

Programa de Doctorado en Ingeniería Matemática,
Estadística e Investigación Operativa por la
Universidad Complutense de Madrid y la
Universidad Politécnica de Madrid



Simulación Social mediante Inteligencia Artificial y Teoría de Juegos

*Aplicación al estudio del efecto de las emociones
en las comunicaciones en redes sociales*

TESIS DOCTORAL

Óscar Serrano Cuéllar

Directores:
Juan Antonio Tejada Cazorla
José Francisco Vélez Serrano

Octubre 2025



UNIVERSIDAD
COMPLUTENSE
MADRID

**DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS
PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR**

D./Dña. _____, estudiante en el Programa de Doctorado _____, de la Facultad de _____ de la Universidad Complutense de Madrid, como autor/a de la tesis presentada para la obtención del título de Doctor y titulada:

y dirigida por: _____

DECLARO QUE:

La tesis es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, de acuerdo con el ordenamiento jurídico vigente, en particular, la Ley de Propiedad Intelectual (R.D. legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, modificado por la Ley 2/2019, de 1 de marzo, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), en particular, las disposiciones referidas al derecho de cita.

Del mismo modo, asumo frente a la Universidad cualquier responsabilidad que pudiera derivarse de la autoría o falta de originalidad del contenido de la tesis presentada de conformidad con el ordenamiento jurídico vigente.

En Madrid, a ____ de _____ de 20____

Fdo.: _____

Esta DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD debe ser insertada en la primera página de la tesis presentada para la obtención del título de Doctor.

Dedicatoria

Dedico esta Tesis a ...

Agradecimientos

I want to thank...

Resumen

Simulación Social mediante Inteligencia Artificial y Teoría de Juegos.

Introducción. En un mundo cada vez más globalizado, resulta cada vez más difícil sustraerse a la influencia que ejercen los demás sobre un mismo. Es por ello que el estudio de cómo el individuo se ve afectado por los sujetos con los que se relaciona para generar un comportamiento global tiene un interés especial. Esto es especialmente cierto en la actualidad, con un uso generalizado de modernas plataformas digitales que nos permiten una inmediatez en la comunicación. La difusión de noticias falsas, por ejemplo, puede ejercer un efecto determinante a la hora de generar un clima de opinión en la sociedad.

Estudios recientes apuntan a que las emociones juegan un papel importante en procesos de difusión viral en una red social como éste, y determinan en buena parte nuestras decisiones. Por ello nos planteamos el estudio de este tipo de fenómenos de difusión, en los que las emociones pueden ejercer un papel clave. Para ello enriqueceremos la Tesis con el planteamiento un modelo de difusión emocional propio, adaptando algunas teorías que hemos encontrado en la literatura.

El estudio de estructuras de comunicación ha recaido tradicionalmente en la Teoría de grafos. Sin embargo los tiempos computacionales de cualquier algoritmo que opere sobre dicho tipo de representación puede hacerse impracticable, especialmente con redes grandes. Es por ello que en esta tesis nos planteamos el estudio de Redes Sociales mediante el empleo de técnicas de Inteligencia Artificial. En concreto planteamos como aportación metodológica la utilización de procedimientos de visión artificial. El procedimiento consistirá en generar patrones de comunicación por Simulación y entrenar con ellos Redes Neuronales diseñadas para procesar información visual. El objetivo: detectar y clasificar los agentes que se encuentran en la red. Esto nos permite estudiar las zonas de conflictividad en la red.

Objetivos y resultados. Un primer objetivo consistirá en reproducir fenómenos de difusión viral en los que las emociones juegan un papel determinante.

Emplearemos métodos de Simulación, donde los agentes utilizan procedimientos de Aprendizaje por Refuerzo para tomar decisiones y en los que las emociones juegan un papel determinante. Nos apoyaremos en la Teoría de Juegos para diseñar el mecanismo de interacción entre ellos: dilemas sociales iterados.

Un segundo objetivo posterior consistirá en procesar la información generada por simulación mediante Redes Neuronales adaptadas a trabajar con información visual. Estudiaremos el papel de cada tipo de agente que introduciremos a la hora de generar patrones de comunicación diferenciales. Al ser los datos generados por simulación, disponemos de la identidad de cada agente y los patrones que genera en la red. Por ello emplearemos Aprendizaje Supervisado para entrenar las neuronas.

Conclusiones. En primer lugar hemos procedido al diseño de un agente emocional basándonos en modelos existentes en la literatura. Los resultados experimentales nos permiten detectar en las simulaciones resultados similares a los obtenidos por agentes típicos en Teoría de Juegos como Tit-for-Tat.

El análisis posterior lo hemos llevado a cabo empleando Redes Neuronales utilizadas en Procesos de Segmentación de Imágenes. Los experimentos llevados a cabo con redes regulares conducen a resultados muy buenos en detección de agentes. Esto es especialmente acentuado en la fase de difusión intermedia, cuando aún podrían llevarse a cabo medidas correctoras.

Abstract

Social Simulation by means of Artificial Intelligence and Theory of Games.

Introduction. In an increasingly globalized world, it is becoming more and more difficult to escape the influence exerted by others on oneself. This is why the study of how the individual is affected by the subjects with whom he or she relates to generate global behavior is of special interest. This is especially true nowadays, with a widespread use of modern digital platforms that allow us an immediacy in communication. The dissemination of fake news, for example, can exert a determining effect in generating a climate of opinion in society.

Recent studies suggest that emotions play an important role in viral diffusion processes in a social network such as this one, and largely determine our decisions. For this reason, we propose to study this type of diffusion phenomena, in which emotions can play a key role. To this end, we will enrich the Thesis with our own model of emotional diffusion, adapting some theories that we have found in the literature.

The study of communication structures has traditionally relied on graph theory. However, the computational times of any algorithm operating on this type of representation can become impractical, especially with large networks. That is why in this thesis we propose the study of Social Networks using Artificial Intelligence techniques. Specifically, we propose as a methodological contribution the use of artificial vision procedures. The procedure will consist of generating communication patterns by simulation and training with them Neural Networks designed to process visual information. The objective: to detect and classify the agents in the network. This allows us to study the areas of conflict in the network.

Objectives and results. A first objective will be to reproduce viral diffusion phenomena in which emotions play a determining role.
We will use Simulation methods, where agents use Reinforcement Learning procedures to make decisions and in which emotions play a determinant role. We will rely on Game Theory to design the interaction mechanism between them: iterated social dilemmas.

A second objective will consist of processing the information generated by simulation using Neural Networks adapted to work with visual information. We will study the role of each type of agent that we will introduce when generating differential communication patterns. As the data is

generated by simulation, we have the identity of each agent and the patterns it generates in the network. Therefore, we will use Supervised Learning to train the neurons.

Conclusions. First of all, we have proceeded to the design of an emotional agent based on existing models in the literature. The experimental results allow us to detect in the simulations results similar to those obtained by typical agents in Game Theory such as Tit-for-Tat.

The subsequent analysis has been carried out using Neural Networks used in Image Segmentation Processes. Experiments carried out with regular networks lead to very good results in agent detection. This is especially accentuated in the intermediate diffusion phase, when corrective measures could still be carried out.

Índice de figuras

1.1.	<i>Grupos emocionales detectados en el estudio Framingham (Fowler, J. H. & Christakis, N. A. (2008) [56]).</i>	2
1.2.	<i>Representación en una red regular del Juego Iterado del Dilema del prisionero (los agentes que cooperan aparecen señalados como 'C').</i>	2
1.3.	<i>Segmentación de una imagen - Facebook IA (Deep Learning for Generic Object Detection: A Survey - Liu, L. et al (2020)) [107].</i>	3
1.4.	<i>La segmentación es en el fondo un problema de clasificación correcta de los píxeles de los que está compuesta la imagen, en cada uno de los distintos tipos de entidades en cuyo reconocimiento ha sido entrenada la red neuronal: (class 0: grass, 1: person, 2: sheep, 3:dog).</i>	3
1.5.	<i>Representación en una red regular de la difusión de la cooperación.</i>	4
1.6.	<i>Partiendo de la representación de la activación emocional de la red social (codificandola con colores), buscamos un resultado: ser capaces de clasificar los nodos de la red (mutante:1).</i>	4
1.7.	<i>Representación a través de un grafo de las relaciones en una Red Social.</i>	5
1.8.	<i>Algunos tipos de estructuras de una Red Social.</i>	6
1.9.	<i>Representación como cuadrícula de una red regular.</i>	6
1.10.	<i>Rejilla de simulación (grid). Aparece resaltada una de sus celdas y su vecindario inmediato.</i>	7
1.11.	<i>Simulación de la aparición de segregación racial mediante autómatas.</i>	7
1.12.	<i>Evolución de la cultura tras 20.000, 40.000 y 80.000 ciclos (Axelrod(1997) [3]).</i>	8
1.13.	<i>a) Dilema de los bienes públicos b) Dilema tragedia de los comunes.</i>	8
1.14.	<i>Hipótesis.</i>	11
1.15.	<i>Esquema de los bloques que conforman la Tesis.</i>	13
2.1.	<i>Representación de una secuencia de decisiones. Partimos en la etapa 0 de un determinado estado inicial X_0, y cada decisión conduce a un nuevo estado ($X_1, X_2, \dots, X_k, \dots, X_N$). Las decisiones que se pueden tomar son propias de cada estado ($u_k \in U(x_k)$), y llevan aparejadas unas consecuencias, que en este caso (al ser un entorno determinista) son del tipo: $g(x_k, u_k)$.</i>	18
2.2.	<i>Si el número de estados-acciones es reducido podemos emplear una Representación tabular.</i>	21
2.3.	<i>Manejar explícitamente el valor de cada par estado acción puede ser muy costoso. Una solución consiste en utilizar una función que aproxime la relación entre los valores.</i>	21
2.4.	<i>Aproximación $\tilde{Q}^*(i, u)$ cuando el sistema se encuentra en un determinado estado $x_k = i$, y la decisión subóptima a la que conduce: \tilde{u}. (Adaptado de Bertsekas, D. (2019) [26])</i>	22

ÍNDICE DE FIGURAS

2.5. En un entorno determinista, la función de valor se encuentra optimizando la suma del costo/ganancia a un paso ($g_{ij}(u)$, $u \in \{u_1, \dots, u_n\}$) más el valor del estado destino $J_{k+1}^*(j)$	29
2.6. En un entorno aleatorio, la función de valor se encuentra optimizando el valor esperado de la suma del costo/ganancia a un paso ($g_{ij}(u)$, $u \in \{u_1, \dots, u_s\}$) más el valor óptimo del estado destino ($J_{k+1}^*(1), \dots, J_{k+1}^*(n)$). Suponemos que la incertidumbre reside en el estado final al que se llega tras tomar la decisión, no en el coste asociado a dicha transición ($g_{ij}(u)$).	30
2.7. Interpretación geométrica del operador de Bellman T_μ . (Ver código Python en el Apéndice).	32
2.8. Interpretación geométrica del operador de Bellman T . (Ver código Python en el Apéndice).	33
2.9. Representación del supuesto de contractividad (se utiliza una sola dimensión para facilitar la visualización). También aparece indicado el punto fijo J_μ , que cumple $T_\mu J = J$ (Adaptado de Bertsekas, D. (2022) [21] - pag. 8).	41
2.10. Ecuación de Bellman. En un entorno no determinista, la función de valor se encuentra minimizando/maximizando la esperanza: del costo/ganancia a un paso ($g_{ij}(u)$, $u \in \{u_1, u_2, \dots, u_s\}$) más el valor óptimo del estado destino ($J^*(j)$, $j = 1, 2, \dots, n$).	46
3.1. Representación del algoritmo VI (Value Iteration) y el punto al que convergen la secuencia de iteraciones. El punto fijo J^* , que cumple $TJ^* = J$, aparece señalado en el gráfico (Adaptado de Bertsekas, D. (2022) [21] - pag. 8 y 71).	53
3.2. Representación del algoritmo PI (Policy Iteration) y el punto al que convergen la secuencia de iteraciones. El punto fijo J^* , que cumple $TJ^* = J$, aparece señalado en el gráfico (Adaptado de Bertsekas, D. (2022) [21] - pag. 71).	58
3.3. Empleo de costes intermedios para estimar el valor de estados visitados en secciones de la trayectoria muestral. En la figura el estado i_0 ha sido visitado m veces.	63
3.4. Simulando K trayectorias ($1, 2, \dots, k, \dots, K$) podemos obtener una estimación de la función de valor en los estados que son visitados en ellas (en la figura aparece un estado i_0).	64
3.5. Acotación de los valores r_t que proporciona el algoritmo	69
3.6. Empleamos un decaimiento geométrico para los pesos (Adaptado de Barto, R. S. & Sutton, A. G. (2018) [181] - pag. 290).	71
3.7. Dinámica del sistema	77
4.1. Esquema de las disciplinas que confluyen en el Reinforcement Learning.	85
4.2. Esquema de las equivalencias entre la notación empleada en Programación Dinámica y la utilizada en Aprendizaje por Refuerzo.	86
4.3. Transiciones entre estados.	89
4.4. El agente dispone, como información del Entorno, de los valores (s_t, r_t) , donde s_t alude al estado en el instante ' t ', y r_t el feedback en forma de recompensa. En base a estos valores, lleva a cabo una acción: a_t . Esto se traduce en una transición a otro estado s_{t+1} . La consecuencia para el agente consiste en recibir una recompensa: r_{t+1}	89
4.5. Conforme hacemos crecer los pasos hacia delante que tenemos en cuenta a la hora de realizar la estimación, nos aproximamos al caso límite: el método de Monte Carlo, que ha de disponer de toda la trayectoria antes de realizar los cálculos.	94
4.6. Algunos de los algoritmos empleados en Aprendizaje por Refuerzo (Reinforcement Learning).	95

4.7. Esquema de las interacciones de un agente implementando un sistema actor-crítico (Adaptado de: Tsitsiklis, J. & Bertsekas, D. (1996) [27] - pag. 36).	101
4.8. Representación de diversos tipos de procesos.	104
5.1. Algunas funciones de activación usuales (donde: $net = W \cdot input + b$).	114
5.2. Representación gráfica del procesamiento de información que realiza una Red Neuronal. Se han empleado unos inputs de dimensión ' p ': $\mathbf{x} = (x_1, x_2, \dots, x_p)$ y dos capas ocultas de neuronas. La neurona final genera un valor \mathbf{y} . En todos los casos se ha empleando una función de activación tangente hiperbólica.	114
5.3. Obtención del vector de parámetros óptimo θ^* .	115
5.4. Algunas arquitecturas de Redes Neuronales (Esquemas adaptados de Principe, J. C. et al. (1999) [146], Goodfellow, I. et al. (2016) [62] y Brunton, S. L. & Kutz, J. N. (2022) [38]).	117
5.5. Funcionamiento del bufer, y extracción posterior de una muestra para entrenar la Red.	118
5.6. Funcionamiento del bufer, y extracción de una muestra ponderada por el efecto que la observación tiene en el error.	121
5.7. Estructura de la Red Neuronal empleada en Dueling DQN.	122
5.8. Esquema de la implementación del modelo A3C.	125
6.1. Entorno multiagente. La acción conjunta de todos los agentes (\bar{a}_t), genera una señal por parte del entorno en forma de recompensa (\bar{r}_t) y una alteración de su estado (s_t), lo que sirve a los agentes como refuerzo para optimizar la elección de sus acciones (gráfico adaptado de Nowe et al. (2012) [130]).	129
6.2. vonNeumann Morgenstern (FOTO)	131
6.3. Esquema del tipo de modelos según el número de agentes y el tipo de conjunto de estados. Además, cuando el agente no tiene la posibilidad de conocer con precisión el estado del sistema, nos encontramos ante POMDP (Partially Observable Markov Decision Processes).	132
6.4. Esquema de la matriz de pagos en un juego matricial.	134
6.5. Esquema con la matriz de pagos de un juego matricial (r_{ij}, c_{ij}), y las probabilidades de cada estrategia ($\alpha, 1 - \alpha; \beta, 1 - \beta$) por jugador (J_1, J_2 : jugador fila/columna).	141
7.1. Prueba de modelo de las transiciones emocionales de un agente.	153
7.2. Si el número de estados-acciones es reducido podemos emplear una Representación tabular.	158
7.3. Manejar explícitamente el valor de cada par estado acción puede ser muy costoso. Una solución consiste en utilizar una función que aproxime la relación entre los valores.	158
7.4. Aproximación $\tilde{Q}^*(i, u)$ cuando el sistema se encuentra en un determinado estado $x_k = i$, y la decisión subóptima a la que conduce: \tilde{u} . (Adaptado de Bertsekas, D. (2019) [26])	159

ÍNDICE DE FIGURAS

Índice de cuadros

2.1. <i>Expresión de la función $J_\pi(x_0)$ en los escenarios más comunes.</i>	19
2.2. <i>Esquema con las técnicas de resolución del problema de optimización planteado.</i> . .	24
3.1. <i>Representación tabular del conjunto de estados-acciones.</i>	51
4.1. <i>Equivalencias entre los términos empleados en Reinforcement Learning y Programación Dinámica (PD) (y Control Óptimo).</i> (Adaptado de Bertsekas, D. (2019) [26] - pag. 44).	87
6.1. <i>Principales algoritmos clásicos multiagente</i>	140
D.1. <i>Representación tabular de la función $Q(s, a)$.</i>	179
D.2. <i>Representación tabular de la función $Q(s, a)$. Algoritmo Q-Learning. Actualizamos mediante el máximos valor de la función para el estado s'.</i>	182

ÍNDICE DE CUADROS

Índice general

Declaración de autoría y originalidad	III
Dedicatoria	v
Agradecimientos	VII
Resumen	IX
Abstract	XI
Índice de figuras	xv
Índice de cuadros	xvii
1. Introducción	1
1.1. Conceptos	5
1.2. Antecedentes	7
1.3. Aportaciones de la Tesis	9
1.4. Hipótesis y objetivos	10
1.4.1. El tipo de información influye sobre el grado de difusión	10
1.4.2. Las emociones actúan como un mecanismo de aprendizaje	10
1.5. Estructura de la tesis. - vol I	12
1.6. Estructura de la tesis - vol. II	13
2. Modelización de la toma de decisiones secuenciales	17
2.1. Conceptos y notación	17
2.1.1. Representación de la función de valor.	20
2.1.2. Clasificación de los métodos de solución.	23
2.2. Análisis mediante Programación Dinámica (PD)	25
2.3. Estudio de la convergencia del algoritmo PD	31
2.4. Ecuación de Bellman	44
3. Métodos de solución	51
3.1. Solución exacta: Programación Dinámica	52
3.1.1. Iteración de valor (<i>Value Iteration</i>)	52
3.1.2. Iteración de estrategia (<i>Policy Iteration</i>)	57

ÍNDICE GENERAL

3.2.	Solución aproximada: Métodos de Simulación I (<i>Value based methods</i>)	62
3.2.1.	Método de Montecarlo	63
3.2.2.	Método de las diferencias temporales (<i>TD</i>)	66
3.2.3.	Algoritmo Q-Learning	74
3.3.	Solución aproximada: Métodos de Simulación II (<i>Policy based methods</i>)	81
3.3.1.	Método 1: Optimización del vector de parámetros	81
3.3.2.	Método 2: Utilizar Aprendizaje Supervisado	83
4.	Aprendizaje por refuerzo (RL)	85
4.1.	Conceptos	88
4.2.	Proceso de Decisiones de Markov (MDP)	90
4.3.	MDP Parcialmente Observable (POMDP)	91
4.4.	Funciones para la toma de decisiones del agente	92
4.4.1.	Funciones de valor	92
4.4.2.	Función de estrategia	93
4.5.	Estimación de las funciones de valor	94
4.5.1.	Método de Montecarlo (MC)	95
4.5.2.	Método de las diferencias temporales (TD)	95
4.6.	Estimación de la función de estrategia	95
4.7.	Value based Methods.	97
4.7.1.	SARSA	97
4.7.2.	Q-learning	98
4.8.	Policy Methods.	99
4.8.1.	REINFORCE	99
4.8.2.	REINFORCE with reward-to-go	100
4.8.3.	REINFORCE with baseline	100
4.9.	Actor-Critic Methods.	101
4.10.	Transferencia de aprendizaje: el problema de la representación	101
4.11.	Otras líneas de investigación.	102
4.11.1.	Imitation Learning	102
4.11.2.	Meta Reinforcement Learning	102
4.11.3.	Hierarchical Reinforcement Learning	102
4.11.4.	Imagination	102
4.12.	Procesos de Decisión de Markov	104
4.13.	MDP Parcialmente Observable (POMDP)	107
5.	Aprendizaje por refuerzo profundo (Deep RL)	113
5.1.	Redes Neuronales	113
5.2.	Entrenamiento de una red neuronal	115
5.3.	Arquitecturas de Redes Neuronales	116
5.4.	Utilización de Redes Neuronales en RL	118
5.5.	Value based Methods.	118
5.5.1.	Deep Q-Network (DQN)	118
5.5.2.	Double Deep Q Network (DDQN)	120
5.5.3.	Deep Q-Network with Prioritized Experience Replay (PER)	121
5.5.4.	Dueling Deep Q-Network (Dueling DQN) (D2QN)	122
5.5.5.	Deep Recurrent QN (DRQN)	122
5.6.	Policy Methods.	123
5.6.1.	Proximal Policy Optimization (PPO)	123

5.6.2. Trust Region Policy Optimization (TRPO)	123
5.6.3. Actor-Critic using Kronecker-factored Trust Region (ACKTR)	123
5.7. Actor-critic Methods	124
5.7.1. Advantage Actor-Critic (A2C)	124
5.7.2. Asyncronous Advantage Actor-Critic (A3C)	125
5.7.3. Deep Deterministic Policy Gradient (DDPG)	125
5.7.4. Twin Delayed Deep Deterministic Policy Gradient (TD3)	126
5.7.5. Soft Actor-Critic (SAC)	126
6. Aprendizaje por refuerzo Multiagente (MARL)	129
6.1. Introducción	130
6.2. Conceptos y formalización	132
6.3. Concepto de solución en MARL: Equilibrio de Nash	135
6.4. Clasificación de los algoritmos de solución I: según el tipo de entrenamiento	137
6.5. Clasificación de los algoritmos de solución II: según el tipo de interacción	138
6.6. Algoritmos mixtos clásicos	141
6.6.1. Algoritmo IGA (Intinitesimal Gradiente Ascent)	141
6.6.2. Algortimo PHC (Policy Hill Climbing)	142
6.6.3. Algoritmo WoLF-PHC	143
6.7. Algoritmos mixtos recientes	147
6.7.1. Mean Field Q-Learning	147
7. Diseño de un agente dotado de emociones	151
7.1. Teorías sobre emociones	151
7.2. Influencia de las emociones en el flujo de información	151
7.3. Agente emocional	152
7.4. Integración en el esquema de RL	153
7.5. Representación del conocimiento	153
7.6. Transferencia de conocimiento y <i>feature representation</i>	154
7.6.1. Representación de la función de valor.	157
8. Estudio de los agentes influyentes en una Red Social	161
8.1. Aplicación de Aprendizaje por refuerzo	161
8.2. Análisis estratégico de un agente emocional	161
8.3. Viralización de información	162
9. Resultados	163
9.1. Análisis estratégicos	163
9.2. Viralización de información en una red social	164
9.3. Distorsión de la información y consecuencias sobre la difusión	165
9.4. Polarización en una Red Social	165
10. Conclusiones	167
10.1. Efecto de las emociones en las comunicaciones	167
10.2. Viralización de la información	167

APÉNDICES:

ÍNDICE GENERAL

A. Estadística y Probabilidad	169
B. Teoría de Juegos	171
C. Redes Sociales	173
D. Código PYTHON	175
D.1. Implementación del algoritmo SARSA	175
D.2. Implementación del algoritmo Q-Learning	181
D.3. Análisis Geométrico del Operador de Bellman J_μ	182
D.4. Resolución de Sistemas Lineales - Cálculo de J_{μ^k}	184
Referencias	187
Índice alfabético	205
Lista de Símbolos y Acrónimos	207
Glosario	211

Capítulo 1

Introducción

El grado de globalización actual de nuestra sociedad facilita la influencia entre individuos. El desarrollo de las plataformas de comunicación on-line potencia aún más el proceso, debido especialmente a la inmediatez en el contacto que permiten. La información que fluye por la red tiene el efecto potencial de moldear nuestra forma de interpretar los acontecimientos a nuestro alrededor e influir en nuestra conducta, con los riesgos que esto conlleva. Es por ello que el estudio de la forma más extrema de difusión como es la difusión viral tiene un interés especial.

Algunos estudios apuntan, por ejemplo, a su papel en fenómenos como la movilización política y la generación de un clima de opinión (Bond et al. (2012)) [31]. Un peligro cada vez más real es la utilización de métodos de comunicación masiva como un medio para la manipulación. Así lo atestiguan estudios sobre la propagación de noticias falsas (Vosoughi et al. (2018)) [193], o la polarización de la población en posiciones políticas extremas, con el uso de *bots* para inflamar los contenidos de la red (Stella et al.) [178].

La modelización de este tipo de procesos de difusión viral es fundamental. Entender los mecanismos que subyacen, puede hacernos capaces de desarrollar medidas que puedan hacer frente a efectos dañinos como los que se acaban de mencionar.

Los estudios clásicos sobre estructuras de comunicación han recurrido fundamentalmente a su representación mediante grafos. Sin embargo el tratamiento algorítmico de este tipo de objetos matemáticos puede llegar a ser muy complejo, generando tiempos computacionales muy elevados, tal y como ponen de manifiesto trabajos clásicos en la difusión de información. Tenemos por ejemplo el planteamiento del problema de viralizar un producto de marketing en sitios de compartir información por Domingos & Richardson (2001) [50] y (2002) [149]. El Modelo de referencia lo tenemos en Kempe, Kleinberg y Tardos ([87] (2003) y [86] (2005)), donde se aborda la complejidad computacional elevada del problema.

Es por esto por lo que nos planteamos la introducción de aportaciones metodológicas que permitan estudiar el comportamiento de una red social sin acudir a este tipo de algoritmos que operan directamente sobre la estructura de la red.

El proceso de investigación que hemos llevado a cabo está centrado en dos objetivos:

1. Simulación de procesos de difusión viral en una red social.
2. Detección de los individuos (agentes) influyentes de la red.

Si observamos algunos de los resultados de investigaciones académicas, encontramos además que en muchos de estos tipos de procesos de difusión parecen tener un papel esencial las emociones. Algunas investigaciones reflejan este fenómeno de la difusión de emociones (Miller, M. et al. (2011) [112]), y estudios clásicos como el de Framingham permiten descubrir relaciones interesantes, como que los estados emocionales parecen contagiar a las personas cercanas (Fowler, J. & Christakis, N. (2008) [56]) (ver fig. 1.1).

La modelización que llevaremos a cabo de procesos de comunicación, se llevará a cabo sobre redes sencillas en forma de cuadrícula, introduciendo el efecto de las emociones de los agentes.

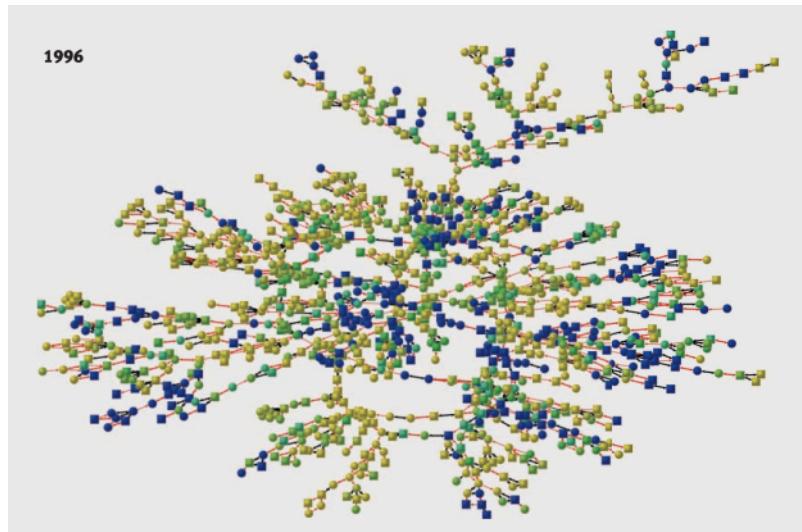


Figura 1.1: *Grupos emocionales detectados en el estudio Framingham (Fowler, J. H. & Christakis, N. A. (2008) [56]).*

FASE de SIMULACIÓN: Para integrar las emociones en la red, emplearemos como desencadenante de todo el proceso de actividad en la Red Social un juego: el Dilema del Prisionero, que los agentes jugarán de manera iterada. Pese a su aparente sencillez, permite esquematizar el tipo de relaciones que podríamos encontrar: la alegría por ser correspondido, el dolor de la traición, el gozo por vengar una afrenta previa o el pesar por haber tratado injustamente a un oponente. La difusión de la cooperación vendría a representar un proceso de difusión viral (fig. 1.2).

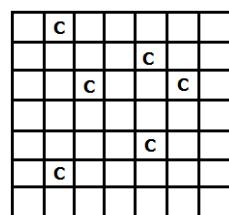


Figura 1.2: *Representación en una red regular del Juego Iterado del Dilema del prisionero (los agentes que cooperan aparecen señalados como 'C').*

El análisis estratégico del juego lo llevaremos a cabo introduciendo agentes mutantes que alteren la actividad de la red social, de modo similar a como lo harían en una red de comunicaciones quienes perturben las interacciones introduciendo conflictividad ('*trolls*'), o intentan imponer un clima de opinión.

FASE de DETECCIÓN DE AGENTES: utilizaremos técnicas de Inteligencia Artificial. El procedimiento que emplearemos consistirá en codificar la actividad emocional de la red social mediante colores, transformando la red regular de la que partiremos en una imagen. Posteriormente utilizaremos técnicas de procesamiento de imágenes para detectar patrones. Es decir, estos procesos de interacción simulados nos servirán para entrenar distintos tipos de Redes Neuronales, con la intención de detectar los patrones de actividad que generan los diferentes tipos de agentes.



Figura 1.3: Segmentación de una imagen - Facebook IA (*Deep Learning for Generic Object Detection: A Survey - Liu, L. et al (2020)*) [107].

La técnica elegida en concreto será la de Segmentación de imágenes (ver fig. 1.3). Este tipo de procedimientos nos permiten entrenar una Red Neuronal para detectar cierto tipo de entidades, localizarlas y trazar su silueta en la imagen.

Si observamos más detenidamente, vemos en la figura 1.4 que la detección de la forma de las distintas entidades que existen en la imagen, es en el fondo un problema de clasificación de todos y cada uno de los píxeles que forman parte de ella.

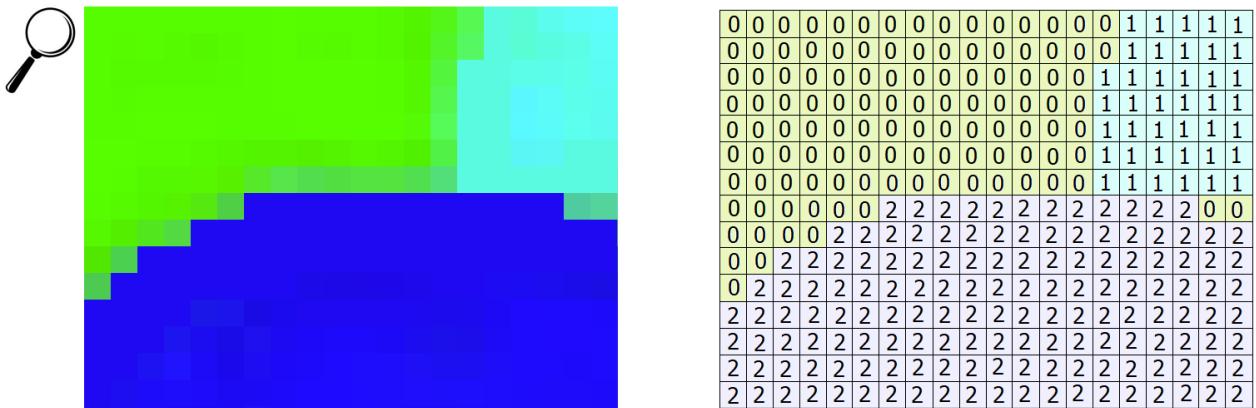


Figura 1.4: La segmentación es en el fondo un problema de clasificación correcta de los píxeles de los que está compuesta la imagen, en cada uno de los distintos tipos de entidades en cuyo reconocimiento ha sido entrenada la red neuronal: (class 0: grass, 1: person, 2: sheep, 3:dog).

Nosotros emplearemos dicho procedimiento para tratar de clasificar todos los nodos de la red y desvelar con ello la identidad de los nodos conflictivos. Todo lo anterior se condensa y concreta en las siguientes líneas.

Problema a tratar. El problema que pretendemos abordar es el estudio de la actividad de difusión viral en una Red Social. La situación que proponemos es la existencia de varios tipos de agentes. Uno de ellos intenta imponer un clima conflictivo al resto de agentes, de naturaleza emocional, e influenciables, y que pueden verse arrastrados a la situación que se pretende imponer. Nos interesa especialmente detectar la presencia de estos mutantes y su localización en la Red.

En primer lugar utilizaremos el Dilema del Prisionero Iterado sobre una cuadrícula, en linea con estudios clásicos sobre difusión de la cooperación, para representar un fenómeno de difusión viral (fig. 1.5). La naturaleza del juego nos permite introducir un agente emocional que reaccionará a los distintos resultados del juego, llevando a cabo un aprendizaje a lo largo del tiempo. Estos agentes representan una masa fácilmente influenciable que facilita en este caso la propagación de la conducta cooperativa, pero que pueden verse arrastrados en sentido opuesto. La introducción de agentes mutantes introduce el factor de conflictividad.

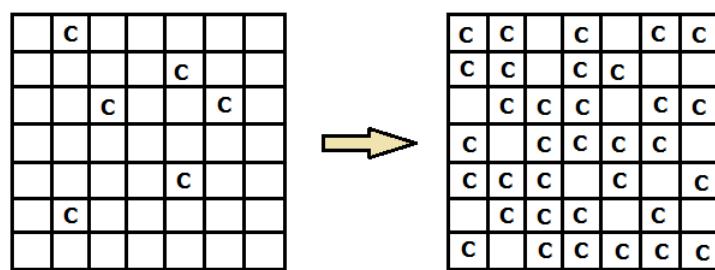


Figura 1.5: Representación en una red regular de la difusión de la cooperación.

Posteriormente el objetivo consistirá en detectar estos focos de conflictividad, donde los agentes están intentado perturbar la actividad, y generar una polarización de la red, y en último extremo evitar la expansión de la estrategia cooperativa.

La actividad emocional de la red será representada mediante imágenes, las cuales servirán para entrenar Redes Neuronales en la detección de los patrones de actividad que generan los agentes.

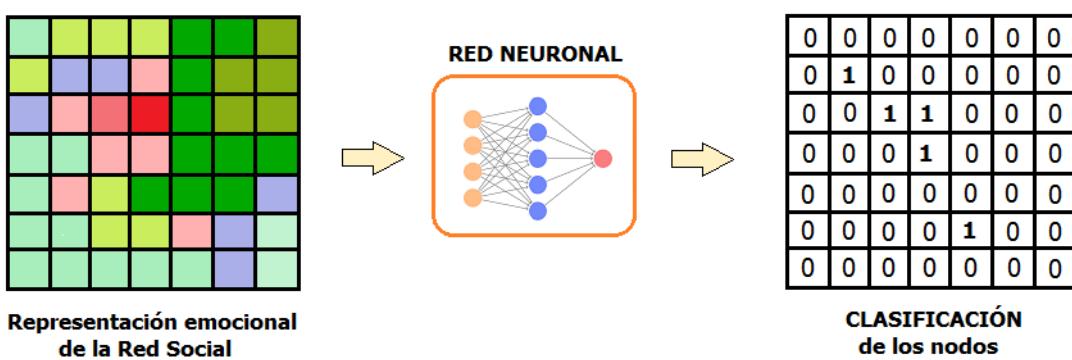


Figura 1.6: Partiendo de la representación de la activación emocional de la red social (codificandola con colores), buscamos un resultado: ser capaces de clasificar los nodos de la red (mutante:1).

Empleando técnicas de segmentación de imágenes trataremos de clasificar todos los nodos de la red y desvelar con ello la identidad de los nodos conflictivos (ver fig. 1.6). Al centrar el esfuerzo en la fase de entrenamiento, la detección de agentes se realiza automáticamente, evitando los largos tiempos aparejados a los algoritmos sobre grafos.

1.1. Conceptos

Un aspecto clave en cualquier situación social que podamos imaginar es la interacción y comunicación entre individuos: la transmisión de ideas o la adopción de prácticas comunes. En lo que sigue definimos algunos conceptos que aparecerán a lo largo del presente trabajo.

Representación de una Red Social.

Una sociedad puede interpretarse como un conjunto de múltiples elementos que interactúan entre sí. Bajo esta consideración aparece de manera natural un objeto matemático: el grafo. Definido de manera rigurosa consiste en un conjunto G que especifica el conjunto de aristas (E) y vértices (V) que componen la red [8].

Cada uno de los vértices vendría a representar a un individuo o **agente**, y las artistas que los interconectan representan las relaciones existentes entre ellos.

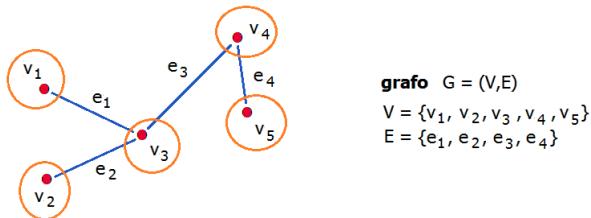


Figura 1.7: *Representación a través de un grafo de las relaciones en una Red Social.*

Vemos en la Figura 1.7, que una simple inspección visual puede proporcionar información del nivel de complejidad de las relaciones existentes en la Red Social. Si atendemos al grado de los vértices vemos que el grado de v_3 es igual a 3. El resto de los nodos tiene grado 1 menos v_4 que tiene grado 2.

Interacción.

En sentido general puede tratarse de la transmisión de hábitos, habilidades, creencias, ideas, valores, por ejemplo. La adopción de una determinada conducta (hábito o idea) puede modelizarse como un proceso de contagio, y resulta importante el estudio de cómo se extienden dichos comportamientos en una determinada población.

Individuos influyentes.

Entendemos que existe influencia cuando un individuo (agente) altera su comportamiento como consecuencia de las interacciones que lleva a cabo con otros individuos con los que está en contacto.

En sentido general se trata de la adopción de un determinado hábito, idea o comportamiento, o también la adquisición de un determinado producto, como consecuencia de las relaciones que entabla cada individuo con los que le rodea.

La primera dificultad consiste en distinguir si los individuos se comportan como lo hacen fruto de la influencia entre ellos, o existe una similitud previa, tal como han recalcado algunos autores (Anagnostopoulos, A. et al. (2008) [1] y Crandall, D. et al (2008) [47]).

Partiremos de la hipótesis de que efectivamente existe influencia entre agentes. Algunos de los estudios que han detectado este efecto son los llevados a cabo en sistemas de recomendación (Leskovec et al. (2006) [101]), así como los estudios llevados a cabo sobre la influencia de comunicaciones en redes sociales y el grado de movilización política (Bond, R.M. et al (2012) [31]).

Estructura de una Red Social.

Si atendemos al tipo de conexiones que unen a los agentes de la red social podemos distinguir varios tipos de redes (ver fig. 1.8). Las redes 'mundo pequeño' (*small world*) son aquellas en las que todos los nodos están interconectados a través de un número reducido de conexiones. En una red aleatoria los enlaces entre cada par de nodos se crean con una cierta probabilidad ' p '. Otro tipo de red interesante es la red libre de escala. En este tipo de redes, pese a que el grado de conexión es bastante bajo, existen algunos nodos diferenciales que están conectados a un gran número de nodos, lo que les convierte en claves para la difusión de información por ejemplo.

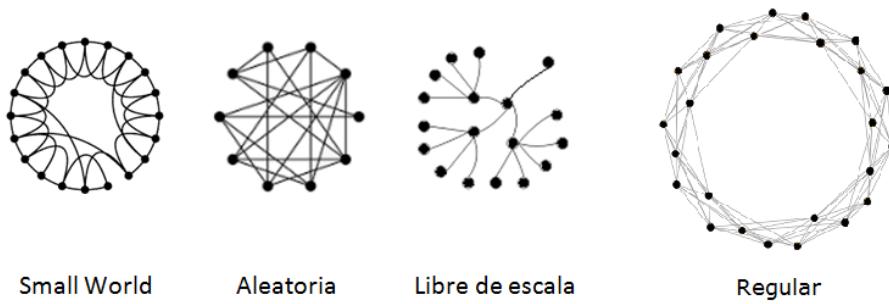


Figura 1.8: *Algunos tipos de estructuras de una Red Social.*

El tipo de red para el que desarrollaremos el presente trabajo es un tipo de red regular, que tiene la característica de que puede ser representada como una rejilla o '*grid*'. Esto nos permitirá tratar la red directamente como una imagen. A su vez sobre este tipo de estructuras es sobre la que se han desarrollado los trabajos clásicos sobre, por ejemplo, la evolución de la cooperación en Teoría de Juegos (tema que trataremos extensamente más adelante).

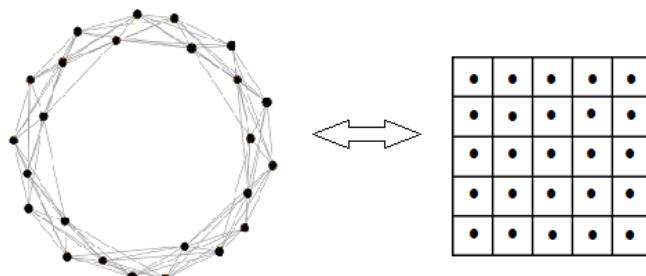


Figura 1.9: *Representación como cuadrícula de una red regular.*

Por tanto, la tesis se desarrollará sobre una estructura de red regular que admite una representación en cuadrícula, tal y como aparece en la figura 1.9, planteándonos posteriormente la extensión a otros tipos de estructuras más complejas.

1.2. Antecedentes

Los orígenes de la simulación computacional de fenómenos sociales habría que situarlos en la idea de von Neumann de los autómatas celulares [123]. Básicamente se trata de una rejilla regular formada por celdas, cada una de las cuales puede tomar un número finito de estados, y donde en cada uno de una serie de pasos sucesivos, el estado de cada celda varía de acuerdo a ciertas reglas y la influencia del estado de las celdas vecinas. La figura 1.10 permite hacerse una idea.

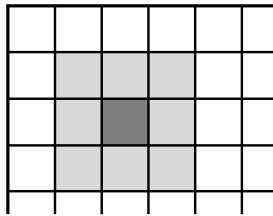


Figura 1.10: *Rejilla de simulación (grid)*. Aparece resaltada una de sus celdas y su vecindario inmediato.

Pese a su aparente sencillez, esta representación puede modelizar procesos bastante complejos. Buscando la representación de la evolución de un ser vivo, el procedimiento sería puesto en práctica en el trabajo clásico de John Conway: 'El Juego de la vida' (Conway, J. (1970) [61]), poniendo de manifiesto que reglas de aplicación local pueden generar la emergencia de patrones globales en el tablero. Thomas Schelling emplearía la misma técnica posteriormente para el estudio del surgimiento de la segregación en una sociedad (Schelling, T. (1981) [161]), tal como se puede ver en la figura 1.11. Vemos que basta con inducir en los individuos una preferencia por otros vecinos cercanos similares para observar este efecto de separación. Es decir: agentes que sólo procesan información local, generan efectos macroscópicos en toda la red (Schelling, T. - "Micromotives and Macrobbehavior" (2006) [162]).

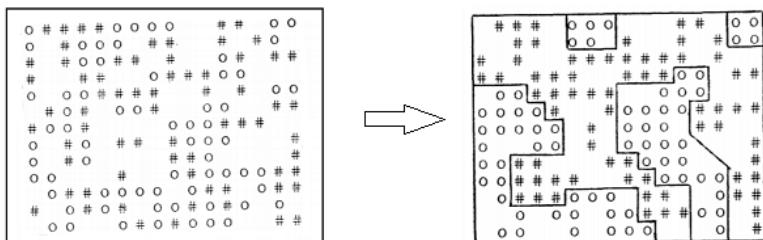


Figura 1.11: *Simulación de la aparición de segregación racial mediante autómatas*.

Simulación de fenómenos sociales y difusión de la cooperación. El concepto de autómata sería extendido gradualmente de manera satisfactoria a otras áreas como la Teoría de Juegos (juegos iterados basados en el Dilema del Prisionero), dando lugar a análisis sobre el surgimiento de la cooperación por Axelrod en su clásico estudio sobre el tema (Axelrod, R. (1981) [4]). Análogo enfoque sería posteriormente utilizado en el campo de la Biología por Martin Nowak (Nowak, M. (2006) [127]), y John Maynard Smith (Smith, J. M. (1979) [177]), cuyos resultados arrojarían luz sobre la manera en que evolucionan las especies.

Axelrod, años después de su estudio sobre la cooperación, llevaría a cabo análisis sobre la diseminación de la cultura empleando nuevamente simulaciones computacionales (Axelrod, R. (1997) [3]). En su artículo, el autor trata el mecanismo de influencia social convergente empleando un modelo basado en agentes. Con él pretende explicar el fenómeno consistente en que, pese a la tendencia a la convergencia en creencias, actitudes y comportamiento (fruto del contacto continuado entre personas), las diferencias entre individuos y grupos persisten en cierto grado (fig. 1.12).

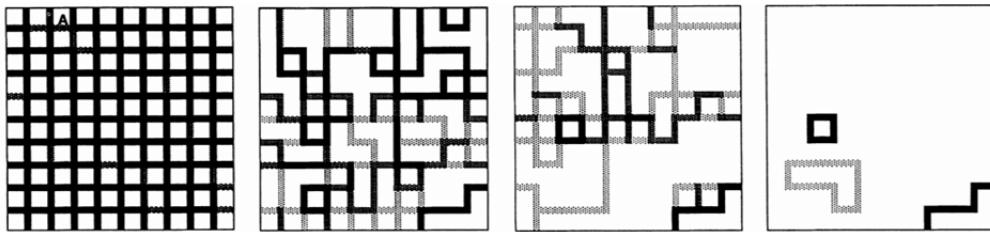


Figura 1.12: *Evolución de la cultura tras 20.000, 40.000 y 80.000 ciclos (Axelrod(1997) [3]).*

Planteado explícitamente como un dilema social secuencial tenemos el trabajo de referencia de Leibo et al. (2017) [100], que utiliza métodos de aprendizaje por refuerzo. Siguiendo este trabajo inicial tenemos una serie de artículos que explotan algunos de los dilemas que plantea, para los que desarrolla un entorno para generar las simulaciones (ver figura 1.13). Tenemos por ejemplo el estudio de Eccles, T. (2019) [51] sobre el aprendizaje de la reciprocidad; o el de Hughes, E. (2018) [80], en el que introduce la aversión a la desigualdad como factor que induce la aparición de comportamientos prosociales.

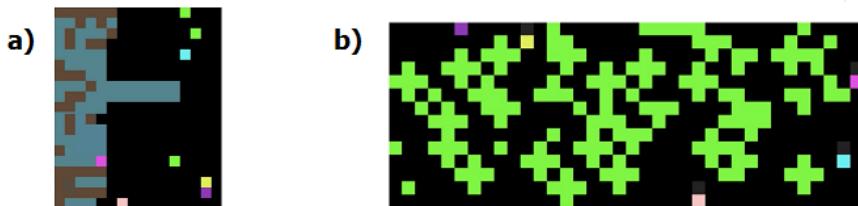


Figura 1.13: a) *Dilema de los bienes públicos* b) *Dilema tragedia de los comunes*.

Otros trabajos destacan por introducir el grado de influencia social como motivación de los agentes a la hora de tomar decisiones (Jaques, N. et al. (2019)) [83], o la formación de grupos de agentes como favorecedora de la emergencia de la reciprocidad (Baker, B. (2020) [6]).

Estudio de las emociones y la difusión de la cooperación. Entre los estudios que se centran en el papel de las emociones y su papel a la hora de inducir la cooperación en el Dilema del Prisionero tenemos por ejemplo los trabajos sobre sentimientos morales y su efecto en el dilema del prisionero iterado en un sistema multiagente (Bazzan, A. et al. (1998) [11]. En este linea los autores proponen un marco de trabajo para el desarrollo de agentes dotados de emociones en el Dilema del Prisionero iterado (Bazzan, A. & Bordini, R. (2001)) [9]. Siguiendo estas lineas tenemos aportaciones como la evolución del comportamiento en agentes dotados de sentimientos morales profundizado en Bazzan, A. et al. (2002) [10], o sobre el papel de los lazos sociales en el desarrollo de la cooperación en el dilema del prisionero iterado tenemos el estudio llevado a cabo por Bazzan, A et al. (2011) [12].

El impulso de imitar estrategias favorables presente en algunos trabajos clásicos sobre evolución de la cooperación, ha sido sustituido por la imitación de emociones, como vía de elevar el bienestar social en juegos espaciales e inducir la propagación de la cooperación (Szolnoki, A. (2011) [184]).

Basándose en procedimientos de aprendizaje por refuerzo tenemos otros trabajos: empleando las emociones como una motivación intrínseca tenemos Sequeira, P. et al. (2011) [168]), y sobre el diseño de este tipo de señales intrínsecas capaces de guiar el aprendizaje en Sequeira, P. et al. (2014) [169]). Podemos encontrar también un modelo con algunas emociones en Broekens, J. et al. (2015) [36]. Y sobre la extensión en el tiempo de las estrategias cooperadoras Peysakhovich, A. & Lerer, A. (2017) [140] y (2018) [141]. En idéntico sentido tenemos también el trabajo ligando de emociones y aprendizaje por refuerzo de Yu et al. (2015) [205].

Considerando la importancia del contagio de emociones y la competición entre estados emocionales, y el papel que podría jugar en la formación de la opinión pública tenemos el enfoque de Fan et al. (2018) [54].

1.3. Aportaciones de la Tesis

Quizás hacer algún comentario sobre la importancia de la investigación. Problemas computacionales en el tratamiento de redes grandes. Transformar en imágenes y utilizar redes neuronales entrenadas en visión por computadora para detectar de manera automática los puntos críticos de la red.

Hay redes para las cuales este tipo de metodología puede ser apropiada. En la actualidad disponemos de métodos de aprendizaje automático que permiten un tratamiento del lenguaje y etiquetar nodos de red con sentimientos. Además algunas redes específicas (X - Twitter, Shina Weibo, etc.) disponen de procedimientos automáticos para etiquetado de estado emocional. Esto permitiría aplicar a este tipo de redes de comunicación los procedimientos que se han desarrollado.

Nos planteamos la introducción de aportaciones metodológicas que permitan estudiar el comportamiento de una red social sin acudir a algoritmos sobre el grafo.

- 1) Planteamiento de un modelo propio de difusión en el que las emociones juegan un papel esencial. Construimos agentes mediante Aprendizaje por Refuerzo, que interactúan entre sí mediante un juego (Teoría de Juegos), lo que permite expresar su sistema de emociones.
- 2) Simulación de la actividad emocional de la red. Codificamos en colores las interacciones entre agentes que se relacionan mediante el juego (dilema del prisionero), cuyas reacciones se van moldeando mediante Aprendizaje por refuerzo, donde las señales de aprendizaje provienen de las emociones de los agentes
- 3) Entrenamiento de Redes Neuronales en la detección de patrones de actividad propios de agentes conflictivos que introducimos en el juego. La aportación metodológica para la detección y clasificación de los agentes de la red será la utilización de Técnicas de Segmentación de imágenes.

1.4. Hipótesis y objetivos

Las hipótesis en que nos basaremos para desarrollar este trabajo se centran en que las interacciones locales en las que intervienen emociones generan un efecto macroscópico que podemos estudiar. Esto nos permite asociar ciertos patrones de comunicaciones a distintos tipos de agentes, los cuales pueden ser localizados en la red por el efecto que provocan. Los trabajos en que nos basamos se describen a continuación, para posteriormente desgranar los objetivos sobre los que vamos a trabajar. [1.4.2 INTERACCION LOCAL / 1.4.1 EFECTO MACROSCÓPICO]

1.4.1. El tipo de información influye sobre el grado de difusión

Artículos como el llevado a cabo en Twitter (Romero, D. M. et al. (2011) [152]) apuntan al fuerte efecto diferencial del tipo de información, sobre el grado de difusión. Otros estudios también apuntan en este mismo sentido (Berger, J. & Milkman, K. (2012) [20]) (también recogido en su libro: [19]), y en concreto, el papel fundamental que ejercen las emociones sobre muchas de nuestras decisiones (Berger, J. (2011) [18]). En este sentido tenemos estudios que buscan cuantificar este efecto como en Miller, M. et al. (2011) [112], sobre los hiperenlaces y cómo se van difundiendo las emociones.

Así mismo tenemos que otros resultados revelan que las noticias tristes se difunden más rápido que las buenas (Wu et al. (2011) [201]). Estudios clásicos llevados a cabo en Facebook y Twitter como el de Ferrara, E. & Yang, Z. (2015) muestran también que las emociones se contagian [55] y que resulta posible detectar la creación de grupos homogéneos en la red y de afinidad según las emociones.

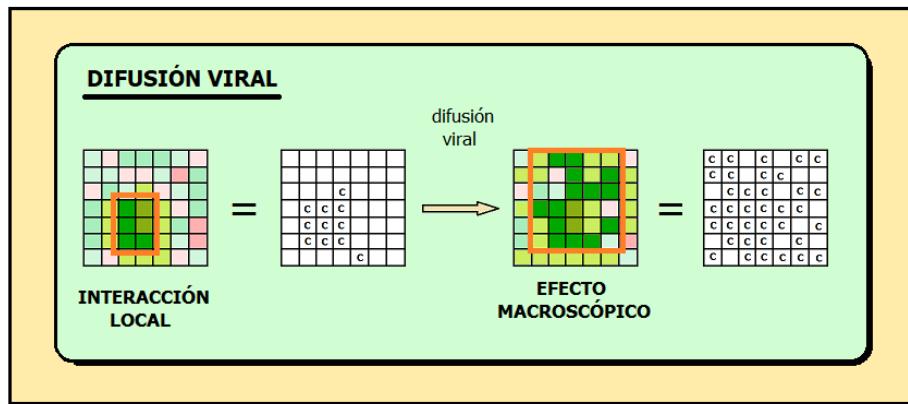
Estudios clásicos como el de Framingan también apuntan a un fuerte efecto contagio (Fowler, J. H. & Christakis, N. A. (2008) [56]). Este efecto no sólo se circunscribe a nuestras relaciones sociales físicas. Algunos estudios revelan que las emociones tienen un fuerte efecto contagioso en las redes sociales *on-line* y plataformas de contacto virtual (Miller et al. (2011) [112]). Estudios llevados a cabo igualmente en Facebook (Kramer et al. (2012) [91] y Kramer et al (2014) [92]) apuntan en el mismo sentido. Otras plataformas como LiveJournal (Zafarani et al. (2010) [207]) también corroboran los análisis anteriores.

1.4.2. Las emociones actúan como un mecanismo de aprendizaje

Sobre la naturaleza de las emociones, su funcionamiento como un refuerzo sobre la conducta, y su efecto sobre la toma de decisiones tenemos estudios como los de Rolls, Edmund T. (2014) [151]. Basándonos en ellos encontramos justificación para el diseño del agente mediante Aprendizaje por refuerzo.

Algunos autores como Ledoux (2020 [99]), resaltan el papel de cada emoción para resolver situaciones con una naturaleza muy específica. La relación de emociones como la ira/miedo con escenarios de amenaza y conflicto (Plutchick, R. (2001) [142]), así como el papel de la alegría en la conducta prosocial, justifica la modelización de relaciones emocionales como uno de los mecanismos de interacción que plantea la Teoría de Juegos.

En concreto el Dilema del Prisionero puede adaptarse bien. Podemos ligar cada emoción con el análisis estratégico que realiza el jugador antes de cada iteración.

Figura 1.14: *Hipótesis*.

■ HIPÓTESIS

H1 : Las emociones son capaces de generar procesos de difusión en una red social

H2 : Las interacciones locales entre agentes generan efectos macroscópicos en la red social

■ OBJETIVO GENERAL 1

Estudio de las interacciones estratégicas entre agentes dotados de emociones, y la expansión viral de su influencia. Esto se traduce en unos objetivos específicos que se detallan a continuación.

OBJETIVOS ESPECÍFICOS

-OE1.1 : Plantear un modelo de difusión viral en el que las emociones jueguen un papel determinante. Emplear para ello juegos de Teoría de Juegos como mecanismo de interacción, y Aprendizaje por Refuerzo en los agentes para la toma de decisiones.

-OE1.2 : Llevar a cabo un estudio de la difusión de la cooperación y el papel que juegan la introducción de mutantes. Relacionar los resultados con el desempeño de agentes clásicos (como Tit-for-Tat).

■ OBJETIVO GENERAL 2

Estudio de los fenómenos de difusión de comportamientos en una Red Social, donde los agentes guían sus decisiones por el impulso de sus emociones.

OBJETIVOS ESPECÍFICOS:

-OE2.1 : Traducir la simulación de los procesos de difusión en imágenes y entrenar con ellas redes neuronales especializadas en visión por computadora.

-OE2.2 : Identificación de los individuos influyentes, planteándolo como un problema de aprendizaje supervisado, empleando los datos generados en las simulaciones.

-OE2.3 : Determinar la viabilidad de generalizar el procedimiento de análisis a redes sociales más complejas.

1.5. Estructura de la tesis. - vol I

La metodología que hemos empleado no llevará a centrarnos en estos primeros capítulos en la parte de **Simulación**. Teniendo en cuenta esta estructura, la tesis comienza con la presentación del problema que pretendemos resolver.

Tras abordar la implementación de agentes inteligentes para la generación de simulaciones, pasaremos al a centrarnos en el empleo de métodos de **Inteligencia Artificial** para el estudio del flujo de información en una red. Esta parte hará un uso intensivo de técnicas de Aprendizaje Automático (especialmente deep learning).

La organización por capítulos se detalla a continuación.

- Capítulo 1
Quizás planteamiento del modelo teórico. Modelo markoviano de la difusión emociones/- cooperación. Nos permite plantear nuestro modelo de difusión. Utilización de Aprendizaje por Refuerzo.
- Capítulo 2
Empezamos con el Diseño experimental. Utilizamos agentes mutantes que perturban la difusión normal en la Red Social. Difusión viral y resultados con agentes mutantes. Análisis comparado con agentes Tit-for-Tat.
- Capítulo 3
Redes estáticas. Utilizamos redes neuronales que procesan imágenes aisladas.
- Capítulo 4
Redes dinámicas. Utilizamos redes neuronales que procesan secuencias de imágenes.
- Capítulo 5 En este capítulo presentamos los **resultados** del estudio y todos los aspectos relativos a la programación que hemos llevado a cabo en Simulación Social.
- Capítulo 6 En este capítulo esquematizaremos las **conclusiones** a las que hemos llegado, así como las pondremos en relación a los objetivos e hipótesis que habíamos establecido al inicio del trabajo.

El texto finaliza con algunos apéndices en los que se detallan algunos aspectos técnicos y de formalización de las herramientas que hemos empleado. Posteriormente aparecen los libros y artículos de investigación referenciados en el trabajo. Finalmente aparecen los símbolos y acrónimos, así como un glosario con los términos técnicos más utilizados.

1.6. Estructura de la tesis - vol. II

La metodología que hemos empleado no llevará a centrarnos en estos primeros capítulos en la parte de **Simulación**. Teniendo en cuenta esta estructura, la tesis comienza con la presentación del problema que pretendemos resolver.

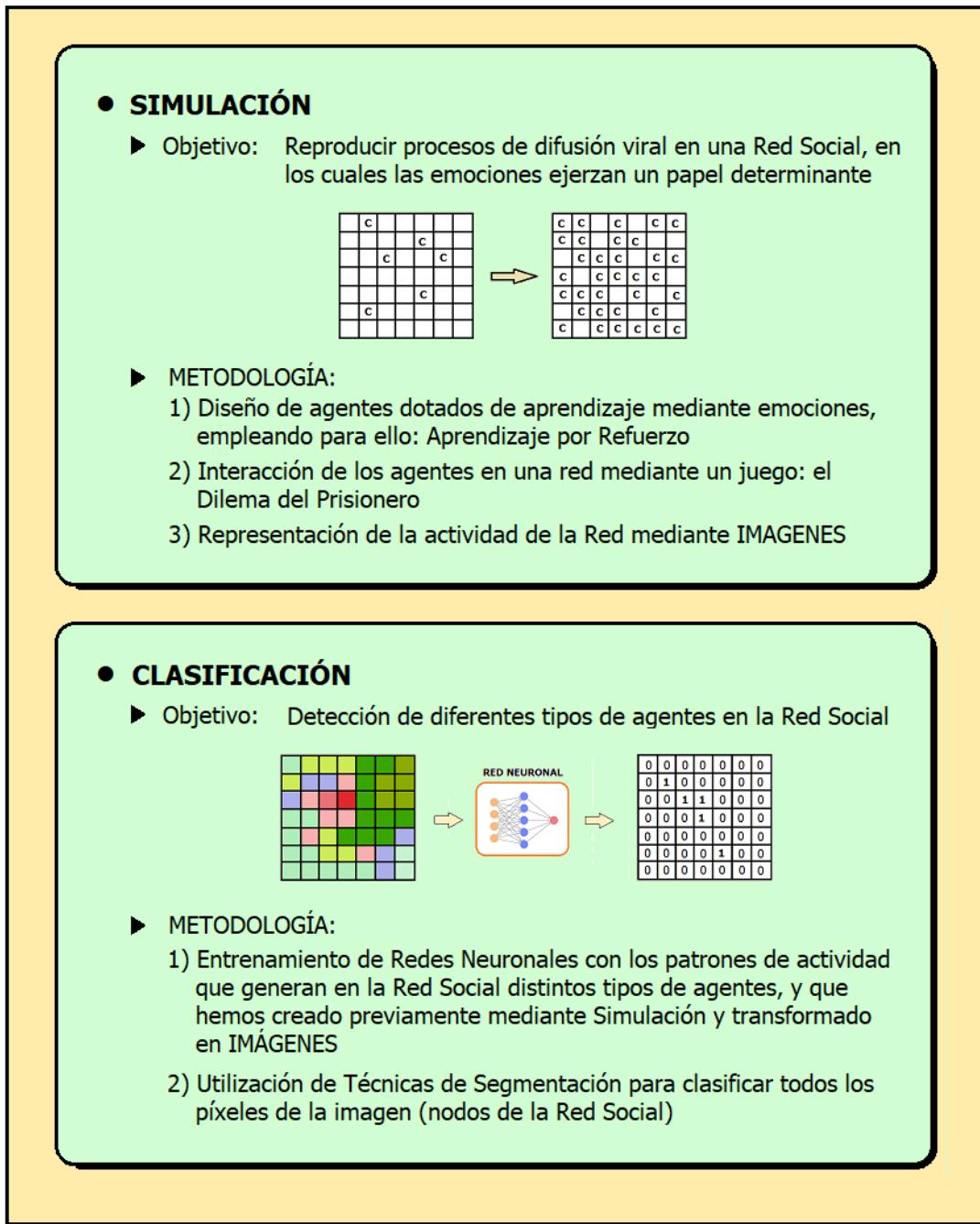


Figura 1.15: Esquema de los bloques que conforman la Tesis.

En concreto, la formalización matemática ocupará especialmente los capítulos 2 y 3; que tratan de la toma de decisiones y los métodos de optimización matemáticos que se encuentran detrás. Una vez disponemos de la base matemática estaremos en condiciones de abordar un tipo especial de problema de cuya solución se ocupa el Aprendizaje por Refuerzo en el capítulo 4: los procesos de decisiones de Markov (MDP). Esto nos capacitará para llevar a cabo simulaciones con agentes

CAPÍTULO 1. INTRODUCCIÓN

dotados del tipo de inteligencia emocional que buscamos. Las interacciones entre agentes se beneficiarán del empleo de la Teoría de Juegos en el capítulo 5.

Esta primera parte nos permite centrarnos en la fase de SIMULACIÓN. Los siguientes capítulos van dirigidos a la presentación de las Redes Neuronales y su utilidad como herramienta. En el capítulo 6 veremos que se pueden emplear como aproximadores funcionales en Aprendizaje por Refuerzo. En los siguientes capítulos dirigiremos su utilización a la parte que nos interesa: la CLASIFICACIÓN de los agentes de la red social mediante redes Neuronales.

La organización detallada por capítulos aparece a continuación.

- **Capítulo 1 . Modelización de decisiones secuenciales.**

Planteamos el problema de toma de decisiones por etapas a lo largo del tiempo. Asociamos un valor a cada estado según las decisiones que queden por delante desde ese punto. Planteamos un algoritmo de Programación Dinámica y estudiamos su convergencia. La formalización matemática se apoya en el llamado operador de Bellman. Veremos que el algoritmo planteado converge al coste óptimo, que es el punto fijo de la ecuación de Bellman.

- **Capítulo 2 . Métodos de solución.**

En el primer capítulo planteamos la relevancia del estudio del efecto de las emociones en la difusión de información en una red social. El problema que pretendemos analizar requiere del empleo de métodos de Simulación, por lo que analizamos las principales técnicas que aparecen en la literatura. Tras señalar las deficiencias proponemos algunas mejoras.

Vemos cómo los métodos de programación Dinámica conducen a la solución exacta del problema que hemos planteado. Sin embargo en problemas complejos, la búsqueda de solución por este sistema puede ser impracticable. Por ello trataremos los métodos aproximados de solución.

- **Capítulo 3 . Aprendizaje por Refuerzo.**

En este capítulo nos centraremos en el estudio del problema de las decisiones secuenciales en unas condiciones muy específicas, como es la ausencia de memoria. Esto nos llevará a un tipo especial de modelos: los procesos de decisiones de Markov. Veremos la aplicación de los métodos vistos para la búsqueda de la decisión óptima, y los distintos procedimientos que se ha desarrollado. En concreto métodos aproximados de solución, los cuales se basan en simulación.

- **Capítulo 4 . Aprendizaje por Refuerzo Multiagente.**

Todos los métodos tratados previamente tratan el problema con un sólo decisor. En el presente capítulo se trata de enfocar el tipo de objetivo cuando en el entorno en estudio operan en realidad de manera simultanea varios decisores. En este capítulo juega un papel especial la Teoría de Juegos. Una vez planteados los distintos métodos estamos en condiciones de generar simulaciones de agentes emocionales.

Una vez que tenemos generados procesos virales mediante simulación, buscamos entrenar redes neuronales con ellos. El objetivo: introducir diversos tipos de agentes en la red, y que las

redes neuronales sean capaces mediante entrenamiento de interiorizar los patrones de actividad diferenciales entre los agentes. Los capítulos que siguen se centran por tanto en la clasificación de agentes mediante técnicas de Inteligencia Artificial. La técnicas que emplearemos en concreto son técnicas de Segmentación de imágenes.

- Capítulo 5 . **Redes Neuronales**.

En este capítulo introducimos las Redes Neuronales y su aplicación, por ejemplo, a la aproximación de funciones. Esto es de utilidad en Aprendizaje por Refuerzo cuando pretendemos tratar un elevado número de acciones del decisor y estados.

Resumen - Ideas principales

Los creación de un clima de opinión social puede verse como un fenómeno de difusión viral en una red social. La difusión de noticias falsas ejercen un efecto determinante en muchas ocasiones, impulsado en muchos casos por el contenido emocional, como ponen de manifiesto algunas investigaciones. Es por esto por lo que el estudio de este tipo de fenómenos puede ayudarnos a afrontar el peligro de la manipulación que conllevan.

El tratamiento matemático de este tipo de problemas es complejo, por lo que la tesis se centrará en la introducción de mejoras metodológicas que faciliten la solución a este tipo de problemas.

1. El procedimiento que emplearemos pasa por **simular** procesos de difusión viral empleando dilemas de la Teoría de Juegos para representar las interacciones entre agentes de una red social.
2. Transformaremos la actividad de una red social en imágenes, y con estos resultados entrenaremos redes neuronales especializadas en segmentar imágenes. Con ello buscamos la **detección de agentes** conflictivos de la red.
3. Algunas redes (*X - Twitter, China Weibo, LiveJournal por ejemplo*) disponen de algún tipo de etiquetado emocional, y en la actualidad las modernas técnicas de Aprendizaje Automático permiten un análisis de sentimientos, por lo que disponemos de un área abundante en el que aplicar los resultados que se deriben de la tesis.

Capítulo 2

Modelización de la toma de decisiones secuenciales

La situación que pretendemos analizar es la de una agente dotado de capacidad de aprendizaje que interactúa con su entorno a lo largo del tiempo. El agente percibe de alguna manera aspectos de lo que le rodea, y toma decisiones con la finalidad de conseguir un objetivo, ya sea maximizar las recompensas que recibe de su entorno o minimizar los costes que acarrean las decisiones que toma.

Las disciplinas que han abordado tradicionalmente este tipo de sistemas que evolucionan por etapas son la Programación Dinámica y la Teoría del Control Óptimo. En lo que sigue expresaremos un poco más formalmente las ideas anteriores y presentaremos los resultados que nos serán de utilidad a la hora de modelizar matemáticamente los fenómenos de comunicación que hemos planteado.

2.1. Conceptos y notación

Inicialmente planteamos un modelo de la forma más general posible, presentando los resultados más destacados para el tratamiento de este tipo de problemas (ver “*Dynamic Programming and Optimal Control*” Bertsekas, D. (vol. I (2017) [24] & vol. II (2012) [25]) y “*Simulation-Based Optimization*” Gosavi, A. (2015) [63], siguiendo especialmente la notación del primero de ellos). Tras la formalización de la situación que pretendemos estudiar, presentaremos el tipo de soluciones que existen para este tipo de problemas.

Suparemos que un determinado agente toma decisiones siguiendo una determinada regla de decisión en tiempos discretos. El agente se encuentra en un determinado estado, y las decisiones que toma tienen como consecuencia el alterar su estado, y el proporcionarle algún tipo de coste o beneficio. El agente busca optimizar el resultado que obtiene. Siguiendo este planteamiento manejaremos la notación que aparece a continuación.

k : índice del tiempo (en horizonte finito tendríamos: $k = 0, 1, \dots, N$)

x_k : estado del sistema (agente) en el instante k , donde: $x_k \in X \quad \forall k$

$U(x_k)$: representa el conjunto de acciones disponibles en el estado x_k .

u_k : decisión tomada en el instante k , donde: $u_k \in U(x_k) \quad \forall k$.

También podemos expresar que la decisión es una función de la información de que se dispone (es decir, del estado del sistema): $u_k = \mu(x_k)$. También notado μ_k .

π : estrategia (*policy*), alude a la colección de reglas de decisión de cada periodo: $\{\mu_0, \mu_1, \dots, \mu_{N-1}\}$. Si la regla de decisión no varía de etapa en etapa ($\pi = \{\mu, \mu, \mu, \dots\}$), se dice que π es **estacionaria**.

w_k : perturbación aleatoria, donde $w_k \sim p(\cdot|x_k, u_k)$ y las w_k son variables aleatorias independientes $\forall k$

En base a la información de que se dispone (x_k) se toma una decisión (u_k), que conduce al sistema a un nuevo estado: x_{k+1} (pudiendo verse afectada la transición a dicho nuevo estado por algún tipo de perturbación aleatoria (w_k)). Siendo en este caso:

$$x_{k+1} = f(x_k, u_k, w_k) \quad k = 0, 1, \dots, N-1$$

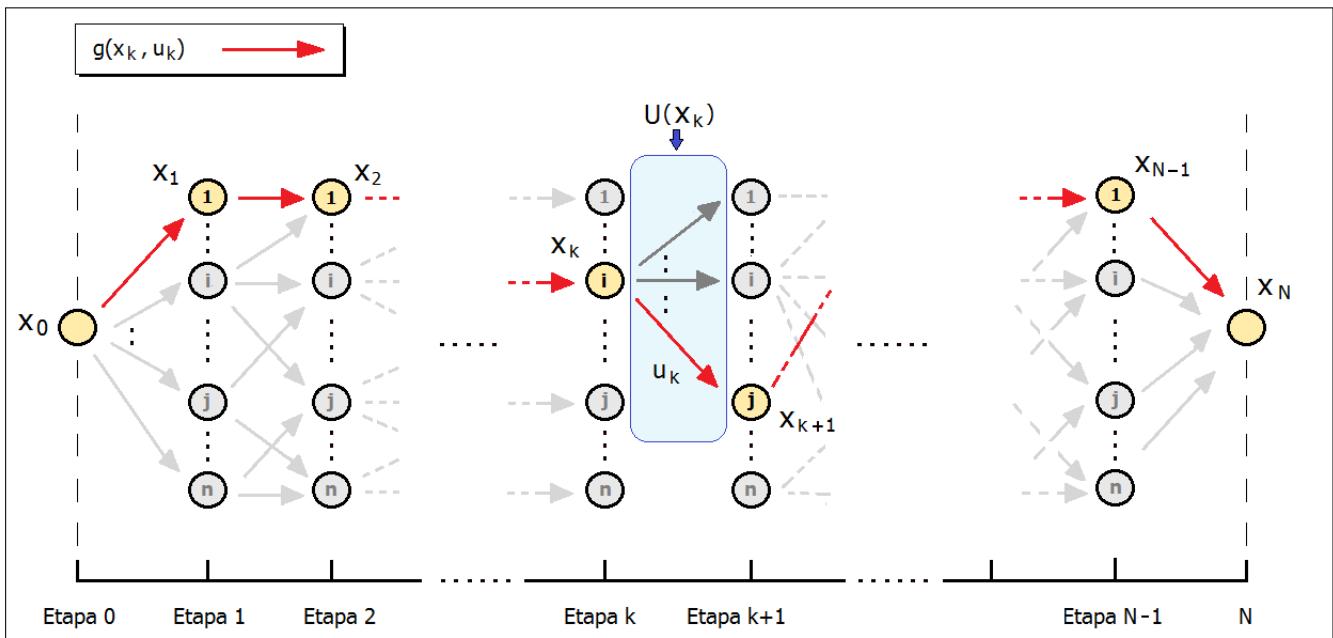


Figura 2.1: *Representación de una secuencia de decisiones.* Partimos en la etapa 0 de un determinado estado inicial X_0 , y cada decisión conduce a un nuevo estado ($X_1, X_2, \dots, X_k, \dots, X_N$). Las decisiones que se pueden tomar son propias de cada estado ($u_k \in U(x_k)$), y llevan aparejadas unas consecuencias, que en este caso (al ser un entorno determinista) son del tipo: $g(x_k, u_k)$.

Las consecuencias para el decisor se concretan en algún tipo de coste (o beneficio), que puede verse influido también por algún tipo de componente aleatorio (w_k).

$g(x_k, u_k, w_k)$: coste de la decisión u_k en x_k que conduce al sistema a un nuevo estado x_{k+1}

En la figura 2.1 podemos ver una representación de un proceso de toma de decisiones como el que hemos descrito. El objetivo consiste en seleccionar en cada estado la decisión más apropiada, de tal manera que las consecuencias para el decisor sean óptimas. Para ello necesitamos expresarlo mediante una función de valor, tal y como aparece a continuación.

J(x₀): Representa el Coste (o beneficio) acumulado cuando se llevan a cabo el conjunto de decisiones que se han tomado $\{u_0, u_1, \dots, u_{N-1}\}$. También notado: $J(x_0; u_0, u_1, \dots, u_{N-1})$ o $J_\pi(x_0)$

J*(x₀): Valor óptimo que buscamos. J^* es en realidad una función, que nos proporciona el coste óptimo para cada estado inicial. Tenemos que: $J^*(x) = \min_{\pi \in \Pi} J_\pi(x), x \in X$.

El horizonte en el que podemos movernos puede ser finito o infinito (tal y como aparece en el cuadro 2.1), y pueden aparecer componentes aleatorios.

Teniendo todo esto en cuenta, el objetivo consistirá en obtener el máximo valor de las recompensas acumuladas que se reciben, o en el caso de costes obtener el valor mínimo. El problema quedaría representado como:

$$\begin{aligned} \text{min./max. : } & J_\pi(x_0) \\ \text{s.a. : } & x_{k+1} = f(x_k, u_k, w_k) \quad \forall k \\ & u_k \in U(x_k) \end{aligned}$$

Vemos que la expresión como un problema de optimización tiene en cuenta las transiciones entre los estados y las decisiones que son posibles de tomar en cada instante actúan como restricciones en el problema.

	Horizonte finito	Horizonte infinito
Entorno Determinista	$J_\pi(x_0; u_0, u_1, \dots, u_{N-1}) =$ $g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \underbrace{\mu_k(x_k)}_{u_k})$	$J_\pi(x_0) = \lim_{N \rightarrow \infty}$ $\sum_{k=0}^{N-1} \alpha^k \cdot g_k(x_k, \underbrace{\mu_k(x_k)}_{u_k})$
Entorno Estocástico	$J_\pi(x_0; u_0, u_1, \dots, u_{N-1}) =$ $\mathbb{E} \left[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \underbrace{\mu_k(x_k)}_{u_k}, \omega_k) \right]$	$J_\pi(x_0) = \lim_{N \rightarrow \infty}$ $\mathbb{E} \left[\sum_{k=0}^{N-1} \alpha^k \cdot g_k(x_k, \underbrace{\mu_k(x_k)}_{u_k}, \omega_k) \right]$ <p style="text-align: center;">factor de descuento: $\alpha \in (0, 1)$</p>

Cuadro 2.1: Expresión de la función $J_\pi(x_0)$ en los escenarios más comunes.

Una vez que hemos representado matemáticamente el problema que pretendemos resolver, pasamos a mencionar las técnicas que existen y que analizaremos en los siguientes apartados.

2.1.1. Representación de la función de valor.

La formulación del problema tal y como se ha presentado en el apartado anterior (ver fig. 2.1) permite visualizar intuitivamente el proceso de toma de decisiones secuenciales; que en un caso de minimización, se traduce en el proceso de búsqueda del camino con menor coste entre dos puntos. A continuación veremos que operando adecuadamente sobre la expresión de la función de valor podemos plantear una estructura recursiva que nos permite plantear un algoritmo para este tipo de problemas y facilita la obtención de la solución óptima.

Para ello trataremos los siguientes puntos: - Expresión recurrente de la función de valor. - Implementación computacional. - Solución exacta vs solución aproximada.

Expresión recurrente de la función de valor. Si partimos de la expresión general de la función de valor, en el caso no determinista:

$$J_\pi(x_0) = \lim_{N \rightarrow \infty} \mathbb{E} \left[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \underbrace{\mu_k(x_k)}_{u_k}, \omega_k) \right]$$

tomando $g_N(x_N) = 0$ para simplificar, podemos extraer el valor de la primera transición y dejar agrupados el resto de términos de la siguiente manera:

$$J_\pi(x_0) = \lim_{N \rightarrow \infty} \mathbb{E} \left[\underbrace{g(x_0, u_0, w_0)}_{\text{valor transición 1}} + \underbrace{\sum_{k=1}^{N-1} g_k(x_k, u_k, \omega_k)}_{\text{resto de transiciones}} \right]$$

Esta expresión sugiere una forma de abordar la búsqueda de las acciones que conducen al valor óptimo: dividir el problema en optimizaciones a un paso. Primero manejamos explícitamente el coste asociado a la primera transición y el estado destino queda etiquetado con el valor del resto de transiciones que quedan por delante.

Si suponemos para simplificar la notación que nos encontramos en un caso discreto, y estacionario, donde en cada estado disponemos del mismo conjunto de decisiones, tendremos lo siguiente: un conjunto de estados $X = \{1, 2, \dots, i, \dots, j, \dots, n\}$ y un espacio de acciones $U = \{1, 2, \dots, s\}$, tendremos $g(x_k = i, u_k = u, x_{k+1} = j) = g_{ij}(u)$, dada la decisión $u \in U$. Entonces la función de valor para una estrategia π : $J_\pi(i)$ $i \in X$, adoptaría la siguiente forma $\forall i \in X$:

$$J_\pi(i) = \underbrace{\mathbb{E}[g_{ij}(u) + J_\pi(j)]}_{Q(i,u)}$$

Donde hemos introducido la notación $Q(i, u)$ para representar el valor esperado del coste de la transición a un paso más el valor del estado siguiente al que se transiciona.

Este procedimiento nos permite acudir a la conocida como **Ecuación de Bellman**, como veremos más adelante, y que permite resolver este tipo de problemas. La forma que adopta dicha ecuación en un problema de minimización es la siguiente:

$$J^*(i) = \min_u \underbrace{\mathbb{E}[g_{ij}(u) + J^*(j)]}_{Q^*(i,u)}$$

La expresión a minimizar también se le conoce como *factor Q*, notado $Q^*(i, u)$. A partir de la obtención recursiva de los valores óptimos de cada estado ($J^*(i), \forall i \in X$) tendremos que las acciones óptimas son:

$$\mu(i) = \arg \min_u \mathbb{E}[g_{ij}(u) + J^*(j)]$$

Implementación. En el plano de la implementación, cuando la función de valor (también la función de estrategia) se definen sobre un conjunto de acciones y estados de pequeña dimensión, los algoritmos emplean una **representación tabular** (ver fig. 7.2). Cada acción-estado se puede reflejar como las filas y columnas de una matriz. Sin embargo, cuando su número es muy elevado (ver fig. 7.3) se suelen emplear la **representación mediante aproximadores funcionales**, como es el caso de las Redes Neuronales.

$Q(i, u)$	1	\dots	\dots	s
1				
\vdots				
i				
\vdots				
n				

Figura 2.2: Si el número de estados-acciones es reducido podemos emplear una Representación tabular.

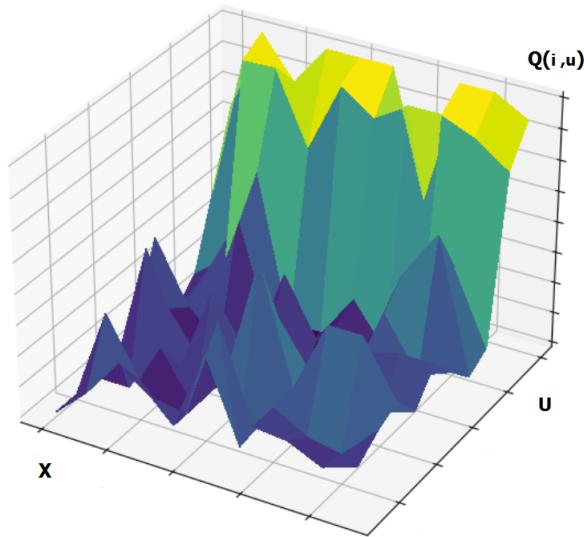


Figura 2.3: Manejar explícitamente el valor de cada par estado acción puede ser muy costoso. Una solución consiste en utilizar una función que aproxime la relación entre los valores.

Una de las herramientas más empleadas en la actualidad son las Redes Neuronales que se emplean como aproximadores funcionales. Siempre que se opte por emplear una aproximación se ha de ser muy cuidadoso con la calidad de la representación

Solución exacta vs solución aproximada Cuando el número de estados/acciones crece suficientemente, las exigencias computacionales hacen impracticable el obtener la solución exacta. Por ello se ha de acudir a la obtención de una solución subóptima, mediante la sustitución de $J^*(\cdot)$ por $\tilde{J}(\cdot; \theta)$, donde θ es un vector de parámetros. En la expresión de los costes óptimos el resultado que tendríamos para cada uno de los estados ($i \in X$) sería el que aparece a continuación:

$$\tilde{J}^*(i) = \min_u \underbrace{\mathbb{E} [g_{ij}(u) + \tilde{J}^*(j)]}_{\tilde{Q}^*(i, u)}$$

lo que conducirá a una decisión subóptima: \tilde{u} :

$$\tilde{u}(i) = \arg \min_u \mathbb{E} [g_{ij}(u) + \tilde{J}^*(j)]$$

En la figura 7.4 podemos ver una representación de esta situación. Vemos que la desviación respecto de la decisión óptima puede llegar a ser bastante significativa si el ajuste no es bueno.

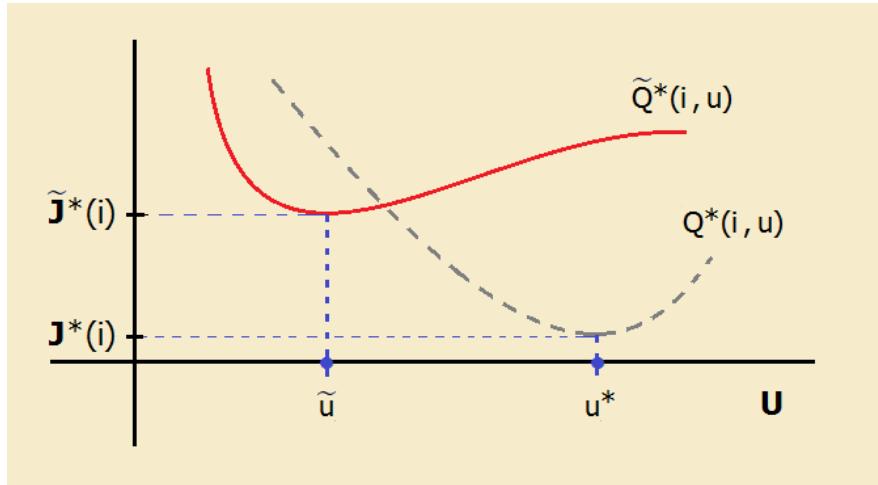


Figura 2.4: Aproximación $\tilde{Q}^*(i, u)$ cuando el sistema se encuentra en un determinado estado $x_k = i$, y la decisión subóptima a la que conduce: \tilde{u} . (Adaptado de Bertsekas, D. (2019) [26])

Si la situación en la que nos encontramos aconseja emplear una aproximación de la función de valor, tendremos que tener en cuenta algunas cuestiones como:

- 1) La estructura de la función \tilde{J} . Usualmente se emplea alguna familia paramétrica de funciones como las Redes Neuronales. Notado: $\tilde{J}(\cdot; \theta)$, donde θ es un vector de parámetros.
- 2) Cómo calcular el vector θ para que el error sea mínimo ($J^*(\cdot) - \tilde{J}(\cdot; \theta)$) y se consiga el mejor ajuste posible al verdadero valor óptimo.

En el siguiente apartado aparece una clasificación de los distintos métodos para encontrar los valores óptimos de los estados. Se empieza por métodos que emplean **Programación Dinámica** y que conducen a una solución exacta. Estos algoritmos también pueden emplearse para obtener una solución aproximada sin más que iterar hasta una cercanía aceptable al verdadero valor.

Posteriormente se enumeran algunas técnicas que emplean métodos de **Simulación** para obtener una solución aproximada.

2.1.2. Clasificación de los métodos de solución.

En lo que sigue presentamos esquemáticamente los métodos de obtención de la estrategia óptima, y que se desarrollarán en los capítulos que siguen.

- **Métodos basados en un modelo**

Las técnicas que podríamos englobar bajo este epígrafe nos permiten encontrar la estrategia óptima μ^* , pero siempre y cuando conozcamos de manera precisa todos los elementos que intervienen en el problema. Esto implica disponer de todas las probabilidades de transición $p(x_{k+1} = j | x_k = i, u_k = u)$ (si es un problema no determinístico), así como las funciones de decisión $\mu(i)$, $i \in X$ y de recompensa/costos $g(x_k, u_k, w_k)$.

Los algoritmos clásicos se basan en técnicas de **Programación Dinámica** y nos permiten encontrar la política óptima μ^* ; es decir: la acción más adecuada en cada estado, de tal manera que la acumulación de recompensas se vuelva máxima (o mínima en el caso de manejar costes). Los dos principales son los siguientes:

A. Iteración de valor (Value iteration).

Se emplea la función de valor con valores iniciales arbitrarios, y sucesivamente se van actualizando, empleando para ello la llamada **ecuación de Bellman**.

B. Iteración de estrategia (Policy iteration).

El algoritmo comienza directamente con una estrategia arbitraria y procede iterativamente hasta obtener una estrategia óptima cuando ya no es posible obtener mejoras. Consta de:

- I. Paso '**policy evaluation**': Dada la estrategia actual (π), obtenemos los valores que consigue ($J_\pi(i)$, $i \in X$).

- II: Paso '**policy improvement**': Se procede a obtener una nueva estrategia (π') que consiga mejores valores ($J_{\pi'}$) que los conseguidos con la estrategia previa (J_π).

- **Métodos libres de modelo**

Por el contrario los métodos conocidos como libres de modelo, nos permiten enfrentarnos a la resolución del problema cuando no disponemos del conocimiento de elementos esenciales como las probabilidades de transición. Se emplea Simulación.

I. Métodos basados en la función de valor (Value based methods)

En este caso buscamos obtener una aproximación de la función de valor óptima (\tilde{J}^*), o bien la función de valor de una estrategia en particular (\tilde{J}_π). Los dos métodos que siguen pueden integrarse como primer paso (policy evaluation) dentro del procedimiento de Iteración de estrategia.

a) Método de Montecarlo

b) Método de las Diferencias Temporales. Uno de los algoritmos que emplean esta técnica, como es Q-Learning, también se emplea como método de iteración de valor.

II. Métodos basados en la función de estrategia (Policy based methods)

En este caso el procedimiento consiste en aproximar la función de estrategia, a través de la selección de alguna familia paramétrica de funciones (como puede ser una red neuronal), y ajustar sus parámetros mediante la simulación del sistema.

Las técnicas de resolución del problema de optimización que hemos planteado queda esquematizadas en el siguiente cuadro resumen 2.2.

Métodos basados en Modelo <i>(Model based methods)</i>	Tipo de técnica	Tipo de solución
■ Value Iteration	Programación Dinámica	Exacta
■ Policy Iteration I. Policy Evaluation II. Policy Improvement	Programación Dinámica	Exacta
Métodos libres de Modelo <i>(Model free methods)</i>	Tipo de técnica	Tipo de solución
Value based - Método de Montecarlo - Método de las Diferencias Temporales	Simulación	Aproximada
Policy based	Simulación	Aproximada

Cuadro 2.2: Esquema con las técnicas de resolución del problema de optimización planteado.

En los apartados que siguen pasamos a analizar detenidamente el problema desde el punto de vista de la *Programación Dinámica*, y la forma de resolverlo iterativamente.

Ésto nos llevará al estudio del llamado **operador de Bellman**, que nos permitirá analizar la convergencia de los algoritmos hacia los valores correctos de la función de valor.

Esto nos permite pasar a analizar la solución del problema, y estar en condiciones de presentar los dos algoritmos principales de Programación Dinámica (Iteración de valor e Iteración de estrategia). Finalmente se presentan los métodos de solución aproximada, que emplean métodos de simulación para obtener el resultado.

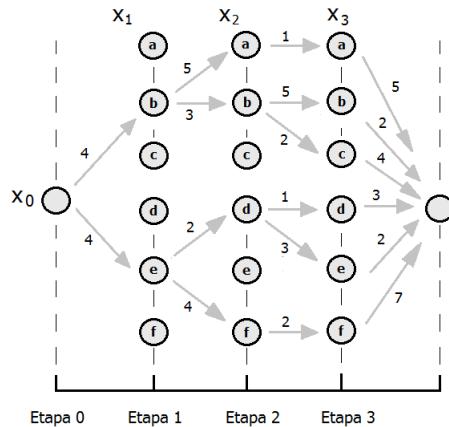
2.2. Análisis mediante Programación Dinámica (PD)

La **Programación Dinámica** es una técnica de resolución de problemas complejos, que consiste en dividir un problema principal en subproblemas que se solapan, de tal manera que combinando secuencialmente las soluciones de cada uno de ellos podemos obtener la solución del problema original.

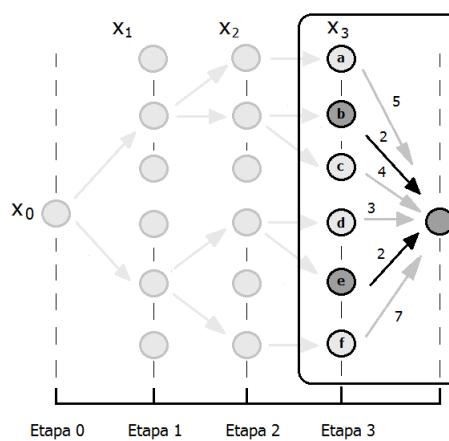
El ejemplo siguiente permite visualizar el procedimiento. Vemos cómo podemos evitar arrastrar la enumeración exhaustiva acumulando a lo largo de cada ruta y optimizando a un sólo paso.

EJEMPLO 2.1: Decisiones en entorno determinista y con horizonte finito

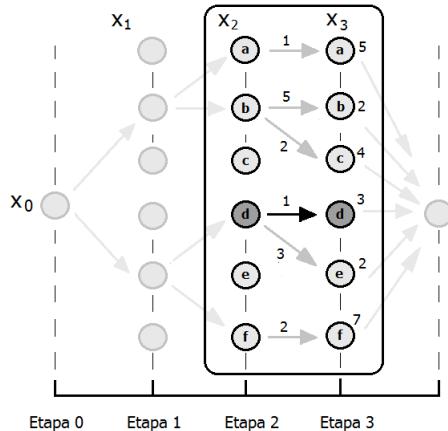
Imaginemos que nos encontramos ante el problema de determinar la secuencia de decisiones óptimas en una serie finita de etapas ($K=0,1,2,3$). La siguiente figura representa un sencillo caso. X_0, X_1, X_2, X_3 son el estado del sistema en cada etapa. Sobre cada arco aparece el coste asociado a la transición en cuestión. El objetivo consiste en llegar desde X_0 al final con el mínimo coste acumulado.



La forma de plantear el problema desde la perspectiva de la Programación Dinámica consiste en dividir el problema principal en problemas más sencillos que impliquen un sólo paso o etapa y resolverlo olvidándose del resto del problema. Empezando por el final tenemos:

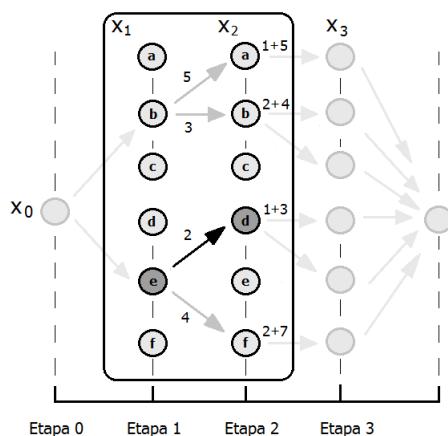


El valor de cada arco final se asocia a cada uno de los nodos en la etapa 3 ($k=3$) y se pasa al siguiente subproblema, que corresponde a la decisión en la etapa 2 ($k=2$). El nuevo subproblema aparece representado en la siguiente figura. Para resolverlo tenemos en cuenta el coste del arco y el del nodo al que conduce.



En la figura anterior podemos observar que desde el nodo 'd' se puede ir a dos nodos (el 'd' y el 'e'), siendo el que vuelve a pasar por 'd' el que conduciría a un menor coste ($1+3$) frente a la decisión de optar por el otro arco que conlleva un coste de ($3+2$). Como estamos minimizando el coste, será el menor valor el que etiquete el nodo y represente el coste futuro cuando se actúa optimamente a partir de ese punto.

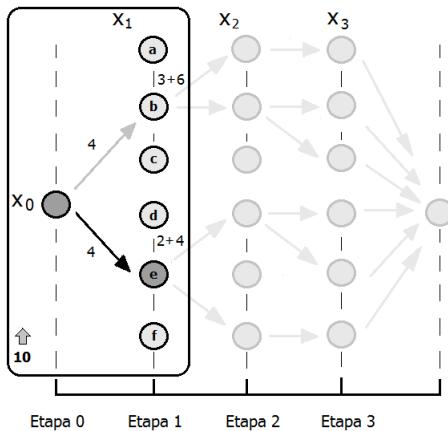
En el caso del nodo 'b' ocurre algo parecido, siendo ahora las posibilidades las de pasar a 'b' o 'c'. La decisión de pasar de 'b' a 'c' es la que tiene menor coste ($2+4$) frente a mayor valor de ($5+2$), por lo que será el primer valor el que etiquete el nodo. El proceso aparece representado a continuación.



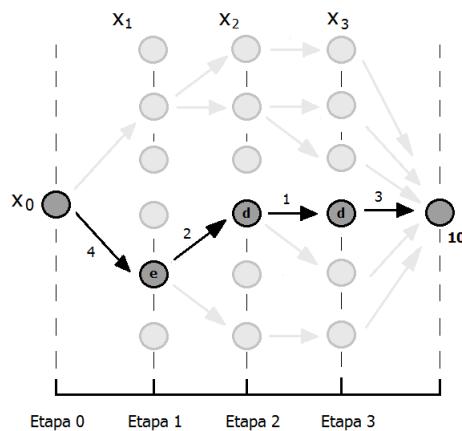
Vemos como en cada etapa el problema se ha reducido a manejar un coste presente más un valor del coste óptimo de todas las etapas que quedan por delante. Ahora es facil ver los valores que etiquetarán los nodos, ya que vuelve a ocurrir algo parecido.

Al nodo 'b' le asociamos el valor ($3+6$) fruto de elegir ($3+(2+4)$) frente a ($5+(1+5)$). El caso del nodo 'e' es igual: ($2+(1+3)$) frente a ($4+(2+7)$). Esta decisión aparece resaltada al ser la de menor valor.

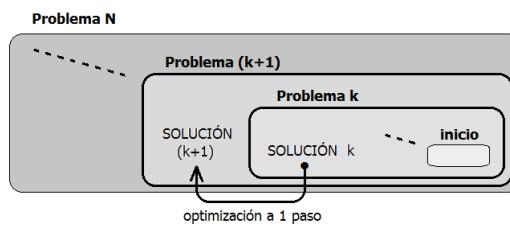
En el siguiente paso llegamos al último subproblema. Ahora basta ver cuál de las dos decisiones que se toman en X_0 es la óptima, ya que aparejado al nodo destino tenemos la etiqueta del menor coste correspondiente a las etapas que quedan por delante. Tenemos que la decisión óptima consiste en ir al nodo 'e', lo que nos conduce ya al coste final óptimo de $(4+(2+4))$.



El proceso anterior nos ha llevado secuencialmente hacia atrás para encontrar el camino con menor coste. Una vez que lo hemos encontrado, basta en recorrer a la inversa los costes óptimos para tener la secuencia de decisiones que se han de tomar en cada etapa, tal y como muestra la figura siguiente (Camino de menor coste: (Nodo inicial)-e-d-d-(Nodo final)).



Una solución se construye sobre la que le precede, resolviendo así un problema cada vez mayor.



Este tipo de técnica es especialmente útil para la resolución de problemas de toma de decisiones por etapas. Su formulación se la debemos a Richard Bellman, cuyas ideas fueron recogidas en diversos artículos sobre el tema, especialmente: "The Theory of Dynamic Programming" (1954) [15], y posteriormente en su libro "Dynamic programming and modern control theory" (1965) [16].

La idea que plantea este método es la de resolver un problema a una etapa, optimizando a un solo paso. En cada iteración se tiene en cuenta el coste de la transición actual y un coste terminal, que corresponde al valor óptimo del resto de etapas que quedan por delante.

Esta idea intuitiva se concreta en el conocido como Principio de Optimalidad, debido a Bellman; y que viene a decir que si tenemos una secuencia óptima para un problema, cualquier secuencia extraída de ella es también óptima para el subproblema correspondiente. La expresión formal aparece en la siguiente definición.

PRINCIPIO DE OPTIMALIDAD

Sea $\{u_0^*, u_1^*, \dots, u_{N-1}^*\}$ una secuencia óptima de acciones, que conjuntamente con x_0 determinan la correspondiente secuencia de estados $\{x_1^*, x_2^*, \dots, x_N^*\}$. Si consideramos el subproblema que comienza en el estado x_k^* en el instante k



y deseamos minimizar el coste futuro ('cost-to-go') desde el instante k al instante N , que adopta la siguiente expresión:

$$g_k(x_k^*, u_k) + \sum_{m=k+1}^{N-1} g_m(x_m, u_m) + g_N(x_N)$$

sobre las variables de decisión $\{u_k, \dots, u_{N-1}\}$ donde $u_m \in U_m(x_m)$, $m = k, \dots, N - 1$. Entonces, dada la secuencia de decisiones óptima del problema completo:

$$\{u_0^*, u_1^*, \dots, u_{k-1}^*, \color{red}{u_k^*}, \dots, \color{red}{u_{N-1}^*}\}$$

la secuencia de acciones truncada $\{\color{red}{u_k^*}, \dots, \color{red}{u_{N-1}^*}\}$ es óptima para el subproblema en cuestión.

Básicamente lo que afirma el principio anterior es que, dada una secuencia de decisiones óptimas para un problema de decisiones secuenciales, cualquier secuencia que elijamos sigue siendo óptima en el subproblema al que están asociadas.

Esto se refleja en el coste asociado a cada secuencia. Llamamos al coste futuro ('cost-to-go') desde un instante k , mediante la expresión $J(x_k; u_k, \dots, u_{N-1})$, que recoge el valor de la secuencia de decisiones en cuestión en el subproblema que comienza en ese instante k .

$$J(x_k; u_k, \dots, u_{N-1}) = g_N(x_N) + \sum_{m=k}^{N-1} g_m(x_m, u_m)$$

El proceso de encontrar los valores óptimos secuencialmente, y posteriormente reconstruir el conjunto de decisiones que conducen a ellos, se concreta en el siguiente algoritmo (Algoritmo 1), que resuelve un problema aprovechando la solución del subproblema inmediatamente anterior.

Algoritmo 1 - Búsqueda de acciones óptimas (caso determinista y horizonte finito)

begin-

 // PASO 1: Obtención de las funciones J^*

$$J_N^*(x_N) = g_N(x_N) \quad \forall x_N$$

for $k = N - 1$ **to** 0 **do**

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} [g_k(x_k, u_k) + J_{k+1}^*(f_k(x_k, u_k))] \quad \forall x_k$$

$\underbrace{_{x_{k+1}}}_{x_{k+1}}$

// PASO 2: Obtención de la secuencia óptima de decisiones

$$u_0^* \in \arg \min_{u_0 \in U_0(x_0)} [g_0(x_0, u_0) + J_1^*(f_0(x_0, u_0))]$$

$\underbrace{_{x_1}}_{x_1^*}$

for $k = 1$ **to** $N - 1$ **do**

$$u_k^* \in \arg \min_{u_k \in U_k(x_k)} [g_k(x_k^*, u_k) + J_{k+1}^*(f_k(x_k^*, u_k))]$$

$\underbrace{_{x_{k+1}}}_{x_{k+1}^*}$

Una vez tenemos las funciones de valor óptimo: $\{J_N^*(x_N), J_{N-1}^*(x_{N-1}), \dots, J_0^*(x_0)\}$ vemos que secuencialmente y realizando los cálculos hacia atrás, estamos en disposición de especificar la secuencia de decisiones óptimas que conducen a dichos valores: $\{u_0^*, \dots, u_{N-1}^*\}$. Tenemos además que el valor $J_0^*(x_0)$ obtenido en la última etapa es igual al coste óptimo $J^*(x_0)$.

Caso determinista y horizonte finito. En la figura 2.5 tenemos una representación del proceso de optimización a un paso. Vemos que para tener el valor ($J_k^*(i)$) de estado 'i' ($i = 1, 2, \dots, n$), elegiremos la mejor acción (supongamos que $u_k \in \{1, \dots, n\} \forall k$, y llamemos al conjunto de acciones sin referencia a etapa: $\{u_1, u_2, \dots, u_n\}$). Para ello sopesamos el coste que conlleva dicha acción ($g_{ij}(u)$ donde $u \in \{u_1, \dots, u_n\}$) y el valor del estado al que conduce ($J_{k+1}^*(j)$, $j = 1, \dots, n$), y que representa el resultado de actuar óptimamente desde ese estado en adelante).

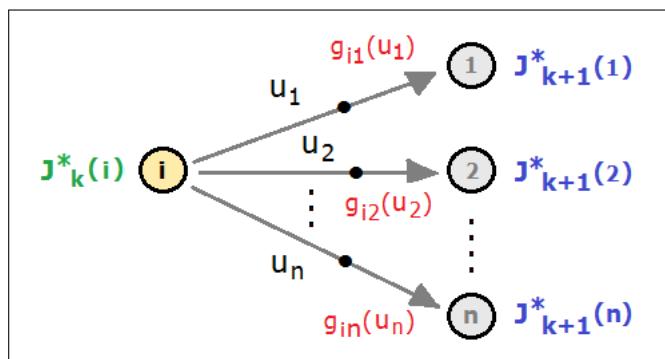


Figura 2.5: En un entorno determinista, la función de valor se encuentra optimizando la suma del costo/ganancia a un paso ($g_{ij}(u)$, $u \in \{u_1, \dots, u_n\}$) más el valor del estado destino $J_{k+1}^*(j)$.

Algoritmo 2 - Programación Dinámica (PD) (caso GENERAL)

```

begin-
     $J_N^*(x_N) = g_N(x_N) \forall x_N$ 
    for  $k = N - 1$  to 0 do
         $J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \mathbb{E} [g_k(x_k, u_k) + J_{k+1}^*(f_k(x_k, u_k))] \quad \forall x_k$ 
        
$$\underbrace{\hspace{10em}}_{x_{k+1}}$$

    }

```

En la medida en que las acciones óptimas quedan determinadas directamente a través de la secuencia de valores óptimos de los estados ($J^*(i)$), nos centraremos en el cálculo de estos últimos. El método general mediante **Programación Dinámica** es el que recoge el Algoritmo 2 (al que nos referiremos abreviadamente como Algoritmo PD, y que aparece mencionado como tal a lo largo del texto).

Caso estocástico y horizonte finito. En la figura 2.6 tenemos una representación del proceso de optimización a un paso con un espacio de estados finito. Para tener el valor óptimo de estado 'i', elegiremos la mejor acción ($u \in \{u_1, \dots, u_s\}$) sopesando el valor esperado del coste que conlleva dicha acción ($\sum_{j=1}^n p_{ij} \cdot g_{ij}(u)$) y el valor del estado al que conduce ($J_{k+1}^*(1), J_{k+1}^*(2), \dots, J_{k+1}^*(n)$), y que representa el resultado de actuar óptimamente a partir de dicho punto).

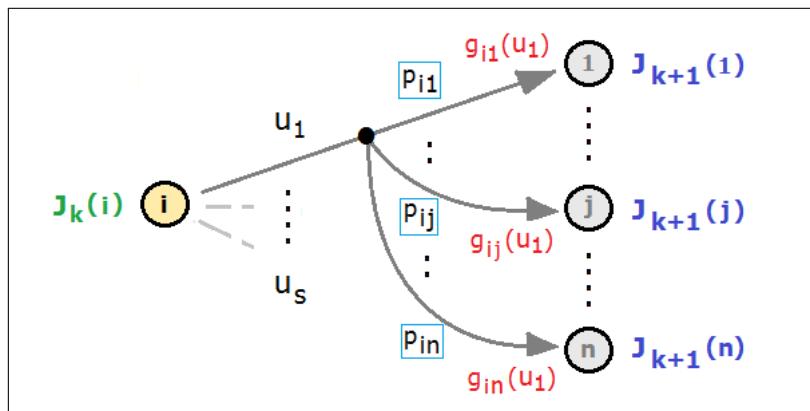


Figura 2.6: En un entorno aleatorio, la función de valor se encuentra optimizando el valor esperado de la suma del costo/ganancia a un paso ($g_{ij}(u)$, $u \in \{u_1, \dots, u_s\}$) más el valor óptimo del estado destino ($J_{k+1}^*(1), \dots, J_{k+1}^*(n)$). Suponemos que la incertidumbre reside en el estado final al que se llega tras tomar la decisión, no en el coste asociado a dicha transición ($g_{ij}(u)$).

El siguiente paso consistirá en analizar la convertencia del algoritmo de Programación Dinámica (Algoritmo PD), ya que será la base para construir los dos procedimientos para la obtención del valor exacto de la función de valor: El procedimiento de **Iteración de valor** y el procedimiento de **Iteración de estrategia**.

Para lograr este objetivo presentaremos en los siguientes apartados las dos herramientas de las que nos serviremos: el **operador de Bellman** y la **ecuación de Bellman**.

2.3. Estudio de la convergencia del algoritmo PD

El algoritmo que acabamos de concretar, acumula secuencialmente los costes futuros y en cada etapa construye la solución de un problema mayor. Ésto nos permite obtener la secuencia de decisiones óptima.

Para analizar el algoritmo y comprobar la convergencia se introduce una representación de un sólo paso a través del denominado **operador de Bellman** (T), y que aplicado sobre un vector de costes/beneficios se representa como TJ .

De todos los casos presentados en el cuadro 2.1, nos interesa el caso en el que aparece un factor de descuento α , que viene a reflejar que las consecuencias futuras de las decisiones que se tomen en etapas posteriores nos importan menos que las consecuencias inmediatas que surgen en el presente.

En este caso, cuando los costes/beneficios por etapa están acotados, el operador de Bellman tiene una propiedad importante tal y como veremos: T es una función contractiva, lo que implica que el operador de Bellman tiene un único punto fijo que es J^* (la función de coste/beneficio óptimo). La definición formal es la siguiente.

DEFINICIÓN 2.1: Operador de Bellman

Para cualquier función $J : X \rightarrow \mathbb{R}$, definimos la función T (llamada operador de Bellman) que consiste en aplicar un paso del algoritmo de Programación Dinámica sobre J de la siguiente manera:

$$(TJ)(x) = \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot J(f(x, u, w))] \quad \forall x \in X$$

Si la regla de decisión que gobierna la selección de acciones es estacionaria - es decir, no cambia de etapa a etapa -, se suele notar el operador como: T_μ , cuya expresión es:

$$(T_\mu J)(x) = \mathbb{E}[g(x, u, w) + \alpha \cdot J(f(x, u, w))] \quad \forall x \in X$$

TJ viene a representar la función de coste óptimo para el subproblema de una sola etapa, en el que hay un coste inmediato que viene dado por g , y un coste terminal por αJ .

Para representar la aplicación reiterada del algoritmo se emplea la composición:

$$(T^k J)(x) = (T(T^{k-1} J))(x) \quad \forall x \in X, \quad k = 1, 2, \dots$$

donde:

$$(T^0 J)(x) = J(x) \quad \forall x \in X$$

En el ejemplo que aparece a continuación nos enfrentamos al caso en el que dado vector de costes de partida $J = (J(1), J(2), \dots, J(n))$, donde $X = 1, 2, \dots, n$, se aplica el Algoritmo PD, lo que nos proporcionará las componentes de TJ ($(TJ)(i)$). Posteriormente vemos las propiedades del operador de Bellman.

Previamente analizamos la interpretación geométrica del Operador de Bellman.

Operador T_μ . El operador T_μ opera sobre los vectores de coste, por lo que tiene tantas dimensiones como estados tiene el sistema ($T_\mu(i)$, $i \in X$). Si observamos su expresión general, que en el caso de espacio de estados finito sería: $J(i) = \sum_{j=1}^n p_{ij}(u) \cdot [g_{ij}(u) + \alpha \cdot J(j)]$, donde $u = \mu(i)$, tenemos que es lineal en J . Vemos en la figura 2.7 que es un plano inclinado.

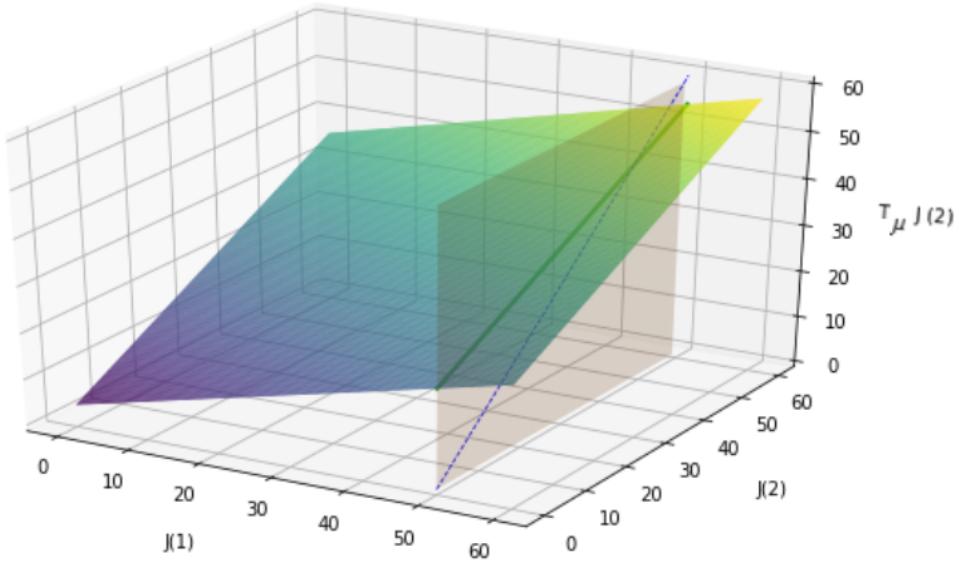
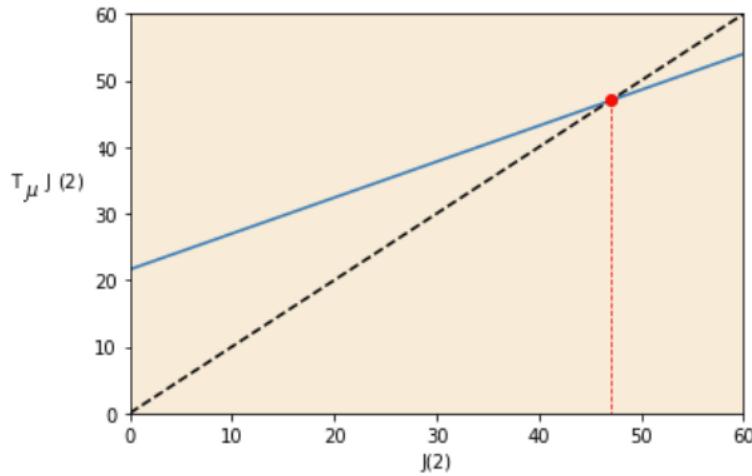
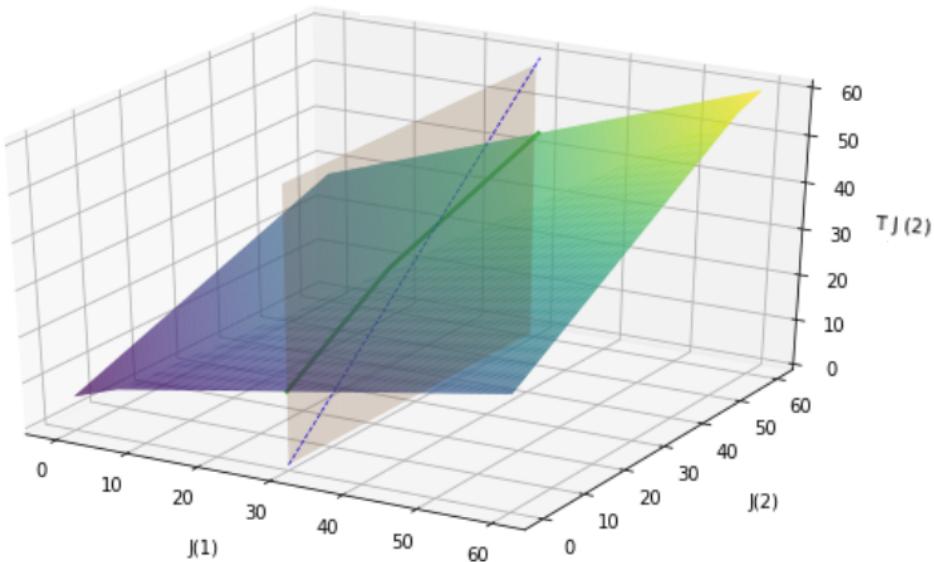
Operador de Bellman T_μ **Proyección en el plano**

Figura 2.7: Interpretación geométrica del operador de Bellman T_μ . (Ver código Python en el Apéndice).

Podemos proyectar en un plano perpendicular a los ejes, fijando un valor por ejemplo para la componente $J(1)$. Aparece en negro discontinuo la bisectriz del primer cuadrante, que refleja los puntos en los que $T_\mu J = J$. El punto en el que se corta la recta T_μ y la bisectriz, representa el valor al que conduce la estrategia μ . Es decir J_μ .

Operador T . El operador T opera igualmente sobre los vectores de coste, por lo que tiene nuevamente tantas dimensiones como estados ($T(i)$, $i \in X$). Sin embargo la toma de mínimos lo convierte ahora en no lineal. Su expresión general en el caso de espacio de estados finito sería la siguiente: $J(i) = \min_u \sum_{j=1}^n p_{ij}(u) \cdot [g_{ij}(u) + \alpha \cdot J(j)]$.

Operador de Bellman T



Proyección en el plano

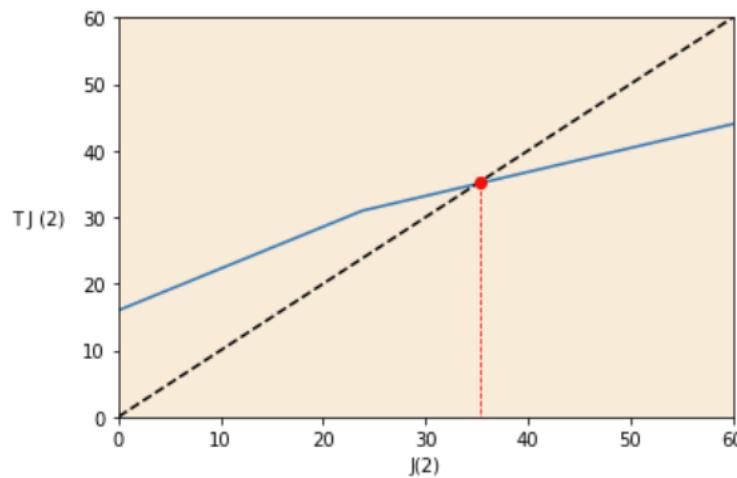
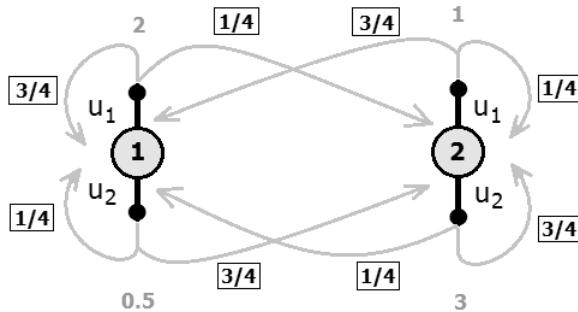


Figura 2.8: Interpretación geométrica del operador de Bellman T . (Ver código Python en el Apéndice).

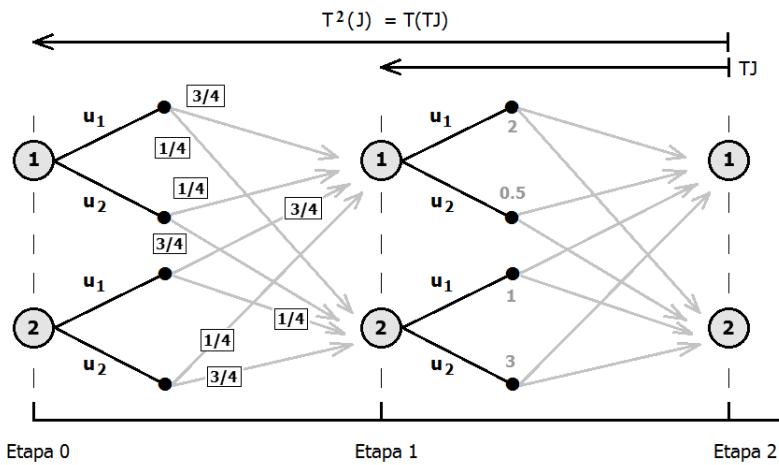
Vemos en la figura 2.8 que tiene forma cóncava y es lineal a trozos (generalmente con tantos fragmentos lineales como decisiones tenga el decisor ($u \in X(i)$)). Si proyectamos en un plano perpendicular vemos el punto en el que se corta TJ con la bisectriz. Este punto lo simbolizaremos como J^* , es el punto fijo y representa el coste óptimo.

EJEMPLO 2.2: Aplicación del Algoritmo PD

Supongamos que deseamos obtener las decisiones óptimas de un sistema que pasa por 2 estados $X = 1, 2$, y en el que existen dos posibles elecciones para el decisor: $U(1) = U(2) = \{u_1, u_2\}$. Las probabilidades de transición: $p_{11}(u_1) = 3/4$, $p_{12}(u_1) = 1/4$, $p_{11}(u_2) = 1/4$, $p_{12}(u_2) = 3/4$ y $p_{21}(u_1) = 3/4$, $p_{22}(u_1) = 1/4$, $p_{21}(u_2) = 1/4$, $p_{22}(u_2) = 3/4$ aparecen en el gráfico siguiente, así como los costes asociados a cada decisión: $g_{11}(u_1) = g_{12}(u_1) = 2$, $g_{11}(u_2) = g_{12}(u_2) = 0,5$, $g_{21}(u_1) = g_{22}(u_1) = 1$, $g_{21}(u_2) = g_{22}(u_2) = 3$.



Supongamos que nos interesa estudiar su evolución en 2 etapas. Representado como una secuencia de decisiones, podemos ver el cálculo del valor de cada estado como la aplicación reiterada del operador de Bellman sobre el vector de costes previo.



Incluyendo el factor de descuento α , la expresión para el cálculo del operador de Bellman quedaría tal y como aparece a continuación.

$$(TJ)(i) = \min_{u \in \{u_1, u_2\}} \left[\sum_{j=1}^2 p_{ij}(u) \cdot [g_{ij}(u) + \alpha \cdot J(j)] \right], \quad i = 1, 2$$

$$(T^2J)(i) = T(TJ)(i) = \min_{u \in \{u_1, u_2\}} \left[\sum_{j=1}^2 p_{ij}(u) \cdot [g_{ij}(u) + \alpha \cdot (TJ)(j)] \right], \quad i = 1, 2$$

Basta sustituir los valores en las expresiones anteriores para obtener los valores de cada estado.

PASO 1

Tomando $\alpha = 1$ y el vector de costes $J = (0, 0)$, tenemos el siguiente resultado:

$$\begin{aligned} (TJ)(1) &= \min \{ p_{11}(u_1) \cdot [g_{11}(u_1) + \alpha \cdot J(1)] + p_{12}(u_1) \cdot [g_{12}(u_1) + \alpha \cdot J(2)] , \\ &\quad p_{11}(u_2) \cdot [g_{11}(u_2) + \alpha \cdot J(1)] + p_{12}(u_2) \cdot [g_{12}(u_2) + \alpha \cdot J(2)] \} \\ &= \min \{ 3/4 \cdot [2 + 0] + 1/4 \cdot [2 + 0] , 1/4 \cdot [1/2 + 0] + 3/4 \cdot [1/2 + 0] \} \\ &= \min \{ 2, 1/2 \} = 1/2 \end{aligned}$$

$$\begin{aligned} (TJ)(2) &= \min \{ p_{21}(u_1) \cdot [g_{21}(u_1) + \alpha \cdot J(1)] + p_{22}(u_1) \cdot [g_{22}(u_1) + \alpha \cdot J(2)] , \\ &\quad p_{21}(u_2) \cdot [g_{21}(u_2) + \alpha \cdot J(1)] + p_{22}(u_2) \cdot [g_{22}(u_2) + \alpha \cdot J(2)] \} \\ &= \min \{ 3/4 \cdot [1 + 0] + 1/4 \cdot [1 + 0] , 1/4 \cdot [3 + 0] + 3/4 \cdot [3 + 0] \} \\ &= \min \{ 1, 3 \} = 1 \end{aligned}$$

PASO 2

Teniendo en cuenta el resultado obtenido ($(TJ) = (1/2, 1)$), aplicamos una nueva iteración:

$$\begin{aligned} (T^2J)(1) &= \min \{ p_{11}(u_1) \cdot [g_{11}(u_1) + (TJ)(1)] + p_{12}(u_1) \cdot [g_{12}(u_1) + (TJ)(2)] , \\ &\quad p_{11}(u_2) \cdot [g_{11}(u_2) + (TJ)(1)] + p_{12}(u_2) \cdot [g_{12}(u_2) + (TJ)(2)] \} \\ &= \min \{ 3/4 \cdot [2 + 1/2] + 1/4 \cdot [2 + 1] , 1/4 \cdot [1/2 + 1/2] + 3/4 \cdot [1/2 + 1] \} \\ &= \min \{ 21/8, 11/8 \} = 11/8 \end{aligned}$$

$$\begin{aligned} (T^2J)(2) &= \min \{ p_{21}(u_1) \cdot [g_{21}(u_1) + (TJ)(1)] + p_{22}(u_1) \cdot [g_{22}(u_1) + (TJ)(2)] , \\ &\quad p_{21}(u_2) \cdot [g_{21}(u_2) + (TJ)(1)] + p_{22}(u_2) \cdot [g_{22}(u_2) + (TJ)(2)] \} \\ &= \min \{ 3/4 \cdot [1 + 1/2] + 1/4 \cdot [1 + 1] , 1/4 \cdot [3 + 1/2] + 3/4 \cdot [3 + 1] \} \\ &= \min \{ 13/8, 31/8 \} = 13/8 \end{aligned}$$

Entonces:

$$(T^2J)(1) = 11/8 = 1.375 \quad \text{y} \quad (T^2J)(2) = 13/8 = 1.625$$

Vemos que empezando en el primer estado y actuando en lo que sigue de manera óptima se obtiene un coste ligeramente menor que si lo hacemos desde el estado dos.

En los siguientes apartados analizaremos más detenidamente el comportamiento del operador de Bellman. Esto nos permitirá llegar a la propiedad de **contractividad** de dicho operador, lo que permite demostrar la convergencia del algoritmo de optimización a un paso que hemos estado analizando. Y posteriormente desarrollaremos dos herramientas para la resolución de este tipo de problemas: la **Ecuación de Bellman** y la **Ecuación de Optimalidad de Bellman**.

LEMA 2.1: Propiedad de monotonicidad del Operador de Bellman

Para cualesquiera funciones $J : X \rightarrow \mathbb{R}$ y $J' : X \rightarrow \mathbb{R}$, tal que:

$$\forall x \in X, J(x) \leq J'(x)$$

y dada cualquier estrategia estacionaria $\mu : X \rightarrow \mathcal{U}$, tenemos que, $\forall x \in X, k = 1, 2, \dots$ se verifica el siguiente resultado:

$$(T^k J)(x) \leq (T^k J')(x)$$

$$(T_\mu^k J)(x) \leq (T_\mu^k J')(x)$$

Demostración:

Probamos por inducción, suponiendo que se cumple desde $K=1$ hasta m , y comprobaremos que se cumple para $(m+1)$:

k=1

$$\forall x \in X, J(x) \leq J'(x) \implies$$

$$(TJ)(x) = \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot J(f(x, u, w))] \leq$$

$$\min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot J'(f(x, u, w))] = (TJ')(x)$$

k=m

$$J(x) \leq J'(x) \implies (T^m J)(x) \leq (T^m J')(x)$$

k=m+1

$$(T^{m+1} J)(x) = \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot T^m J(f(x, u, w))]$$

$$(T^{m+1} J')(x) = \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot T^m J'(f(x, u, w))]$$

y como por suposición :

$$(T^m J)(x) \leq (T^m J')(x) \quad \forall x \in X$$

sólo puede ser:

$$(T^{m+1} J)(x) \leq (T^{m+1} J')(x)$$

QED ----- ■

Y la siguiente propiedad importante es la siguiente.

LEMA 2.2: Propiedad de variación constante del operador de Bellman

Dada una función $J : X \rightarrow \mathbb{R}$, una estrategia estacionaria (μ) , tal que $\mu \in \mathcal{M}$, un escalar r , y $x \in X$, tenemos que $\forall k$ se cumple:

$$(T^k(J + re))(x) \leq (T^k J)(x) + \alpha^k \cdot r$$

$$(T_\mu^k(J + re))(x) \leq (T_\mu^k J)(x) + \alpha^k \cdot r$$

Demostración:

Probamos por inducción, suponiendo que se cumple desde $K=1$ hasta m , y comprobaremos que se cumple para $(m+1)$. Basándonos en la siguiente relación:

$$(TJ)(x) = \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot J(f(x, u, w))] \quad \forall x \in X$$

k=1

$$\begin{aligned} (T(J + re))(x) &= \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot (J(f(x, u, w)) + re)] \\ &= \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot J(f(x, u, w)) + (\alpha r e)] \\ &= \underbrace{\min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot J(f(x, u, w))]}_{(TJ)(x)} + \alpha \cdot r \\ &= (TJ)(x) + \alpha \cdot r \end{aligned}$$

k=m

$$(T^m(J + re))(x) = \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + \alpha \cdot (J(f(x, u, w)) + re)]$$

k=m+1

$$(T^m(J + re))(x)$$

QED

Entre las propiedades del operador T , se encuentra el que es una función contractiva respecto de una norma máxima. Este hecho está detrás de la convergencia del algoritmo de programación dinámica.

Conceptos de Análisis

DEFINICION

Sea un espacio vectorial real Y , definimos el operador **norma** (*simbolizado* : $\| \cdot \|$) que permite determinar la magnitud de cada vector: $\| y \| \quad \forall y \in Y$. Entonces se verifican las siguientes propiedades:

- a) $\| y \| \geq 0 \quad \forall y \in Y$
- b) $\| y \| = 0 \iff y = 0$
- c) $\| ay \| = |a| \| y \| \quad \forall a \in \mathbb{R}$
- d) $\| y + z \| \leq \| y \| + \| z \| \quad \forall y, z \in Y$

DEFINICION

El espacio vectorial real Y bajo la norma ($\| \cdot \|$), se dice **completo** si toda secuencia de Cauchy ($\{y_k\} \subset Y$) es convergente. Es decir:

$$\bar{y} \in Y \implies \| y_k - \bar{y} \| \rightarrow 0$$

DEFINICION

$B(X)$ denotará el espacio de las funciones real valoradas F ($F : X \rightarrow \mathbb{R}$) X tal que $F(x)/v(x)$ está acotada $\forall x \in X$. Entonces, dada una función $F : B(X) \rightarrow B(X)$ se dice **contractiva** con módulo de la contracción ρ ($\rho \in (0, 1)$) si cumple:

$$\| Fy - Fz \| \leq \rho \| y - z \|, \quad \forall y, z \in Y$$

TEOREMA

Dado un espacio vectorial real completo Y bajo una norma ($\| \cdot \|$), y una función **contractiva** $F : B(X) \rightarrow B(X)$. Entonces la función F cumple la siguiente ecuación:

$$y = Fy$$

cuya solución y^* es única, llamada **punto fijo** de F .

Además la secuencia $\{y_k\}$ generada por la iteración: $y_{k+1} = Fy_k$ converge a y^* , comenzando por un punto inicial arbitrario y_0 .

TEOREMA

Sea $F : B(X) \rightarrow B(X)$, una **función contractiva** con módulo $\rho \in (0, 1)$. Entonces existe un único valor $J^* \in B(X)$ tal que:

$$J^* = FJ^*$$

además:

- a) Tenemos que se cumple la siguiente relación:

$$\| F^k J - J^* \| \leq \rho^k \| J - J^* \|, \quad k = 1, 2, \dots$$

- b) La secuencia converge al punto fijo

$$\{F^k J\} \rightarrow J^* \quad \forall J \in B(X)$$

PROPOSICIÓN 2.1: Contractividad de T respecto de una norma $\|\cdot\|_\xi$

Partiendo del caso en el que todas las estrategias (policies) estacionarias son propias, tenemos que existe un vector con componentes positivas $\xi = (\xi_1, \dots, \xi_n)$ de tal manera que los operadores T y T_μ son funciones contractivas con respecto a la norma máxima ponderada $\|\cdot\|_\xi$ para todas las estrategias estacionarias μ . Es decir:

$$\| TJ - T\bar{J} \|_\xi \leq \beta \cdot \| J - \bar{J} \|_\xi$$

donde:

$$\| J \|_\xi = \max_{i=1,\dots,n} \frac{|J(i)|}{\xi(i)}$$

En particular existe un cierto valor $\beta < 1$ de tal manera que:

$$\sum_{j=1}^n p_{ij}(u) \leq \beta \cdot \xi(i), \quad \forall i, u \in U(i)$$

Demostración:

Partimos de un problema de búsqueda del camino más corto (SPP - 'Shortest Path Problem') que construimos con las mismas probabilidades de transición que el problema de partida que nos interesa. Los costes de transición los ponemos a -1 .

Sea $\hat{J}(i)$ el coste óptimo asociado al estado i en este nuevo problema. Tenemos que $\forall i = 1, \dots, n$ y toda estrategia estacionaria μ :

$$\begin{aligned} \hat{J}(i) &= -1 + \min_{u \in U(i)} \sum_{j=1}^n p_{ij}(u) \cdot \hat{J}(j) \\ &\leq -1 + \sum_{j=1}^n p_{ij}(\mu(i)) \cdot \hat{J}(j) \end{aligned}$$

Definimos el vector ξ como la solución del problema de Programación Dinámica, e intentaremos llegar al resultado de que cumple la propiedad que buscamos.

Tomamos por tanto: $\xi(i) = -J(i)$, $i = 1, \dots, n$.

Entonces $\forall i$, tenemos que $\xi(i) \geq 1$, y para toda estrategia estacionaria μ se cumple:

$$\sum_{j=1}^n p_{ij}(\mu(i)) \cdot \xi(j) \leq \xi(i) - 1 \leq \beta \cdot \xi(i), \quad i = 1, \dots, n \tag{2.1}$$

donde β se define:

$$\beta = \max_{i=1,\dots,n} \frac{\xi(i) - 1}{\xi(i)} < 1.$$

Para cualquier estrategia estacionaria μ , y estado i , y cualesquiera vectores J y \bar{J} , aplicando la relación 2.1 tenemos:

$$\begin{aligned} |(T_\mu J)(i) - (T_\mu \bar{J})(i)| &= \left| \sum_{j=1}^n p_{ij}(\mu(i)) \cdot (J(j) - \bar{J}(j)) \right| \\ &\leq \sum_{j=1}^n p_{ij}(\mu(i)) \cdot |J(j) - \bar{J}(j)| \\ &\leq \left(\sum_{j=1}^n p_{ij}(\mu(i)) \cdot \xi(j) \right) \cdot \left(\max_{j=1,\dots,n} \frac{|J(j) - \bar{J}(j)|}{\xi(j)} \right) \\ &\leq \beta \cdot \xi(i) \cdot \left(\max_{j=1,\dots,n} \frac{|J(j) - \bar{J}(j)|}{\xi(j)} \right) \end{aligned}$$

Dividiendo ambos lados de la desigualdad por $\xi(i)$, y tomando el máximo sobre i en el lado izquierdo tenemos:

$$\max_{i=1,\dots,n} \frac{|(T_\mu J)(i) - (T_\mu \bar{J})(i)|}{\xi(i)} \leq \beta \cdot \max_{j=1,\dots,n} \frac{|J(j) - \bar{J}(j)|}{\xi(j)}$$

de tal manera que T_μ es una contracción con respecto a la norma máxima ponderada $\|\cdot\|_\xi$. esto conduce también a:

$$(T_\mu J)(i) \leq (T_\mu \bar{J})(i) + \beta \cdot \xi(i) \cdot \max_{j=1,\dots,n} \frac{|J(j) - \bar{J}(j)|}{\xi(j)}$$

tomando el mínimo sobre μ en ambos lados:

$$(TJ)(i) \leq (T\bar{J})(i) + \beta \cdot \xi(i) \cdot \max_{j=1,\dots,n} \frac{|J(j) - \bar{J}(j)|}{\xi(j)} \quad (2.2)$$

e intercambiando el papel de J y \bar{J} :

$$(T\bar{J})(i) \leq (TJ)(i) + \beta \cdot \xi(i) \cdot \max_{j=1,\dots,n} \frac{|J(j) - \bar{J}(j)|}{\xi(j)} \quad (2.3)$$

basta combinar las relaciones 2.2 y 2.3 para obtener:

$$|(TJ)(i) - (T\bar{J})(i)| \leq \beta \cdot \xi(i) \cdot \max_{j=1,\dots,n} \frac{|J(j) - \bar{J}(j)|}{\xi(j)}$$

Dividiendo ambos lados por $\xi(i)$ y tomando el máximo sobre i en el lado derecho, tenemos que T es una contracción con respecto a $\|\cdot\|_\xi$.

QED ----- ■

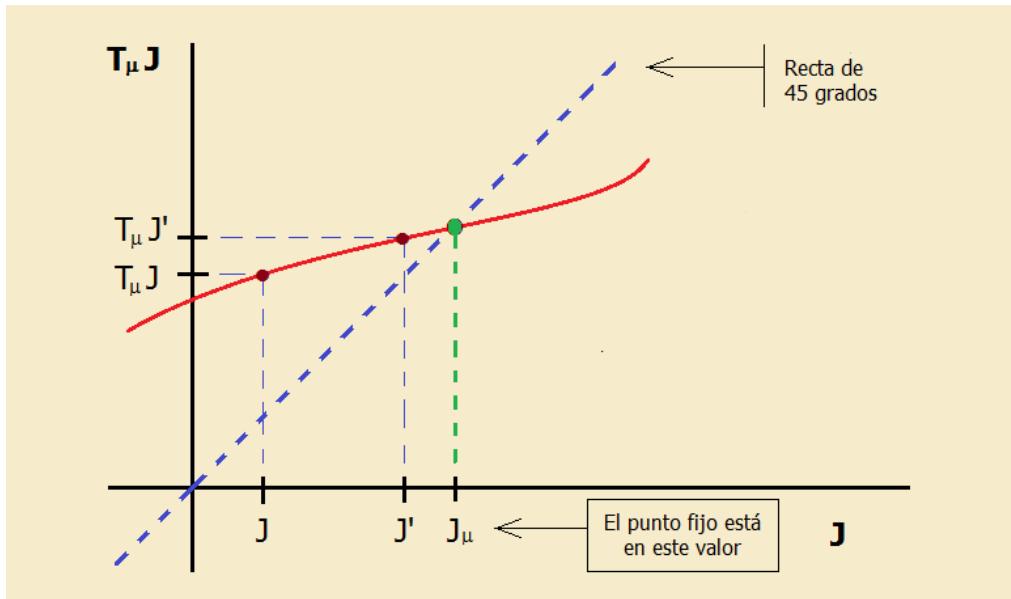


Figura 2.9: Representación del supuesto de contractividad (se utiliza una sola dimensión para facilitar la visualización). También aparece indicado el punto fijo J_μ , que cumple $T_\mu J = J$ (Adaptado de Bertsekas, D. (2022) [21] - pag. 8).

En la figura 2.9 podemos observar una representación de la contractividad. Como se comprobará más adelante, esta propiedad tiene un papel esencial en la convergencia de los algoritmos que vamos a tratar. A continuación analizamos más detenidamente la propiedad contractiva aplicada al caso del operador de Bellman.

PROPOSICIÓN 2.2: Propiedad contractiva del operador de Bellman

Los operadores de Bellman T y T_μ son funciones contractivas de módulo α con respecto a la norma máxima (pag. 190 RL Bertekas):

$$\| J \| = \max_i | J(i) |$$

Verificándose que para todo par de vectores J y J' , tenemos que:

$$\| TJ - TJ' \| \leq \alpha \| J - J' \|$$

$$\| T_\mu J - T_\mu J' \| \leq \alpha \| J - J' \|$$

Demostración:

Sea:

$$c = \max_{x \in X} | J(x) - J'(x) |$$

Entonces tenemos que:

$$J(x) - c \leq J'(x) \leq J(x) + c \quad i = 1, \dots, n$$

Basta aplicar las proposiciones sobre monotonicidad y para obtener:

$$(TJ)(i) - \alpha \cdot c \leq (TJ')(i) \leq (TJ)(i) + \alpha \cdot c \quad i = 1, \dots, n$$

Se sigue de ello que:

$$|(TJ)(i) - (TJ')(i)| \leq \alpha \cdot c \quad i = 1, \dots, n$$

El caso en que empleamos T_μ es directo sin mas que tener en cuenta que en este caso sólo tenemos una decisión en i que es $\mu(i)$.

QED ----- ■

Este resultado nos permite estudiar el comportamiento del algoritmos de programación dinámica.

PROPOSICIÓN 2.3: Convergencia del algoritmo de Programación Dinámica

Para cualquier función acotada $J : X \rightarrow \mathbb{R}$, tenemos que $\forall x \in X$ se verifica el siguiente resultado:

$$J^*(x) = \lim_{N \rightarrow \infty} (T^N J)(x)$$

Demostración:

$\forall N \in \mathbb{N}^+, \forall x_0 \in X$ y estrategia $\pi = \{\mu_0, \mu_1, \dots\}$

$$\begin{aligned} J_\pi(x_0) &= \lim_{N \rightarrow \infty} \mathbb{E}_w \left[\sum_{k=0}^K \alpha^k \cdot g(x, \mu_k(x_k), w) \right] = \\ &\mathbb{E}_w \left[\sum_{k=0}^{N-1} \alpha^k \cdot g(x, \mu_k(x_k), w) \right] + \underbrace{\lim_{K \rightarrow \infty} \mathbb{E}_w \left[\sum_{k=N}^K \alpha^k \cdot g(x, \mu_k(x_k), w) \right]}_{\text{buscamos una cota para esta expresión}} \end{aligned}$$

Si tenemos en cuenta que los valores están acotados:

$$\left| \lim_{K \rightarrow \infty} \mathbb{E}_w \left[\sum_{k=N}^K \alpha^k \cdot g(x, \mu_k(x_k), w) \right] \right| \leq M \sum_{k=N}^{\infty} \alpha^k = M \cdot \frac{\alpha^N}{1-\alpha}$$

Si añadimos un coste terminal $\alpha^N \cdot J(x_N)$ a la n -ésima etapa, y llamamos:

$$\begin{cases} b = M \cdot \frac{\alpha^N}{1-\alpha} + \alpha^N \cdot \max_{x \in X} J(x) \\ -b = -M \cdot \frac{\alpha^N}{1-\alpha} - \alpha^N \cdot \max_{x \in X} J(x) \end{cases}$$

podemos expresar:

$$J_\pi(x_0) - b \leq \mathbb{E}_w \left[\alpha^N \cdot J(x_N) + \sum_{k=0}^{N-1} \alpha^k \cdot g(x, \mu_k(x_k), w) \right] \leq J_\pi(x_0) + b$$

y tomando el mínimo sobre π , tenemos que $\forall x_0$ y N :

$$J^*(x_0) - b \leq (T^N J)(x_0) \leq J^*(x_0) + b$$

basta tomar $\lim_{N \rightarrow \infty}$ para tener:

$$J^*(x_0) \leq (T^N J)(x_0) \leq J^*(x_0)$$

QED ----- ■

PROPOSICIÓN 2.4: Ratio de Convergencia

Para cualquier par de funciones acotadas $J : X \rightarrow \mathbb{R}$ y $J' : X \rightarrow \mathbb{R}$, tenemos que $\forall k = 0, 1, \dots$ se verifica el siguiente resultado:

$$\max_{x \in X} |(T^k J)(x) - (T^k J')(x)| \leq \alpha^k \cdot \max_{x \in X} |(T J)(x) - (T J')(x)|$$

Demostración:

Sea:

$$c = \max_{x \in X} |J(x) - J'(x)|$$

de tal manera que $\forall x \in X$ se cumple:

$$J(x) - c \leq J'(x) \leq J(x) + c$$

Aplicando T^k y empleando los resultados de monotonicidad y variación constante, tenemos que $\forall x \in X$:

$$(T^k J)(x) - \alpha^k c \leq (T^k J')(x) \leq (T^k J)(x) + \alpha^k c$$

De donde se sigue que $\forall x \in X$:

$$| (T^k J)(x) - (T^k J')(x) | \leq \alpha^k c$$

QED ----- ■

2.4. Ecuación de Bellman

Asociado a cada estrategia π existe un vector que contiene el valor de cada uno de los estados. Dichos valores representan los costes/ganancias que se obtienen comenzando en cada uno de ellos y siguiendo en adelante la estrategia en consideración. Estos valores se obtienen resolviendo una ecuación que recibe el nombre de **Ecuación de Bellman para la estrategia μ** . Su principal utilidad reside en que nos permite mejorar la estrategia y obtener otra que conduzca a mejores resultados.

Ecuación de Bellman para una estrategia μ

La ecuación de Bellman para una determinada estrategia μ se obtiene mediante la siguiente expresión general:

$$J_\mu(x) = \mathbb{E}_w [g(x, u, w) + \alpha \cdot J_\mu(f(x, u, w))] \quad \forall x \in X$$

Dicha expresión proporciona un sistema lineal de ecuaciones, cuya solución es el valor óptimo de cada uno de los estados.

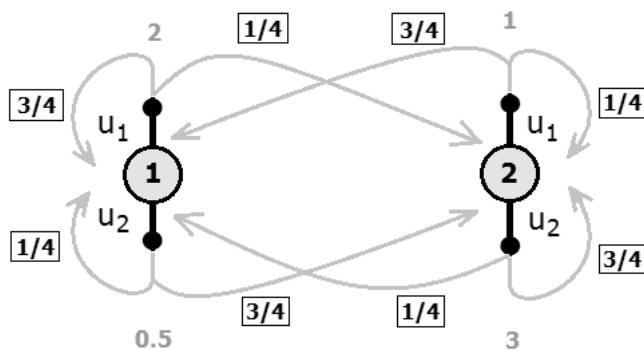
A continuación presentamos un ejemplo de la Ecuación de Bellman para el cálculo del valor de una estrategia en particular.

EJEMPLO 2.3: Ecuación de Bellman para una estrategia μ

En el caso de espacio de estados y espacio de acciones finitos, la ecuación de Bellman para una estrategia μ queda:

$$J_\mu(i) = \sum_{j=1}^n p_{ij}(\mu(i)) \cdot [g_{ij}(\mu(i)) + \alpha \cdot J_\mu(j)], \quad i = 1, 2, \dots, n$$

Si tomamos los datos del ejemplo 2.2: $p_{11}(u_1) = 3/4$, $p_{12}(u_1) = 1/4$, $p_{11}(u_2) = 1/4$, $p_{12}(u_2) = 3/4$ y $p_{21}(u_1) = 3/4$, $p_{22}(u_1) = 1/4$, $p_{21}(u_2) = 1/4$, $p_{22}(u_2) = 3/4$; así como los costes asociados: $g_{11}(u_1) = g_{12}(u_1) = 2$, $g_{11}(u_2) = g_{12}(u_2) = 0,5$, $g_{21}(u_1) = g_{22}(u_1) = 1$, $g_{21}(u_2) = g_{22}(u_2) = 3$.

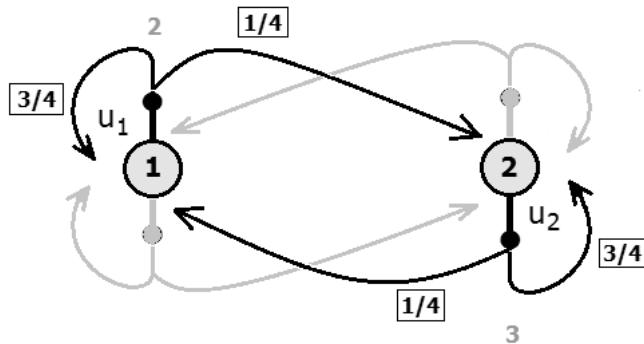


Nos planteamos encontrar el valor asociado a cada estado cuando se parte de él y se sigue a continuación una determinada estrategia μ .

Supongamos que dicha estrategia μ es determinista y viene dada por la relación que aparece a continuación:

$$\mu(s) = \begin{cases} u_1 & \text{si } s = 1 \\ u_2 & \text{si } s = 2 \end{cases}$$

es decir:



Si utilizamos la ecuación de Bellman para la estrategia μ que hemos visto previamente, tendremos la siguiente expresión:

$$J_\mu(i) = \sum_{j=1}^2 p_{ij}(\mu(i)) \cdot [g_{ij}(\mu(i)) + \alpha \cdot J_\mu(j)], \quad i = 1, 2$$

que desarrollada da lugar al siguiente sistema de ecuaciones:

$$\begin{aligned} J_\mu(1) &= p_{11}(u_1) \cdot [g_{11}(u_1) + \alpha \cdot J_\mu(1)] + p_{12}(u_1) \cdot [g_{12}(u_1) + \alpha \cdot J_\mu(2)] \\ J_\mu(2) &= p_{21}(u_2) \cdot [g_{21}(u_2) + \alpha \cdot J_\mu(1)] + p_{22}(u_2) \cdot [g_{22}(u_2) + \alpha \cdot J_\mu(2)] \end{aligned}$$

sin más que sustituir la correspondiente matriz de probabilidades asociada a la estrategia μ del enunciado:

$$\begin{bmatrix} p_{11}(u_1) & p_{12}(u_1) \\ p_{21}(u_2) & p_{22}(u_2) \end{bmatrix} = \begin{bmatrix} 3/4 & 1/4 \\ 1/4 & 3/4 \end{bmatrix}$$

así como $\alpha = 9/10$, y los valores $g_{11}(u_1) = g_{12}(u_1) = 2$ y $g_{21}(u_2) = g_{22}(u_2) = 3$, tendríamos:

$$\begin{cases} 0.325 J_\mu(1) - 0.225 J_\mu(2) = 2 \\ -0.225 J_\mu(1) + 0.325 J_\mu(2) = 3 \end{cases}$$

que es un sistema lineal de ecuaciones, cuya solución es:

$$J_\mu(1) \approx 24.12 \quad \text{y} \quad J_\mu(2) \approx 25.96$$

Tenemos por tanto que siguiendo la estrategia μ , el estado 2 del que partir es ligeramente más beneficioso en términos del valor descontado acumulado futuro.

Caso estocástico y espacio de estados finito. Tal y como veremos a continuación, la expresión de Bellman es una vía para encontrar la estrategia óptima. En la figura 2.10 podemos observar la búsqueda de la decisión óptima y el cálculo de los valores óptimos de los estados.

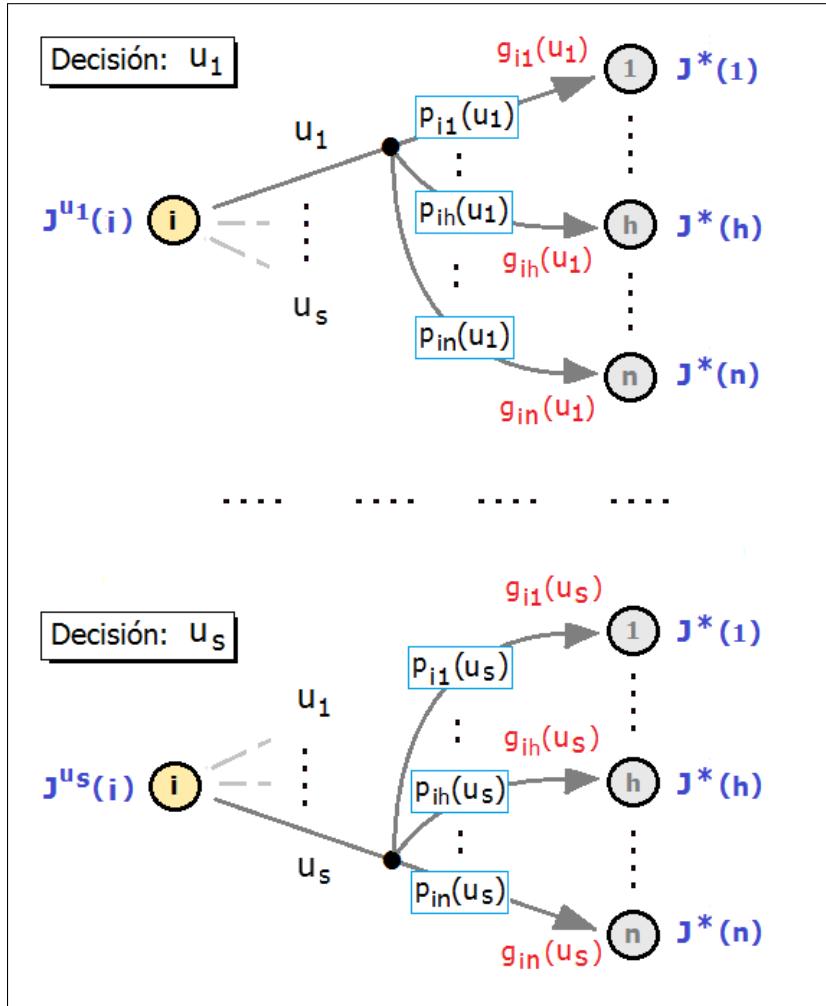


Figura 2.10: *Ecuación de Bellman.* En un entorno no determinista, la función de valor se encuentra minimizando/maximizando la esperanza: del costo/ganancia a un paso ($g_{ij}(u)$, $u \in \{u_1, u_2, \dots, u_s\}$) más el valor óptimo del estado destino ($J^*(j)$, $j = 1, 2, \dots, n$).

Tomando el caso con espacio de estados finito (X_1, X_2, \dots, X_n) y entorno aleatorio, vemos que la optimización a un paso ha de tener en cuenta el valor esperado.

Si denominamos $J^u(i)$ al valor del estado i cuando se toma la decisión u (donde $u \in \{u_1, \dots, u_s\}$) y a partir de ahí se actúa optimamente, el valor óptimo ($J^*(i)$) se obtendrá sin más que estudiar el mínimo de dichos valores: $J^*(i) = \min \{J^{u_1}(i), J^{u_2}(i), \dots, J^{u_s}(i)\}$. La acción óptima: $u \in U(i)$, será aquella que en el estado en cuestión conduce a dicho resultado.

El proceso anterior se generaliza en la llamada **Ecuación de optimalidad de Bellman**, que aparece a continuación en su forma más general.

ECUACIÓN DE BELLMAN (Ecuación de optimalidad)

Dado un estado $x \in X$, una decisión $u \in U(x)$, y un estado destino al que conduce dicha decisión $f(x, u, \omega)$, el valor óptimo del estado de partida se obtiene como:

$$J^*(x) = \min_{u \in U(x)} \mathbb{E}_w [g(x, u, w) + \alpha \cdot J^*(f(x, u, w))] \quad \forall x \in X$$

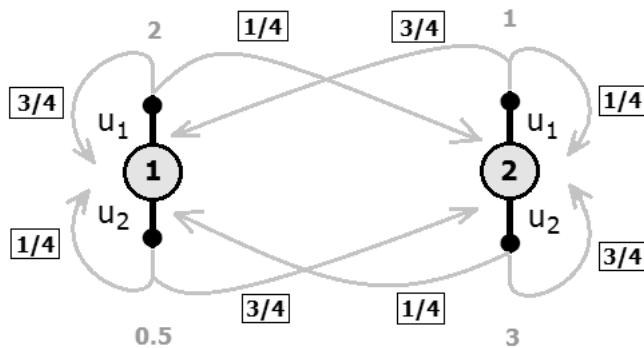
La interpretación de la Ecuación de Optimalidad de Bellman en términos de Programación Dinámica, consiste en que el valor óptimo del estado i ($J^*(i)$) es el mínimo del coste de la etapa actual más el valor esperado óptimo para el resto de etapas teniendo en cuenta el estado destino j , cuyo valor es $J^*(j)$.

EJEMPLO 2.4: Ecuación de Bellman (ecuación de optimalidad)

Imaginemos que nos encontramos ante el problema de determinar la secuencia de decisiones óptimas en un problema de decisiones secuenciales, para lo cual necesitamos obtener el valor de cada estado para actuar óptimamente. En el caso de un conjunto de estados finito: X_1, X_2, \dots, X_n (o más abreviadamente $i = 1, 2, \dots, n$) tendríamos:

$$J^*(i) = \min_{u \in U(i)} \left[\sum_{j=1}^n p_{ij}(u) \cdot [g_{ij}(u) + \alpha \cdot J^*(j)] \right], \quad i = 1, 2, \dots, n$$

Si para cada estado el conjunto de decisiones es: $U(i) = \{u_1, u_2, \dots, u_k\}$, el valor óptimo del estado i se obtendría tras aplicar la expresión que acabamos de ver. El procedimiento quedaría representado en la siguiente figura, donde vemos los dos principales elementos que entran en juego: un coste inmediato asociado a la decisión, y un coste óptimo del estado destino: $J^*(1), \dots, J^*(j), \dots, J^*(n)$.



Si los valores resultantes, correspondientes a cada acción posible en el estado i , son como aparece en la figura superior tendríamos que: $J^*(i) = \min \{ J^1(i), J^2(i), \dots, J^k(i) \}$. La acción óptima es aquella $u \in U(i)$ que conduce a dicho resultado.

PROPOSICIÓN 2.5: J^* es la única solución a la ecuación de Bellman

La función de coste óptimo J^* , satisface la siguiente expresión, que recibe el nombre de Ecuación de Bellman:

$$J^*(x) = \min_{u \in U(x)} \mathbb{E}_w [g(x, u, w) + \alpha \cdot J^*(f(x, u, w))] \quad \forall x \in X$$

o equivalentemente:

$$J^*(x) = TJ^*$$

Es decir; J^* es solución de la ecuación de Bellman, y aún es más: entre la clase de funciones acotadas es la única solución.

Demostración:

$$\begin{aligned} J^*(x) - M \cdot \frac{\alpha^N}{1-\alpha} &\leq (T^N J_0)(x) \leq J^*(x) + M \cdot \frac{\alpha^N}{1-\alpha} \\ (TJ)^*(x) - M \cdot \frac{\alpha^{N+1}}{1-\alpha} &\leq (T^{N+1} J_0)(x) \leq J^*(x) + M \cdot \frac{\alpha^{N+1}}{1-\alpha} \end{aligned}$$

basta tomar $\lim_{N \rightarrow \infty}$ para tener:

$$\lim_{N \rightarrow \infty} (T^{N+1} J_0)(x) = J^*(x)$$

de donde tenemos que:

$$J^* = TJ^*$$

QED ----- ■

TEOREMA 2.1: J^* es el punto fijo del operador de Bellman

Dada una función $v : X \rightarrow \mathbb{R}$ donde $v(x) > 0$, $\forall x \in X$. $B(X)$ denotará el espacio de las funciones real valoradas J ($J : X \rightarrow \mathbb{R}$) X tal que $J(x)/v(x)$ está acotada $\forall x \in X$.

Sea $T : B(X) \rightarrow B(X)$ el **operador de Bellman**, donde T es una función contractiva con módulo $\alpha \in (0, 1)$. Entonces existe un único valor $J^* \in B(X)$ tal que:

$$J^* = TJ^*$$

además:

a) Tenemos que se cumple la siguiente relación:

$$\| T^k J - J^* \| \leq \alpha^k \| J - J^* \|, \quad k = 1, 2, \dots$$

b) La secuencia converge al punto fijo

$$\{T^k J\} \rightarrow J^* \quad \forall J \in B(X)$$

Resumen - Ideas principales

En el capítulo hemos planteado la modelización del proceso de toma de decisiones secuenciales por etapas. Una vez expresado el problema que pretendemos abordar, planteamos las herramientas para su resolución. En concreto en resultados de programación dinámica.

1. Analizamos la evaluación de estrategias.
2. Posteriormente analizamos la generación del óptimo de entre todas ellas.
3. La iteración del algoritmo de Programación Dinámica ($J_{k+1} = TJ_k$) converge a la función de coste óptimo J^* , que es solución única de la ecuación de Bellman $J = TJ$.

Capítulo 3

Métodos de solución

En el capítulo anterior hemos abordado el análisis y la formalización del problema de la toma de decisiones secuenciales; así como la presentación de formas de búsqueda de la solución. En el presente capítulo nos adentraremos en el estudio de los distintos métodos que se han desarrollado para encontrar el valor óptimo que buscamos.

Si recordamos los análisis que llevamos a cabo, la manera de encontrar los valores óptimos consiste en dividir recursivamente el problema y optimizar a un sólo paso. La ecuación de Bellman viene a recoger esta idea y su solución conduce a los valores óptimos de los estados que buscamos. En ciertas condiciones existe una solución única que es el punto fijo de la ecuación.

Distinguiremos:

- **Solución exacta.** Utiliza métodos de Programación Dinámica. Requiere el conocimiento de las probabilidades de transición entre los estados
- **Solución aproximada.** Se basa en el empleo de métodos de Simulación.

	a_1	\dots	a_j	\dots	a_m
s_1					
:					
s_i			Valor de (s_i, a_j)		
:					
s_n					

Cuadro 3.1: *Representación tabular del conjunto de estados-acciones.*

Cuando el cardinal de los conjuntos de estados (\mathcal{S}) y acciones/decisiones del decisor (\mathcal{A}) son finitos y no muy grandes ($|\mathcal{S}| = n$ y $|\mathcal{A}| = m$) se emplea una representación matricial de los valores asociados a cada combinación estado-acción (**representación tabular** - Ver tabla 3.1). En el caso contrario en el que el cardinal es muy elevado, se emplean métodos de aproximación funcional de la función de valor. A menos que se especifique lo contrario el análisis que llevaremos a cabo se centrará en el primer caso.

3.1. Solución exacta: Programación Dinámica

Si disponemos de toda la información del comportamiento del sistema es posible encontrar la función de valor de manera exacta, lo que nos permite obtener la secuencia de decisiones óptimas. La aplicación de las ideas que hemos visto de programación dinámica genera 2 tipos de algoritmos, que son clásicos en este ámbito:

-Algoritmo de Iteración de valor (*Value Iteration*).

-Algoritmo de Iteración de Estrategias (*Policy Iteration*).

A continuación pasamos a detallarlos.

3.1.1. Iteración de valor (*Value Iteration*)

Comenzamos dando la expresión formal del algoritmo y enunciando el principal resultado referente a la convergencia del algoritmo.

ITERACIÓN DE VALOR (*V.I. - Value Iteration*)

La aplicación reiterada del algoritmo DP para generar una secuencia $T^k J$ comenzando por un valor J , se denomina **value iteration** y es el principal método para el cálculo del vector de coste óptimo futuro J^* . (pag. 24 Neuro-Dynamic Pr).

$$J_{k+1}(x) = \min_{u \in U(x)} \mathbb{E}_w [g(x, u, w) + \alpha \cdot J_k(f(x, u, w))] \quad k = 0, 1, \dots$$

(Observar que ahora, considerando el horizonte infinito, los órdenes temporales aparecen cambiados, ya que se empieza con el primer estado, para así poder encadenar hacia delante de manera indefinida. Generalmente empleando una condición inicial: $J_0(x) = 0 \forall x$)

La siguiente proposición presenta el principal resultado relativo a la convergencia de este algoritmo a los verdaderos valores óptimos de los estados.

PROPOSICIÓN 3.1: Convergencia del algoritmo de Iteración de Valor (*V.I.*)

Dadas unas condiciones iniciales cualesquiera: $J_0(x) \in B(X)$, la secuencia: $\{J_k(x)\} = \{TJ(x), T^2J(x), \dots\}$ generada por el algoritmo de Iteración de Valor (*Value Iteration*), y donde T es una función contractiva, converge a la función de coste óptimo $J^*(x) \quad \forall x \in X$.

$$J_k \longrightarrow J^*(x)$$

Además la secuencia de errores $| (T^k J)(x) - J^*(x) |$ está acotada por una constante múltiplo de α^k , $\forall x \in X$. Es decir: (vol. II - pag. 120)

$$| (T^k J)(x) - J^*(x) | \leq \alpha^k | J(x) - J^*(x) | \quad k = 0, 1, 2, \dots$$

En la siguiente figura podemos observar la representación geométrica de la evolución del algoritmo. Se comprueba cómo la secuencia de estimaciones generadas por el algoritmo converge al punto fijo.

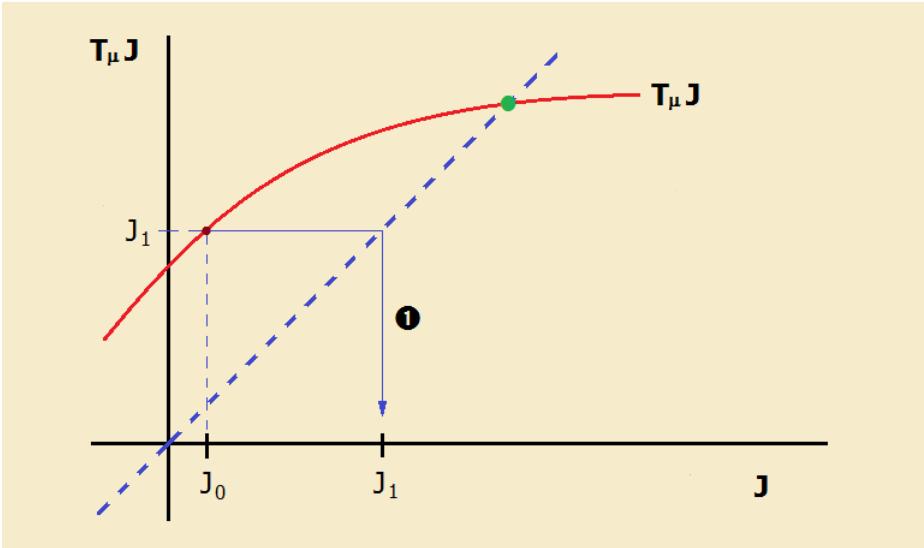
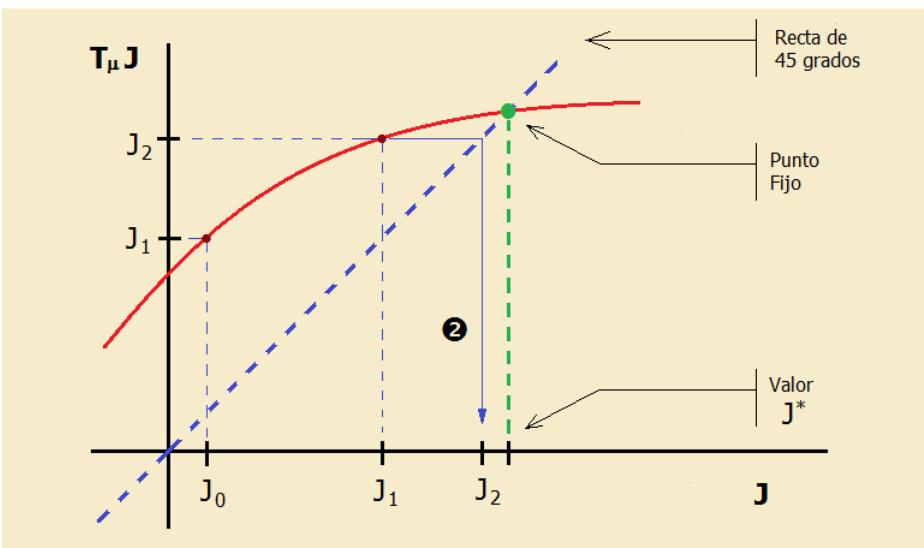
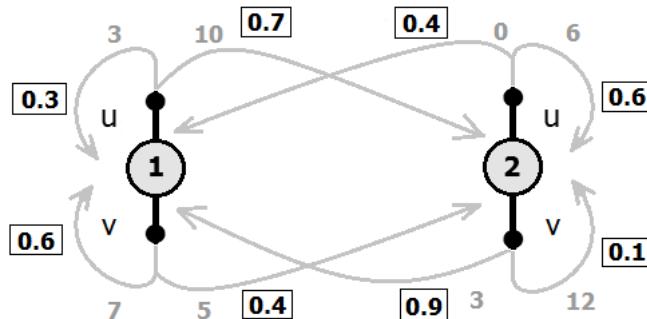
Inicio:**Iteración:**

Figura 3.1: Representación del algoritmo VI (Value Iteration) y el punto al que convergen la secuencia de iteraciones. El punto fijo J^* , que cumple $TJ^* = J$, aparece señalado en el gráfico (Adaptado de Bertsekas, D. (2022) [21] - pag. 8 y 71).

El siguiente ejemplo permite ilustrar el algoritmo de Iteración de Valor (Adaptado de un ejemplo propuesto en *Reinforcement Learning*” (2018) Cap. 2 - Barto, Andrew G. & Sutton, Richard S. [181]).

EJEMPLO 3.1: Value Iteration (espacio de estados finito)

Supongamos un sistema con un conjunto de estados finito: X_1, X_2 (o más breviadamente $i = 1, 2$), y acciones: $U(i) = \{u, v\}$. La dinámica de transiciones viene dada por el siguiente gráfico (Ejemplo desarrollado a partir de Bertsekas, D. (2022) [21] - pag. 30):



Supongamos entonces que las probabilidades de transición son: $p_{11}(u) = 0.3$, $p_{12}(u) = 0.7$, $p_{11}(v) = 0.6$, $p_{12}(v) = 0.4$ y $p_{21}(u) = 0.4$, $p_{22}(u) = 0.6$, $p_{21}(v) = 0.9$, $p_{22}(v) = 0.1$.

Junto a los costes asociados: $g_{11}(u) = 3$, $g_{12}(u) = 10$, $g_{11}(v) = 7$, $g_{12}(v) = 5$, $g_{21}(u) = 0$, $g_{22}(u) = 6$, $g_{21}(v) = 3$, $g_{22}(v) = 12$.

Imaginemos que nos encontramos ante el problema de determinar la secuencia de decisiones que minimizan el coste. El valor óptimo del estado i ($i = 1, 2$) cumple la siguiente expresión:

$$J^*(i) = \min_{a \in \{u, v\}} \left[\sum_{j=1}^2 p_{ij}(a) \cdot [g_{ij}(a) + \alpha \cdot J^*(j)] \right], \quad i = 1, 2$$

Es decir, en el óptimo $J^* = \{J^*(1), J^*(2)\}$ se verifica que: $TJ^* = J^*$. Al ser ecuaciones no lineales no podemos resolverlo directamente, y necesitamos algoritmos iterativos como el de Iteración de Valor: $J_{k+1} = T \cdot J_k$. Desarrollando la expresión anterior tenemos:

$$\begin{aligned} J_{k+1}(1) &= \min \{ p_{11}(u) \cdot [g_{11}(u) + \alpha \cdot J_k(1)] + p_{12}(u) \cdot [g_{12}(u) + \alpha \cdot J_k(2)] , \\ &\quad p_{11}(v) \cdot [g_{11}(v) + \alpha \cdot J_k(1)] + p_{12}(v) \cdot [g_{12}(v) + \alpha \cdot J_k(2)] \} \end{aligned}$$

$$\begin{aligned} J_{k+1}(2) &= \min \{ p_{21}(u) \cdot [g_{21}(u) + \alpha \cdot J_k(1)] + p_{22}(u) \cdot [g_{22}(u) + \alpha \cdot J_k(2)] , \\ &\quad p_{21}(v) \cdot [g_{21}(v) + \alpha \cdot J_k(1)] + p_{22}(v) \cdot [g_{22}(v) + \alpha \cdot J_k(2)] \} \end{aligned}$$

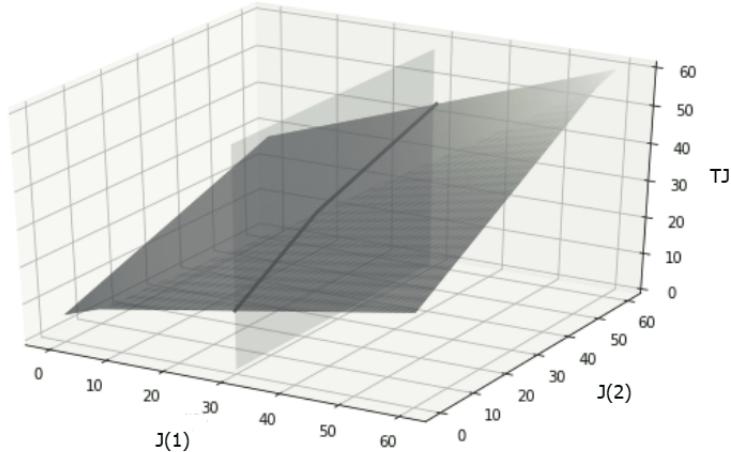
sustituyendo las probabilidades de transición y los costes tenemos:

$$\begin{aligned} J_{k+1}(1) &= \min \{ 0.3 \cdot [3 + 0.9 \cdot J_k(1)] + 0.7 \cdot [10 + 0.9 \cdot J_k(2)] , \\ &\quad 0.6 \cdot [7 + 0.9 \cdot J_k(1)] + 0.4 \cdot [5 + 0.9 \cdot J_k(2)] \} \end{aligned} \tag{3.1}$$

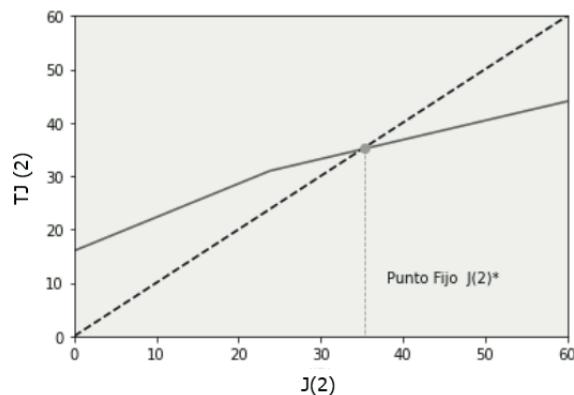
$$\begin{aligned} J_{k+1}(2) &= \min \{ 0.4 \cdot [0 + 0.9 \cdot J_k(1)] + 0.6 \cdot [6 + 0.9 \cdot J_k(2)] , \\ &\quad 0.9 \cdot [3 + 0.9 \cdot J_k(1)] + 0.1 \cdot [12 + 0.9 \cdot J_k(2)] \} \end{aligned} \tag{3.2}$$

Esta expresión nos permite encadenar los valores hasta el fin del algoritmo.

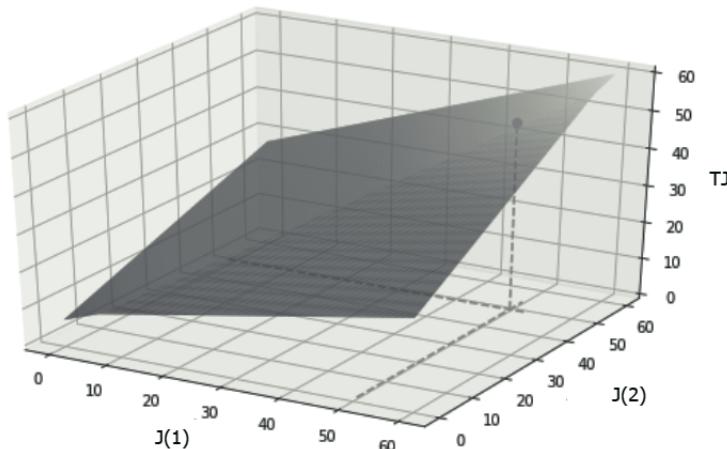
Para tener una idea intuitiva del proceso podemos representar gráficamente los valores $(J(1), J(2), TJ)$ aplicando 3.1 y 3.2.



Si fijamos un valor arbitrario $J(1) = 30$, por ejemplo, podemos ver el corte de dicho plano en la curva TJ . En la figura siguiente añadimos la recta de 45° grados a la proyección sobre el plano, lo que nos permite identificar el punto al que tendería el proceso iterativo.



El punto fijo al que tiende el algoritmo de Iteración de Valor (VI) aparece en la siguiente figura, junto al valor que alcanza la curva TJ (Ver código PYTHON en el Apéndice).



Se suele tomar 0 como valor inicial del algoritmo, aunque tomar un valor arbitrario más cercano para realizar menos iteraciones converge igualmente (Bertsekas, D. (2020) [22] - pag. 23). Por ejemplo: $J_0(1) = J_0(2) = 10$. Sustituyendo en las ecuaciones 3.1 y 3.2 y tenemos:

$$\begin{aligned} J_1(1) &= TJ_0(1) = \min \{ 0.3 \cdot [3 + 0.9 \cdot 10] + 0.7 \cdot [10 + 0.9 \cdot 10] , \\ &\quad 0.6 \cdot [7 + 0.9 \cdot 10] + 0.4 \cdot [5 + 0.9 \cdot 10] \} \\ &= \min \{ 0.3 \cdot 12 + 0.7 \cdot 19 , 0.6 \cdot 16 + 0.4 \cdot 14 \} \\ &= \min \{ 3.6 + 13.3 , 9.6 + 5.6 \} = \min \{ 16.9, 15.2 \} = 15.2 \end{aligned}$$

$$\begin{aligned} J_1(2) &= TJ_0(2) = \min \{ 0.4 \cdot [0 + 0.9 \cdot 10] + 0.6 \cdot [6 + 0.9 \cdot 10] , \\ &\quad 0.9 \cdot [3 + 0.9 \cdot 10] + 0.1 \cdot [12 + 0.9 \cdot 10] \} \\ &= \min \{ 0.4 \cdot 9 + 0.6 \cdot 15 , 0.9 \cdot 12 + 0.4 \cdot 14 \} \\ &= \min \{ 3.6 + 9 , 10.8 + 5.6 \} = \min \{ 12.6, 16.4 \} = 12.6 \end{aligned}$$

Aplicando el algoritmo iterativo VI (Value Iteration) (representado gráficamente en la figura 3.1) tenemos los siguientes resultados numéricos.

	TJ(1)	TJ(2)
0	10.000000	10.000000
1	15.200000	12.600000
2	18.944000	15.876000
3	22.145120	18.992880
...
77	50.573678	47.402946
78	50.574847	47.404115
79	50.575899	47.405167
80	50.576845	47.406114

Podemos observar en el fragmento de salida que los valores se van estabilizando hacia la iteración 80. El punto fijo es $J^* = (J^*(1), J^*(2)) = (50.59, 47.41)$.

Como se puede comprobar en el ejemplo anterior, el algoritmo de **Iteración de valor** proporciona una secuencia de valores que progresivamente se aproximan a los valores correctos. La principal dificultad que entraña este algoritmo reside en que el número de iteraciones necesarias es infinito, y por lo tanto hemos de quedarnos con una aproximación que consideremos razonablemente cercana al valor límite.

Para solventar este problema existe el siguiente algoritmo que permite ir acercándose sucesivamente a través de un procedimiento de evaluación de una estrategia de partida y su mejora posterior. Se trata del algoritmo de **Iteración de estrategias (Policy Iteration)**.

3.1.2. Iteración de estrategia (*Policy Iteration*)

El algoritmo que presentamos se basa igualmente en las ideas y conceptos de Programación Dinámica, y permite obtener la solución exacta. Implica dos tipos de pasos (*Policy Evaluation* y *Policy Improvement*) que pasamos a detallar a continuación.

ITERACIÓN DE ESTRATEGIA (*PI - Policy Iteration*)

Consiste en empezar por una determinada estrategia, mediante la cual se lleva a cabo un paso de evaluación J . A partir de estos valores se procede a una mejora estratégica eligiendo las acciones que conducen a un mejor valor que el que se tenía (*policy improvement step*) (pag. 29 Neuro-Dynamic Pr) y (pag. 202 RL).

- **Policy Evaluation**

Computar $J_{\mu^k}(x) \forall x \in X$ como solución del sistema de ecuaciones lineales de Bellman asociadas a la estrategia (*policy*) μ :

$$J_{\mu^k}(x) = \mathbb{E}_w [g(x, \mu^k(x), w) + \alpha \cdot J_{\mu^k}(f(x, \mu^k(x), w))] \quad \forall x \in X$$

- **Policy Improvement**

Posteriormente calcular la nueva estrategia μ^{k+1} .

$$\mu^{k+1} \in \arg \min_{u \in U(x)} \mathbb{E}_w [g(x, u, w) + \alpha \cdot J_{\mu^k}(f(x, u, w))] \quad \forall x \in X$$

El proceso de Evaluación-Mejora se repite empleando la nueva estrategia obtenida (μ^{k+1}) para resolver el sistema de ecuaciones de Bellman, y posteriormente se vuelve a determinar la estrategia mejorada correspondiente a esa solución.

Los pasos finalizan cuando: $J_{\mu^{k+1}}(x) = J_{\mu^k}(x) \forall x \in X$, momento en el que disponemos de la estrategia óptima (μ^k).

La siguiente proposición recoge el principal resultado sobre convergencia a los valores límites del algoritmo que acabamos de presentar.

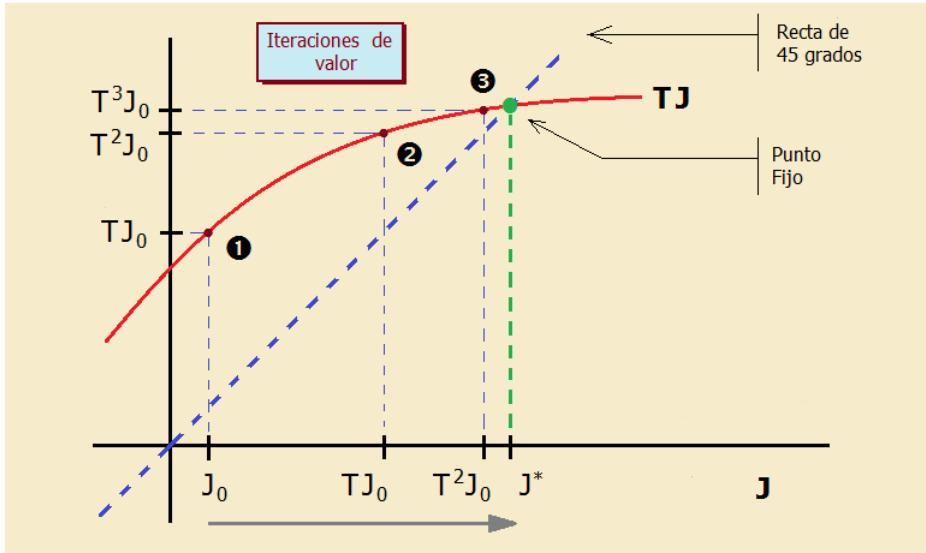
PROPOSICIÓN 3.2: Convergencia del algoritmo de Iteración de Estrategias

El algoritmo de Iteración de Estrategias (*Policy Iteration*) genera una secuencia de estrategias con mejora creciente y finaliza con la obtención de la estrategia óptima.

En la siguiente figura podemos observar la representación geométrica de la evolución del algoritmo. Se comprueba cómo la secuencia de estimaciones generadas por el algoritmo converge al punto fijo.

En el gráfico de la parte de arriba podemos observar algunas iteraciones de valor. En el gráfico abajo podemos analizar una iteración de estrategia. Vemos como ésta obtiene una valor que se acerca al valor óptimo, que es el punto fijo J^* .

VI (Value Iteration):



PI (Policy Iteration):

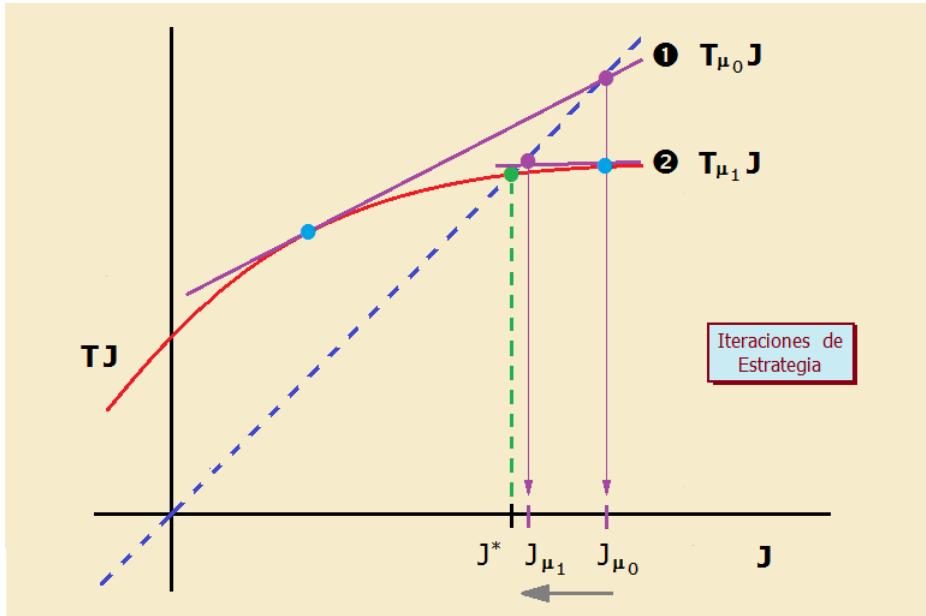


Figura 3.2: Representación del algoritmo PI (Policy Iteration) y el punto al que convergen la secuencia de iteraciones. El punto fijo J^* , que cumple $TJ^* = J$, aparece señalado en el gráfico (Adaptado de Bertsekas, D. (2022) [21] - pag. 71).

El siguiente ejemplo permite ilustrar el algoritmo de Iteración de Estrategia (Policy Iteration).

EJEMPLO 3.2: Policy Iteration (espacio de estados finito)

En el algoritmo Policy Iteration, en el caso de un espacio de estados finito las reglas de actualización adoptarían la forma:

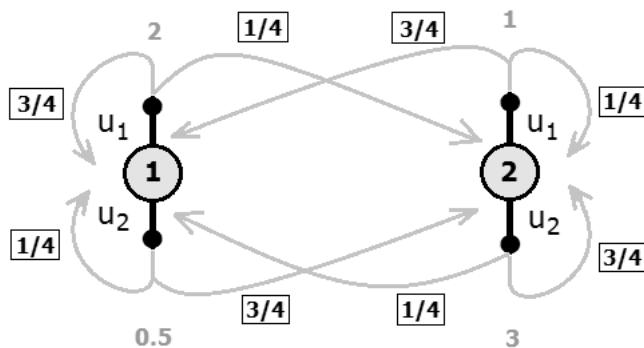
Policy Evaluation:

$$J_{\mu^k}(i) = \sum_{j=1}^n p_{ij}(\mu^k(i)) \cdot [g_{ij}(\mu^k(i)) + \alpha \cdot J_{\mu^k}(j)] \quad (3.3)$$

Policy Improvement:

$$\mu^{k+1}(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^n p_{ij}(u) [g_{ij}(u) + \alpha \cdot J_{\mu^k}(j)] \quad (3.4)$$

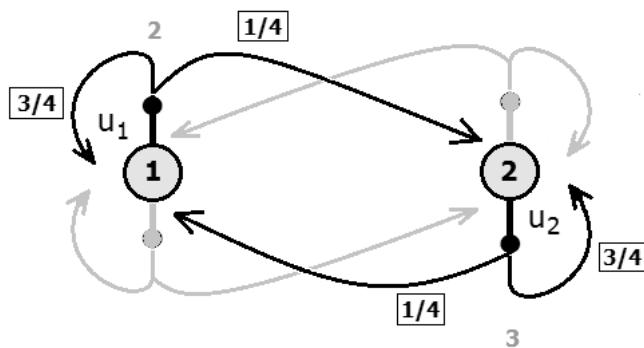
El proceso se repite hasta que la variación en los valores de la función de valor no sea apreciable:
Ver: pag. 202 de RL - Bertsekas. Procedemos a aplicarlo al siguiente ejemplo.



Para ilustrar la aplicación de la técnica acudimos al siguiente ejemplo (Ver: Bertsekas, D. P. (2012) [25] - pag. 99).

Supongamos que la estrategia μ es determinista y viene dada por la relación que aparece a continuación:

$$\boxed{k=0} \quad \mu^0(i) = \begin{cases} u_1 & \text{si } i = 1 \\ u_2 & \text{si } i = 2 \end{cases}$$



Teniendo en cuenta que: $p_{11}(u_1) = 3/4$, $p_{12}(u_1) = 1/4$, $p_{11}(u_2) = 1/4$, $p_{12}(u_2) = 3/4$ y $p_{21}(u_1) = 3/4$, $p_{22}(u_1) = 1/4$, $p_{21}(u_2) = 1/4$, $p_{22}(u_2) = 3/4$; los costes asociados: $g_{11}(u_1) = g_{12}(u_1) = 2$, $g_{11}(u_2) = g_{12}(u_2) = 0,5$, $g_{21}(u_1) = g_{22}(u_1) = 1$, $g_{21}(u_2) = g_{22}(u_2) = 3$, así como $\alpha = 9/10$, y los valores $g_{11}(u_1) = g_{12}(u_1) = 2$ y $g_{21}(u_2) = g_{22}(u_2) = 3$, procedemos a iniciar el algoritmo..

Policy Evaluation:

Expresamos una ecuación para cada estado ($i = 1, 2$) aplicando 3.3, y teniendo en cuenta que $\mu^0(1) = u_1$ y $\mu^0(2) = u_2$.

$$\begin{aligned} J_{\mu^0}(1) &= p_{11}(u_1) \cdot [g_{11}(u_1) + \alpha \cdot J_{\mu^0}(1)] + p_{12}(u_1) \cdot [g_{12}(u_1) + \alpha \cdot J_{\mu^0}(2)] \\ J_{\mu^0}(2) &= p_{21}(u_2) \cdot [g_{21}(u_2) + \alpha \cdot J_{\mu^0}(1)] + p_{22}(u_2) \cdot [g_{22}(u_2) + \alpha \cdot J_{\mu^0}(2)] \end{aligned}$$

Tenemos que las probabilidades son las siguientes:

$$\begin{bmatrix} p_{11}(u_1) & p_{12}(u_1) \\ p_{21}(u_2) & p_{22}(u_2) \end{bmatrix} = \begin{bmatrix} 3/4 & 1/4 \\ 1/4 & 3/4 \end{bmatrix}$$

Por lo que sustituyendo y operando convenientemente el sistema de ecuaciones queda:

$$\begin{cases} 0.325 J_{\mu^0}(1) - 0.225 J_{\mu^0}(2) = 2 \\ -0.225 J_{\mu^0}(1) + 0.325 J_{\mu^0}(2) = 3 \end{cases}$$

Cuya solución es (24.09, 25.9) (Ver Apéndice con el código Python). Se expone la del libro:

$$J_{\mu^0}(1) \approx 24.12 \quad \text{y} \quad J_{\mu^0}(2) \approx 25.96$$

Policy Improvement:

Procedemos a analizar las posibles mejoras de la estrategia de partida aplicando 3.4.

$$\begin{aligned} TJ_{\mu^0}(1) &= \min \{3/4 \cdot [2 + 0.9 \cdot \mathbf{24.12}] + 1/4 \cdot [2 + 0.9 \cdot \mathbf{25.96}], \\ &\quad 1/4 \cdot [0.5 + 0.9 \cdot \mathbf{24.12}] + 3/4 \cdot [0.5 + 0.9 \cdot \mathbf{25.96}]\} \\ &= \min \{17.78 + 6.34, 5.55 + 17.9\} = \min \{24.12, 23.45\} = 23.45 \end{aligned}$$

$$\begin{aligned} TJ_{\mu^0}(2) &= \min \{3/4 \cdot [1 + 0.9 \cdot \mathbf{24.12}] + 1/4 \cdot [1 + 0.9 \cdot \mathbf{25.96}], \\ &\quad 1/4 \cdot [3 + 0.9 \cdot \mathbf{24.12}] + 3/4 \cdot [3 + 0.9 \cdot \mathbf{25.96}]\} \\ &= \min \{17.03 + 6.09, 6.18 + 19.77\} = \min \{23.12, 25.95\} = 23.12 \end{aligned}$$

de aquí podemos extraer una mejor estrategia:

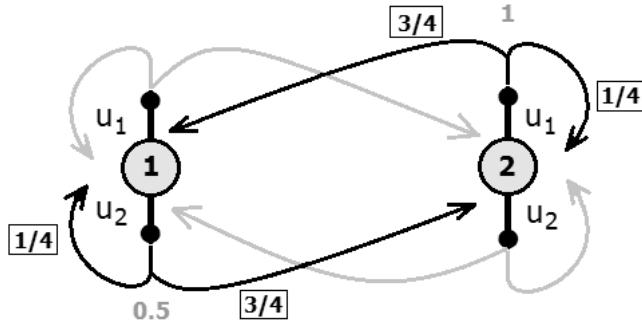
$$u_2 = \arg \min_{u \in \{u_1, u_2\}} TJ_{\mu^0}(1) \implies \mu^1(1) = u_2$$

$$u_1 = \arg \min_{u \in \{u_1, u_2\}} TJ_{\mu^0}(2) \implies \mu^1(2) = u_1$$

Ahora procedemos a la siguiente iteración del algoritmo.

k=1

$$\mu^1(i) = \begin{cases} u_2 & \text{si } i = 1 \\ u_1 & \text{si } i = 2 \end{cases}$$



Policy Evaluation:

Aplicamos nuevamente la ecuación 3.3, considerando ahora que: $\mu^1(1) = u_1$ y $\mu^1(2) = u_2$.

$$\begin{aligned} J_{\mu^1}(1) &= p_{11}(u_2) \cdot [g_{11}(u_2) + \alpha \cdot J_{\mu^1}(1)] + p_{12}(u_2) \cdot [g_{12}(u_2) + \alpha \cdot J_{\mu^1}(2)] \\ J_{\mu^1}(2) &= p_{21}(u_1) \cdot [g_{21}(u_1) + \alpha \cdot J_{\mu^1}(1)] + p_{22}(u_1) \cdot [g_{22}(u_1) + \alpha \cdot J_{\mu^1}(2)] \end{aligned}$$

sustituyendo los valores y operando convenientemente, el sistema de ecuaciones queda:

$$\begin{cases} 0.775 J_{\mu^1}(1) - 0.675 J_{\mu^1}(2) = 0.5 \\ -0.675 J_{\mu^1}(1) + 0.775 J_{\mu^1}(2) = 1 \end{cases}$$

Cuya solución es (7.327, 7.672) (Ver Apéndice con el código Python):

$$J_{\mu^1}(1) \approx 7.33 \quad \text{y} \quad J_{\mu^1}(2) \approx 7.67$$

Policy Improvement:

Procedemos a analizar las posibles mejoras de la estrategia de partida aplicando 3.4.

$$\begin{aligned} TJ_{\mu^0}(1) &= \min \{ 3/4 \cdot [2 + 0.9 \cdot 7.33] + 1/4 \cdot [2 + 0.9 \cdot 7.67], \\ &\quad 1/4 \cdot [0.5 + 0.9 \cdot 7.33] + 3/4 \cdot [0.5 + 0.9 \cdot 7.67] \} \\ &= \min \{ 6.45 + 2.22, 1.77 + 5.55 \} = \min \{ 8.67, 7.32 \} = 7.32 \\ TJ_{\mu^0}(2) &= \min \{ 3/4 \cdot [1 + 0.9 \cdot 7.33] + 1/4 \cdot [1 + 0.9 \cdot 7.67], \\ &\quad 1/4 \cdot [3 + 0.9 \cdot 7.33] + 3/4 \cdot [3 + 0.9 \cdot 7.67] \} \\ &= \min \{ 5.7 + 1.97, 2.4 + 7.42 \} = \min \{ 7.67, 9.83 \} = 7.67 \end{aligned}$$

Como tanto $J_{\mu^1}(1) = TJ_{\mu^1}(1)$, como $J_{\mu^1}(2) = TJ_{\mu^1}(2)$, tenemos que μ^1 es óptima y entonces: $J_{\mu^1} = J^*$. Donde: $J^* = (J^*(1), J^*(2)) = (7.33, 7.67)$.

El algoritmo Policy Iteration conduce a la estrategia:

$$\mu^*(i) = \begin{cases} u_2 & \text{si } i = 1 \\ u_1 & \text{si } i = 2 \end{cases}$$

3.2. Solución aproximada: Métodos de Simulación I (*Value based methods*)

Si repasamos el camino la secuencia de pasos que se han dado, tenemos que hemos llegado a 2 métodos computacionales que nos permiten obtener la solución exacta del problema de Programación Dinámica que planteamos al principio:

- **Value Iteration** Consiste en la iteración reiterada con el algoritmo PD. La secuencia de valores que genera ($T^k J$) a partir de un cierto conjunto de valores iniciales (J_0) converge al vector de valores óptimos (J^*). Requiere generalmente un número infinito de iteraciones, aunque en algunos casos se puede lograr con un número finito de pasos (como en el problema determinista del Camino más corto)
- **Policy Iteration** Se compone de dos pasos (*Policy evaluation* y *Policy improvement*), y es capaz de generar una secuencia de estrategias mejoradas.

En la medida en que ahora tratamos el caso en el que no disponemos de un modelo que disponga explícitamente de la estructura de costes y las probabilidades de transición entre los estados, pero disponemos de un simulador del sistema; podemos plantearnos diversas vías para obtener estimaciones mediante Simulación.

- Podemos utilizar simulaciones repetidas para generar estimaciones de las probabilidades de transición
- O bien podemos generar diversas trayectorias, y estimar $J_\mu(i)$, sin manejar explícitamente las probabilidades

La vía por la que optaremos es la segunda. En lo referente al cálculo de los valores, hay que tener en cuenta aspectos prácticos, como son el almacenamiento en memoria.

Cuando el espacio de estados es de pequeña dimensión y se calcula explícitamente el valor $J(i)$ para cada uno de los estados ($i \in X$) estamos ante el caso de **representación tabular**. Si por el contrario el número de estados es muy elevado el almacenamiento y tratamiento explícito de todos los valores puede volverse impráctico, por lo que se suele acudir a alguna forma de *aproximación funcional*, que denominamos **representación compacta**. Aquí se ajusta una función que depende de un conjunto de parámetros, y mediante ella se puede aproximar cualquier valor.

Los 2 principales métodos para la obtención del valor de cada estado son:

- **Método de Montecarlo** Utiliza simulaciones de trayectorias completas de transiciones, y solo al final estima los valores $J(i)$, $i \in X$.
- **Método de las Diferencias Temporales (TD: Temporal Differences)** Es un método de aprendizaje secuencial que actualiza paso a paso la función de valor.

Con estos dos métodos podemos construir versiones de los algoritmos previamente tratados para el caso del cálculo exacto. El estudio de la convergencia se realiza acudiendo generalmente a resultados que pueden explotar la monotonicidad contractividad de las reglas de actualización que se emplean.

3.2.1. Método de Montecarlo

Partimos de una estrategia estacionaria (y propia) μ , y pretendemos encontrar el vector de valor/costes asociado $J^\mu(i)$, (donde: $i \in \{1, 2, \dots, n\}$). Al espacio de estados puede añadirse un estado absorbente '0', que permite representar el problema como SPP (*Shortest Path Problem*).

La metodología que emplearemos será la mencionada previamente: generar por simulación trayectorias muestrales, y emplearlas para obtener estimaciones de valor de todos y cada uno de los estados que son visitados en la trayectoria. Simbolizaremos como $c(i, m)$ el coste acumulado tras la visita m -ésima al estado i .

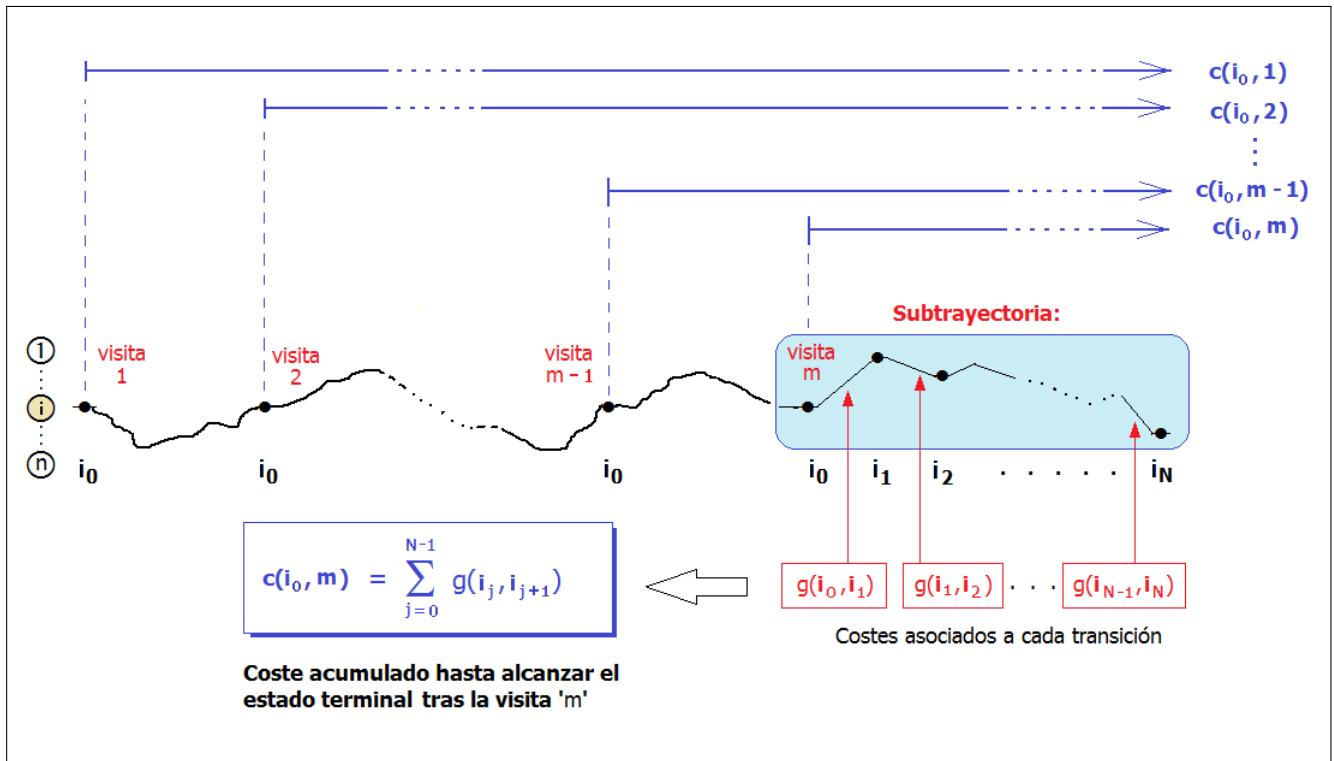


Figura 3.3: Empleo de costes intermedios para estimar el valor de estados visitados en secciones de la trayectoria muestral. En la figura el estado i_0 ha sido visitado m veces.

Si suponemos que trayectorias diferentes son independientes, y que cada una de ellas ha sido generada por el proceso de Markov asociado a μ . Tenemos que:

$$J^\mu(i) = \mathbb{E}[c(i, m)] \quad \forall i, \forall m$$

que será estimado mediante la media muestral. Suponiendo n visitas al estado i tenemos:

$$J(i) = \frac{1}{n} \sum_{m=1}^n c(i, m)$$

Las figuras 3.3 y 3.4 representan el proceso de simulación. Podemos intuir que se presenta un problema cuando se producen varias visitas al mismo estado en la misma trayectoria. Vemos que los costes acumulados son dependientes entre sí, ya que son fragmentos unos de otros.

La consecuencia que se deriba de ello es que el estimador media muestral ($J(i)$) pasa a ser sesgado, además de que aumenta la dispersión de las estimaciones que genera. Sin embargo cuando se producen infinitas visitas a los estados, este efecto pasa a ser despreciable y las estimaciones convergen a los valores correctos (Tsitsiklis, J. & Bertsekas, D. (1996) [27]).

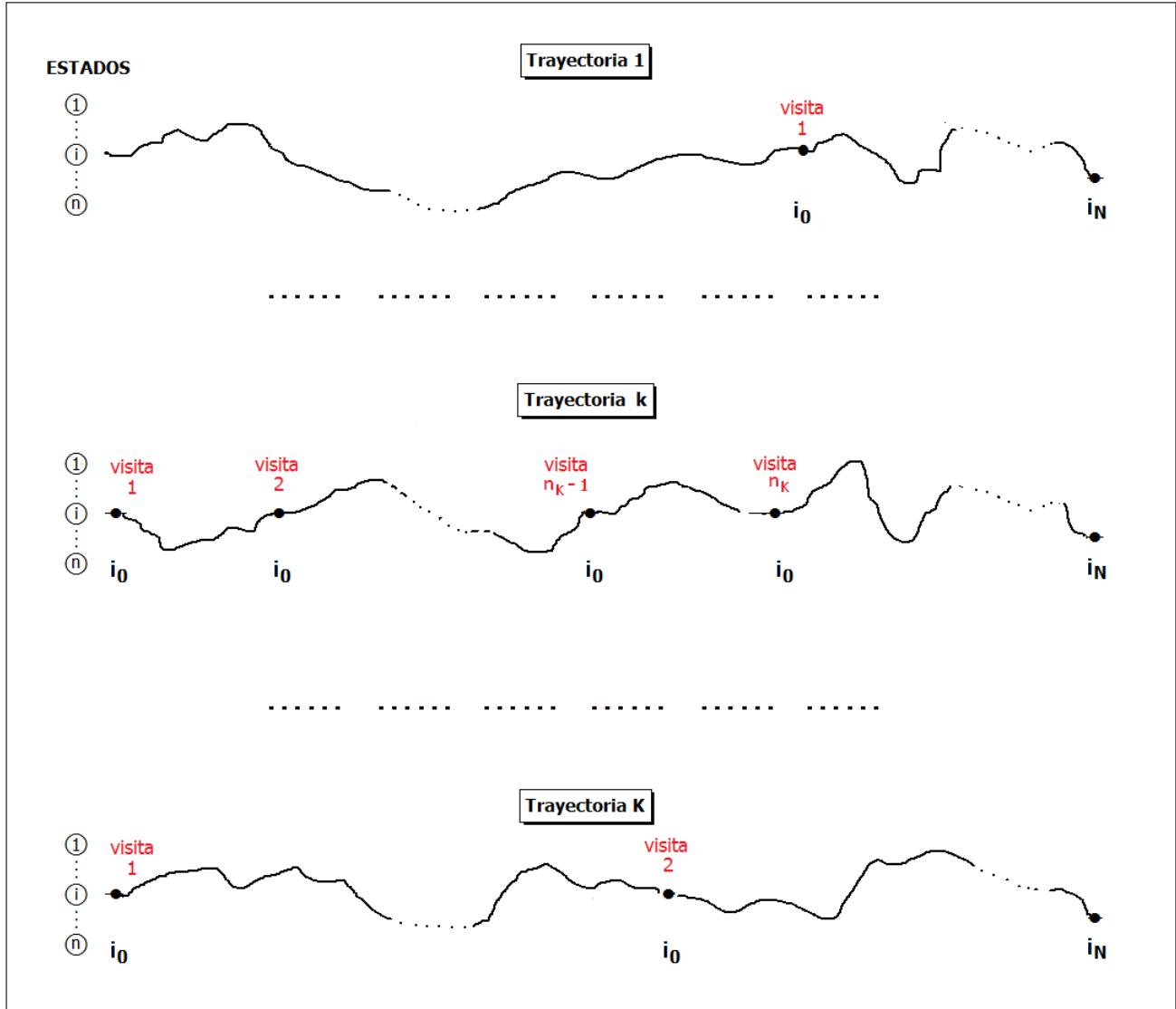


Figura 3.4: Simulando K trayectorias ($1, 2, \dots, k, \dots, K$) podemos obtener una estimación de la función de valor en los estados que son visitados en ellas (en la figura aparece un estado i_0).

Las medias muestrales que nos permiten obtener las estimaciones del valor asociado a cada estado, también pueden ser calculadas de manera iterativa. Para ello, dado un estado i , y comenzando con $J(i) = 0$ se emplea la siguiente expresión de actualización:

$$J(i) := J(i) + \gamma_m \cdot [c(i, m) - J(i)] \quad m = 1, 2, \dots$$

donde:

$$\gamma_m = \frac{1}{m} \quad m = 1, 2, \dots$$

En el caso que se ha comentado de varias visitas al mismo estado, podemos conseguir disminuir el sesgo que se genera en el estimador, incrementando el número de trayectorias que intervienen en el cálculo de la media muestra. La expresión que tiene en cuenta este hecho aparece a continuación.

- **Every visit Montecarlo** En este caso la trayectoria incluye varias pasadas por el estado i en cuestión, por lo que tendremos que tenerlo en cuenta a la hora de realizar la estimación de la función de valor óptimo.

$$\begin{aligned} \lim_{K \rightarrow \infty} \frac{\sum_{\{k|n_k \geq 1\}} \sum_{m=1}^{n_k} c(i, m, k)}{\sum_{\{k|n_k \geq 1\}} n_k} &= \lim_{k_i \rightarrow \infty} \frac{\frac{1}{K_i} \sum_{\{k|n_k \geq 1\}} \sum_{m=1}^{n_k} c(i, m, k)}{\frac{1}{K_i} \sum_{\{k|n_k \geq 1\}} n_k} \\ &= \frac{\mathbb{E} \left[\sum_{m=1}^{n_k} c(i, m, k) \mid n_k \geq 1 \right]}{\mathbb{E} [n_k \mid n_k \geq 1]} \end{aligned}$$

Como se tiene por el siguiente resultado que:

$$\begin{aligned} \frac{\mathbb{E} \left[\sum_{m=1}^{n_k} c(i, m, k) \mid n_k \geq 1 \right]}{\mathbb{E} [n_k \mid n_k \geq 1]} \\ = \mathbb{E} [c(i, 1, k) \mid n_k \geq 1] = J^\mu(i) \end{aligned}$$

se sigue que efectivamente la expresión converge al vector de costes asociados a la estrategia μ , que es el resultado que estábamos buscando.

$$\lim_{K \rightarrow \infty} \frac{\sum_{\{k|n_k \geq 1\}} \sum_{m=1}^{n_k} c(i, m, k)}{\sum_{\{k|n_k \geq 1\}} n_k} = J^\mu(i)$$

- **First visit Montecarlo** En este caso sólo se produce una pasada por el estado i , por lo que no aparece la dificultad mencionada previamente.

$$\frac{\sum_{\{k|n_k \geq 1\}} c(i, 1, k)}{K_i}$$

El método que acabamos de ver emplea la información recogida una vez a finalizado la trayectoria simulada. Sin embargo, tal como señalamos previamente, también se puede calcular incrementalmente sobre la marcha la estimación de la función de valor ($J(i)$). Es lo que se conoce como **diferencias temporales**, y que pasamos a analizar a continuación.

3.2.2. Método de las diferencias temporales (*TD*)

En el cálculo de la función de valor mediante **programación dinámica** establecíamos de manera exacta el valor que tiene cada estado ($J(i) \forall i \in \mathcal{S}$), empleando para ello algoritmos iterativos del tipo: TJ (donde T es un operador contractivo, y J la función de valor). La solución que obteníamos era el punto fijo del operador ($J^* \text{ donde } TJ^* = J^*$).

Cuando no conocemos las probabilidades de transición entre los estados en un entorno aleatorio, no es posible medir de manera precisa la expresión TJ que estamos manejando, por lo que se hace necesario acudir a otras vías. En este caso: los **algoritmos estocásticos iterativos** son de gran utilidad.

Para hacernos una idea de la naturaleza de la técnica, supongamos que deseamos obtener el valor esperado (μ) de una determinada característica aleatoria: $X \sim f(x)$ donde $\mathbb{E}[X] = \mu$. La vía más apropiada consiste en acudir a una muestra de observaciones independientes (que desde el punto de vista matemático son n variables aleatorias: X_1, X_2, \dots, X_n tal que $X_i \sim f(x)$), y a partir de las realizaciones concretas obtener la media muestral.

Si observamos el siguiente desarrollo:

$$\begin{aligned}\bar{X}_n &= \frac{1}{n} \cdot \sum_{i=1}^n X_i \\ &= \frac{1}{n} \cdot \left[X_n + \sum_{i=1}^{n-1} X_i \right] \\ &= \frac{1}{n} \cdot \left[X_n + (n-1) \cdot \frac{1}{(n-1)} \sum_{i=1}^{n-1} X_i \right] \\ &= \frac{1}{n} \cdot (X_n + (n-1) \cdot \bar{X}_{n-1}) \\ &= \frac{1}{n} \cdot (X_n + n \cdot \bar{X}_{n-1} - \bar{X}_{n-1}) \\ &= \bar{X}_{n-1} + \frac{1}{n} \cdot (X_n - \bar{X}_{n-1})\end{aligned}$$

es decir:

$$\underbrace{\bar{X}_n}_{\text{Estimación nueva}} = \underbrace{\bar{X}_{n-1}}_{\text{Estimación previa}} + \frac{1}{n} \cdot \underbrace{(X_n - \bar{X}_{n-1})}_{\text{Corrección}}$$

Tenemos que la media (\bar{X}_k) para un determinado tamaño muestral k se puede calcular iterativamente y de manera secuencial, ajustando la estimación previamente obtenida (\bar{X}_{k-1}) con la información que incorpora la última observación ($X_k - \bar{X}_{k-1}$).

En la fórmula superior, esta corrección se realiza con un factor que es proporcional al tamaño del conjunto de observaciones (llamémoslo α que en este caso: $\alpha = 1/n$).

La idea que se encuentra detrás del ejemplo anterior se puede aplicar al caso que estamos estudiando y permite, como veremos, estimar las funciones de valor.

Si recordamos: estamos manejando funciones de valor J que nos permitan establecer el valor que tiene cada estado ($J(i) \forall i \in \mathcal{X}$), empleando para ello algoritmos iterativo del tipo: TJ (donde J es un operador contractivo). La solución es el punto fijo del operador ($TJ^* = J^*$). Cuando no es posible medir precisamente la forma funcional que estamos manejando, se hace necesario acudir a otro tipo de herramientas. En este caso: los algoritmos estocásticos iterativos son de gran utilidad.

Algoritmos Estocásticos Iterativos

Buscamos resolver un sistema de ecuaciones del tipo: $Hr = r$, donde $H : \mathbb{R}^n \rightarrow \mathbb{R}^n$. El vector r^* que satisface dicha expresión ($Hr^* = r^*$) recibe el nombre de punto fijo de F .

Cuando surge el problema de que las actualizaciones a cada paso incorporan ruido, se ha de acceder a métodos **estocásticos iterativos** (Robbins, H. & Monro, S. (1951)) [150]. La regla de actualización consiste en:

$$r_{t+1} = (1 - \gamma_t) \cdot r_t + \gamma_t \cdot [Hr_t + \omega_t]$$

donde cada uno de los elementos tomaría la forma:

$$r_{t+1}(i) = (1 - \gamma_t(i)) \cdot r_t(i) + \gamma_t(i) \cdot [(Hr_t)(i) + \omega_t(i)]$$

Cuando la Función H es **contractiva** con respecto a la norma máxima, o satisface el supuesto de monotonidad se dan las condiciones para que pueda aparecer la convergencia.

DEFINICIÓN.

Dado cualquier vector positivo ξ , definimos la **norma máxima ponderada** $\|\cdot\|_\xi$ como:

$$\|r\|_\xi = \max_i \frac{|r(i)|}{\xi(i)}$$

DEFINICIÓN.

Cuando todos los componentes de ξ son iguales a 1, la norma resultante se denomina **norma máxima** y se denota por $\|\cdot\|_\infty$.

$$\|r\|_\infty = \max_i |r(i)|$$

DEFINICIÓN.

Dada una función $H : \mathbb{R}^n \rightarrow \mathbb{R}^n$, se dice que es una función **pseudocontractiva** respecto a una norma máxima ponderada si existe algún $r^* \in \mathbb{R}^n$, un vector positivo $\xi = (\xi_1, \xi_2, \dots, \xi_n) \in \mathbb{R}^n$, y una constante $\beta \in [0, 1)$ de tal manera que:

$$\|Hr - r^*\|_\xi \leq \beta \cdot \|r - r^*\|_\xi \quad \forall r$$

expresado:

$$\frac{|(Hr)(i) - r^*(i)|}{\xi(i)} \leq \beta \cdot \max_j \frac{|r(j) - r^*(j)|}{\xi(j)} \quad \forall i, r$$

Algoritmos Estocásticos Iterativos - *Demostración*

El cumplimiento de la **pseudocontractividad** implica que r^* es un punto fijo de H , es decir: $Hr^* = r^*$. Además no puede haber otros puntos fijos, ya que si existiera cualquier otro vector r tal que $Hr = r$, tendríamos que: $\|r - r^*\|_\xi = \|Hr - r^*\|_\xi \leq \beta \cdot \|r - r^*\|_\xi$, y como $\beta < 1$ tenemos que forzosamente $\|r - r^*\|_\xi = 0$ y por tanto $r = r^*$.

PROPOSICIÓN.

Sea r_t , una secuencia de valores generada por la regla de actualización que hemos enunciado, entonces: r_t converge a r^* con probabilidad 1 si se cumplen los siguientes supuestos:

- a) Los *stepsizes* $\gamma_t(i)$ son no negativos y cumplen $\forall i :$

$$\sum_{t=0}^{\infty} \gamma_t(i) = \infty , \quad \sum_{t=0}^{\infty} \gamma_t^2(i) < \infty$$

- b) Sea \mathcal{F}_t la historia del algoritmo hasta el instante t , entonces los términos de ruido $\omega_t(i)$ verifican: $\mathbb{E}[\omega_t(i) | \mathcal{F}_t] = 0$ y dada una norma $\|\cdot\|$ sobre \mathbb{R}^n existen A y B tal que:

$$\mathbb{E}[\omega_t^2(i) | \mathcal{F}_t] \leq A + B \cdot \|r_t\| \quad \forall i, t.$$

- c) La función H es una función semi-contractiva ponderada con respecto a la norma máxima

Demostración:

Intuitivamente veamos que, si partimos de $r^ = 0$ y H es pseudo-contractiva con respecto a la norma máxima $\|\cdot\|_\infty$, tenemos que:*

$$|Hr(i)| \leq \beta \cdot \max_i |r(i)| \quad \forall i, r.$$

Además sin pérdida de generalidad, suponemos que $w_t(i) = 0 \quad \forall i, t$, y los stepsizes $\gamma_t(i)$ son la unidad (cuando el valor $r(i)$ es actualizado) o cero (en caso contrario).

Comenzamos la secuencia de valores con un vector r_0 que: $|r_0(i)| \leq c$, para una cierta constante c . Tenemos entonces que r_0 queda encajado dentro de un cubo de lado $2c$ sobre cada dimensión, centrado en el origen (ver fig. 3.5).

Supongamos que r_t reside dentro del cubo, y sus valores son actualizados según $r_{t+1}(i) = (Hr_t)(i)$. Debido a la pseudocontractividad de H tenemos que: $|r_{t+1}(i)| \leq \beta c < c$. Es decir partiendo del cubo n -dimensional inicial, todas las actualizaciones se mantienen dentro de él.

Suponiendo que cada componente es actualizada infinitas veces, los cubos son progresivamente más pequeños y convergen a 0.

QED ----- ■

Algoritmos Estocásticos Iterativos - *Demostración*

En el caso de que la función H sea una función **contractiva**; sabemos que ha de verificar:

$$\| Hr - H\bar{r} \|_{\xi} \leq \beta \cdot \| r - \bar{r} \|_{\xi} \quad \forall r, \bar{r}.$$

lo que garantizada la existencia del punto fijo r^* , y convierte automáticamente a H en **pseudo-contractiva**.

La demostración intuitiva que hemos visto nos permite abordar la cuestión de la convergencia de los algoritmos basados en la iteración de valor, que son expresados como un algoritmo estocástico iterativo. (Para ver la demostración en detalle acudir al APÉNDICE).

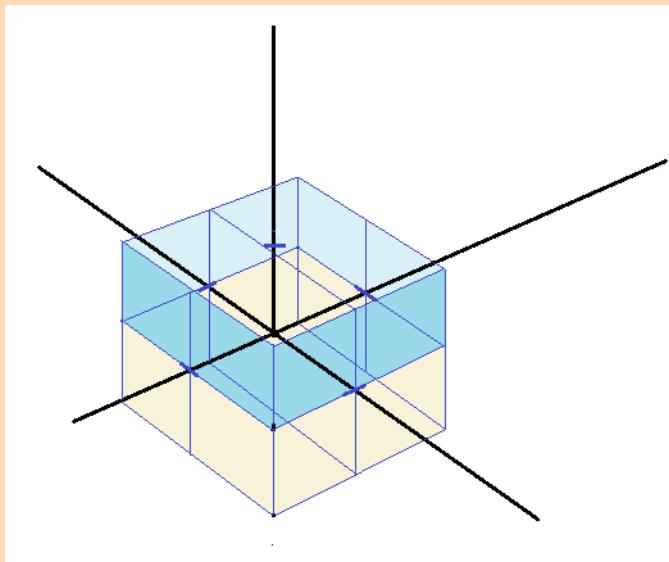


Figura 3.5: *Cubo*

Utilizaremos estos resultados para el análisis de la convergencia del algoritmo Q-Learning en las secciones que siguen.

Los algoritmos de Aprendizaje por Refuerzo que se empezaron a desarrollar fueron identificados por algunos autores como una generalización de estos Algoritmos Estocásticos Iterativos que acabamos de repasar (Tsitsiklis, J. (1994) [191]). Para un tratamiento unificado de la optimización estocástica ver: Powell, B. (2019) [144] y posteriormente Powell, B. (2022) [145].

Este resultado es de utilidad para establecer la convergencia de métodos de aproximación estocástica basados en algoritmos de iteración de valor (*value iteration*) del tipo $J := TJ$, especialmente en el caso en el que aparece un factor de descuento (α) o en los problemas SPP (*Shortest Path Problem*) donde todas las estrategias son propias (Bertsekas, D. & Tsitsiklis, J. (1996) [27]).

Método de las Diferencias Temporales (Temporal Difference - TD)

Método computacional adecuado cuando no disponemos de un modelo explícito del sistema ni de la estructura de costes/beneficios. La regla de actualización consiste en:

$$J(i_k) := J(i_k) + \gamma \cdot d_l \quad k = 0, \dots, l$$

donde:

$$d_l = g(i_l, i_{l+1}) + J(i_{l+1}) - J(i_l) \quad l = k, \dots, N-1$$

Una vez que disponemos de una trayectoria completa (i_0, i_1, \dots, i_N) , los valores estimados de cada estado $J(i_k)$, $k=0, \dots, N-1$ se actualizan sin más que aplicar $\forall i_k \in X$ la regla:

$$J(i_k) := J(i_k) + \gamma [g(i_k, i_{k+1}) + g(i_{k+1}, i_{k+2}) + \dots + g(i_{N-1}, i_N) - J(i_k)] \quad (3.5)$$

Dicha regla de actualización puede ser reescrita introduciendo términos que se cancelan entre sí, de la siguiente forma (y aprovechando la convención de que $J(i_N) = 0$):

$$\begin{aligned} J(i_k) &:= J(i_k) + \gamma [[g(i_k, i_{k+1}) + \cancel{J(i_{k+1})} - J(i_k)] \\ &\quad + [g(i_{k+1}, i_{k+2}) + \cancel{J(i_{k+2})} - \cancel{J(i_{k+1})}] \\ &\quad + \dots \dots \dots \dots \dots \dots \dots \dots \dots \\ &\quad + [\underbrace{g(i_{N-1}, i_N) + J(i_N) - \cancel{J(i_{N-1})}}_{\text{diferencia temporal } d_{N-1}}]] \end{aligned}$$

y en teniendo en cuenta la expresión siguiente, que llamaremos **diferencia temporal**:

$$d_k := \underbrace{g(i_k, i_{k+1}) + J(i_{k+1})}_{\text{estimación del valor faltante}} - \underbrace{J(i_k)}_{\text{estimación actual}}$$

admite la expresión equivalente:

$$J(i_k) := J(i_k) + \gamma [d_k + d_{k+1} + \dots + d_{N-1}]$$

De esta manera podemos actualizar los valores incrementalmente:

$$J(i_k) := J(i_k) + \gamma \cdot d_l, \quad l=k, \dots, N-1$$

PROPOSICIÓN 3.3: Convergencia del algoritmo TD

La secuencia de estimaciones generada por el algoritmo de las diferencias temporales (Temporal Difference - TD) converge con probabilidad 1 al valor óptimo.

Demostración:

Ver: Bertsekas, D. & Tsitsiklis, J. (1996) [27].

QED ----- ■

La ecuación anterior 3.5 es el resultado de aplicar el método de aproximación estocástica de Robbins-Monroe (Robbins, H. & Monroe, S. (1951) [150]) para resolver una ecuación del tipo:

$$J_\mu(i_k) = \mathbb{E} \left[\sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1}) \right], \quad i_k \in X$$

Si en lugar de utilizar todos los costes/beneficios acumulados de la trayectoria completa, como en el método de Monte Carlo, utilizamos la ecuación de Bellman a un paso:

$$J_\mu(i_k) = \mathbb{E}[g(i_k, i_{k+1}) + J_\mu(i_{k+1})]$$

el método de aproximación estocástica conduce a la regla de actualización:

$$J(i_k) := J(i_k) + \gamma \cdot [g(i_k, i_{k+1}) + J(i_{k+1}) - J(i_k)]$$

Este procedimiento se puede generalizar para cualquier número de pasos l , estando en este caso interesados en resolver unas ecuaciones del tipo:

$$J_\mu(i_k) = \mathbb{E} \left[\sum_{m=0}^l g(i_{k+m}, i_{k+m+1}) + J_\mu(i_{k+l+1}) \right]$$

Como la elección de este valor es arbitrario, se ha desarrollado un método que emplea un promedio ponderado de las ecuaciones de Bellman, utilizando un decaimiento geométrico en los valores de los pesos (ver fig. 3.6). Es el conocido como método TD(λ), que pasamos a desarrollar a continuación.

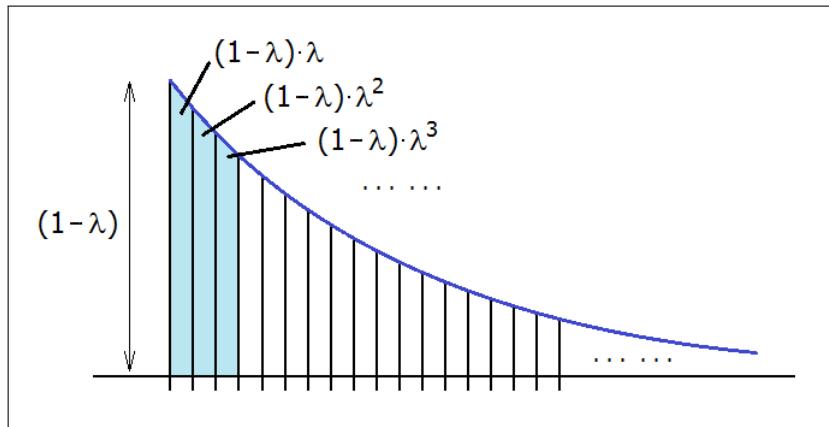


Figura 3.6: Empleamos un decaimiento geométrico para los pesos (Adaptado de Barto, R. S. & Sutton, A. G. (2018) [181] - pag. 290).

- **TD(λ)**

A continuación expresamos el método de las Diferencias Temporales parametrizado por el parámetro λ . Partiendo de la expresión cuyo valor queremos obtener, que sería la siguiente:

$$J_\mu(i_k) = (1 - \lambda) \cdot \mathbb{E} \left[\sum_{l=0}^{\infty} \lambda^l \cdot \left(\sum_{m=0}^l g(i_{k+m}, i_{k+m+1}) - J_\mu(i_{k+l+1}) \right) \right] \quad (3.6)$$

Operando sobre los sumatorios podemos obtener una expresión mucho más conveniente que permite simplificar la forma de la ecuación.

En concreto, si desarrollamos:

$$\sum_{l=0}^{\infty} \lambda^l \cdot \left(\sum_{m=0}^l g(i_{k+m}, i_{k+m+1}) - J_\mu(i_{k+l+1}) \right) \quad (3.7)$$

tenemos:

l=0

$$\lambda^0 \cdot [g(i_k, i_{k+1}) - J_\mu(i_{k+1})] +$$

l=1

$$\lambda^1 \cdot [g(i_k, i_{k+1}) + g(i_{k+1}, i_{k+2}) - J_\mu(i_{k+2})] +$$

l=2

$$\lambda^2 \cdot [g(i_k, i_{k+1}) + g(i_{k+1}, i_{k+2}) + g(i_{k+2}, i_{k+3}) - J_\mu(i_{k+3})] +$$

...

Vemos que el primer coste/beneficio $g(i_k, i_{k+1})$ (*en color rojo*) aparece multiplicado por todas las potencias de λ .

m=0

$$g(i_k, i_{k+1}) \cdot \sum_{l=0}^{\infty} \lambda^l = g(i_k, i_{k+1}) \cdot \frac{1}{1-\lambda}$$

El segundo coste/beneficio de la trayectoria: $g(i_{k+1}, i_{k+2})$ (*en color azul*) aparece multiplicado por las potencias de λ a partir de λ^1 .

m=1

$$g(i_{k+1}, i_{k+2}) \cdot \sum_{l=1}^{\infty} \lambda^l = g(i_{k+1}, i_{k+2}) \cdot \lambda \cdot \frac{1}{1-\lambda}$$

Con el siguiente valor de la trayectoria: $g(i_{k+2}, i_{k+3})$ ocurre igual pero a partir de λ^2 .

m=2

$$g(i_{k+2}, i_{k+3}) \cdot \sum_{l=2}^{\infty} \lambda^l = g(i_{k+2}, i_{k+3}) \cdot \lambda^2 \cdot \frac{1}{1-\lambda}$$

Por tanto podemos sumar primero en el índice m , multiplicando cada término por la secuencia de productos en λ que le corresponde; intercambiando el orden en el doble sumatorio. Esto nos permite convertir la expresión 3.7 en la 3.8:

$$\sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1}) \sum_{l=m}^{\infty} \lambda^l - \sum_{l=0}^{\infty} \lambda^l J_\mu(i_{k+l+1}) \quad (3.8)$$

Si tenemos en cuenta que: $(1 - \lambda) \cdot \sum_{l=m}^{\infty} \lambda^l = \lambda^m$ e incorporamos los cambios anteriores en [3.6](#), entonces tenemos que la expresión adopta la siguiente forma:

$$\begin{aligned}
 J_\mu(i_k) &= (1 - \lambda) \cdot \mathbb{E} \left[\sum_{l=0}^{\infty} \lambda^l \cdot \left(\sum_{m=0}^l g(i_{k+m}, i_{k+m+1}) - J_\mu(i_{k+l+1}) \right) \right] \\
 &= (1 - \lambda) \cdot \mathbb{E} \left[\sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1}) \sum_{l=m}^{\infty} \lambda^l - \sum_{l=0}^{\infty} \lambda^l J_\mu(i_{k+l+1}) \right] \\
 &= \mathbb{E} \left[(1 - \lambda) \cdot \sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1}) \sum_{l=m}^{\infty} \lambda^l + \sum_{l=0}^{\infty} J_\mu(i_{k+l+1}) \cdot (\lambda^l - \lambda^{l+1}) \right] \\
 &= \mathbb{E} \left[\sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1}) \cdot \underbrace{(1 - \lambda) \cdot \sum_{l=m}^{\infty} \lambda^l}_{\lambda^m} + \sum_{l=0}^{\infty} J_\mu(i_{k+l+1}) \cdot (\lambda^l - \lambda^{l+1}) \right] \\
 &= \mathbb{E} \left[\sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1}) \cdot \lambda^m + \sum_{l=0}^{\infty} J_\mu(i_{k+l+1}) \cdot \lambda^l - \sum_{l=0}^{\infty} J_\mu(i_{k+l+1}) \cdot \lambda^{l+1} \right]
 \end{aligned}$$

Como los índices de los sumatorios son mudos, cambiamos a un índice común: l por m , e introducimos restado y sumado $J_\mu(i_k)$ para completar el término en λ^0 que falta en el tercer sumatorio. Esto nos permite extraer el factor común $\sum_{m=0}^{\infty} \lambda^m$:

$$= \mathbb{E} \left[\sum_{m=0}^{\infty} \lambda^m \cdot \left(g(i_{k+m}, i_{k+m+1}) + J_\mu(i_{k+m+1}) - J_\mu(i_{k+m}) \right) \right] + J_\mu(i_k)$$

Sobre esta expresión, el método basado en la aproximación estocástica de Robbins-Monro conduce a la siguiente regla de actualización $\forall i_k \in X$:

$$J(i_k) := J(i_k) + \gamma \cdot \sum_{m=k}^{\infty} \lambda^{m-k} \cdot d_m$$

■ TD Método general

Partiendo de una trayectoria i_0, i_1, \dots y las correspondientes diferencias temporales (d_m), pasamos a exponer un método general basado en ellas. En la expresión que sigue aparecen los llamados **coeficientes de elegibilidad** (*eligibility coefficients*): $z_m(i)$.

$$J(i_k) := J(i_k) + \gamma \cdot \sum_{m=k}^{\infty} z_m(i) \cdot d_m$$

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

3.2.3. Algoritmo Q-Learning

Una forma alternativa de plantear la toma de decisiones secuenciales consiste en asociar, no un valor óptimo a cada estado, sino explícitamente a la combinación del estado en consideración junto a cada una de las acciones que pueden llevarse a cabo desde él. Entre las alternativas que han planteado diversos autores encontramos los **valores Q**, o las **ventajas**.

Factor Q

Factor Q (*Q-factor* - o también valor estado-acción (Bertsekas, D. (2019) [26] y Bertsekas, D. (2020) [23])). En un MDP (Markov Decision Process) con espacio de estados finito vendría a representar al valor (coste/ganancia) asociado a un estado i cuando se toma una decisión u y a partir de ese momento se continúa actuando de manera óptima. Tendríamos:

$$Q(i, u) = \sum_{j=1}^n p_{ij}(u) \cdot [g(i, u, j) + J(j)], \quad i = 1, 2, \dots, n, \quad u \in U(i)$$

y que en un problema con descuento el Q-factor óptimo se representaría tal y como aparece en la siguiente expresión (Bertsekas, D. (2019) [26] - pag. 253):

$$Q^*(i, u) = \sum_{j=1}^n p_{ij}(u) \cdot [g(i, u, j) + \alpha \cdot J^*(j)], \quad i = 1, 2, \dots, n, \quad u \in U(i)$$

Utilización con Métodos basados en modelo (Programación Dinámica). Utilizando la notación más general, el Q-factor del par estado-acción (x_k, u_k) en el instante k , notado: $Q_k(x_k, u_k)$ vendría dado por la siguiente expresión:

$$Q_k(x_k, u_k) = \mathbb{E} [g_k(x_k, u_k, w_k) + J_{k+1}(x_k, u_k, w_k)] \quad k = 0, 1, 2, \dots$$

donde J_{k+1} es la función de coste óptimo futuro para la etapa $k+1$. Y $Q_k(x_k, u_k)$ representa el coste obtenido al llevar a cabo la acción u_k en el estado x_k , y proceder a continuación a seguir una estrategia óptima (Bertsekas, D. (2020) pag. 344 Vol. I [24]).

Empleando los valores $Q_k(x_k, u_k)$ el algoritmo de Programación Dinámica se puede expresar de manera equivalente como:

$$J_k(x_k) = \min_{u \in U_k(x_k)} Q_k(x_k, u_k)$$

Se puede considerar una alternativa a la *Iteración de Valor (VI)* (del tipo: $J_{k+1} = T J_k$) si conocemos las probabilidades de transición, en cuyo caso (ver Bertsekas, D. (2012) [25] - pag. 96) en el caso con descuento y número de estados finitos tendríamos:

$$Q_{k+1}(i, u) = \sum_{j=1}^n p_{ij}(u) \cdot [g_{ij}(u) + \alpha \cdot \underbrace{\min_{v \in U(j)} Q_k(j, v)}_{J_k(j)}], \quad i = 1, 2, \dots, n, \quad u \in U(i)$$

(*Observar que en este caso el subíndice alude a la iteración y no a la etapa como en el algoritmo PD, y los índices por tanto van hacia delante y no hacia atrás*)

En la literatura tiene especial importancia el primero de los enfoques. Se puede emplear como alternativa a la función de valor J que hemos visto en el apartado de soluciones mediante Programación Dinámica, pero su verdadera importancia surge cuando no disponemos de un conocimiento del sistema y nos hemos de limitar a observarlo o realizar simulaciones con él, que es el enfoque que trataremos en este apartado.

El principal algoritmo para la estimación de los valores Q es el algoritmo Q-Learning, que fue desarrollado por Watkins, C. Dayan, P. (1992) [198].

Ventaja (*Advantage*)

Se denomina **Ventaja** (*Advantage*) (Baird, L.C. (1993) [5] y Harmon, M.E. et al. (1994) [68]) de la acción u en el estado x , notado: $A_k(x, u)$ a la siguiente expresión:

$$A_k(x, u) = Q_k(x, u) - \min_{u \in U_k(x_k)} Q_k(x, u)$$

Este valor permite comparar las distintas acciones, con el añadido de que tiene un rango de valores mucho más reducido (Bertsekas, D. (2020) pag. 346 Vol. I [24]). Representa la mejora obtenida mediante la elección de una determinada acción en un cierto estado respecto del promedio obtenido cuando se tienen en cuenta todas las acciones posibles.

Podemos deducir la expresión empleando la Ecuación de Bellman. Esto nos permite obtener una expresión para actualizar los factores $Q^*(i, u)$. Partiendo de la expresión de los costes óptimos:

$$J^*(i) = \min_{u \in U(i)} \underbrace{\left[\sum_{j=1}^n p_{ij}(u) \cdot [g_{ij}(u) + \alpha \cdot J^*(j)] \right]}_{Q^*(i, u)}, \quad i = 1, 2, \dots, n$$

y teniendo en cuenta que :

$$J^*(j) = \min_{v \in U(j)} Q^*(j, v)$$

resulta:

$$Q^*(i, u) = \sum_{j=1}^n p_{ij}(u) \cdot \left[g_{ij}(u) + \alpha \cdot \min_{v \in U(j)} Q^*(j, v) \right], \quad i = 1, 2, \dots, n \quad (3.9)$$

Entonces el algoritmo **value iteration** puede ser expresado en término de los Q-factores.

$$Q(i, u) := \sum_{j=1}^n p_{ij}(u) \cdot \left[g_{ij}(u) + \alpha \cdot \min_{v \in U(j)} Q(j, v) \right], \quad \forall(i, u)$$

o en la forma más general, introduciendo un paso de aprendizaje $\gamma \in (0, 1]$:

$$Q(i, u) := (1 - \gamma) \cdot Q(i, u) + \gamma \cdot \sum_{j=1}^n p_{ij}(u) \cdot \left[g_{ij}(u) + \alpha \cdot \min_{v \in U(j)} Q(j, v) \right]$$

En esta expresión necesitamos conocer los valores de las probabilidades de transición $p_{ij}(u)$. El algoritmo **Q-Learning** es una aproximación de este algoritmo donde en lugar de calcular el valor esperado, empleamos un único valor muestral:

$$Q(i, u) := (1 - \gamma) \cdot Q(i, u) + \gamma \cdot \left[g_{ij}(u) + \alpha \cdot \min_{v \in U(j)} Q(j, v) \right]$$

El método **Q-Learning** vendría a ser la *aproximación estocástica de Robbins-Monro* de la ecuación 3.9. El algoritmo genera una secuencia de valores $Q_t(i, u)$ que convergen a los verdaderos valores $Q^*(i, u)$, siempre y cuando cada par estado acción (i, u) se visite con frecuencia infinita y la tasa de aprendizaje (γ) disminuya convenientemente.

Vemos que ahora no necesitamos conocer explícitamente los detalles de cómo opera el sistema tales como disponer de los valores de las probabilidades de transición. Experimentalmente accedemos a los nuevos valores observados j y $g_{ij}(u)$, tomando como partida el valor actual (i, u) .

Q-Learning

Método de cálculo del valor óptimo de cada par estado-acción $((i, u))$ y que es adecuado cuando no disponemos de un modelo explícito del sistema ni de la estructura de costes/-beneficios.

Q-Learning es análogo al algoritmo VI (*Value Iteration*), y actualiza directamente estimaciones de Q-Factores asociados con la estrategia óptima. (Bertsekas, D. & Tsitsiklis, J. (1996) [27] - pag. 245). Se suelen emplear las dos expresiones siguientes equivalentes:

$$Q(i, u) := Q(i, u) + \gamma \left[g(i, u, j) + \min_{v \in U(j)} Q(j, v) - Q(i, u) \right]$$

$$Q(i, u) := (1 - \gamma)Q(i, u) + \gamma \left[g(i, u, j) + \min_{v \in U(j)} Q(j, v) \right]$$

que en el caso con factor de descuento $\alpha \in [0, 1)$ quedaría esta última:

$$Q(i, u) := (1 - \gamma)Q(i, u) + \gamma \left[g(i, u, j) + \alpha \cdot \min_{v \in U(j)} Q(j, v) \right]$$

Representación tabular. Computacionalmente se requiere guardar explícitamente cada uno de los valores de la función (para todo par (i, u)) en una matriz, por lo que puede ser inadecuado cuando se dispone de conjuntos de estados o de acciones muy grandes. En este caso se suelen emplear métodos de aproximación.

Representación compacta. Empleamos aproximadores funcionales, como una Red Neuronal, para obtener valores aproximados $\tilde{Q}(i, u)$ (Ver por ejemplo: Bertsekas, D. & Tsitsiklis, J. (1996) [27] - pag. 337).

EJEMPLO 3.3: Algoritmo Q-Learning

Sea un determinado sistema representado tal y como aparece en la figura 3.7. Tenemos dos estados $X = \{1, 2\}$ y un conjunto de acciones $U(i) = \{u_1, u_2\}$, $i = 1, 2$.

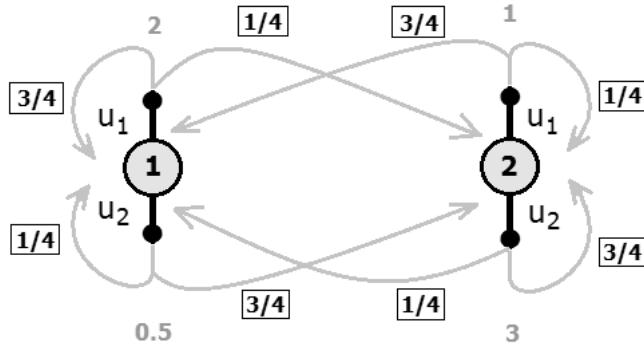


Figura 3.7: *Dinámica del sistema*

Si definimos el Q-factor óptimo correspondiente al par (i, u) , notado: $Q^*(i, u)$, como aparece en la siguiente expresión:

$$Q^*(i, u) = \sum_{j=0}^n p_{ij}(u) \cdot [g(i, u, j) + J^*(j)], \quad i = 1, 2, \dots, n \quad (3.10)$$

Podemos obtener una expresión alternativa para la ecuación de Bellman:

$$J^*(i) = \min_{u \in U(i)} Q^*(i, u), \quad i = 1, 2, \dots, n \quad (3.11)$$

Empleando esta nueva expresión de la ecuación de Bellman, podemos introducir la expresión 3.11 (de los costes óptimos $J^*(i)$) en la ecuación 3.10 que define $Q^*(i, u)$, obteniéndose la siguiente expresión..

$$Q^*(i, u) = \sum_{j=0}^n p_{ij}(u) \cdot [g(i, u, j) + \min_{v \in U(j)} Q^*(j, v)], \quad i = 1, 2, \dots, n$$

Expresión iterativa: Método computacional adecuado cuando no disponemos de un modelo explícito del sistema ni de la estructura de costes/beneficios. Es análogo al VI, y actualiza directamente estimaciones de Q-Factores asociados con la estrategia óptima. (pag. 245 Neuro Dynamic).

$$Q(i, u) := (1 - \gamma)Q(i, u) + \gamma \left[g(i, u, j) + \min_{v \in U(j)} Q(j, v) \right]$$

Convergencia: La secuencia de estimaciones generada por el algoritmo Q-Learning está acotada con probabilidad 1 y converge por tanto a Q^* . Pag. 249 NeuroDynamic.

Ahora abordamos la cuestión relativa a la convergencia del algoritmo.

PROPOSICIÓN 3.4: Convergencia del algoritmo Q-Learning

La aplicación del algoritmo *Q-Learning* en base a la utilización reiterada de la siguiente regla de actualización (pag. 249 NeuroDynamic):

$$Q_{t+1}(i, u) = (1 - \gamma_t) \cdot Q_t(i, u) + \gamma_t \cdot \left[g(i, u, j) + \min_{v \in U(j)} Q_t(j, v) \right] \quad (3.12)$$

donde las tasas de aprendizaje (*stepsizes*) γ_t pueden depender de (i, u) ($\gamma_t(i, u)$), cumplen que son no negativas y $\forall (i, u)$:

$$\sum_{t=0}^{\infty} \gamma_t(i, u) = \infty, \quad \sum_{t=0}^{\infty} \gamma_t^2(i, u) < \infty$$

genera una secuencia de estimaciones que converge a los valores correctos $Q^*(i, u)$ con probabilidad 1 para todo par (i, u) .

Demostración:

Planteamos el algoritmo Q-Learning con la estructura general de un algoritmo de aproximación estocástica del tipo:

$$r_{t+1} = (1 - \gamma_t) \cdot r_t + \gamma_t \cdot [Hr_t + w_t]$$

donde cada uno de los estados $i = 1, 2, \dots, n$ se actualiza según:

$$r_{t+1}(i) = (1 - \gamma_t(i)) \cdot r_t(i) + \gamma_t(i) \cdot [Hr_t(i) + r_t(i)]$$

Si la función H es una contracción con respecto a una norma máxima ponderada y cumple ciertas propiedades como la monotonidad, estaremos en condiciones de asegurar la convergencia al punto fijo de la función de interés.

En nuestro caso partimos de la definición de los Q-factores y de la aplicación del operador de Bellman (T) de programación dinámica. Basta tomar $H = T$, donde:

$$(HQ)(i, u) = \sum_{j=1}^n p_{ij}(u) \cdot \left[g_{ij}(u) + \alpha \cdot \min_{v \in U(j)} Q^*(j, v) \right]$$

Ahora introducimos $(HQ)(i, u)$ autocancelándose en la expresión 3.12 y agrupamos términos tal y como aparece a continuación:

$$\begin{aligned} Q_{t+1}(i, u) &= (1 - \gamma_t(i, u)) \cdot Q_t(i, u) \\ &+ \gamma_t(i, u) \cdot \left[\underbrace{(HQ_t)(i, u) + g_{ij}(u) + \min_{v \in U(j)} Q^*(j, v) - (HQ_t)(i, u)}_{w_t(i, u)} \right] \end{aligned}$$

Esto nos permite obtener la expresión del algoritmo Q-Learning que buscábamos, como un algoritmo estocástico iterativo:

$$Q_{t+1}(i, u) = (1 - \gamma_t(i, u)) \cdot Q_t(i, u) + \gamma_t(i, u) \cdot [(HQ_t)(i, u) + w_t(i, u)]$$

Bastará ahora comprobar que se cumplen las condiciones para la aplicación de los resultados de convergencia:

Perturbación aleatoria: Sea $\mathcal{F}_t = \{r_0, \dots, r_t, w_0, \dots, w_{t-1}, \gamma_0, \dots, \gamma_t\}$ la historia del algoritmo hasta el instante t . Teniendo en cuenta que: $\text{Var}(\omega_t(i) | \mathcal{F}_t) = \mathbb{E}[\omega_t(i) | \mathcal{F}_t] + \mathbb{E}[\omega_t^2(i) | \mathcal{F}_t]$.

Entonces, dada una norma $\|\cdot\|$ sobre \mathbb{R}^n existen $A, B \in \mathbb{R}$, de tal manera que los términos de perturbación aleatoria $\omega_t(i)$ verifican:

$$\left\{ \begin{array}{l} 1) \text{ Esperanza nula de } \omega_t \\ \mathbb{E}[\omega_t(i) | \mathcal{F}_t] = 0 \\ \\ 2) \text{ Varianza acotada} \\ \mathbb{E}[\omega_t^2(i) | \mathcal{F}_t] \leq A + B \cdot \|r_t\| \quad \forall i, t. \end{array} \right.$$

vemos que se cumplen las condiciones relativas a la esperanza y varianza condicional de los términos $\omega_t(i)$ (para más detalles ver: Tsitsiklis, J. (1994) [191]).

Función H: Bastará con demostrar que H es una función semicontractiva respecto de una norma máxima ponderada (o una función contractiva respecto de una norma máxima), para estar en condiciones de aplicar los resultados de convergencia de los algoritmos estocásticos iterativos.

Si partimos del supuesto de que todas las estrategias son propias, entonces existen coeficientes positivos $\xi(i), i \neq 0$ y un escalar $\beta \in [0, 1)$ tales que (dem. pag. 22 Neuro-Dynamic):

$$\begin{aligned} & \sum_{j=1}^n p_{ij}(u) \cdot \xi(j) \leq \beta \cdot \xi(i) \quad \forall i, u \in U(i) \\ & \|Q\|_\xi = \min_{i, u \in U(i)} \frac{|Q(i, u)|}{\xi(i)} \\ & |(HQ)(i, u) - (H\bar{Q})(i, u)| \leq \sum_{j=1}^n p_{ij}(u) \cdot |\min_{v \in U(j)} Q(j, v) - \min_{v \in U(j)} \bar{Q}(j, v)| \\ & \leq \sum_{j=1}^n p_{ij}(u) \cdot \max_{v \in U(j)} |Q(j, v) - \bar{Q}(j, v)| \\ & \leq \sum_{j=1}^n p_{ij}(u) \cdot \|Q - \bar{Q}\|_\xi \cdot \xi(j) \\ & \leq \beta \cdot \|Q - \bar{Q}\|_\xi \cdot \xi(i) \end{aligned}$$

de donde se tiene que:

$$\frac{|(HQ)(i, u) - (H\bar{Q})(i, u)|}{\xi(i)} \leq \beta \cdot \|Q - \bar{Q}\|_{\xi}$$

tomando el mínimo:

$$\min_{i, u \in U(i)} \frac{|(HQ)(i, u) - (H\bar{Q})(i, u)|}{\xi(i)} \leq \beta \cdot \|Q - \bar{Q}\|_{\xi}$$

tenemos:

$$\|(HQ) - (H\bar{Q})\|_{\xi} \leq \beta \cdot \|Q - \bar{Q}\|_{\xi}$$

por lo que H es una función contractiva respecto de la norma máxima ponderada, y de ellos se desprende la convergencia al punto fijo de la función.

QED ----- ■

La demostración anterior se debe a Tsitsiklis, J. (1994) [191] (ver también: Bertsekas, D. & Tsitsiklis, J. (1996) [27] - pag. 247). Sobre la convergencia del algoritmo Q-Learning, otra de las demostraciones la podemos encontrar en Jaakkola, T. et al. (1994) [81]. Para más información (Bertsekas, D. (2022) [21] - pag. 112). (Para un tratamiento más moderno de la convergencia de los algoritmos de aproximación estocástica tenemos Powell, W. B. (2022) [145] - pag. 256).

Empleando el método de Ecuaciones Diferenciales Ordinarias (ODE - *Ordinary Differential Equations*) tenemos el desarrollo llevado a cabo por Borkar, V. S. & Meyn, S. P. (2000) [32] (ver también: Gosavi, A. (2015) [63] - pag. 404 y Bertsekas, D. & Tsitsiklis, J. (1996) [27] - pag. 171).

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

3.3. Solución aproximada: Métodos de Simulación II (*Policy based methods*)

Procedemos a tratar el caso de la aproximación en el espacio de estrategias. Utilizaremos la notación y el esquema seguido por “*Reinforcement Learning and Optimal Control*” (Bertsekas, D. (2019) [26] - Capítulo 5).

Como hemos señalado previamente estamos tratando:

- Métodos de aproximación mediante **Funciones de valor**. Lo tratamos en el apartado anterior y nos basábamos en los conceptos de Programación Dinámica, y empleando la estructura de los algoritmos Value Iteration (V.I.) y Policy Iteration (P.I.).
- Métodos de aproximación mediante la **Función de Estrategia**. En este caso no nos basaremos en los resultados de Programación Dinámica, sino que utilizaremos métodos generales de optimización, como el método de descenso por el gradiente y la búsqueda aleatoria (random search). Podemos utilizar dos métodos:
 - 1) La vía consistirá en parametrizar la expresión de μ y optimizar respecto de una medida del rendimiento que ofrece.
 - 2) Plantear el problema como uno de Aprendizaje Supervisado y emplear mínimos cuadrados

3.3.1. Método 1: Optimización del vector de parámetros

En primer lugar procedemos a parametrizar la función de decisiones. Suponemos que depende de parámetro o vector de parámetros r :

$$\tilde{\mu}(r) \quad \text{donde : } \tilde{\mu}(r) = (\tilde{\mu}(1, r), \dots, \tilde{\mu}(n, r))$$

Teniendo en cuenta que el valor del estado inicial (i) en una trayectoria, cuando se actúa guiado por una función de estrategia $\tilde{\mu}(r)$ viene dado por: $J_{\tilde{\mu}(r)}(i)$, optimizar el valor esperado en el caso no determinista se traduce en:

$$\min_r \mathbb{E} [J_{\tilde{\mu}(r)}(i)]$$

- **Método del Gradiente.** Emplearíamos la actualización:

$$r^{k+1} = r^k - \gamma^k \cdot \nabla_r J_{\tilde{\mu}(r^k)}(i_0) \quad k = 0, 1, \dots$$

Si no se dispone del gradiente se puede aproximar por diferencias finitas de valores de la función de coste $J_{\tilde{\mu}(r^k)}(i_0)$. Si el problema es estocástico los valores de la función de coste se pueden obtener por Simulación de Montecarlo. El problema reside en que este método introduce mucho ruido, y son necesarias muchas muestras a promediar para obtener estimaciones del gradiente suficientemente precisas, lo que es muy ineficiente.

- **Método del Gradiente Incremental.** En este caso se trata de plantear el problema de optimización como un problema de Optimización Estocástica, permitiendo estrategias aleatorizadas. Si suponemos que Z es un subconjunto de \mathbb{R}^m , F es una función de \mathbb{R}^m , z es un valor de Z , y \mathcal{P}_z una distribución de probabilidad sobre Z , la expresión general sería:

$$\min_{p \in \mathcal{P}_z} \mathbb{E}_p [F(z)]$$

y restringiendo el problema introduciendo la parametrización por r :

$$\min_r \mathbb{E}_{p(z;r)} [F(z)]$$

En este caso tendríamos que la regla de actualización sería del tipo:

$$r^{k+1} = r^k - \gamma^k \cdot \nabla_r (\mathbb{E}_{p(z;r^k)} [F(z)]) \quad (3.13)$$

Que convertimos en una expresión que maneja el gradiente respecto a r del logaritmo de la función de probabilidad evaluado en r sin más que operar convenientemente. En el caso de una distribución discreta tendríamos:

$$\begin{aligned} \nabla (\mathbb{E}_{p(z;r)} [F(z)]) &= \nabla \left(\sum_{z \in Z} p(z; r) \cdot F(z) \right) \\ &= \sum_{z \in Z} \nabla p(z; r) \cdot F(z) \\ &= \sum_{z \in Z} p(z; r) \cdot \frac{\nabla p(z; r)}{p(z; r)} \cdot F(z) \\ \boxed{\frac{\nabla p}{p} = \nabla(\log p)} \quad \longrightarrow &= \sum_{z \in Z} p(z; r) \cdot \nabla(\log(p(z; r))) \cdot F(z) \\ &= \mathbb{E}_{p(z;r)} [\nabla(\log(p(z; r))) \cdot F(z)] \end{aligned}$$

Es decir la expresión para el cálculo del gradiente queda:

$$\nabla_r (\mathbb{E}_{p(z;r^k)} [F(z)]) = \mathbb{E}_{p(z;r^k)} [\nabla(\log(p(z; r))) \cdot F(z)]$$

que se sustituiría en la regla de actualización 3.13.

Empleando una linea base (*baseline*).

También puede introduciendo una linea de referencia (llamémosla b) en la expresión del cálculo del gradiente, lo que permite reducir la variabilidad, y por tanto aumentar la eficiencia. La expresión quedaría como sigue:

$$\nabla_r (\mathbb{E}_{p(z;r^k)} [F(z)]) = \mathbb{E}_{p(z;r^k)} [\nabla(\log(p(z; r))) \cdot [F(z) - b]]$$

e igualmente sustituiríamos en la ecuación 3.13.

La forma general del algoritmo del Gradiente quedaría tal como sigue.

Algoritmo 3 - Método del Gradiente

```

begin
    Inicializar  $r, \gamma$ 
    repeat
        Obtener una muestra de z:  $z^k \sim p(z; r^k)$ 
        Calcular el gradiente:  $\nabla(\log(p(z^k; r^k)))$ 
        Iterar:
             $r^{k+1} = r^k - \gamma^k \cdot \nabla_r(\log(p(z^k; r^k)) \cdot [F(z)])$ 
    until (optimo)

```

3.3.2. Método 2: Utilizar Aprendizaje Supervisado

Ahora para obtener una aproximación de la función de estrategia podríamos obtener observaciones del sistema de interés. Tendríamos q pares de valores estado-acción: (i^s, u^s) , $s = 1, \dots, q$, lo que nos permitiría converirlo en una problema de Aprendizaje Supervisado. Posteriormente se llevaría a cabo el ajuste siguiendo cualquiera de las dos siguientes formas, que son las más usuales.

Aplicando Mínimos Cuadrados. (Least Squares). El problema de mínimos cuadrados quedaría tal como sigue (podría añadirse además un término de regularización, para asegurarnos de que el modelo no crezca excesivamente en complejidad):

$$\min_r \sum_{s=1}^q \| u^s - \tilde{\mu}(i^s, r) \|^2 \quad (3.14)$$

Aplicando Interpolación.. Para ello se deben especificar para cada $i \in \{i^1, \dots, i^s\}$ una distribución de probabilidad $\{\phi_{i1}, \dots, \phi_{is}\}$, y emplear como $\tilde{\mu}$ (ver más detalles Bertsekas, D. (2019) [26] - capítulo 5 y 6):

$$\tilde{\mu}(i) = \sum_{s=1}^q \phi_{is} u^s \quad (3.15)$$

Para obtener la colección de observaciones podríamos aplicar cualquiera de los dos siguientes métodos.

- Mediante un experto. En este caso μ se entrena para adaptarse al comportamiento de un experto, por lo que se supone que los valores u^s son valores lo más óptimos posibles ('near optimal') dado el estado i^s , que han sido generados expresamente para el aprendizaje.
- Generando muestras mediante simulación. En este caso partiríamos de un número de estados muestrales lo más elevado posible: i^s , $s = 1, \dots, q$. Posteriormente generaríamos las acciones u^s , $s = 1, \dots, q$ a través de un proceso de minimización a un paso, ya sea utilizando la aproximación de la función de valor (a un paso o varios hacia adelante) \tilde{J} :

$$u^s = \arg \min_{u \in U(i^s)} \sum_{j=1}^q p_{ij}(u) \cdot \left[g_{i^s j}(u) + \alpha \cdot \tilde{J}(j) \right]$$

o con los Q-factores aproximados:

$$u^s = \arg \min_{u \in U(i^s)} \tilde{Q}(i^s, u, r)$$

Los métodos empleando la función de estrategia (*Policy Methods*) se tratarán extensamente en el capítulo dedicado a Aprendizaje por Refuerzo (*Reinforcement Learning*), cuando se trate el algoritmo REINFORCE.

Resumen - Ideas principales

Esto son un conjunto de ideas resumidas que permiten hacerse una idea de todo aquello que se ha analizado en el capítulo.

1. La iteración del algoritmo de Programación Dinámica ($J_{k+1} = TJ_k$) converge a la función de coste óptimo J^* , que es solución única de la ecuación de Bellman $J = TJ$.
2. También se emplearán redes recurrentes.
3. Además utilizaremos un tipo especial de redes recurrentes, como son las redes LSTM.
Su uso es muy extendido en este tipo de modelizaciones.

Capítulo 4

Aprendizaje por refuerzo (RL)

El Aprendizaje por refuerzo (*Reinforcement Learning*) es un área del Aprendizaje Automático (*Machine Learning*) (ver fig. 4.1) cuyo objetivo es la resolución de problemas de toma de decisiones secuenciales óptimas. Pese a que sus inicios se remontan a 1950, la auténtica explosión ha surgido alrededor del periodo que va de 2013 al presente. En estos años se han conseguido progresos realmente notables en el aprendizaje de tareas que requieren inteligencia, como son los juegos.

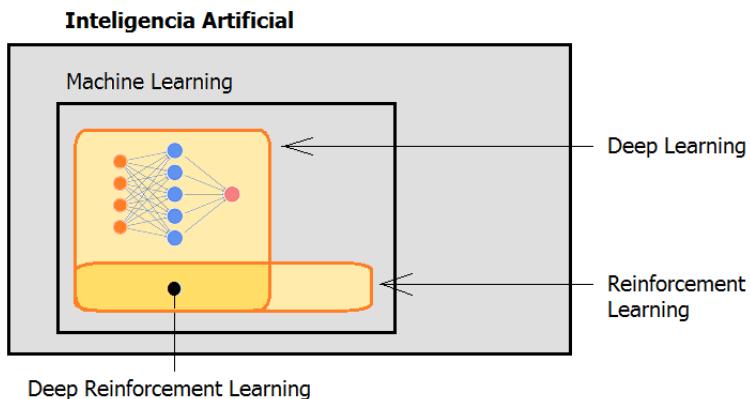


Figura 4.1: Esquema de las disciplinas que confluyen en el Reinforcement Learning.

En esta evolución ha tenido mucho que ver la madurez del Aprendizaje Profundo (*Deep Learning*), que consiste en el empleo de Redes Neuronales de varias capas, que en este ámbito se emplean generalmente como aproximadores funcionales. Esto ha dado lugar, de hecho, a la aparición de un área propia dentro del Aprendizaje por Refuerzo, como es del Aprendizaje por Refuerzo profundo (*Deep Reinforcement Learning*).

Originariamente los problemas de secuencias de decisiones eran abordados con el empleo de técnicas como las que proporciona la **Programación Dinámica**, tal y como hemos analizado extensamente en el capítulo 2. Los primeros resultados en este sentido quedaron recogidos en los artículos clásicos: Bellman (1952) [15] y Bellman (1957) [14].

El principal obstáculo para su aplicación reside en que el conocimiento preciso de todas las reglas que gobiernan el sistema es un requisito imprescindible. Es por ello que algunos autores comenzaron a explorar un nuevo enfoque: la aproximación computacional del aprendizaje a través de la interacción, con un aprendizaje orientado a objetivos y en el que las simulaciones juegan un

papel fundamental (Barto, A. & Sutton, R. (2018) [181]) ¹.

De la mano del Aprendizaje por Refuerzo (*Reinforcement Learning*), los investigadores comenzaron a lograr que agentes entrenados mediante estas técnicas, superasen el rendimiento humano en habilidades complejas como las que se ponen en práctica en juegos de mesa o de ordenador, y sin conocimiento previo de las reglas. Los primeros avances importantes surgieron a principios de los años 90, en juegos como el Backgammon (Tesauro (1994)) [187].

El impulso definitivo llegaría alrededor de 2010, con la fundación de DeepMind ² por: Demis Hassabis, Shane Legg y Mustafa Suleyman. Su objetivo fundamental: ampliar la Inteligencia Artificial, desarrollando algoritmos de propósito general que permitan desarrollar agentes dotados de inteligencia. Esta Inteligencia Artificial General permitiría abordar la resolución de diferentes tareas, no sólo las circunscritas a un dominio muy concreto como hasta ahora.

Algunos de los logros alcanzados los encontramos por ejemplo en el aprendizaje con consolas de videojuegos Atari (Bellemare (2012) [13], y posteriormente Mnih et al. (2013) [115] y Mnih et al. (2015) [114]), o en el juego de estrategia Go. Las sucesivas generaciones AlphaGo y AlphaGo Zero han logrado batir al campeón mundial en esta disciplina (David Silver et al. (2016) [171] y (2017) [172]).

	PROGRAMACIÓN DINÁMICA	Aprendizaje por Refuerzo
Espacio de Estados y Acciones	$i \in X, u \in U$	$s \in \mathcal{S}, a \in \mathcal{A}$
Estrategia	$\mu(i)$	$\pi(s) \quad \pi(a s)$
Función de Valor	$J(i) \quad Q(i, u)$	<ul style="list-style-type: none"> ■ <u>Valor Exacto:</u> $v^\pi(s) \quad q^\pi(s, a)$ ■ <u>Estimación:</u> $V^\pi(s) \quad Q^\pi(s, a)$

Figura 4.2: Esquema de las equivalencias entre la notación empleada en Programación Dinámica y la utilizada en Aprendizaje por Refuerzo.

El **Aprendizaje por Refuerzo** se construye en gran parte sobre los resultados que hemos visto de **Programación Dinámica**, por lo que existen una serie de relaciones que conviene resaltar antes de continuar, y que quedan reflejadas en la figura 4.2.

Las funciones de valor a partir de ahora acumulan lo que denominaremos recompensas, y convierten el problema en uno de maximización.

¹WEB Sutton: <http://incompleteideas.net/>

²WEB: <https://deepmind.com/>

En cuanto al objetivo sigue siendo encontrar la estrategia del agente que hace óptimas las decisiones. Para aclarar los matices que diferencian ambos enfoques resumimos en la cuadro 4.1 las equivalencias entre términos.

Aprendizaje por Refuerzo <i>(Reinforcement Learning)</i>	Programación Dinámica y Control Óptimo
Agente	Decisor
Ambiente	Sistema
Recompensa	Coste
Planificación	Resolución de un problema de PD con un modelo matemático conocido del sistema
Aprendizaje	Resolución de un problema de PD sin manejar un modelo explícito
Autoaprendizaje (self-learning o self-play)	Resolución de un problema de PD utilizando alguna forma de iteración de valor
Predicción	Evaluación de una estrategia
Control	Encontrar la mejor estrategia
Backup	Aplicación del operador de Bellman en algún estado
Sweep	Aplicación del operador de Bellman a todos los estados
Ground Truth	Evidencias empíricas o información proporcionada mediante observaciones directas

Cuadro 4.1: *Equivalencias entre los términos empleados en Reinforcement Learning y Programación Dinámica (PD) (y Control Óptimo).* (Adaptado de Bertsekas, D. (2019) [26] - pag. 44).

En la medida en que en Aprendizaje por Refuerzo empleamos la experimentación, y la información se extrae de las interacciones de un agente con su entorno, ahora no disponemos de un modelo explícito del sistema. Es lo que se denomina *model-free*, y emplea extensamente métodos de aprendizaje para buscar la estrategia óptima, tal y como veremos.

La estrategia que está siendo evaluada y aprendida se denomina **target policy**, en contraposición a **behavior policy**, que es empleada para generar las decisiones. Por ello los métodos que emplean la misma estrategia que emplean para explorar se denominan **on-policy**, mientras que si son distintas se llaman **off-policy**.

En los apartados siguientes pasamos a analizar algunos conceptos más en detalle, y a desarrollar la formalización matemática.

4.1. Conceptos

Nos enfrentamos al problema de determinar la secuencia de decisiones que conducen a un mejor resultado para un decisor, cuando este interactúa con un entorno cuya dinámica desconoce. La única información (feedback) sobre la bondad de sus acciones la recibe en forma de algún tipo de refuerzo o recompensa. La notación y terminología que emplearemos se detalla a continuación.

-Agente Es el encargado de tomar decisiones.

-Ambiente (*Environment*) Es todo aquello que rodea al agente, y de donde obtiene la información (en términos de recompensa) que dirige su aprendizaje.

-Estado (s , donde $s \in \mathcal{S}$) Alude a la situación en que se encuentra el agente. El conjunto de todos los posibles estados se simboliza por \mathcal{S} . En la medida en que el estado del agente varía secuencialmente se suele introducir el tiempo como subíndice. Una forma de notar que el agente se encuentra en el estado s , en el instante t sería: $S_t = s$.

Además cada estado puede ser caracterizado por un conjunto de características (*features*) que miden algún aspecto que interesa considerar. $\phi : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}^d$, tal que $\phi(s, a, s') = (\phi_1, \dots, \phi_d)$ (donde $s, s' \in \mathcal{S}$ y $a \in \mathcal{A}$).

$$\phi(s, a, s') = (\phi_1(s, a, s'), \phi_2(s, a, s'), \dots, \phi_d(s, a, s')) \quad (4.1)$$

Es decir, cada característica adopta la forma $\phi_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, $i = 1, \dots, d$.

-Acción (a , donde $a \in \mathcal{A}$) Es la respuesta del agente ante la información que recibe del entorno. Para introducir el tiempo, y expresar que en el instante t se tomó la decisión a , la notación quedaría: $A_t = a$.

-Recompensa (*reward*: r) Es el estímulo que recibe el agente de parte del entorno tras llevar a cabo una acción a , lo que conlleva pasar de un estado s a otro s' .

Formalmente tendríamos una función $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, tal que $R(s, a, s') = r$ (donde $s, s' \in \mathcal{S}$ y $a \in \mathcal{A}$).

Para expresar que en un determinado instante t se recibió una recompensa r tendríamos: $R_t = r$.

Generalmente nos interesa la ganancia acumulada, llamémosla G , a partir de un determinado instante de tiempo t ($t+1, t+2, \dots, t+T$), por lo que es normal emplearlo como subíndice de G : G_t . Para facilitar el tratamiento matemático se suele tomar un horizonte infinito ($T = \infty$). Para reflejar que el efecto de los momentos futuros influye menos que el momento presente se toma un *factor de descuento*, $\gamma \in [0, 1]$.

$$G_t = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \dots + \gamma^T \cdot R_{t+T} = \sum_{i=0}^T \gamma^i \cdot R_{t+1+i} \quad (4.2)$$

-Política o Estrategia (*policy*: π) Es una función que define la manera de comportarse del agente, especificando la acción a llevar a cabo en cada estado. $\pi : \mathcal{S} \rightarrow \mathcal{A}$, tal que $\pi(s) = a$ (donde $s \in \mathcal{S}$ y $a \in \mathcal{A}$).

Puede ser una función determinista como acabamos de ver, aunque también puede ser una distribución de probabilidad. En este caso se suele notar: $\pi(a|s)$.

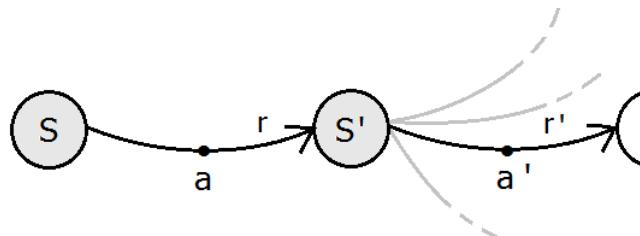


Figura 4.3: *Transiciones entre estados.*

-Probabilidades de transición. En el ámbito de un Proceso de Decisión de Markov, recoge la probabilidad de pasar de un estado s a otro s' tras llevar a cabo una acción a . Notado por algunos autores como: $\mathcal{P}_{s,s'}^a$, aunque también simplemente: $p(s'|s, a) = p(S_{t+1} = s' | S_t = s, A_t = a)$.

Estas probabilidades habrán de cumplir la propiedad de **Markov**, que consiste en que la probabilidad de pasar a un estado s' sólo depende lo ocurrido en el estado inmediatamente anterior, y no de toda su historia pasada.

-Trayectoria. Un conjunto de observaciones secuenciales (*estado, acción, recompensa*) forman una trayectoria (τ). Para especificar abreviadamente que se trata de una realización concreta ($S_t = s, A_t = a, R_t = r$), notémoslo acompañando el subíndice al valor observado: $\tau = \{(s_0, a_0, r_0), (s_1, a_1, r_1), (s_2, a_2, r_2), \dots\}$.

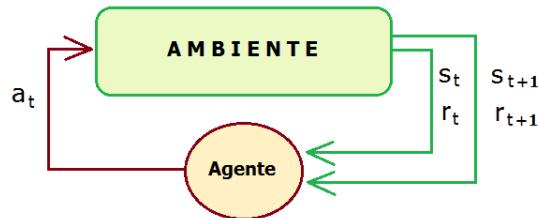


Figura 4.4: *El agente dispone, como información del Entorno, de los valores (s_t, r_t) , donde s_t alude al estado en el instante 't', y r_t el feedback en forma de recompensa. En base a estos valores, lleva a cabo una acción: a_t . Esto se traduce en una transición a otro estado s_{t+1} . La consecuencia para el agente consiste en recibir una recompensa: r_{t+1} .*

Finalmente, la dinámica que se observa en una situación como la que estamos modelizando quedaría resumida tal y como refleja la Figura 4.4.

4.2. Proceso de Decisiones de Markov (MDP)

El modelo apropiado para representar un proceso de decisiones secuenciales es el conocido como Proceso de Decisiones de Markov (MDP - *Markov Decision Process*). En él las probabilidades de transición entre estados dependen únicamente del estado actual y de la decisión que se toma. La definición formal aparece a continuación.

DEFINICIÓN 4.1: Proceso de Decisiones de Markov (MDP)

Decimos que se ha definido un **Proceso de Decisiones de Markov (MDP - Markov Decision Process)**, representado por la tupla: $(\mathcal{S}, \mathcal{A}, R, p)$, cuando quedan especificados cada uno de los siguientes elementos:

- *Espacio de estados: \mathcal{S} . Una transición quedará representada mediante el estado inicial s y el estado destino s' , (donde $s, s' \in \mathcal{S}$).*
- *Espacio de acciones: \mathcal{A} . ($a \in \mathcal{A}$).*
- *Función de Recompensa: R .*

$$\begin{aligned} \mathbf{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} &\longrightarrow \mathbb{R} \\ (s, a, s') &\longmapsto r \end{aligned}$$

- *Probabilidades de Transición: $p(\cdot)$. Si suponemos definidas las variables aleatorias en tiempo discreto ($t = 1, 2, \dots$): S_t, A_t que toman valores respectivamente en los conjuntos anteriores ($S_t \in \mathcal{S}, A_t \in \mathcal{A}$), y H_t representa la historia hasta t , entonces:*

$$P(S_{t+1} = s' | H_t) = P(S_{t+1} = s' | S_t = s, A_t = a) = p(s'|s, a)$$

Cuando no se conocen las probabilidades de transición hemos de acudir a métodos experimentales. Una vía por la que han optado algunos autores (Moore, A. & Atkeson, C. (1993) [118]) consiste en obtener estimaciones explícitas de los parámetros del MDP; es decir, de las probabilidades de transición y de las recompensas, y entonces aplicar los métodos de Programación Matemática.

Sin embargo, generalmente se emplean métodos de aprendizaje que no necesitan manejar expresamente las probabilidades, y es el enfoque que desarrollaremos. Con estos métodos podemos obtener la estimación del valor de cada estado, y determinar las acciones óptimas que conducen a ellos.

Entre los algoritmos de aprendizaje que se utilizan, podemos distinguir algoritmos ***off-line***, que llevan a cabo la computación antes del proceso de control/decisión (Un ejemplo lo tenemos en el uso de Redes Neuronales, que necesitan un procesamiento por lotes y por tanto la información ha de ser acumulada y almacenada previamente). Por el contrario los algoritmos ***on-line*** llevan a cabo la computación cuando el proceso comienza, y los datos van estando disponibles.

Existen casos en los que el agente no puede determinar con exactitud el estado del sistema. Para estos casos existe una variante de este modelo, que es el Proceso de Markov Parcialmente Observable (POMDP - Partial Observable Markov Decision Process).

4.3. MDP Parcialmente Observable (POMDP)

El Proceso de Markov Parcialmente Observable (POMDP - Partially Observable Markov Decision Process) es una generalización del Proceso de Markov (MDP), donde permitimos la existencia de incertidumbre sobre el estado del sistema. Una primera recopilación de resultados la podemos encontrar en Monahan, G.E. (1982) [117].

La falta de conocimiento sobre el estado del sistema se suele sustituir por un estado interno, una creencia del agente sobre las condiciones exteriores (Cassandra, A. R. et al. (1994) [42]). Esto se traduce en una distribución de probabilidad ($b(\cdot)$) sobre \mathcal{S} basada en las observaciones llevadas a cabo (z , donde $z \in \mathcal{Z}$).

DEFINICIÓN 4.2: Proceso de Decisiones de Markov Parcialmente Observable

Decimos que se ha definido un **Proceso de Decisiones de Markov Parcialmente Observable (POMDP)** - Partially Observable Markov Decision Process), notado $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, R, p, o)$, cuando quedan especificados cada uno de los siguientes elementos:

- Espacio de estados: \mathcal{S} . Una transición quedará representada mediante el estado inicial s y el estado destino s' (donde $s, s' \in \mathcal{S}$), aunque no es observable por el agente.
- Espacio de acciones: \mathcal{A} . ($a \in \mathcal{A}$).
- Espacio de observaciones: \mathcal{Z} . ($z \in \mathcal{Z}$).
- Función de Recompensa.

$$\begin{aligned} \mathbf{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} &\longrightarrow \mathbb{R} \\ (s, a, s') &\longmapsto r \end{aligned}$$

Y si suponemos definidas las variables aleatorias en tiempo discreto ($t = 1, 2, \dots$): S_t, A_t, Z_t que toman valores respectivamente en los conjuntos anteriores ($S_t \in \mathcal{S}, A_t \in \mathcal{A}, Z_t \in \mathcal{Z}$), y H_t el historial pasado de transiciones hasta el instante t , entonces las probabilidades que gobiernan la dinámica del sistema vienen dadas por:

- Probabilidades de Transición: $p(\cdot)$.

$$P(S_{t+1} = s' | H_t) = P(S_{t+1} = s' | S_t = s, A_t = a) = p(s'|s, a)$$

- Probabilidades de las observaciones: $o(\cdot)$.

$$P(Z_{t+1} = z | H_t) = P(Z_{t+1} = z | S_t = s, A_t = a) = o(z|s, a)$$

Algunos métodos de solución los podemos encontrar además en Kaelbling, L. P. et al. (1998) [84]. También en Jaakkola, T. et al. (1994) [82], quienes se centran en identificar los parámetros del MDP subyacente para aplicar luego los métodos tradicionales de Programación Dinámica.

En lo que sigue analizaremos en profundidad la implementación de métodos de solución para el Proceso de Decisiones de Markov.

4.4. Funciones para la toma de decisiones del agente

Los algoritmos de Aprendizaje por Refuerzo (*Reinforcement Learning*) implican generalmente la estimación de **funciones de valor**, de manera análoga a como hacíamos en Programación Dinámica (con la función $J(i)$, $i \in X$). La intención es la de disponer de una medida de la bondad de un estado, o de una acción llevada a cabo en dicho estado. La idea de bondad alude en este caso al valor acumulado descontado de las recompensas que recibe el agente desde el instante en consideración en adelante.

La **función de estrategia**, por otro lado, nos permite relacionar cada estado con la acción más apropiada para llevar a cabo, y en conjunción con la función de valor (o por si sola), permite abordar el problema de la optimización de las decisiones del agente.

Empezaremos definiendo la primera de las funciones, para a continuación pasar a tratar la segunda de ellas.

4.4.1. Funciones de valor

$v^\pi(s)$ (*Valor del estado 's'*). Es el retorno o ganancia esperada cuando se parte desde el estado 's', y el agente se comporta a partir de dicho punto siguiendo la estrategia fijada por π . Es una forma de reflejar la bondad del estado 's'. Formalmente:

DEFINICIÓN 4.3: Función de valor de estado (*State value function*)

Se denomina **función de valor** de un estado 's' bajo una estrategia π , notado $v^\pi(s)$ a la siguiente expresión:

$$v^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

que refleja el valor descontado esperado que supone para el agente el encontrarse en un estado s y siguiendo una estrategia π en su objetivo de maximizar el valor obtenido.

$$\begin{aligned} v^\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-1} R_T + \dots \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \cdot \underbrace{(R_{t+2} + \gamma R_{t+3} + \dots + \gamma^{T-2} R_T) + \dots}_{G_{t+1}} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_t + \gamma \cdot G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \cdot \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma \cdot \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \cdot \sum_{s',r} p(s',r|s,a) [r + \gamma \cdot v^\pi(s')] \end{aligned} \tag{4.3}$$

Lo normal en el tipo de problemas que estamos tratando es que no se disponga de las probabilidades transición entre los estados ($p(s',r|s,a)$) por lo que el cálculo exacto no se puede llevar a cabo. Analizaremos posteriormente la forma de estimar $v^\pi(s)$ a partir de experimentación o simulación.

DEFINICIÓN 4.4: Función de valor estado-acción

Se denomina **función de valor** de un estado ' s ' cuando se lleva a cabo una acción ' a ', bajo una estrategia π (notado $q^\pi(s, a)$) a la siguiente expresión:

$$q^\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

que refleja el valor descontado esperado que supone para el agente el encontrarse en un estado s y siguiendo una estrategia π en su objetivo de maximizar el valor obtenido.

$q^\pi(s, a)$ (*Valor del estado ' s ' y la acción ' a '*). Representa la bondad de llevar a cabo la acción ' a ' cuando el agente se encuentra en el estado ' s ' y se comporta siguiendo una estrategia π ..

$$\begin{aligned} q^\pi(s, a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_t + \gamma \cdot G_{t+1} \mid S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma \cdot v^\pi(s')] \end{aligned} \quad (4.4)$$

Nuevamente nos enfrentamos al problema de no disponer de las probabilidades de transición ($p(s', r | s, a)$), por lo que utilizaremos técnicas de estimación que analizaremos más adelante.

Una vez disponemos del valor asociado a cada estado en función de las acciones que condujeron al agente a él, el proceso de construir la estrategia óptima es sencillo. Para ello acudimos a la función de estrategia ('policy').

4.4.2. Función de estrategia

La situación que pretendemos analizar es la de una agente dotado de capacidad de aprendizaje que interactúa con su entorno a lo largo del tiempo. El agente percibe de alguna manera aspectos de lo que le rodea, y toma decisiones con la finalidad de conseguir un objetivo, que en este caso consiste en maximizar las recompensas que recibe de su entorno.

DEFINICIÓN 4.5: Función de estrategia

La función de estrategia es la función que asigna a cada estado $s \in \mathcal{S}$ la acción a llevar a cabo cuando el agente se encuentra en él: $\pi(s) = a \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$.

Para cualquier Proceso de Decisiones de Markov (MDP) existe una estrategia (policy) que es óptima π^* , tal que: $v^{\pi^*}(s) \geq v^\pi(s) \quad \forall \pi$. La expresión en términos de la función valor estado-acción sería:

$$\pi^*(s) = \arg \max_{a' \in \mathcal{A}} q^\pi(s, a') \quad (4.5)$$

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

4.5. Estimación de las funciones de valor

Para obtener una aproximación a la función de estrategia óptima se emplean métodos de aprendizaje y se basan exclusivamente en la recolección de experiencias. Cuando nuestro objetivo consiste simplemente en estimar una de las funciones de estado para una cierta estrategia π , estamos ante un problema de **predicción**. Mientras que si buscamos el valor óptimo de la función de estrategia estamos ante un problema de **control**. Según su naturaleza podemos distinguir:

Método de Montecarlo: alude a una clase de algoritmos que promedian las recompensas para cada estado-acción a partir de trayectorias o episodios completos (Ver figura 4.5).

Métodos de aprendizaje por Diferencias Temporales (TD) : son un tipo de algoritmos que actualizan los valores paso a paso, empleando lo que se llama el error o diferencia temporal, que es la diferencia entre una estimación adelante (a un paso o a varios) y el valor estimado al paso corriente. Este procedimiento de obtener una estimación en base a otra estimación de pasos siguientes recibe el nombre de *Bootstrapping*. Cuantos más periodos hacia adelante consideremos más nos aproximamos al método de Montecarlo (ver fig. 4.5).

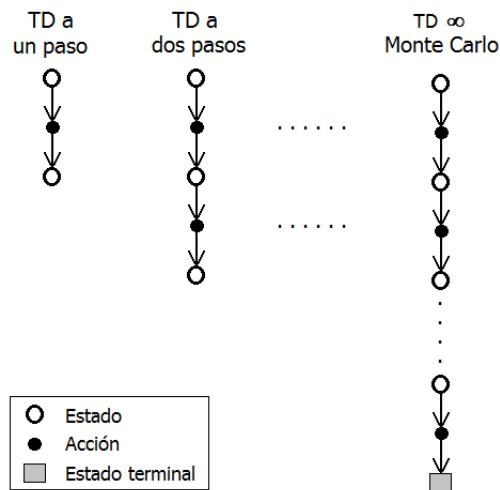


Figura 4.5: Conforme hacemos crecer los pasos hacia adelante que tenemos en cuenta a la hora de realizar la estimación, nos aproximamos al caso límite: el método de Monte Carlo, que ha de disponer de toda la trayectoria antes de realizar los cálculos.

Según sea el objetivo del proceso de aprendizaje podemos distinguir:

-**Método de las funciones de valor** (*Value based methods*). Se emplean algoritmos que estiman alguna de las funciones que hemos visto (la función de estado, o la función de estado-acción).

-**Método de la función de estrategia** (*Policy based methods*). Una forma alternativa consiste en construir una forma parametrizada de la función de estrategia ($\pi_\theta(s)$), y optimizar directamente sobre ella. Un pequeño esquema lo podemos encontrar en la figura 4.6.

-**Método del actor-crítico** (*Actor-Critic methods*). Consiste en aprender simultáneamente una aproximación tanto a la función de valor como a la función de estrategia. El 'crítico' alude a la función de valor, que se encarga de juzgar la bondad de los valores que genera el 'actor' (la

función de estrategia) (Ver figura ??).

El procedimiento que se aplica en el caso de los métodos de Diferencias Temporales está inspirado en la Programación Dinámica. Si pretendemos encontrar el valor de un determinado estado en un proceso de Markov, la ecuación de Bellman establece una relación del tipo: $Valor(s) = R(s, a, s') + \gamma Valor(s')$.

Esto se traduce en que el valor de un determinado estado 's' se puede expresar como la suma de la Recompensa inmediata (resultado de llevar a cabo la transición de s a s' por medio de la ejecución de la acción a) más el valor descontado del siguiente estado s' . De dicha expresión surgen un conjunto de ecuaciones, resultado de aplicar recursivamente dicha relación. La aplicación hacia atrás, una vez llegado al estado terminal del que podemos establecer su valor, conduce a la obtención del valor deseado.

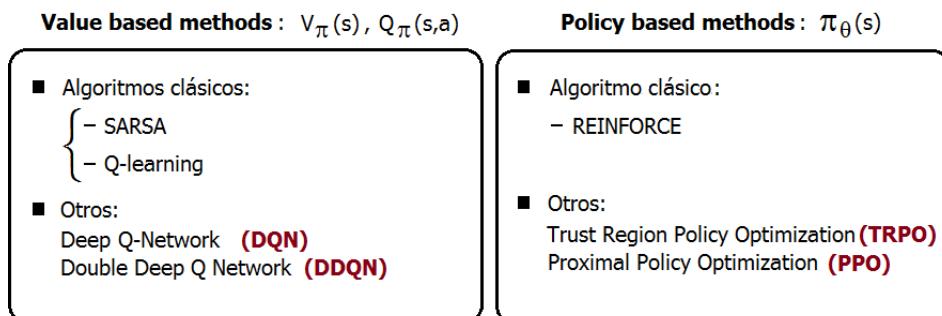


Figura 4.6: Algunos de los algoritmos empleados en Aprendizaje por Refuerzo (Reinforcement Learning).

4.5.1. Método de Montecarlo (MC)

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

4.5.2. Método de las diferencias temporales (TD)

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

4.6. Estimación de la función de estrategia

La situación que pretendemos analizar es la de una agente dotado de capacidad de aprendizaje que interactúa con su entorno a lo largo del tiempo. El agente percibe de alguna manera aspectos

de lo que le rodea, y toma decisiones con la finalidad de conseguir un objetivo, que en este caso consiste en maximizar las recompensas que recibe de su entorno.

La situación que pretendemos analizar es la de una agente dotado de capacidad de aprendizaje que interactúa con su entorno a lo largo del tiempo. El agente percibe de alguna manera aspectos de lo que le rodea, y toma decisiones con la finalidad de conseguir un objetivo, que en este caso consiste en maximizar las recompensas que recibe de su entorno.

Objetivo

El objetivo consiste en encontrar la estrategia óptima ($\pi^*(s)$), que será aquella que conduzca a unas recompensas acumuladas máximas, a lo largo de las trayectorias (τ) que genera la estrategia en cuestión ($\tau \sim \pi$). Como las recompensas se evalúan en términos de esperanza se notará: $\mathbb{E}_{\tau \sim \pi}$ o abreviadamente: \mathbb{E}_π .

Se suele introducir un factor de descuento γ , que refleja que las ganancias futuras tienen un menor valor para nosotros que las presentes. El objetivo deseable para el agente, por tanto, consistirá en maximizar el valor esperado descontado de las recompensas acumuladas. El 'retorno' de una cierta trayectoria ' τ ' vendrá dado por:

$$G(\tau) = R_1 + \gamma \cdot R_2 + \gamma^2 \cdot R_3 + \dots + \gamma^T \cdot r_{T+1} = \sum_{i=0}^T \gamma^i \cdot R_{i+1} \quad (4.6)$$

Y por tanto nos interesa maximizar la expresión siguiente (tomando generalmente $T = \infty$):

$$\mathbb{E}_{\tau \sim \pi}[G(\tau)] = \mathbb{E}_{\tau \sim \pi} \left[\sum_{i=0}^T \gamma^i \cdot R_{i+1} \right] \quad (4.7)$$

El proceso de búsqueda de la mejor estrategia es un proceso iterativo que se concreta en dos pasos, uno de evaluación y otro de mejora.

- Evaluación de las estrategias (*Policy Evaluation (PE)*).
Consiste en calcular la función de valor asociada a la estrategia en consideración π .
- Mejora de la estrategia (*Policy Improvement (PI)*).
Se realiza a través de la selección de la acción que consigue mejores resultados en cada estado, y se realiza iterativamente.

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_t + \gamma \cdot G_{t+1} | s] \quad (4.8)$$

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[R_t + \gamma \cdot G_{t+1} | s, a] \quad (4.9)$$

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

4.7. Value based Methods.

Con los algoritmos que analizamos a continuación podemos aproximar, por medio del aprendizaje, una de las funciones de valor que hemos visto. A partir de las valoraciones que proporciona de los pares (s,a) - en el caso de la función Q -, es inmediato el obtener la estrategia óptima.

4.7.1. SARSA

El algoritmo fue introducido por Rummery & Naranjan (1994) [155]. La regla de actualización de la función de valor estado-acción es la siguiente:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot Q(s', a') - Q(s, a)] \quad (4.10)$$

Algoritmo 4 - SARSA

```

begin
    Inicializar la tasa de aprendizaje  $\alpha$ 
    Inicializar la probabilidad de exploración  $\epsilon$ 
    Inicializar arbitrariamente  $Q(s, a) \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$ 
repeat
    Inicializar  $s$ 
    Elegir acción  $a$ 
    repeat
        Observar:  $r$  y  $s'$ 
        Elegir acción  $a'$  desde  $s'$  ( $\epsilon$ -greedy)
         $Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ 
         $a \leftarrow a'$ 
    until ( $s$  estado Terminal)
until (No más episodios)

```

Regla de actualización 1 (SARSA):

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot Q(s', a') - Q(s, a)]$$

equivalentemente:

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot [r + \gamma \cdot Q(s', a')]$$

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

4.7.2. Q-learning

Aparecen métodos TD en el artículo clásico de Sutton (1988) [180]. La regla de actualización de la función de valor estado-acción es la siguiente:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)] \quad (4.11)$$

Algoritmo 5 - Q-Learning

```

begin
    Inicializar la tasa de aprendizaje  $\alpha$ 
    Inicializar la probabilidad de exploración  $\epsilon$ 
    Inicializar arbitrariamente  $Q(s, a) \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$ 
repeat
    Inicializar  $s$ 
    Elegir acción  $a$ 
    repeat
        Observar:  $r$  y  $s'$ 
        Elegir acción  $a'$  desde  $s'$  ( $\epsilon$ -greedy)
         $Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ 
         $a \leftarrow a'$ 
    until ( $s$  estado Terminal)
until (No más episodios)

```

Regla de actualización 2 (Q-Learning):

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)]$$

equivalentemente:

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s', a')]$$

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

4.8. Policy Methods.

A continuación abordamos el estudio de aquellos algoritmos que permiten el aprendizaje directo de la función de estrategia π . Entre las ventajas se encuentra la convergencia local asegurada a un óptimo local (tal y como demostraron Sutton et al. (2000)) [182].

4.8.1. REINFORCE

Se trata de un algoritmo clásico debido a Ronald J. Williams (1992) [200]. Su utilidad reside en que permite aprender la función π , lo que conduce a obtener como resultado las probabilidades de los cursos de acción a partir de los estados.

La idea sobre la que se fundamenta es la de incrementar sucesivamente (a lo largo del aprendizaje) las probabilidades asociadas a acciones que conducen a buenos resultados. La estrategia óptima será la que maximice el total acumulado de las recompensas descontadas. En términos precisos, la función objetivo a maximizar sera:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi}[R(\tau)] = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T \gamma^t \cdot r_t \right] \quad (4.12)$$

que es la ganancia esperada sobre todas las trayectorias generadas por el agente. Esto se traduce en:

$$\max_{\theta} J(\pi_\theta) \quad (4.13)$$

Para ello acudimos al método de descenso por el gradiente, por lo que los parámetros se verán actualizados a partir de la siguiente regla:

$$\theta \longleftarrow \theta + \alpha \cdot \nabla_{\theta} J(\pi_\theta) \quad (4.14)$$

y donde el gradiente de la función de estrategia (π) adopta la expresión que aparece a continuación:

$$\nabla_{\theta} J(\pi_\theta) = \nabla_{\theta} \mathbb{E}_{\tau \sim \pi}[R(\tau)] = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T R_t(\tau) \cdot \nabla_{\theta} \log \pi_\theta(a_t | s_t) \right] \quad (4.15)$$

Mejora: REINFORCE empleando una linea base.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

Algoritmo 6 - REINFORCE**begin** Iniciarizar la tasa de aprendizaje α Iniciarizar los pesos θ de la red neuronal π_θ **for** $episodio = 1$ **to** MAX **do** Generar una trayectoria $\tau = (s_0, a_0, r_0, \dots, s_T, a_T, r_T)$ empleando la estrategia en vigor π Establecer $\nabla_\theta J(\pi_\theta) = 0$ **for** $t = 0$ **to** T **do**

$$R_t(\tau) = \sum_{t=0}^T \gamma^t \cdot r_t$$

$$\nabla_\theta J(\pi_\theta) \leftarrow \nabla_\theta J(\pi_\theta) + R_t(\tau) \cdot \nabla_\theta \log \pi_\theta(a_t, s_t)$$

/* Actualizamos los parámetros de la red neuronal

$$\theta \leftarrow \theta - \alpha \cdot \nabla_\theta J(\theta)$$

4.8.2. REINFORCE with reward-to-go

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

4.8.3. REINFORCE with baseline

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

4.9. Actor-Critic Methods.

Los métodos que se engloban bajo este epígrafe combinan los métodos de las funciones de valor y el ajuste de una función de estrategia. Los principales modelos son los que se presentan a continuación.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

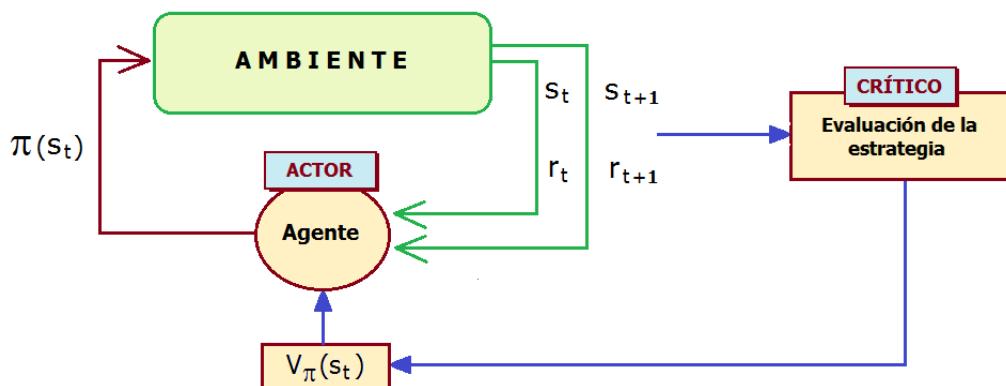


Figura 4.7: Esquema de las interacciones de un agente implementando un sistema actor-crítico (Adaptado de: Tsitsiklis, J. & Bertsekas, D. (1996) [27] - pag. 36).

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

4.10. Transferencia de aprendizaje: el problema de la representación

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los

métodos tradicionales.

4.11. Otras líneas de investigación.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

4.11.1. Imitation Learning

- Dataset Aggregation (DAgger)
- Deep Q-Learning from Demonstrations (DQfd)
- Generative Adversarial Imitation Learning (GAIL)
- Inverse Reinforcement Learning (IRL) Objetivo: Aprender la función de recompensa

4.11.2. Meta Reinforcement Learning

4.11.3. Hierarchical Reinforcement Learning

4.11.4. Imagination

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

DE CARA A LA IMPLEMENTACIÓN:

Algunas webs de interés con los entornos de prueba son Open AI Gym³ y Universe⁴. También han surgido implementaciones en código abierto de los principales algoritmos de aprendizaje por

³WEB Open AI Gym: <https://gym.openai.com/>

⁴WEB Universe: <https://openai.com/blog/universe/>

refuerzo como Stable Baselines⁵, lo que permite automatizar la tarea de aprendizaje de los agentes.

En el ámbito más específico de la Simulación social encontramos abundantes aplicaciones ligadas sobre todo al transporte. Aquí existen multitud de problemas en los que es necesario contar con agentes dotados de inteligencia, y donde el Aprendizaje por refuerzo proporciona buenos resultados.

Encontramos estudios sobre la coexistencia de peatones y conductores, y cómo lograr una reducción de accidentes de transeúntes [102] [53], sobre la regulación óptima del tráfico en las distintas situaciones que puedan provocar los agentes también se han realizado estudios [40]. Del lado de la economía encontramos por ejemplo la simulación de mercados bursátiles [163] o la gestión de recursos comunes [139].

Estudios sobre conflictos e insurgencia [45].

Planteado explícitamente como un dilema social secuencial tenemos el trabajo de referencia de Leibo et al. (2017) [100]. 213

⁵WEB Stable Baselines: <https://stable-baselines.readthedocs.io/en/master/>

4.12. Procesos de Decisión de Markov

En el presente apartado ensamblamos todas las piezas que hemos manejado previamente para así, fijar las condiciones que nos permiten resolver el problema de toma de decisiones óptimas secuenciales, cuando el entorno está sujeto a un comportamiento aleatorio. Para ello damos un conjunto de definiciones introductorias que nos permiten formalizar el tipo de proceso que buscamos.

Proceso estocástico. Si observamos un sistema que evoluciona aleatoriamente, siguiendo una determinada ley de probabilidad, y dicho sistema es observado en tiempos discretos: $n = 0, 1, 2, \dots$, podemos definir una variable aleatoria X_n asociada a cada instante temporal de observación. Esta v.a. vendría a representar el estado del sistema en el instante de observación 'n'. En términos formales se ha definido un **proceso estocástico en tiempo discreto** ($\{X_n, n \geq 0\}$).

El proceso estocástico serían este conjunto de variables aleatorias indexadas, que toman valor en un determinado conjunto \mathcal{S} , llamado el espacio de estados del proceso estocástico.

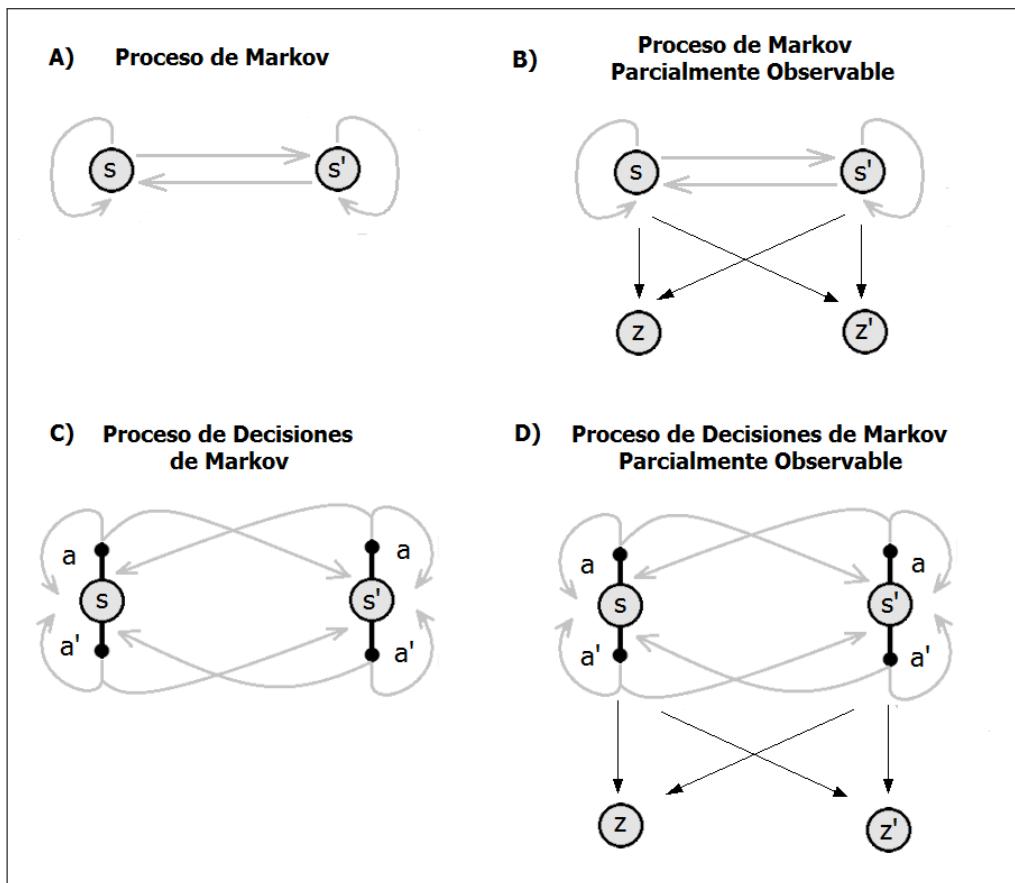


Figura 4.8: Representación de diversos tipos de procesos.

Un tipo especial de proceso es aquel que cumple la propiedad de Markov, que consiste básicamente en que, el pasado es irrelevante a la hora de predecir el futuro y únicamente aporta información el estado actual del sistema. En términos probabilísticos consistiría en que: $p(X_{n+1} = j | X_n = i, X_{n-1}, \dots, X_1, X_0) = p(X_{n+1} = j | X_n = i)$. En este caso el proceso $\{X_n, n \geq 0\}$ recibe el nombre de *Cadena de Markov* (Kulkarni, V.G. (2016) [94]).

Proceso de Markov oculto (parcialmente observable) (HMM - *Hidden Markov Model*).

Este tipo de modelo es apropiado cuando los estados del sistema bajo estudio no son directamente observables ($\{X_n, n \geq 0\}$) por lo que no podemos determinar con precisión el estado del sistema; pero, podemos observar ($\{Y_n, n \geq 0\}$) que depende del proceso de Markov X_n .

En la figura 4.8 podemos observar la representación de varios tipos de procesos, entre ellos un proceso HMM en B).

Proceso de decisiones de Markov. La situación que nos interesa analizar es la de un sistema probabilístico que evoluciona a lo largo del tiempo, y sobre el que podemos influir con la toma de decisiones, para conseguir que el sistema logre un desempeño óptimo (Puterman, M. L. (2014) [147]). La definición formal sería la que aparece a continuación.

DEFINICIÓN 4.6: Proceso de Decisiones de Markov (*MDP*)

Decimos que se ha definido un **Proceso de Decisiones de Markov** (**MDP** - *Markov Decision Process*), representado por la tupla: $(\mathcal{S}, \mathcal{A}, R, p)$, cuando quedan especificados cada uno de los siguientes elementos:

- *Espacio de estados: \mathcal{S} . Una transición quedará representada mediante el estado inicial s y el estado destino s' , (donde $s, s' \in \mathcal{S}$).*
- *Espacio de acciones: \mathcal{A} . ($a \in \mathcal{A}$).*
- *Función de Recompensa: R .*

$$\begin{aligned} \mathbf{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} &\longrightarrow \mathbb{R} \\ (s, a, s') &\longmapsto r \end{aligned}$$

- *Probabilidades de Transición: $p(\cdot)$. Si suponemos definidas las variables aleatorias en tiempo discreto ($t = 1, 2, \dots$): S_t, A_t que toman valores respectivamente en los conjuntos anteriores ($S_t \in \mathcal{S}, A_t \in \mathcal{A}$), y H_t representa la historia hasta t , entonces:*

$$P(S_{t+1} = s' | H_t) = P(S_{t+1} = s' | S_t = s, A_t = a) = p(s'|s, a)$$

Si no se conocen las probabilidades de transición podemos acudir a métodos experimentales. Una posibilidad pasaría por obtener estimaciones explícitas de los parámetros del MDP (Moore, A. & Atkeson, C. (1993) [118]); es decir, de las probabilidades de transición y de las recompensas, y entonces aplicar los métodos de Programación Matemática.

Sin embargo, la opción más habitual consiste en el empleo de métodos de aprendizaje que no necesitan manejar expresamente las probabilidades. Con estos métodos podemos obtener la estimación del valor de cada estado, y determinar las acciones óptimas que conducen a ellos.

Entre los algoritmos de aprendizaje que se utilizan, podemos distinguir algoritmos **off-line**, que llevan a cabo la computación antes del proceso de control/decisión (Un ejemplo lo tenemos en el uso de Redes Neuronales, que necesitan un procesamiento por lotes y por tanto la información ha de ser acumulada y almacenada previamente). Por el contrario los algoritmos **on-line** llevan a cabo la computación cuando el proceso comienza, y los datos van estando disponibles.

EJEMPLO 4.1: MDP

Problema de sustitución de maquinaria. Supongamos que una determinada maquinaria puede pasar por tres estados $\mathcal{S} = \{s_1, s_2, s_3\}$ correspondiéndose con la calidad de la máquina en orden creciente.

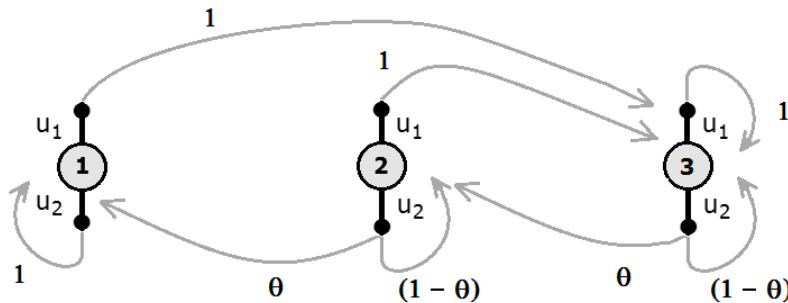
El estado del sistema evoluciona secuencialmente por etapas ($x_k, k = 0, 1, \dots$), y el decisor puede actuar llevando a cabo una de las dos acciones siguientes en cada paso $\mathcal{U} = \{u_1, u_2\}$:

$$\begin{cases} u_1 : "Reemplazar la máquina" \\ u_2 : "Seguir usando la máquina" \end{cases}$$

Supongamos que θ representa la probabilidad de que la maquinaria se deteriore hasta el nivel inferior de calidad. Entonces las probabilidades del proceso de decisiones de Markov son las que aparecen a continuación.

$$P(u_1) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad P(u_2) = \begin{bmatrix} 1 & 0 & 0 \\ \theta & (1-\theta) & 0 \\ 0 & \theta & (1-\theta) \end{bmatrix}$$

Si notamos s_i como i para simplificar notación, el proceso de funcionamiento del sistema puede representarse como refleja el siguiente gráfico, donde las probabilidades asociadas a cada transición aparecen sobre cada arco.



Cada decisión tiene un coste asociado. Si llevamos a cabo la acción u_1 (*reemplazar la máquina*) desde cualquier estado tenemos un coste de $c(x, u_1) = R$. Si por el contrario optamos por la acción u_2 y se deja funcionar sin recambiar, incurrimos en el mismo coste sea cual sea el estado: $c(x, u_2) = c, x = 1, 2, 3$.

El objetivo consiste en encontrar la secuencia de decisiones que permitan minimizar el coste asociado a la operación de la máquina (revisar la notación).

$$\mu^* = \min_{\mu} \mathbb{E}_{\mu} \left[\sum_{k=0}^{N-1} c(x_k, u_k) \mid x_0 \right]$$

Empleando la ecuación de Bellman, utilizamos programación dinámica, sustituyendo recursivamente hacia atrás, empezando por $J_N(i) = 0, i \in X$, y prosiguiendo con $k = N - 1, \dots, 0$:

$$J_k(i) = \min \{ R + J_{k+1}(1), c(i, 2) + \sum_{j=1}^3 P_{ij}(2) \cdot J_{k+1}(j) \} \quad i \in \{1, 2, 3\}$$

4.13. MDP Parcialmente Observable (POMDP)

El Proceso de Markov Parcialmente Observable (POMDP - Partially Observable Markov Decision Process) es una generalización del Proceso de Markov (MDP), donde permitimos la existencia de incertidumbre sobre el estado del sistema. Una primera recopilación de resultados la podemos encontrar en Monahan, G.E. (1982) [117].

La falta de conocimiento sobre el estado del sistema se suele sustituir por un estado interno, una creencia del agente sobre las condiciones exteriores (Cassandra, A. R. et al. (1994) [42]). Esto se traduce en una distribución de probabilidad ($b(\cdot)$) sobre \mathcal{S} basada en las observaciones llevadas a cabo (z , donde $z \in \mathcal{Z}$).

DEFINICIÓN 4.7: Proceso de Decisiones de Markov Parcialmente Observable

Decimos que se ha definido un **Proceso de Decisiones de Markov Parcialmente Observable (POMDP)** - Partially Observable Markov Decision Process), notado $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, R, p, o)$, cuando quedan especificados cada uno de los siguientes elementos:

- Espacio de estados: \mathcal{S} . Una transición quedará representada mediante el estado inicial s y el estado destino s' (donde $s, s' \in \mathcal{S}$), aunque no es observable por el agente.
- Espacio de acciones: \mathcal{A} . ($a \in \mathcal{A}$).
- Espacio de observaciones: \mathcal{Z} . ($z \in \mathcal{Z}$).
- Función de Recompensa.

$$\begin{aligned} \mathbf{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} &\longrightarrow \mathbb{R} \\ (s, a, s') &\longmapsto r \end{aligned}$$

Y si suponemos definidas las variables aleatorias en tiempo discreto ($t = 1, 2, \dots$): S_t, A_t, Z_t que toman valores respectivamente en los conjuntos anteriores ($S_t \in \mathcal{S}, A_t \in \mathcal{A}, Z_t \in \mathcal{Z}$), y H_t el historial pasado de transiciones hasta el instante t , entonces las probabilidades que gobiernan la dinámica del sistema vienen dadas por:

- Probabilidades de Transición: $p(\cdot)$.

$$P(S_{t+1} = s' | H_t) = P(S_{t+1} = s' | S_t = s, A_t = a) = p(s'|s, a)$$

- Probabilidades de las observaciones: $o(\cdot)$.

$$P(Z_{t+1} = z | H_t) = P(Z_{t+1} = z | S_t = s, A_t = a) = o(z|s, a)$$

Algunos métodos de solución los podemos encontrar además en Kaelbling, L. P. et al. (1998) [84]. También en Jaakkola, T. et al. (1994) [82], quienes se centran en identificar los parámetros del MDP subyacente para aplicar luego los métodos tradicionales de Programación Dinámica.

En lo que sigue analizaremos en profundidad la implementación de métodos de solución para el Proceso de Decisiones de Markov.

EJEMPLO 4.2: MDP Parcialmente Observable (POMDP)

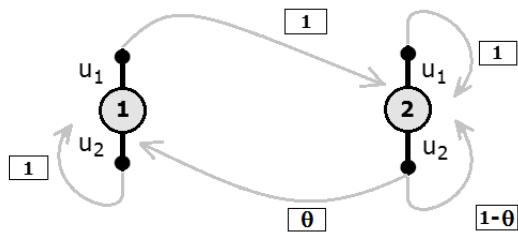
Problema de sustitución de maquinaria. Supongamos que una determinado proceso de producción puede pasar por dos estados $S = \{s_1, s_2\}$ correspondiéndose con:

$$\begin{cases} s_1 : "Funcionamiento defectuoso" \\ s_2 : "Sustitución de maquinaria" \end{cases}$$

El estado del sistema evoluciona secuencialmente por etapas (x_k , $k = 0, 1, \dots$), y el decisor puede actuar llevando a cabo una de las dos acciones siguientes en cada paso $\mathcal{U} = \{u_1, u_2\}$, donde:

$$\begin{cases} u_1 : "Reemplazar la máquina" \\ u_2 : "Seguir usando la máquina" \end{cases}$$

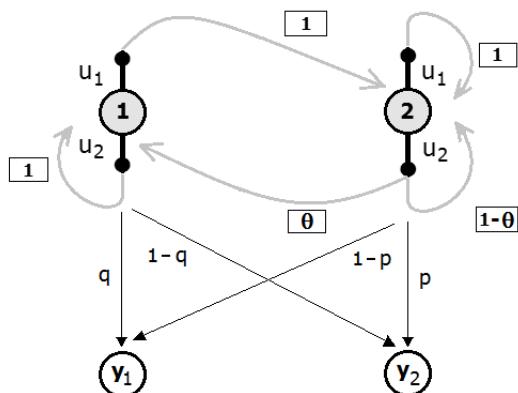
Si notamos s_i como i para simplificar notación, el proceso de funcionamiento del sistema puede representarse como refleja el siguiente gráfico.



La dificultad que surge reside en que el estado x_k no es directamente observable. Únicamente podemos juzgarlo indirectamente en base a la calidad del producto que fabrica: $\mathcal{Y} = \{y_1, y_2\}$.

$$\begin{cases} y_1 : "Producto de mala calidad" \\ y_2 : "Producto de buena calidad" \end{cases}$$

Si la probabilidad de que una máquina defectuosa genere un producto de mala calidad es p ($p(y_1|x=1) = p$) y la probabilidad de que la calidad sea buena $1-p$ ($p(y_2|x=1) = 1-p$); y la probabilidad de que una máquina en estado óptimo genere un producto de mala calidad es $1-q$ ($p(y_1|x=2) = 1-q$) y la probabilidad de que la calidad sea buena q ($p(y_2|x=2) = q$), entonces el sistema quedaría representado como:

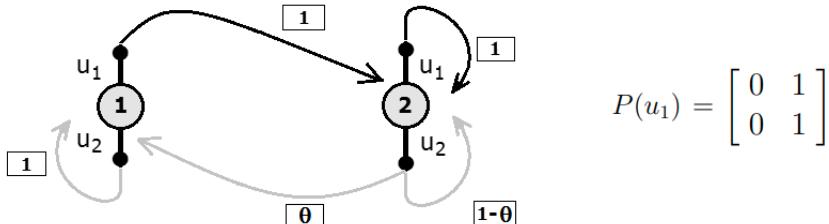


Y ahora tenemos en cuenta las transiciones relacionadas con cada acción que puede tomar el agente.

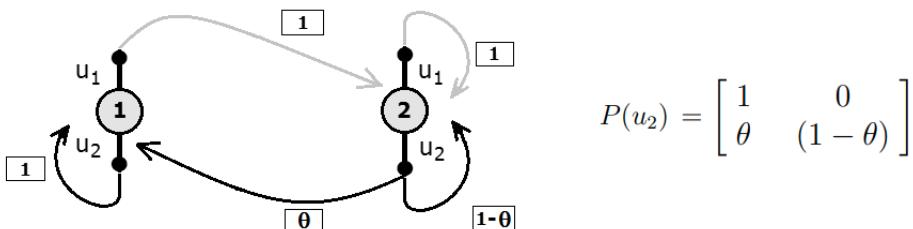
Y ahora tenemos en cuenta las transiciones relacionadas con cada acción que puede tomar el agente.

Para cada una de las decisiones ($u \in \mathcal{U}$), la matriz $P(u)$ recogerá las probabilidades: $P_{ij}(u) = p(x_{k+1} = j | x_k = i, u_k = u) \quad \forall i, j \in \mathcal{X}$.

En concreto, la matriz de probabilidades de transición asociadas a la decisión u_1 :



Y si la decisión tomada es u_2 tenemos que la matriz de probabilidades asociada es la siguiente:



La dificultad que surge reside en que el estado x_k no es directamente observable. Únicamente podemos juzgarlo indirectamente en base a la calidad del producto que fabrica.

Ahora para cada acción $u \in \mathcal{U}$, la matriz $B(u)$ recoge la distribución de probabilidad de las observaciones, asociada a la decisión u . En concreto tenemos que, los elementos que la constituyen son de la forma:

$$B_{iy}(u) = p(y_{k+1} = y | x_{k+1} = i, u_k = u) \quad \forall i \in \mathcal{X}, y \in \mathcal{Y}$$

de esta matriz extraemos las matrices $B_y(u) \ y \in \mathcal{Y}$, colocando en la diagonal de una matriz ($0_{X \times X}$) la probabilidad de que cada uno de los estados del sistema genere la observación en cuestión ' y ':

$$B_y(u) = \begin{pmatrix} B_{1y}(u) & 0 & \cdots & 0 \\ 0 & B_{2y}(u) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_{Xy}(u) \end{pmatrix} \quad (4.16)$$

Lo que en nuestro caso se traduce en (*creo que hay error en el libro, lo corrijo*):

$$\mathbf{B} = \begin{bmatrix} q & (1-q) \\ (1-p) & p \end{bmatrix} \implies B_1 = \begin{bmatrix} q & 0 \\ 0 & (1-p) \end{bmatrix} \quad B_2 = \begin{bmatrix} (1-q) & 0 \\ 0 & p \end{bmatrix}$$

Para obtener la acción óptima necesitamos la **distribución a posteriori**. Llamémosla:

$$\pi_k = [\pi_k(1), \dots, \pi_k(X)]'$$

y que recoge la estimación del agente de la probabilidad de cada estado oculto, dada la información disponible hasta el periodo k ($\mathcal{I}_k = \{\pi_0, u_0, y_1, \dots, u_{k-1}, y_k\}$):

$$\pi_k(i) = p(x_k = i | \mathcal{I}_k) \quad i \in X$$

Tenemos que π_k es un estadístico suficiente, que no crece con la dimensión k . Para su cálculo nos serviremos de T , un filtro HMM (*Hidden Markov Model filter* - filtro de una modelo oculto de Markov). Es decir, el estado de creencia (*belief state o information state*) π_k se obtiene como:

$$\pi_k = T(\pi_{k-1}, y_k, u_{k-1}) \quad \text{donde :}$$

$$T(\pi, y, u) = \frac{B_y \cdot P^t(u) \cdot \pi}{\sigma(\pi, y, u)} \quad \text{siendo : } \sigma(\pi, y, u) = \mathbf{1}_X^t \cdot B_y(u) \cdot P^t(u) \cdot \pi$$

Finalmente para obtener las acciones óptimas nos basamos en el estado de creencia. Si estamos en el periodo k , tendremos que $u_k = \mu_k^*(\pi_k)$.

Para la obtención de la estrategia óptima, vendrá en un problema finito de la aplicación recursiva hacia atrás de las siguientes expresiones.

$$J_k(\pi) = \min_{u \in \mathcal{U}} \{c'_u \pi + \sum_{y \in \mathcal{Y}} J_{k+1}(T(\pi, y, u)) \cdot \sigma(\pi, y, u)\}$$

$$\mu_k^*(\pi) = \arg \min_{u \in \mathcal{U}} \{c'_u \pi + \sum_{y \in \mathcal{Y}} J_{k+1}(T(\pi, y, u)) \cdot \sigma(\pi, y, u)\}$$

lo que se traduce en:

$$J_k(\pi) = \min_{u \in \{u_1, u_2\}} \{c'_u \pi + J_{k+1}(T(\pi, y_1, u)) \cdot \sigma(\pi, y_1, u) + J_{k+1}(T(\pi, y_2, u)) \cdot \sigma(\pi, y_2, u)\}$$

realizando los cálculos recursivamente hacia atrás tendremos:

$$\begin{aligned} J_N(\pi) &= 0 \\ J_{N-1}(\pi) &= \min_{u \in \{u_1, u_2\}} \{c'_u \pi + J_N(T(\pi, y_1, u)) \cdot \sigma_{(\pi, y_1, u)} + J_N(T(\pi, y_2, u)) \cdot \sigma_{(\pi, y_2, u)}\} \\ J_{N-2}(\pi) &= \min_{u \in \{u_1, u_2\}} \{c'_u \pi + J_{N-1}(T(\pi, y_1, u)) \cdot \sigma_{(\pi, y_1, u)} + J_{N-1}(T(\pi, y_2, u)) \cdot \sigma_{(\pi, y_2, u)}\} \\ &\dots \end{aligned}$$

En resumen, encadenando valores recursivamente hacia atrás tenemos:

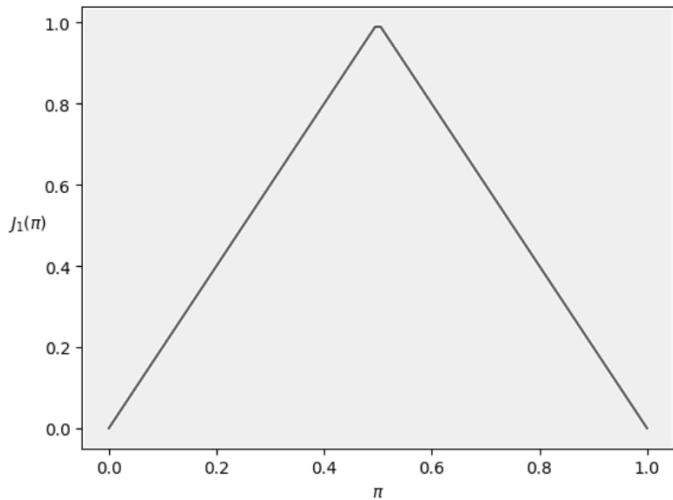
$$J_N(\pi) = 0$$

$$\begin{aligned} J_{N-1}(\pi) &= \min \{c'_{u_1} \pi + J_N(T(\pi, y_1, u_1)) \cdot \sigma_{(\pi, y_1, u_1)} + J_N(T(\pi, y_2, u_1)) \cdot \sigma_{(\pi, y_2, u_1)}, \\ &\quad c'_{u_2} \pi + J_N(T(\pi, y_1, u_2)) \cdot \sigma_{(\pi, y_1, u_2)} + J_N(T(\pi, y_2, u_2)) \cdot \sigma_{(\pi, y_2, u_2)}\} \\ &= \min \{c'_{u_1} \pi, c'_{u_2} \pi\} \end{aligned}$$

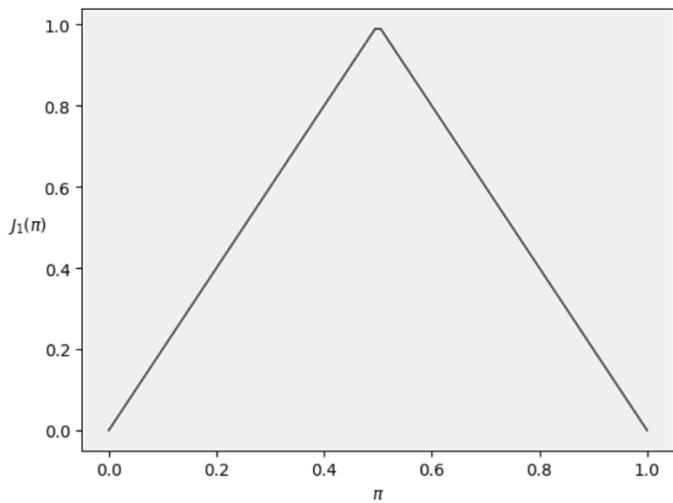
$$\begin{aligned} J_{N-2}(\pi) &= \min \{c'_{u_1} \pi + J_{N-1}(T(\pi, y_1, u_1)) \cdot \sigma_{(\pi, y_1, u_1)} + J_{N-1}(T(\pi, y_2, u_1)) \cdot \sigma_{(\pi, y_2, u_1)}, \\ &\quad c'_{u_2} \pi + J_{N-1}(T(\pi, y_1, u_2)) \cdot \sigma_{(\pi, y_1, u_2)} + J_{N-1}(T(\pi, y_2, u_2)) \cdot \sigma_{(\pi, y_2, u_2)}\} \\ &\dots \end{aligned}$$

Y ahora tenemos en cuenta las transiciones relacionadas con cada acción que puede tomar el agente.

Las primeras iteraciones quedarían:



La gráfica con la evolución quedaría:



Este es el resultado al que se llega.

En el texto de POMDP (Krishnamurthy, V. (2016) [93] - pag. especialmente 151 y también en pag. 222) podemos observar un gráfico en las que aparece un triángulo equilátero inscrito en un primer cuadrante. Como los belief state pertenecen a un simplex de una dimensión menos que el número de estados, esto permite obtener el valor de cada componente a partir del punto representado dentro del triángulo. En un triángulo equilátero los ángulos son de 60º, que es el cálculo que aparece en cada uno de los dos ejes del gráfico.

Resumen - Ideas principales

Esto son un conjunto de ideas resumidas que permiten hacerse una idea de todo aquello que se ha analizado en el capítulo.

1. Las redes neuronales que emplearemos serán fundamentalmente perceptrones.
2. También se emplearán redes recurrentes.
3. Además utilizaremos un tipo especial de redes recurrentes, como son las redes LSTM.
Su uso es muy extendido en este tipo de modelizaciones.

Capítulo 5

Aprendizaje por refuerzo profundo (Deep RL)

Cuando el número de estados y de acciones es reducido, la **representación tabular** de la función de valor ($J(x)$ o $Q(x, u)$) surge de manera natural, tal y como hemos estado utilizando hasta ahora. Sin embargo, cuando el sistema bajo estudio tiene un gran número de estados o acciones, la utilización de matrices es una vía computacionalmente muy costosa. En estas condiciones lo adecuado consiste en optar por una **representación compacta** (ver capítulo 2), empleando aproximadores funcionales.

De manera simplificada nos enfrentamos al problema de construir una función $y = f(x, \theta)$ a partir de unos datos de entrenamiento (x_i, y_i) , de tal manera que dicha función se ajuste a ellos lo mejor posible. En nuestro caso, dada una estrategia μ , los valores podrían ser $(i, J^\mu(i))$ $i \in X$, obtenidos mediante experimentación o simulación.

Este esquema se traduce en encontrar el vector de parámetros θ^* que minimice una cierta función de pérdida (\mathcal{L}) que cuantifica de alguna manera las discrepancias entre lo observado (y) y lo predicho ($f(x, \theta)$). Es, decir:

$$\theta^* = \arg \min_{\theta \in \Theta} \mathcal{L}(\theta)$$

La elección del modelo a utilizar dependerá en gran medida del tipo de problema a tratar, pero el uso de Redes Neuronales es lo más común en la actualidad. Una de las razones reside en que es un aproximador universal (Hornik, et. al (1989) [76]), y la utilización de múltiples capas de neuronas puede ajustar señales muy complejas (Cybenko, G. (1989) [49]). Es por lo que procedemos a abordar este tema en las secciones que siguen.

5.1. Redes Neuronales

La idea principal detrás de las redes neuronales es la de componer recursivamente una función, para generar progresivamente una señal de respuesta cada vez más compleja. Utilizando la siguiente notación (Murphy, K. P. (2022) [120]), sea $f_l(\mathbf{x}) = f(\mathbf{x}, \theta_l)$, donde $l = 1, \dots, L$ la función paramétrica de ajuste sería:

$$f(\mathbf{x}, \boldsymbol{\theta}) = f_L(f_{L-1}(\cdots(f_1(\mathbf{x}))\cdots))$$

La función $f_l(x)$ se denomina **función de activación** y actúa como unidad de procesamiento. Si θ_l son los parámetros de la capa l , llamemos W_l a los pesos que afectan a una neurona en particular. Entonces la función de activación actúa transformando el producto de las entradas (*inputs*) \mathbf{x} por los pesos W_l más una constante arbitraria b . En la figura 5.1 podemos observar algunas de las más habituales.

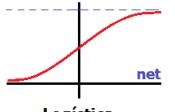
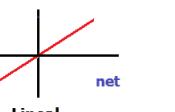
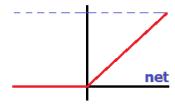
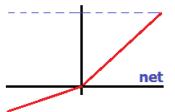
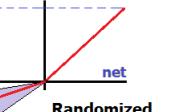
FUNCIÓN de ACTIVACIÓN				
Expresión de la función	$f(\mathbf{x}) = \tanh(\mathbf{x}) = \frac{1 - e^{-\mathbf{x}}}{1 + e^{-\mathbf{x}}}$	$f(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}}$	$f(\mathbf{x}) = \mathbf{x}$	$f(\mathbf{x}) = \begin{cases} 0 & \text{si } \mathbf{x} < 0 \\ 1 & \text{si } \mathbf{x} \geq 0 \end{cases}$
FUNCIÓN de ACTIVACIÓN				
Expresión de la función	$f(\mathbf{x}) = \begin{cases} 0 & \text{si } \mathbf{x} < 0 \\ \mathbf{x} & \text{si } \mathbf{x} \geq 0 \end{cases}$	$f(\mathbf{x}) = \begin{cases} \alpha \mathbf{x} & \text{si } \mathbf{x} < 0 \\ \mathbf{x} & \text{si } \mathbf{x} \geq 0 \end{cases}$ donde $\alpha \leq 1$		

Figura 5.1: Algunas funciones de activación usuales (donde: $net = W \cdot input + b$).

Las interconexiones entre estas unidades de procesamiento quedan apiladas en forma de capas, tal y como refleja la figura 5.2.

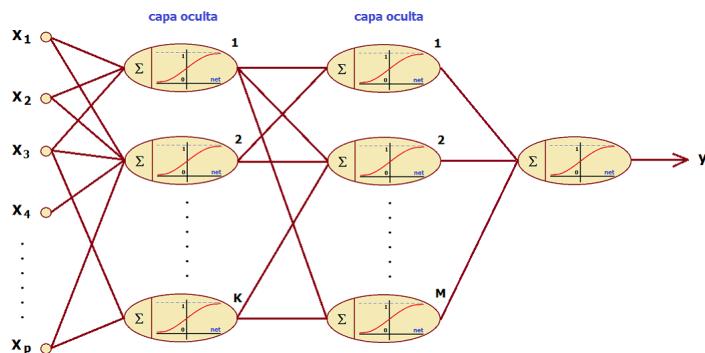


Figura 5.2: Representación gráfica del procesamiento de información que realiza una Red Neuronal. Se han empleado unos inputs de dimensión ' p ': $\mathbf{x} = (x_1, x_2, \dots, x_p)$ y dos capas ocultas de neuronas. La neurona final genera un valor \mathbf{y} . En todos los casos se ha empleando una función de activación tangente hiperbólica.

Tras disponer de una modelo, el paso siguiente consiste ajustar los parámetros para conseguir que la señal que genera la red neuronal produzca los menores errores de predicción, dados los valores respuesta observados. Este proceso recibe el nombre de **entrenamiento**.

5.2. Entrenamiento de una red neuronal

Dados un conjunto de datos muestrales (x_i, y_i) , buscamos ajustar los parámetros de la red neuronal dada por $f(\mathbf{x}, \theta)$. Los parámetros óptimos θ^* son aquellos que hacen que las diferencias $(y_i - f(x_i))$ sean mínimas en algún sentido. La forma de tratar estas diferencias vendrá dada por una función g , que suele consistir en promediar los valores cuadráticos de los errores. Esta función es la llamada **función de pérdida** \mathcal{L} .

Empleando métodos tradicionales de optimización, tenemos que el óptimo se encontrará actualizando los valores de los parámetros en el sentido dado por el mayor gradiente. El proceso esquemáticamente consistiría en:

- 1) Especificación de la función de pérdida :

$$\mathcal{L}(\theta) = g(y - (f(\mathbf{x}, \theta)))$$
- 2) Cálculo del gradiente :

$$\nabla \mathcal{L}(\theta)$$
- 3) Actualización de parámetros de la red:

$$\theta = \theta - \gamma \cdot \nabla \mathcal{L}(\theta)$$

El procedimiento se traduce en un desplazamiento por la curva de error guiados por el valor del gradiente. En la figura 5.3 podemos observar una representación del proceso de optimización.

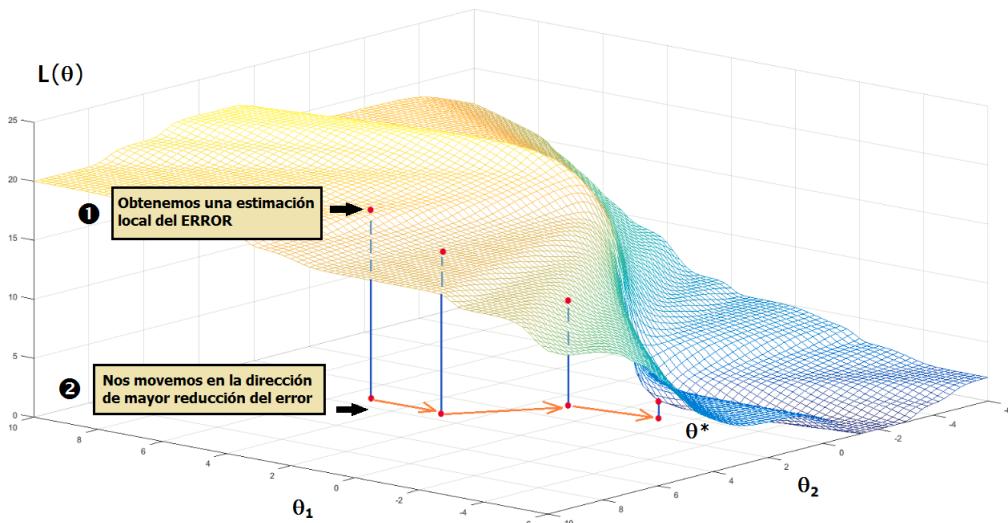


Figura 5.3: Obtención del vector de parámetros óptimo θ^* .

En la medida en que utilizamos el gradiente para avanzar en la actualización de los parámetros de la red, se pueden presentar algunos problemas como la desaparición del gradiente o su crecimiento explosivo que convierte el aprendizaje en muy intangible. Existen métodos que permiten corregir algunas de estas deficiencias y ayudar a que el algoritmo de optimización encuentre el óptimo (ver: Goodfellow, I. et al (2016) [62]).

5.3. Arquitecturas de Redes Neuronales

El origen de las redes neuronales se retrotrae a un artículo de McCulloch, W. & Pitts, W. (1943) [111], en el que los autores plantean un modelo de computación basado en el funcionamiento de las neuronas biológicas. Básicamente se trata de una función umbral que se activa (valor 1) cuando la suma de los inputs por unos ciertos pesos más una constante superan un cierto valor (ver fig. 5.4). Sería algo más tarde cuando se propondría el Perceptrón (Rosenblatt, F. (1958) [153]), como una unidad computacional dotada de aprendizaje, que varía sus parámetros para adaptarse a la información que se le proporciona.

Sus limitaciones en el procesamiento de información, tal como demostraron Minsky, M. & Seymour, P. (1969) [109], hicieron que la comunidad científica perdiera el interés en este tipo de modelos. Sin embargo el descubrimiento, años más tarde, de construcciones más complejas y la forma de entrenarlas óptimamente provocó el resurgimiento de este área de investigación.

En el siguiente esquema se recogen algunas de las principales arquitecturas de Redes Neuronales, y los tipos de datos cada vez más complejos que son capaces de procesar.

- **MLP (Multi-Layer Perceptron).** Este tipo de red se construye conectando neuronas y apilando capas, todas ellas con conexiones exclusivamente hacia adelante (*Feed Forward Network*), y que genera al final una señal de respuesta. Se emplea como un aproximador funcional y se entrena mediante retropropagación de error, tal y como vimos.
Los **autoencoders** son un tipo especial de red multicapa alimentada hacia delante que es entrenada para generar una salida lo más aproximada posible a la entrada que se le proporciona (Bourlard, H. & Kamp, Y. (1988) [33]). Esta red está diseñada limitando la capacidad de las capas ocultas internas, de manera que impida a la red caer en el aprendizaje obvio de reconstruir de manera perfecta la señal de entrada. Esto permite a la red funcionar como una mecanismo para la reducción de las dimensiones de los datos.
- **CNN (Convolutional Neural Network).** Su origen se encuentra en el Neocognitron (Fukushima, K. & Miyake, S. (1982) [59] y su aplicación a detección de patrones Fukushima, K. (1988) [58]), que se basaba en hallazgos sobre el procesamiento de la información visual (Hubel, D. H. & Wiesel, T. N. (1962) [79]). Posteriormente se desarrolló un modo eficiente de aprendizaje por retropropagación del error (no con pesos asignados manualmente) (Le Cun, Y. et al. (1989) [98]). Se utiliza fundamentalmente en el procesamiento de imágenes.
- **RNN (Recurrent Neural Networks).** Este tipo de arquitectura es capaz de procesar datos secuenciales y se debe a Rumelhart, D. E. et al (1985) [154]. Su estructura particular dota a este tipo de red de memoria, ya que los *outputs* en pasos previos vuelven a entrar posteriormente como *inputs*. Esto permite su utilización en áreas que se habían resistido al empleo de redes neuronales tradicionales, como la del procesamiento del lenguaje natural (*NLP - Natural Language Processing*).
Entre las dificultades que se tienen que abordar se encuentra el problema que surge con la desaparición del gradiente o su crecimiento explosivo, que fue detectado por Bengio, Y. et al (1993) [17]. Como consecuencia de ello han surgido variaciones que permiten mantener un flujo de gradiente estable como la red recurrente LSTM (Hochreiter, S. & Schmidhuber, J. (1997) [75]).

Respecto del uso en Aprendizaje por Refuerzo, tenemos que las redes neuronales se han adoptado extensamente como estimadores funcionales (MLP) (ver por ejemplo: Tsitsiklis, J. (1996) [191]), y para la extracción de 'features' (CNN).

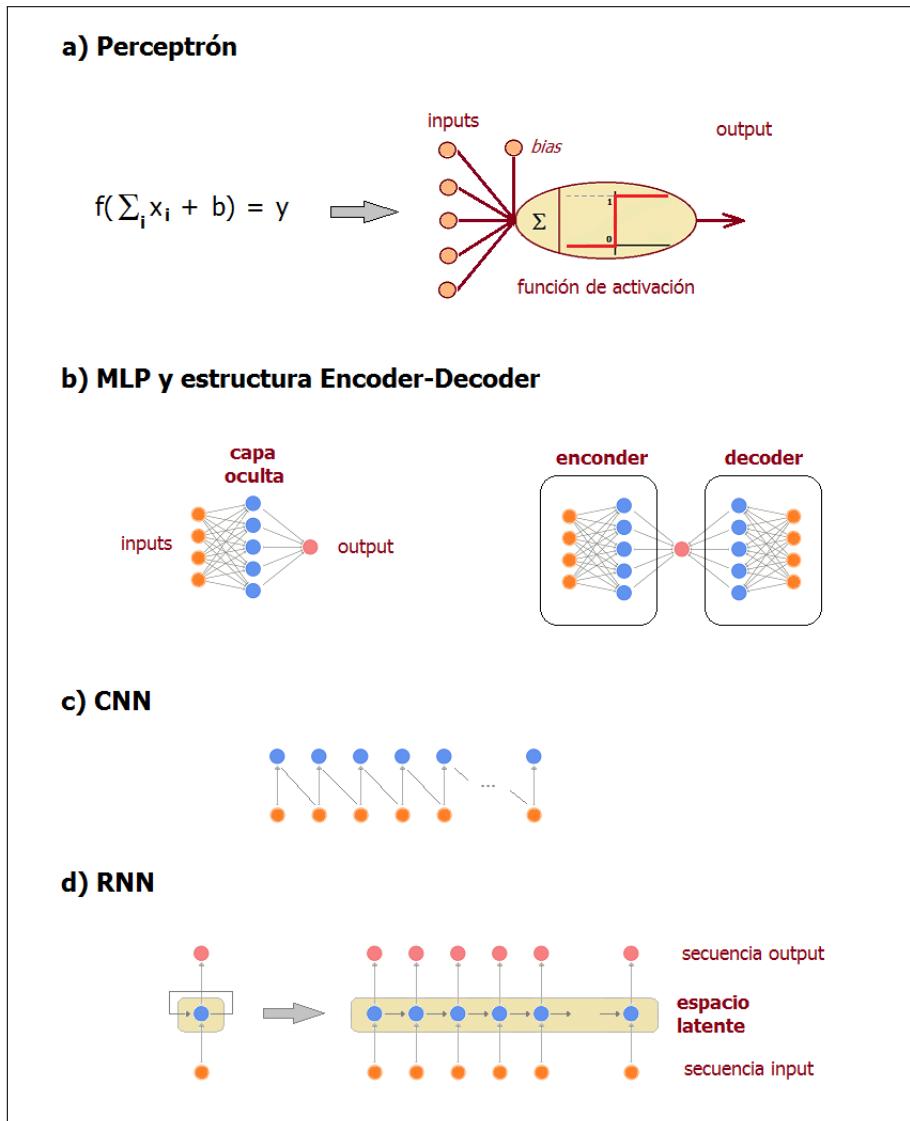


Figura 5.4: Algunas arquitecturas de Redes Neuronales (Esquemas adaptados de Principe, J. C. et al. (1999) [146], Goodfellow, I. et al. (2016) [62] y Brunton, S. L. & Kutz, J. N. (2022) [38]).

A la hora de abordar el estudio del empleo de redes neuronales, conviene clasificar los distintos métodos. Posteriormente distinguiremos entre:

- 1) **Value based methods.** Estimamos la función de valor mediante RN. Generalmente $Q(s,a)$.
- 2) **Policy based methods.** Consiste en parametrizar la función de decisión y en utilizar la red neuronal para estimar dicha función: $\mu(\cdot)$.
- 3) **Actor-Critic methods.** Utilizan una mezcla de los métodos anteriores.

En los siguientes apartados procederemos a presentar algunos de los resultados más notables de aplicación de Redes Neuronales, comenzando primero con la construcción del tipo de función de pérdida que se emplea para el entrenamiento de las redes.

5.4. Utilización de Redes Neuronales en RL

Para construir la función de pérdida $\mathcal{L}(\theta)$ acudimos a la ecuación de Bellman. Para ello se toma en cuenta la diferencia entre el valor estimado del siguiente estado (s') - denominado *target* -, y el valor estimado para el estado actual, que sería el *valor observado*. La función de pérdida se construiría ponderando de alguna forma estos errores, mediante una función g :

$$\mathcal{L}(\theta) = g\left(\underbrace{r + \gamma \cdot \max_{a' \in \mathcal{A}} Q_\theta(s', a')}_{\text{target}} - \underbrace{Q_\theta(s, a)}_{\text{observado}} \right)$$

Como función g , podemos utilizar $g = \mathbb{E}[(\cdot)^2]$, lo que daría lugar al Error Cuadrático Medio (MSE), que es una de las medidas más usuales a utilizar.

5.5. Value based Methods.

Con los algoritmos que analizamos a continuación podemos aproximar, por medio del aprendizaje, una de las funciones de valor que hemos visto. A partir de las valoraciones que proporciona de los pares (s, a) - en el caso de la función Q -, es inmediato el obtener la estrategia óptima.

5.5.1. Deep Q-Network (DQN)

La introducción de Redes Neuronales profundas (Deep Neural Networks) permitió avances significativos en el entrenamiento por refuerzo. El algoritmo DQN es debido a Mnih et al. (2013) [115].

A partir de ahora notaremos $Q_\theta^\pi(s, a)$ a la función de valor estado-acción. Con ello queremos indicar que la red neuronal depende de un vector de parámetros θ .

En la medida en esta red neuronal va a servir para obtener las estimaciones, surge la pregunta acerca de cómo obtener los valores de los parámetros . La solución pasa por emplear un bufer ('replay bufer' o 'experience replay' - notado: \mathcal{D}) que almacene las experiencias que va acumulando el agente. El bufer acumula las transiciones que se han ido generando a lo largo de sucesivos episodios o trayectorias $(\tau_1, \tau_2, \dots, \tau_h)$ (Ver figura 5.5).

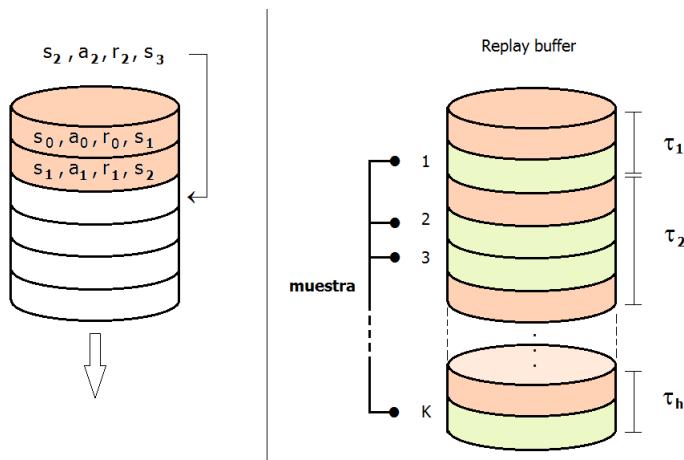


Figura 5.5: Funcionamiento del bufer, y extracción posterior de una muestra para entrenar la Red.

Como las experiencias dentro de un mismo episodio suelen estar altamente correladas , el procedimiento que se emplea es el de seleccionar una muestra aleatoria (*batch* o lote), que será el que se emplee para entrenar la red neuronal. El procedimiento es el habitual: emplear una función de pérdida $L(\theta)$, como puede ser el Error Cuadrático Medio (*Mean Square Error*). El gradiente de dicha función ($\nabla_{\theta}L(\theta)$) nos proporcionará la dirección en la que corregir los parámetros para acercar la salida al objetivo.

$$L(\theta) = MSE = \frac{1}{K} \sum_{i=1}^K \left[r_i + \gamma \cdot \max_{a' \in \mathcal{A}} Q_{\theta_{target}}^{\pi}(s'_i, a') - Q_{\theta}^{\pi}(s_i, a_i) \right]^2 \quad (5.1)$$

$$\theta = \theta - \alpha \cdot \nabla_{\theta}L(\theta) \quad (5.2)$$

Empleando el esquema que hemos venido empleando reiteradamente, los valores estado-acción quedarían, por tanto, actualizados según la siguiente expresión:

$$Q_{\theta}^{\pi}(s, a) \leftarrow \underbrace{Q_{\theta}^{\pi}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a' \in \mathcal{A}} Q_{\theta_{target}}^{\pi}(s', a') - Q_{\theta}^{\pi}(s, a)]}_{\text{target}} \underbrace{-}_{\text{output}} \underbrace{\text{TD(1) error}}_{TD(1) \text{ error}} \quad (5.3)$$

La secuencia de pasos completa quedaría recogida en el algoritmo 2 (incluir referencia).

Algoritmo 7 Deep Q-Network (DQN)

```

begin
    Inicializar la tasa de aprendizaje  $\alpha$ 
    Inicializar el factor de descuento  $\gamma$ 
    /*Llevamos a cabo una serie de episodios o trayectorias (M).
    for EPISODIO = 1 to M do
        /*Muestreamos K valores de bufer
        Recoger K experiencias  $(s_i, a_i, r_i, s'_i, a'_i)$  generadas por la estrategia en vigor  $\pi$ 
        for i = 1 to K do
            if  $(s'_i \text{ es un nodo terminal})$  then
                 $y_i = r_i$ 
            else
                 $y_i = r_i + \gamma \cdot \max_{a' \in \mathcal{A}} Q_{\theta_{target}}^{\pi}(s'_i, a'_i)$ 
            /*Calcular el Error Cuadrático Medio (MSE)
             $L(\theta) = \frac{1}{K} \sum_{t=0}^K [y_i - Q_{\theta}^{\pi}(s_i, a_i)]^2$ 
            /*Actualizar los parámetros de la red
             $\theta = \theta - \alpha \cdot \nabla_{\theta}L(\theta)$ 

```

5.5.2. Double Deep Q Network (DDQN)

Algunos estudios han demostrado una sistemática sobreestimación en el valor de los estados en el caso del proceso de aprendizaje con Q-Learning cuando es combinado con un aproximador funcional como son las Redes Neuronales (Thrun, S. & Schwartz, A. (1993) [188]). La razón reside en la utilización del operador 'max' a la hora de aproximar el valor del siguiente estado en una transición, ya que introduce un sesgo positivo. Además al utilizar los mismos valores tanto para seleccionar una acción como para evaluarla, hace que sea más probable el seleccionar valores sobreestimados, lo que conduce a estimaciones de valor de los estados demasiado optimistas.

Una posible solución consistiría en desacoplar la parte de selección de acciones de la de evaluación, introduciendo un modo alternativo de aproximar el valor esperado máximo (Hasselt, H. (2010) [69]). En este artículo, el autor approxima el valor esperado máximo para cualquier conjunto de variables aleatorias, y posteriormente lo aplica al algoritmo Q-Learning introduciendo un doble estimador (ver también: Hasselt, H. (2011) [70] y Hasselt, H. et al. (2015) [192]).

Esta extensión del modelo DQN recibe el nombre de Double Deep Q-Network (DDQN). La actualización de los pesos obedece a la siguiente expresión (Ecuación 5.4):

Los diferentes términos que intervienen son:

$$Q_{\theta}^{\pi}(s, a) \leftarrow Q_{\theta}^{\pi}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a' \in \mathcal{A}} Q_{\theta'}^{\pi}(s', \arg \max_{a' \in \mathcal{A}} Q(s', a')) - Q_{\theta}^{\pi}(s, a)]$$

$\underbrace{\hspace{100pt}}_{target}$
 $\underbrace{\hspace{100pt}}_{output}$

 $\underbrace{\hspace{200pt}}_{TD(1) \text{ error}}$

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

5.5.3. Deep Q-Network with Prioritized Experience Replay (PER)

Parece razonable centrarse en el conjunto de transiciones para el cual la red genera un error en término de la Diferencia Temporal más grande.

$$\underbrace{r + \gamma \cdot \max_{a' \in \mathcal{A}} Q_{\theta'}^{\pi}(s', a') - Q_{\theta}^{\pi}(s, a)}_{target} - \underbrace{Q_{\theta}^{\pi}(s, a)}_{output}$$

TD(1) error

El algoritmo que procedemos a analizar fue desarrollado por Schaul et al. (2016) [160], y se centra en la cuestión que acabamos de mencionar: utiliza un buffer con una prioridad asignada a cada transición en función del error.

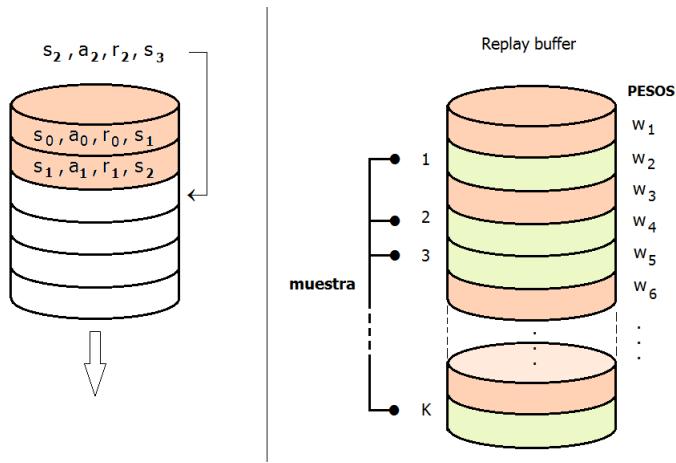


Figura 5.6: Funcionamiento del bufer, y extracción de una muestra ponderada por el efecto que la observación tiene en el error.

Existen dos métodos de priorización de las observaciones de la base de datos. Un primer paso consiste en asignar una valor proporcional al error, y otro emplea la ordenación previa en base al error y asigna posteriormente el rango que ocupa cada observación. Esto genera unos valores p_i

$$\begin{cases} \bullet \text{ Proporcional :} & p_i = | TD \text{ error} | + \epsilon \\ \bullet \text{ Con rango :} & p_i = \frac{1}{rango(i)} \end{cases}$$

con estos p_i procedemos a calcular $P(i)$, donde $\alpha \in [0, 1]$:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Y para evitar el sobreajuste ('overfitting'), se reduce la importancia de transiciones que están siendo muestreadas muchas veces. Esto genera los pesos finales con los que se establece el muestreo.

$$w_i = \left[\frac{1}{N} \cdot \frac{1}{P(i)} \right]^\beta$$

El valor de N representa el tamaño del buffer, $P(i)$ es la probabilidad de la transición a la hora de ser muestreada, y β es un valor que se ajusta mediante *annealing* (empezando por un valor alrededor de 0.4 y haciéndolo tender a 1).

5.5.4. Dueling Deep Q-Network (Dueling DQN) (D2QN)

Si recordamos la definición de Ventaja ('*advantage*'), vemos que nos proporciona una medida de la bondad de una determinada acción a en un estado s en comparación con lo obtenido en promedio con el resto de acciones.

$$A(s, a) = Q(s, a) - V(s)$$

El algoritmo siguiente fue desarrollado por Wang et al. (2016) [197], y hace uso de este valor para calcular la función $Q(s, a)$. Para ello calcula $Q(s, a)$ mediante $V(s) + A(s, a)$.

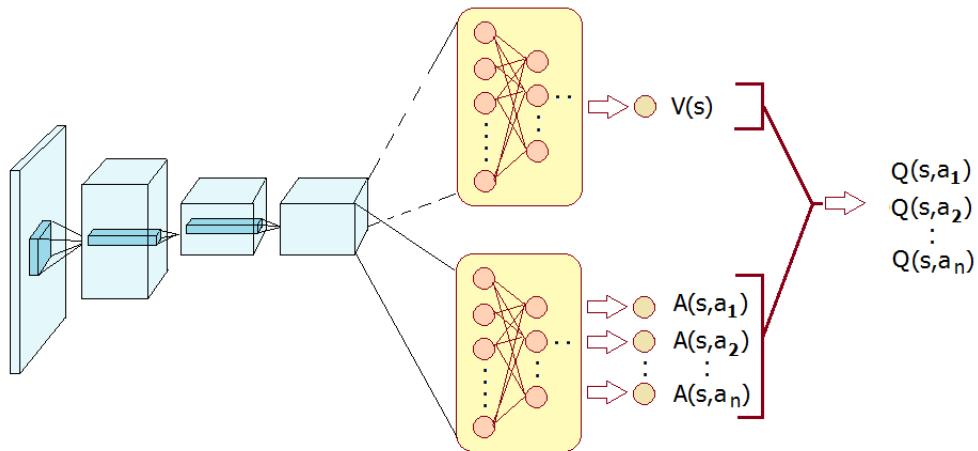


Figura 5.7: Estructura de la Red Neuronal empleada en Dueling DQN.

Se diseña una arquitectura neuronal (que podemos ver en la figura 5.7) cuya salida se bifurca para obtener la función de valor ($V(s)$) por un lado, y por otro lado la ventaja de todas las posibles acciones en ese estado ($A(s, a_i)$). El valor final de la función $Q(s, a)$ se obtiene agregando estos valores generados por la red.

5.5.5. Deep Recurrent QN (DRQN)

El algoritmo fue desarrollado por Hauskchnet et. al (2017) [71].

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

5.6. Policy Methods.

A continuación abordamos el estudio de aquellos algoritmos que permiten el aprendizaje directo de la función de estrategia π . Entre las ventajas se encuentra la convergencia local asegurada a un óptimo local (tal y como demostraron Sutton et al. (2000)) [182].

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

5.6.1. Proximal Policy Optimization (PPO)

El algoritmo fue desarrollado por Schulman et al. (2017) [165].

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

5.6.2. Trust Region Policy Optimization (TRPO)

El algoritmo fue desarrollado por Schulman et al. (2015) [164].

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

5.6.3. Actor-Critic using Kronecker-factored Trust Region (ACKTR)

El algoritmo fue desarrollado por Wu et al. (2017) [202].

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

5.7. Actor-critic Methods

Se han detectado problemas análogos de sobreestimación en este tipo de métodos a los ya mencionados que se producen en Deep Q-Learning, lo que puede conducir al aprendizaje de estrategias subóptimas (Fujimoto, S. et al. (2018) [57]).

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

5.7.1. Advantage Actor-Critic (A2C)

Sobre este tipo de métodos destaca el artículo de Konda y John Tsitsiklis (2000) [89]

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

5.7.2. Asynchronous Advantage Actor-Critic (A3C)

El algoritmo fue desarrollado por Mnih et al. (2016) [113].

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

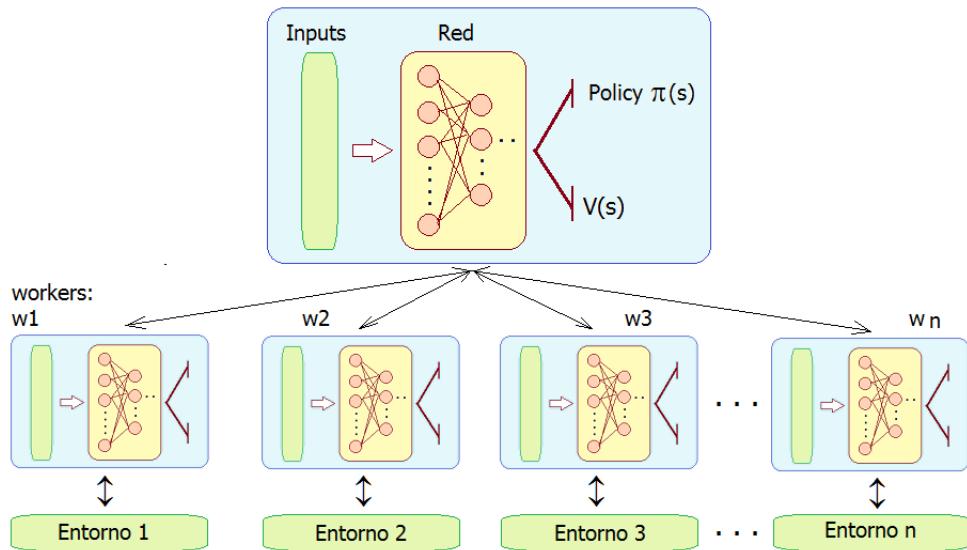


Figura 5.8: Esquema de la implementación del modelo A3C.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

5.7.3. Deep Deterministic Policy Gradient (DDPG)

El algoritmo fue desarrollado por Lillicrap et al. (2016) [103].

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal,

como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

5.7.4. Twin Delayed Deep Deterministic Policy Gradient (TD3)

El algoritmo fue desarrollado por Fujimoto et al. (2018) [57].

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

5.7.5. Soft Actor-Critic (SAC)

El algoritmo fue desarrollado por Haarnoja et al. (2018) [66]. Y más en profundidad sobre este tipo de algoritmos ver también Haarnoja (2018) [67].

En lo que sigue se apuntan algunas de las líneas de investigación que han surgido recientemente y que abordan diferentes cuestiones que permiten mejorar los resultados obtenidos con los métodos tradicionales.

Artículo de McCulloch & Pitts (1943) [111].

Artículo de Rosenblatt, F. (1958) [153].

BENCHMARKS para Multi-agent Deep Reinforcement Learning:

Tenemos algunos entornos que nos permiten testar los algoritmos (Hernández-Leal, P. et al (2019) [73]).

También algunos autores han tratado de llevar a cabo una comparativa en el caso del Deep Reinforcement Learning Multiagente (Papoudakis, G. et al. (2020) [136]).

Se repasan los orígenes de las redes neuronales en las estructuras planteadas en los siguientes artículos: Artículo de McCulloch & Pitts (1943) [111] y Artículo de .

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Resumen - Ideas principales

Esto son un conjunto de ideas resumidas que permiten hacerse una idea de todo aquello que se ha analizado en el capítulo.

1. Las redes neuronales que emplearemos serán fundamentalmente perceptrones.
2. También se emplearán redes recurrentes.
3. Además utilizaremos un tipo especial de redes recurrentes, como son las redes LSTM.
Su uso es muy extendido en este tipo de modelizaciones.

Capítulo 6

Aprendizaje por refuerzo Multiagente (MARL)

Existen situaciones en las que la resolución de una determinada tarea requiere, por su complejidad, coordinación y cooperación simultánea de varios decisores. En otras ocasiones se trata más bien de aprender a tomar decisiones óptimas teniendo en cuenta los efectos que las decisiones de los otros agentes tienen sobre cada uno.

En cualquiera de estos casos nos encontramos ante un problema de toma de decisiones secuenciales en el que existe más de un agente. La solución consiste en extender el planteamiento del Aprendizaje por refuerzo analizado previamente al caso de n decisores.

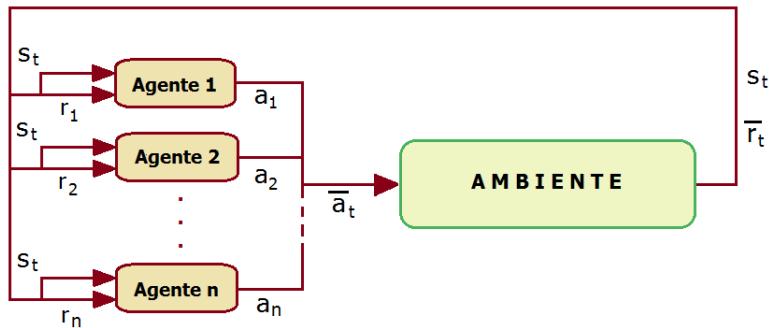


Figura 6.1: *Entorno multiagente. La acción conjunta de todos los agentes (\bar{a}_t), genera una señal por parte del entorno en forma de recompensa (\bar{r}_t) y una alteración de su estado (s_t), lo que sirve a los agentes como refuerzo para optimizar la elección de sus acciones (gráfico adaptado de Nowe et al. (2012) [130]).*

La situación a la que nos enfrentamos queda esquematizada en la figura 6.1. Ahora nos encontramos con un grupo de agentes que comparten un entorno, y que se relacionan entre sí, afectando las decisiones que toman a los resultados que cada uno puede obtener.

En el caso de un sólo agente el objetivo se puede establecer fácilmente a través de la maximización de las recompensas que recibe. Sin embargo, en el caso de múltiples agentes ya no es tan sencillo. Ésto se debe a que los agentes pueden tener intereses encontrados.

6.1. Introducción

En el Aprendizaje por refuerzo multiagente (*Multiagent reinforcement learning*) se han ido generando objetivos divergentes, tal y como han recalcado algunos autores (Soham et al. (2007) [170]). Entre las líneas de investigación algunos trabajos se centran en la búsqueda de equilibrios entre los agentes (enfoque normativo), mientras que otros analizan no tanto cómo deberían actuar los agentes, sino en reproducir la manera en que realmente lo hacen (enfoque descriptivo), por poner un ejemplo.

Por lo tanto antes de comenzar la formalización es necesario aclarar el enfoque que se va adoptar, ya que determina el objetivo que se va a perseguir.

Análisis de la situación. La existencia de varios agentes que se ven forzados a interactuar entre sí, lo que provoca que el curso de acción óptimo dependa de las acciones que llevan a cabo el resto de agentes. Desde este punto de vista el Reinforcement Learning multiagente se adapta al esquema que plantea la **Teoría de Juegos** (von Neumann, J. & Morgenstern, O. - 1944 [119]). Esta disciplina estudia las interacciones estratégicas que se dan entre agentes, que pueden tener intereses enfrentados, a través del empleo de ciertos estereotipos de situaciones (llamadas juegos) que capturan en gran medida el tipo de relaciones que se dan en la vida real. Si atendemos al tipo de relación que se establece entre los jugadores, podemos distinguir entre:

-Juegos Cooperativos. Los jugadores comparten intereses comunes y, por tanto, tienen incentivos para cooperar.

-Juegos Competitivos (o no cooperativos). En este caso los jugadores tienen intereses enfrentados. Podemos distinguir:

- 1) Juegos de suma cero, donde las ganancias de unos suponen las pérdidas de los oponentes.
- 2) Juegos de suma general, donde no se da la relación anterior.

-Juegos Mixtos. En los juegos mixtos existe una mezcla de incentivos de los dos casos anteriores, y pueden aparecer situaciones tanto de cooperación como de enfrentamiento.

El objetivo consiste en encontrar la estrategia óptima, que en esta situación gira en torno a aquella decisión de la que no es posible desviarse unilateralmente para generar una mejor ganancia (*lo que se conoce como Equilibrio de Nash*).

Por otro lado, si el interés se encuentra en cómo los individuos van cambiando sus estrategias con el paso del tiempo, el esquema que mejor se adapta puede ser el que plantea la **Teoría de Juegos Evolutiva**. En este caso consideramos individuos que se replican y el éxito evolutivo hace que unos individuos que adoptan una determinada estrategia se adapten mejor al entorno que otros (Bloembergen, D. et al. (2015) [30]) y sobrevivan. En este caso el concepto de solución óptima está ligado a la estrategia que es inmune a la aparición de estrategias invasoras (*Estrategias Evolutivas Estables*).

Teoría de Juegos

La **Teoría de Juegos** es una rama de las matemáticas y la economía, que surge a partir de los trabajos de John von Neumann y Oskar Morgenstern, y proporciona herramientas para el análisis de decisiones en situaciones en las que aparece el enfrentamiento y la cooperación entre decisores racionales.

Como hemos visto, las situaciones que analiza se denominan juegos y pueden dividirse en:

- | | |
|-----------------|---|
| Tipos de Juegos | $\left\{ \begin{array}{l} \text{- Cooperativos: } \textbf{Mismos intereses} \\ \text{- Competitivos: } \textbf{Intereses enfrentados} \\ \text{- Mixtos} \end{array} \right.$ |
|-----------------|---|



Figura 6.2: **John von Neumann** (izqda.) y **Oskar Morgenstern** (dcha.) (Wikipedia)

En algunas clasificaciones se alude a juegos Cooperativos frente a no Cooperativos, pero en el sentido de que exista la posibilidad de llevar a cabo acuerdos vinculantes antes del juego y que se han de respetar. También podemos estar ante un juego con información perfecta/imperfecta (si el decisor conoce todo lo ocurrido previamente o no), y con información completa/incompleta (si se conocen los cursos de acción, beneficios e intereses del resto de jugadores, o si por el contrario se desconocen algunos elementos del juego).

Algunos conceptos adicionales son:

TIPOS DE ESTRATEGIA: La estrategia es la regla de decisión que emplea el decisor para decidir entre el conjunto de opciones que se le presentan. Podemos distinguir:

-Estrategias puras

Son aquellas que seleccionan una acción de manera determinista.

-Estrategias mixtas

Son aquellas que seleccionan una acción de manera probabilística sobre el conjunto de acciones posibles.

TIPO DE SOLUCIÓN:

-El principal concepto de solución en Teoría de Juegos es el **Equilibrio de Nash** (Nash, J. (1950) [121] y equilibrio en juegos no cooperativos: Nash, J. (1951) [122]), aunque no es el único. Entre otros tenemos el **Equilibrio correlado** (Aumann, R. J. (1987) [2]). También existe el **Equilibrio de Stackelberg** (Björk, P. A. & Vuong, Q. H. (1985) [29]) basado en la teoría de la competencia de Stackelberg, según la cual uno de los agentes tiene una posición dominante por lo que su estrategia determina la de sus competidores.

6.2. Conceptos y formalización

Tal y como hemos visto, un Proceso de Decisión de Markov (MDP) se representa mediante la tupla $(\mathbf{S}, \mathbf{A}, \mathbf{R})$, y queda completamente especificado cuando conocemos cada uno de sus elementos. En el caso multiagente generalizamos dicha definición, introduciendo \mathcal{N} , que representa el conjunto de jugadores. Esto se traduce en que ahora manejamos los siguientes elementos:

-Conjunto de Agentes: $\mathcal{N} = \{1, \dots, n\}$.

-Conjunto de Estados: $\mathcal{S} = \{s_1, \dots, s_m\}$

(En el caso de robots, puede haber una parte que es visible sólo para cada agente. En este caso algunos autores utilizan la notación: $\mathbf{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \dots \times \mathcal{S}_n$ [156]).

-Espacio conjunto de Acciones: $\mathbf{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_n$

-Conjunto de funciones de Recompensa $R_j : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_n \rightarrow \mathbb{R}_{j=1,\dots,n}$

Según el número de agentes 'n', y el número de estados, así como la ausencia de conocimiento sobre el estado del entorno aparecen situaciones diferentes que se formalizan tal y como se esquematiza en la figura 6.3 y se detalla a continuación.

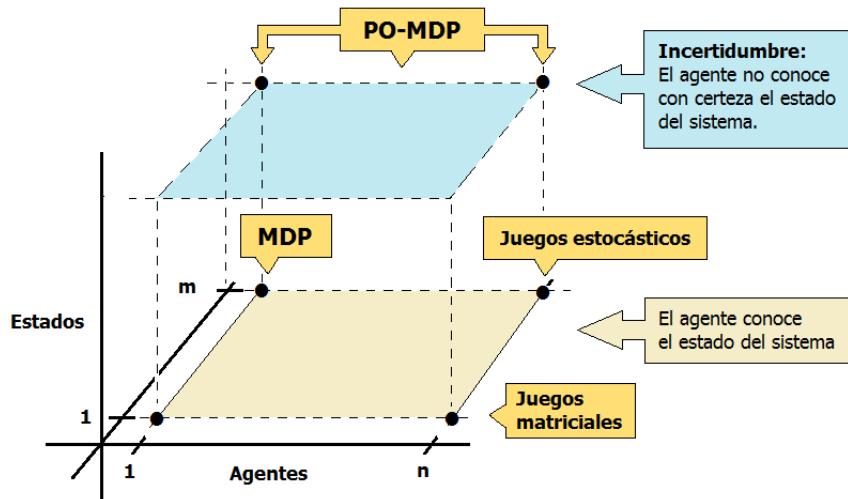


Figura 6.3: Esquema del tipo de modelos según el número de agentes y el tipo de conjunto de estados. Además, cuando el agente no tiene la posibilidad de conocer con precisión el estado del sistema, nos encontramos ante POMDP (Partially Observable Markov Decision Processes).

- **Juegos Matriciales.** (*o juegos estratégicos*). En este caso nos encontramos ante una situación estática, en la que el estado del entorno no juega ningún papel. El número de jugadores es 'n' y cada jugador dispone de un conjunto de acciones disponibles ($\mathcal{A}_i, i = 1, \dots, n$). La recompensa que reciben depende del conjunto de decisiones llevadas a cabo por todos los jugadores: $R_j : \mathbf{A} \rightarrow \mathbb{R}_{j=1,\dots,n}$. El objetivo del agente de maximizar la recompensa recibida, se traduce en encontrar la estrategia óptima que conduce a dicho resultado. Y en la medida en que depende de las decisiones de los demás, la solución pasa por ser la estrategia que es mejor respuesta a las acciones de los demás.

Juegos matriciales clásicos: DILEMAS

Un dilema es un tipo de juego en el que el decisor se enfrenta a una disyuntiva, al estar enfrentados sus intereses con los del otro jugador del que ahora desconocemos sus intenciones. El más conocido de ellos es el llamado Dilema del Prisionero. Su forma actual se debe a Albert Tucker a partir de un experimento psicológico que llevaron a cabo Merrill Flood y Melvin Dresher, dos investigadores de la RAND Corporation (organización encargada de análisis estratégicos) (Poundstone, W. (1993) [143] - capítulo 6).

-Dilema del Prisionero. Dos ladrones son detenidos por la comisión de un delito grave, y la policía no dispone de pruebas, pero sí de indicios que apuntan a su participación. Para esclarecer los hechos, cada uno de ellos es encerrado en una celda sin que exista posibilidad de comunicación. Posteriormente se les plantea por separado un acuerdo.

El dilema al que se enfrenta cada prisionero nace de la naturaleza de la elección a la que se les somete:

- Delatar (por tanto: no cooperar con el compañero). Esto abre la vía de librarse de la acusación y salir libre, recordando sobre el otro la acusación del delito grave.
- No delatar (cooperando con el compañero), lo que implica ser acusado del delito menor que conlleva una pena más suave. Sin embargo el resultado final dependerá de la opción conjunta, ya que la otra persona se encuentra en la misma situación. Si ambos optan por la vía de la no cooperación, y se delatan implicándose mutuamente, serán acusados del delito grave obteniendo la menor satisfacción posible.

El problema reside en que la única opción que permite salir sin cargos es la de Delatar, pero conlleva el riesgo de que el otro, que también se enfrenta a la misma decisión, siga el mismo razonamiento y su decisión termine por llevar a los dos al peor escenario.

		J₂	
		No Cooperar	Cooperar
		(P , P)	(T , S)
J₁	No Coop.	(S , T)	(R , R)
	Coop.		

Matriz de Pagos: (J₁, J₂)

R: Reward (*recompensa*)
T: Temptation (*tentación*)
S: Sucker (*ingenuo, pardillo*)
P: Punishment (*pena*)

La matriz de pagos refleja la estructura de incentivos del juego: la tentación de traicionar (T), la recompensa de la colaboración mutua (R), el castigo por el enfrentamiento (P) o la temida situación de actuar de pardillo (S) y acabar pagando por los dos (Para un análisis de las tensiones de intereses ver: Mady, M. W. & Flache, A. (2002) [108]).

-Juego del Gallina. Éste es otro dilema que refleja el tipo de conflictos que aparecen en las relaciones entre individuos. El juego se basa en el reto entre 2 jugadores, en una carrera de automóviles hacia un precipicio (o uno hacia el otro). El primero que abandona y decide esquivar el choque, pierde y es el 'gallina'. El otro jugador puede frenar y queda como ganador. Pero si ninguno se aparta y siguen hasta el final (esperando en vano que el otro se aparte en el último momento), ambos se estrellan. En el caso de que los dos jugadores decidan apartarse, conseguirán salir ilesos, pero ninguno puede humillar al otro.

La forma usual de representar la interacción entre los agentes, es una matriz en la que se enfrentan sus acciones y se recoge el pago que recibe cada uno de ellos en cada posible decisión (ver figura 6.4).

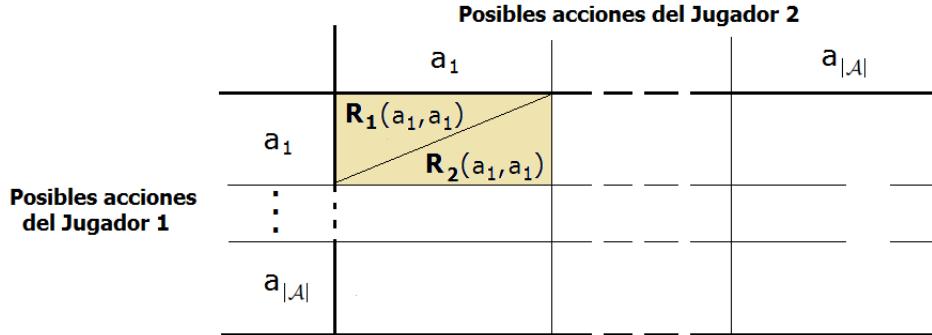


Figura 6.4: Esquema de la matriz de pagos en un juego matricial.

- **Juegos Estocásticos** (*o juegos de Markov*). En un juego estocástico existen 'n' jugadores, que toman decisiones de entre un conjunto de acciones disponibles (\mathcal{A}_i , $i = 1, \dots, n$), por lo tanto son una extensión a múltiples agentes de un proceso de Markov (MDP).

A diferencia de los juegos matriciales, ahora las decisiones que toman los jugadores pueden alterar el estado del sistema, provocando una transición dentro del conjunto de estados (\mathcal{S}). Ahora la función de recompensa adoptará por tanto la forma $R_j : \mathcal{S} \times \mathbf{A} \rightarrow \mathbb{R}_{j=1, \dots, n}$, donde: $\mathbf{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_n$.

En este caso el agente ha de encontrar la estrategia, que proporciona la acción optima en cada estado del sistema. Y 'resolver el juego' se traduce en determinar la estrategia conjunta óptima.

- **Procesos de Decisión de Markov Parcialmente Observables**. Si se permite que exista incertidumbre sobre el estado (debido a la falta de conocimiento del agente) nos encontramos ante un Proceso de Decisiones de Markov Parcialmente Observable (*Partially Observable Markov Decision Process - POMDP* [117]).

En este caso, nos encontramos entonces ante una generalización del Proceso de Decisión de Markov: puede haber desde uno a varios agentes, y donde ahora el agente no puede percibir el entorno con precisión. La vía para modelizar el problema consiste en manejar dos procesos. Por una lado está el estado del sistema $\mathcal{S} = \{s_1, \dots, s_m\}$, que el agente no puede conocer con certeza. Y por otro lado tenemos están las observaciones que construye el agente con la información limitada de que dispone $\mathcal{O} = \{o_1, \dots, o_r\}$.

Algún enfoque alternativo en: "Rollout, Policy Iteration and Distributed Reinforcement Learning" Bertsekas, D. (2021) [23].

6.3. Concepto de solución en MARL: Equilibrio de Nash

Suponemos que en el instante 't' todos los agentes toman una decisión simultáneamente, recibiendo cada uno de ellos una recompensa: $(r_t^j, j=1,...,n)$. La decisión vendrá dada a partir de su estrategia o policy $(\pi^j : \mathcal{S} \rightarrow \Omega(\mathcal{A}^j))$. La estrategia conjunta de todos los agentes vendrá dada por:

$$\boldsymbol{\pi} = [\pi^1, \dots, \pi^n]$$

Entonces, dado un estado inicial 's', la función de valor de un agente 'j' bajo la estrategia conjunta $\boldsymbol{\pi}$, vendrá dada por la siguiente expresión:

$$v_{\pi}^j = \sum_{t=0}^{\infty} \gamma^t \cdot \mathbb{E}_{\pi}[r_t^j | S_0 = s; \pi]$$

y la función de valor estado-acción:

$$Q_{\pi}^j = r^j(s, a) + \gamma \cdot \mathbb{E}_{s \sim \pi}[v_{\pi}^j(s')]$$

Equilibrio de Nash. En lo que sigue analizamos el objetivo que aparece cuando varios agentes interactúan entre sí. Generalmente buscamos una '*policy*' óptima que conduzca a maximizar la función de valor del agente. Como optimizar la función: v_{π}^j del agente 'j' depende de la estrategia conjunta del resto de agentes más él mismo, se acude al concepto de equilibrio de Nash.

DEFINICIÓN 6.1: Equilibrio de Nash

Dado un conjunto de jugadores $\mathcal{N} = \{1, \dots, n\}$, el **equilibrio de Nash** alude a aquella estrategia conjunta de la forma:

$$\boldsymbol{\pi}_* = [\pi_*^1, \pi_*^2, \dots, \pi_*^n]$$

tal que: $\forall s \in \mathcal{S}, j \in \{1, 2, \dots, n\}$ y todas las π^j válidas se cumple:

$$v_{\pi}^j(s, \pi_*) = v^j(s, \pi_*^j, \pi_*^{-j}) \geq v^j(s, \pi^j, \pi_*^{-j})$$

donde π_*^{-j} es la estrategia del resto de agentes:

$$\pi_*^{-j} = [\pi_*^1, \dots, \pi_*^{j-1}, \pi_*^{j+1}, \dots, \pi_*^n]$$

Vemos que cada agente reacciona con la mejor respuesta al resto de jugadores. Hu y Wellman (2003) [78] definieron un procedimiento iterativo para el cálculo del valor que buscamos (Nash Policy).

El proceso cuando el número de agentes crece se va haciendo progresivamente más difícil de realizar. Para evitar este problema algunos autores (Yang et al. (2018)) [204] han propuesto el Mean Field Multi-agent Reinforcement Learning.

EJEMPLO 6.1: Cálculo del Equilibrio de Nash

Imaginemos que nos encontramos ante el problema de determinar la secuencia de decisiones óptimas en una serie finita de etapas ($K=0,1,2,3$). La siguiente figura representa un sencillo caso. X_0, X_1, X_2, X_3 son el estado del sistema en cada etapa. Sobre cada arco aparece el coste asociado a la transición en cuestión. El objetivo consiste en llegar desde X_0 al final con el mínimo coste acumulado.

		J2		
		Delatar	Callar	
		Delatar	(3, 3)	(0, 5)
J1	Callar	(5, 0)	(1, 1)	

Ahora bien si el jugador 1 decide optar por callar tenemos que la matriz de pagos es la que aparece a continuación.

		J2		
		Delatar	Callar	
		Delatar	(3, 3)	(0, 5)
J1	Callar	(5, 0)	(1, 1)	

Tomando ahora el análisis para el jugador número 2 tendremos el siguiente resultado. Aparece la estrategia que le reporta el mejor resultado ante las decisiones del otro jugador.

		J2		
		Delatar	Callar	
		Delatar	(3, 3)	(0, 5)
J1	Callar	(5, 0)	(1, 1)	

		J2		
		Delatar	Callar	
		Delatar	(3, 3)	(0, 5)
J1	Callar	(5, 0)	(1, 1)	

y finalmente la siguiente figura. En ella podemos ver enmarcada la celda donde el cruce de estrategias conduce a un punto en el que a ninguno le conviene desviarse de es estrategia.

		J2		
		Delatar	Callar	
		Eq. Nash	(3, 3)	(0, 5)
J1	Callar	(5, 0)	(1, 1)	

De donde tenemos que el resultado es que el **Equilibrio de Nash** consiste en el delatarse mutuamente.

6.4. Clasificación de los algoritmos de solución I: según el tipo de entrenamiento

El aprendizaje multiagente se enfrenta a retos adicionales, como es el hecho de que el entorno aparezca como no estacionario [134]. Se han abierto múltiples líneas de investigación, tal y como muestran algunos autores (Oroojlooy, A. & Hajinezhad, D. (2021) [133]). El principal problema al que nos enfrentamos es la coordinación y cooperación de los agentes. Siguiendo a Gupta, J. K. et al. (2017) [65], podemos señalar algunas de las soluciones que se han propuesto.

- **Entrenamiento independiente.**

La forma más simple de entrenamiento de un agente en el caso MARL, consiste en que cada uno de ellos lleva a cabo el proceso independientemente de los demás, tratando al resto de los agentes como un elemento del entorno. El problema como ya se ha mencionado previamente consiste en que éste se vuelve no estacionario, lo que provoca la pérdida de garantías sobre la convergencia de muchos algoritmos. (Algunos estudios plantean una medida capaz de reflejar este efecto: Lanctot, M. et al. (2017) [96]).

En el caso de llevar a cabo el aprendizaje no en el espacio de acciones individual , sino en el espacio de acciones conjunto (*joint action learners*); esto supone observar las acciones de los otros y actuar óptimamente respecto de la estrategia que se está estimando que guía el comportamiento de los otros agentes. El método ayuda a generar la cooperación entre ellos, aunque computacionalmente puede ser muy costoso conforme crece el número de agentes.

- **Entrenamiento centralizado y ejecución descentralizada.**

Estudios como el de Tan, M. [185] (1993) muestran los beneficios de la compartición de información durante el entrenamiento, lo que puede acelerar el aprendizaje. Este hecho sugiere la utilización de diversas formas de centralizar el aprendizaje para, por ejemplo, acelerar la aparición de la cooperación. Con esta vía cada agente toma decisiones por su cuenta, aunque se produce cierta coordinación en el entrenamiento.

La vía más extrema consiste en que sólo una unidad es entrenable con la información que se obtiene de todos los agentes, y posteriormente se generan decisiones para cada uno de ellos. El entrenamiento centralizado supone la existencia de un modelo conjunto tanto para las acciones como para las observaciones de los agentes. El problema que plantea el centralizar tanto la ejecución como el entrenamiento reside en el crecimiento exponencial del espacio de observaciones y acciones conforme crece el número de agentes.

- **Aprendizaje concurrente.**

En aprendizaje concurrente se permite el aprendizaje independiente de la función de decisión de cada decisor. En los métodos basados en el gradiente, se emplea la señal de recompensa conjunta para la optimización de las estrategias. Sin embargo se incurre nuevamente en el peligro que ya se ha señalado, y es el de la inestabilidad del aprendizaje.

- **Compartición de parámetros.**

En este caso las experiencias de todos los agentes se emplean para el entrenamiento. Si la representación de la función de estrategia está parametrizada, como por ejemplo a través de una Red Neuronal, los agentes compartirían los parámetros. Este enfoque es adecuado cuando los agentes son muy homogéneos entre sí.

6.5. Clasificación de los algoritmos de solución II: según el tipo de interacción

Siguiendo la clasificación de los métodos de aprendizaje por Refuerzo multiagente (MARL) dada por Busoniu, L. et al. (2018) [39]), tenemos que podemos distinguir:

- **Cooperativo.** En este caso los agentes comparten la misma función de recompensa.

- **Estático** No existe distinción entre estados.

El algoritmo *Joint Action Learners* (JAL) fue desarrollado por Claus, C. & Boutilier, C. (1998) [44], en contraposición a agentes independientes (IL - *Independent Learners*) que aprenden ignorando la presencia de otros agentes. Ahora el aprendizaje se lleva a cabo en el espacio de acciones conjuntas.

No existen garantías sobre la convergencia de dicho algoritmo, por lo que se ha desarrollado una heurística: *Frecuency Maximum Q-Value* (Kapetanakis, S & Kudenko, D. (2004) [85]) con el objetivo de subsanar esta deficiencia.

- **Dinámico** Existen transiciones entre estados.

En este contexto se han desarrollado diversos algoritmos que implican la formación de grupos. Destaca *Team Q-Learning* propuesto por Littman, M. L. (2001) [106], *Distributed Q-Learning* (Lauer, M. & Riedmiller, M. (2000) [97]) que busca una estrategia óptima para el equipo en su conjunto; así como *Optimal Adatpive Learning* (OAL) (Wang, X. & Sandholm, T. (2002) [196]) que está diseñado para converger a NE en un juego por equipos y que además sea el óptimo.

En este tipo de sistemas la complejidad suele crecer rápidamente conforme lo hace el número de agentes, por lo que otros autores han propuesto centrar la coordinación en unos pocos estados y actuar independientemente en el resto. El algoritmo *Sparse Cooperative Q-Learning* se basa en esta idea (Kok, J. R. & Vlassis, N (2004) [88]).

- **Competitivo.** En el caso completamente competitivo tenemos que las ganancias de uno suponen las pérdidas de otro. De entre los algoritmos para este tipo de situación, y en el caso de 2 únicos agentes, tenemos el algoritmo *minimax Q-Learning*. Este algoritmo fue desarrollado por Littman, M. L. (2001) [106] y (1994) [105].

También habría que destacar HAMRL - *Heuristically Accelerated Multi-agent Reinforcement Learning* (Bianchi, R. A. et al. (2013) [28]).

- **Mixto.** En este caso no se impone ninguna restricción sobre la función de recompensa de cada jugador y el tipo de relación con los demás. Los algoritmos desarrollados para este tipo de situaciones suelen estar muy influidos por la Teoría de Juegos.

- **Estático**

En entornos multiagente es deseable un aprendizaje óptimo ante oponentes estacionarios, así como la convergencia a un equilibrio de Nash (NE) cuando todos emplean el mismo aprendizaje. El algoritmo AWESOME - *Adapt When EveryBody is Stationary, Otherwise Move to Equilibrium* (Conitzer,V.& Sandholm, T. (2003) [46]) ofrece buenos resultados, y es apropiado cuando existen más de dos agentes. Requiere que sean visibles las acciones de los otros.

El juego ficticio (*Fictitious play*) es una algoritmo desarrollado por Brown, G. W. (1951) [37], y permite alcanzar equilibrio en un juego coordinado.

Como alternativa, empleando métodos bayesianos, tenemos el algoritmo *Hyper-Q* desarrollado por Tesauro, G. (2003) [186]. Converge a la estrategia óptima si los oponentes emplean estrategias estacionarias.

Como alternativa a los métodos anteriores tenemos algunos algoritmos que emplean métodos basados en el gradiente para la búsqueda directa de una estrategia óptima. Algunos los veremos en detalle a continuación, y destacan entre los principales:

El algoritmo IGA (*Infinitesimal Gradient Ascent*) fue desarrollado por Singh, S. et al. (2000) [176], y una modificación del mismo: WoLF-IGA (Bowling, M. & Veloso, M. (2002) [35]) que emplea tasas de aprendizaje variable.

El algoritmo GIGA (*Generalized Infinitesimal Gradient Ascent*) fue desarrollado por Zinkevich, M. (2003 [209]) para el tratamiento de juegos con más de 2 acciones.

El algoritmo GIGA-WoLF fue desarrollado por Bowling, M. (2005 [34]).

- Dinámico

Debido a que se pierde la garantía de convergencia, el algoritmo Q-Learning no se suele aplicar directamente en entornos multiagente. Sin embargo algunos autores han obtenido resultados satisfactorios (*Single-agent Reinforcement Learning* - Sen, S. et al. (1994) [167]).

En lo que sigue se repasan modelos expresamente diseñados para el caso multi-agente. En concreto, los siguientes casos dependen del concepto de equilibrio de Nash (NE) de Teoría de Juegos (también existe *Asymmetric Q-Learning* (Könönen, V. (2004) [90]) que emplea el equilibrio de Stackeberg).

El algoritmo *Nash Q-Learning* (Hu, J. & Wellman, M. P. (1998) [77]) es la extensión del algoritmo Minimax Q-Learning de juegos estocásticos de suma cero al caso de suma general. Está diseñado para buscar la acción conjunta óptima (NE).

El algoritmo *Correlated Q-Learning* (Greenwald, A. et al. (2003) [64]) es aplicable igualmente al caso de suma general. Y *Friend or Foe Q-Learning* (Littman, M. L., (2001) [104]) ofrece mejoras en la convergencia y se emplean 2 tipos de equilibrio: equilibrio adversario.

Por otro lado tenemos un conjunto de algoritmos que cuya solución es independiente del concepto de equilibrio. El algoritmo WoLF-PHC (*Win or Learn Fast Policy Hill Climbing*) fue desarrollado por Bowling, M. & Veloso, M. (2002) [35]. Utiliza un procedimiento basado en el gradiente (PCH - *Policy Hill Climbing*) y adapta Q-Learning para el empleo de estrategias mixtas. La utilización del principio WoLF (*Win or Learn Fast*) permite introducir una tasa de aprendizaje variable.

El algoritmo PD-WoLF (*Policy Dynamic-based Win or Learn Fast*) desarrollado por Banerjee, B. & Peng, J. (2003) [7] emplea idéntico principio WoLF.

El algoritmo NSPC (*Non-Stationary Converging Policies*) (Weinberg, M. & Rosenschein, J. S. (2004) [199]) se centra en el problema de la no estacionaridad.

Finalmente, el algoritmo EXORL (*Extended Optimal Response Learning*) - Suematsu, N. & Hayashi, A. (2002) [179]) es apto para juegos de suma general.

El benchmarking puede ser complicado. Algunos autores han trabajado en la creación de plataformas que permiten la comparación de resultados de los distintos algoritmos (Papoudakis, et.

al. (2020) [135]).

La perspectiva de Teoría de Juegos puede ser desde el punto de vista de la dinámica que se genera. Nos interesa no tanto como se distribuyen las estrategias, sino cómo evolucionan. Estudio en este sentido tenemos el trabajo de Bloemberguen, D. et al. (2015 [30]).

Algunos otros criterios para clasificar los algoritmos los tenemos en: Canese, L. et al. (2021) [41].

Tipo de tarea:	Juego Estático	Juego Dinámico
COOPERATIVA	<p>Existe un objetivo común, que es compartido por todos los jugadores</p> <p>Los principales algoritmos son:</p> <ul style="list-style-type: none"> ▪ Joint-Action Learners (JAL) [44] ▪ Frecuency MQ-Value (FMQ) [85] 	<p>Los principales algoritmos son:</p> <ul style="list-style-type: none"> ▪ Team-Q [106] ▪ Distributed-Q [97] ▪ Opt. Adaptive L. (OAL) [196] ▪ Sparse CQ-L (SCQL) [88] ▪ FMR Q-L. (FMRQL) [208]
COMPETITIVA	<p>Existen intereses enfrentados entre los jugadores o entre grupos de ellos</p> <p>Los algoritmos que aparecen a continuación no distinguen entre Juego estático o dinámico. Ambos sirven para dos jugadores.</p> <ul style="list-style-type: none"> ▪ Minimax-Q (Littman (1994)) [105] ▪ Heuristically accelerated multi-agent (HAMRL) [28] 	
MIXTA	<p>Pueden aparecer tanto situaciones de cooperación como de enfrentamiento</p> <p>Los principales algoritmos son:</p> <ul style="list-style-type: none"> ▪ AWESOME [46] ▪ Hyper-Q [186] ▪ Fictitious play [37] ▪ IGA [176] ▪ WoLF-IGA [35] ▪ GIGA [209] ▪ GIGA-WoLF [34] 	<p>Los principales algoritmos son:</p> <ul style="list-style-type: none"> ▪ Single-agent RL [167] ▪ Nash-Q [77] ▪ Correlated Eq.-Q [64] ▪ WoLF-PHC [35] ▪ PD-WoLF [7] ▪ NSPC [199] ▪ EXORL [179]

Cuadro 6.1: *Principales algoritmos clásicos multiagente*

6.6. Algoritmos mixtos clásicos

Como hemos analizado en la clasificación de los algoritmos multiagente, este tipo de algoritmos se utilizan cuando los agentes pueden llevar a cabo estrategias cooperativas como de enfrentamiento. En lo que sigue se proporciona una colección de los principales algoritmos, y que fueron enunciados previamente.

6.6.1. Algoritmo IGA (Intinitesimal Gradient Ascent)

Este algoritmo fue desarrollado por Singh et al. (2000) [176]. Se puede aplicar fácilmente al caso de 2 jugadores y dos acciones. En la figura 6.5 podemos observar el esquema de pagos y las probabilidades asociadas que constituyen la estrategia de cada uno de ellos.

$$\begin{array}{c} \text{J}_2 \\ \downarrow \\ \overbrace{\quad\quad\quad}^{\beta \quad 1 - \beta} \\ \text{J}_1 \rightarrow \left\{ \begin{array}{l} \alpha \quad \begin{bmatrix} r_{11}, c_{11} & r_{12}, c_{12} \\ r_{21}, c_{21} & r_{22}, c_{22} \end{bmatrix} \\ 1 - \alpha \end{array} \right. \end{array}$$

Figura 6.5: Esquema con la matriz de pagos de un juego matricial (r_{ij}, c_{ij}) , y las probabilidades de cada estrategia $(\alpha, 1 - \alpha; \beta, 1 - \beta)$ por jugador (J_1, J_2 : jugador fila/columna).

A continuación desarrollamos alguno de los algoritmos para varios jugadores (Schwartz, H. M. (2014) pag. 50).

$$\begin{aligned} V_r(\alpha, \beta) &= \alpha \cdot \beta \cdot r_{11} + \alpha \cdot (1 - \beta) \cdot r_{12} + (1 - \alpha) \cdot \beta \cdot r_{21} + (1 - \alpha) \cdot (1 - \beta) \cdot r_{22} \\ &= + u_r \cdot \alpha \cdot \beta + \alpha \cdot (r_{12} - r_{22}) + \beta \cdot (r_{21} - r_{22}) + r_{22} \\ V_c(\alpha, \beta) &= \alpha \cdot \beta \cdot c_{11} + \alpha \cdot (1 - \beta) \cdot c_{12} + (1 - \alpha) \cdot \beta \cdot c_{21} + (1 - \alpha) \cdot (1 - \beta) \cdot c_{22} \\ &= + u_c \cdot \alpha \cdot \beta + \alpha \cdot (c_{12} - c_{22}) + \beta \cdot (c_{21} - c_{22}) + c_{22} \end{aligned}$$

donde:

$$\begin{aligned} u_r &= r_{11} - r_{12} - r_{21} + r_{22} \\ u_c &= c_{11} - c_{12} - c_{21} + c_{22} \end{aligned}$$

Y tras derivar tenemos:

$$\begin{aligned} \frac{\partial V_r(\alpha, \beta)}{\partial \alpha} &= \beta \cdot u_r + (r_{12} - r_{22}) \\ \frac{\partial V_c(\alpha, \beta)}{\partial \beta} &= \alpha \cdot u_c + (c_{21} - c_{22}) \end{aligned}$$

el algoritmo consiste en actualizar la estrategia aplicando la siguiente regla:

$$\begin{aligned} \alpha_{k+1} &= \alpha_k + \eta \cdot \frac{\partial V_r(\alpha_k, \beta_k)}{\partial \alpha_k} \\ \beta_{k+1} &= \beta_k + \eta \cdot \frac{\partial V_c(\alpha_k, \beta_k)}{\partial \beta_k} \end{aligned}$$

La versión de este algoritmo WoLF-IGA, utiliza el procedimiento WoLF (*Win or Learn Fast*) que consiste básicamente en introducir una tasa de aprendizaje variable, con el objetivo de que éste se vuelva más lento cuando se está ganando y, por el contrario acelerar el aprendizaje cuando se está perdiendo. Vemos que la estrategia se actualiza con la información que proporciona el gradiente junto con esta tasa variable l (donde $l \in [l_{min}, l_{max}]$):

$$\begin{aligned}\alpha_{k+1} &= \alpha_k + \eta \cdot l_k^r \cdot \frac{\partial V_r(\alpha_k, \beta_k)}{\partial \alpha_k} \\ \beta_{k+1} &= \beta_k + \eta \cdot l_k^r \cdot \frac{\partial V_c(\alpha_k, \beta_k)}{\partial \beta_k}\end{aligned}$$

En cuanto a los resultados que ofrece, tenemos que converge al equilibrio de Nash (NE) en estrategias mixtas o estrategias puras en juegos matriciales de suma general (2 jugadores y dos estrategias). La principal dificultad para su implementación reside en que es necesario conocer tanto la estrategia de los oponentes como los pagos que reciben estos, por lo que no emplea aprendizaje descentralizado.

6.6.2. Algoritmo PHC (Policy Hill Climbing)

Fue desarrollado igualmente por Bowling & Veloso (2002) [35] y está basado en el algoritmo Q-Learning. En cuanto a los resultados que ofrece, tenemos que converge a la solución óptima cuando el resto de jugadores emplean una estrategia estacionaria fija. Si el resto de jugadores se encuentran aprendiendo igualmente, entonces la convergencia no está asegurada.

La regla para actualizar el valor de cada acción (desde la iteración t a $t + 1$) para el jugador j , donde α es la tasa de aprendizaje y γ el factor de descuento, se resume en:

$$Q_{t+1}^j(s, a) = (1 - \alpha) \cdot Q_t^j(s, a) + \alpha \cdot \left[r^j + \gamma \cdot \max_{a'} Q_t^j(s, a') \right]$$

La acción de cada jugador se selecciona según la estrategia $\pi_t^j(s, a)$, y su valor se va actualizando en una pequeña cantidad Δ_a :

$$\pi_{t+1}^j(s, a) = \pi_t^j(s, a) + \Delta_a$$

donde:

$$\Delta_a = \begin{cases} \delta & a = \arg \max_{a'} Q_t^j(s, a') \\ -\frac{\delta}{|A_j|-1} & a \neq \arg \max_{a'} Q_t^j(s, a') \end{cases}$$

Es decir la regla significa

Regla de actualización 5. 1 (Policy Hill Climbing (PHC)):

$$Q_{t+1}^j(s, a) \longleftarrow (1 - \alpha) \cdot Q_t^j(s, a) + \alpha \cdot \left[r^j + \gamma \cdot \max_{a'} Q_t^j(s, a') \right]$$

Algoritmo 8 Policy Hill Climbing (PHC)

```

begin
    Inicializar las tasas de aprendizaje  $\alpha \in (0, 1]$ ,  $\delta \in (0, 1]$ 
    Inicializar el factor de descuento  $\gamma \in (0, 1)$ 
    Inicializar  $Q_i(s, a_i) = 0$  y  $\pi_i(s, a_i) = \frac{1}{|\mathcal{A}_i|}$ 
    for iteracion do
        Observar:  $s$ 
        Tomar decisión en base a la estrategia actual:  $a_c$ 
        Observar recompensa y nuevo estado:  $r_i, s'$ 
        Actualizar  $Q_i(s, a_c)$  y  $\pi_i(s, a_i)$ :
        
$$Q_{t+1}^j(s, a) = (1 - \alpha) \cdot Q_t^j(s, a) + \alpha \cdot \left[ r^j + \gamma \cdot \max_{a'} Q_t^j(s, a') \right]$$

        
$$\pi_i(s, a_i) = \pi_i(s, a_i) + \Delta_{sa_i}$$


```

6.6.3. Algoritmo WoLF-PHC

Este algoritmo fue introducido por Bowling & Veloso (2002) [35], para juegos matriciales entre dos jugadores con dos tipos de acciones.

$$Q_{t+1}^j(s, a) = (1 - \alpha) \cdot Q_t^j(s, a) + \alpha \cdot \left[r^j + \gamma \cdot \max_{a'} Q_t^j(s, a') \right]$$

con:

$$\pi_{t+1}^j(s, a) = \pi_t^j(s, a) + \Delta_a$$

donde:

$$\Delta_a = \begin{cases} \delta & a = \arg \max_{a'} Q_t^j(s, a') \\ -\frac{\delta}{|A_j| - 1} & a \neq \arg \max_{a'} Q_t^j(s, a') \end{cases}$$

El valor de δ alterna entre dos valores δ_w (*winning*) y δ_l (*lossing*), dependiendo de si el jugador se encuentra ganando o perdiendo respectivamente. Estos valores cumplen: $\delta_l > \delta_w$, y permiten una adaptación más rápida de la estrategia en el caso de que los resultados del jugador frente al oponente no estén siendo favorables. El criterio que se toma para determinar la posición que se tiene en el juego consiste en comparar el valor esperado actual (bajo la estrategia π) frente a lo que consigue la estrategia promedio ($\bar{\pi}$). Es decir el jugador está en buena posición si:

$$\underbrace{\sum_{a'} \pi_t(a') \cdot Q_{t+1}^j(a')}_{\text{Valor esperado actual}} > \underbrace{\sum_{a'} \bar{\pi}_{t+1}(a') \cdot Q_{t+1}^j(a')}_{\text{Valor esperado de } \bar{\pi}}$$

lo que llevaría a utilizar δ_w . En resumidas cuentas la regla de selección de la tasa de aprendizaje queda:

$$\delta = \begin{cases} \delta_w & \sum_{a'} \pi_t(a') \cdot Q_{t+1}^j(a') > \sum_{a'} \bar{\pi}_{t+1}(a') \cdot Q_{t+1}^j(a') \\ \delta_l & \sum_{a'} \pi_t(a') \cdot Q_{t+1}^j(a') \leq \sum_{a'} \bar{\pi}_{t+1}(a') \cdot Q_{t+1}^j(a') \end{cases}$$

donde la estrategia promedio $\bar{\pi}$ se obtiene como:

$$\bar{\pi}_{t+1}^j(s, a') = \bar{\pi}_t^j(s, a') + \frac{1}{C_{t+1}(s)} \cdot [\pi_t^j(s, a') - \bar{\pi}_t^j(s, a')] \quad \forall a' \in \mathcal{A}_j$$

y donde se actualiza un valor contador: $C_{t+1}(s) = C_t(s) + 1$.

Algoritmo 9 WoLF-PHC

```

begin
    Inicializar las tasas de aprendizaje  $\alpha$ ,  $\delta$ 
    Inicializar el factor de descuento  $\gamma$ 
    Inicializar  $Q_i(s, a_i) = 0$  y  $\pi_i(s, a_i) = \frac{1}{|\mathcal{A}_i|}$ 
    for iteracion do
        Observar: s
        Tomar decisión en base a la estrategia actual:  $a_c$ 
        Observar recompensa y nuevo estado:  $r_i$ ,  $s'$ 
        Actualizar  $Q_i(s, a_c)$  y  $\pi_i(s, a_i)$ :
        
$$Q_{t+1}^j(s, a) = (1 - \alpha) \cdot Q_t^j(s, a) + \alpha \cdot [r^j + \gamma \cdot \max_{a'} Q_t^j(s, a')]$$

        
$$\pi_i(s, a_i) = \pi_i(s, a_i) + \Delta_{sa_i}$$

    
```

Regla de actualización 5. 2 (Win or Learn Fast (WoLF-PHC)):

$$Q_{t+1}^j(s, a) \leftarrow Q_t^j(s, a) = (1 - \alpha) \cdot Q_t^j(s, a) + \alpha \cdot [r^j + \gamma \cdot \max_{a'} Q_t^j(s, a')]$$

Algoritmo WoLF-PHC. Este algoritmo fue introducido por Bowling & Veloso (2002) [35], para juegos matriciales entre dos jugadores con dos tipos de acciones. Empleamos el procedimiento WoLF para lograr la convergencia al **Equilibrio de Nash** en self-play. Recordemos que plantear una tasa de aprendizaje variable implica:

$$Q_{t+1}^j = (1 - \alpha) \cdot Q_t^j(a) + \alpha \cdot \left[r^j + \gamma \cdot \max_{a'} Q_t^j(a') \right]$$

con:

$$\pi_{t+1}^j(a) = \pi_t^j(a) + \Delta_a$$

donde:

$$\Delta_a = \begin{cases} -\delta_a & a = \arg \max_{a'} Q_t^j(a') \\ \sum_{a' \neq a} \delta_{a'} & a \neq \arg \max_{a'} Q_t^j(a') \end{cases}$$

donde:

$$\delta_a = \min \left(\pi_t^j(a), \frac{\delta}{|A_j| - 1} \right)$$

con:

$$\delta = \begin{cases} \delta_w & \sum_{a'} \pi_t(a') \cdot Q_{t+1}^j(a') > \sum_{a'} \bar{\pi}_{t+1}(a') \cdot Q_{t+1}^j(a') \\ \delta_l & \sum_{a'} \pi_t(a') \cdot Q_{t+1}^j(a') \leq \sum_{a'} \bar{\pi}_{t+1}(a') \cdot Q_{t+1}^j(a') \end{cases}$$

y donde:

$$\bar{\pi}_{t+1}^j(a') = \bar{\pi}_t^j(a) + \frac{1}{C_{t+1}} \cdot [\pi_t^j(a') - \bar{\pi}_t^j(a')] \quad \forall a' \in \mathcal{A}_j$$

y donde se actualiza un valor contador: $C_{t+1} = C_t + 1$.

Algoritmo 10 WoLF-IGA

```

begin
    Inicializar las tasas de aprendizaje  $\alpha$ ,  $\delta$ 
    Inicializar el factor de descuento  $\gamma$ 
    Inicializar  $Q_i(s, a_i) = 0$  y  $\pi_i(s, a_i) = \frac{1}{|\mathcal{A}_i|}$ 
    for iteracion do
        Observar: s
        Tomar decisión en base a la estrategia actual:  $a_c$ 
        Observar recompensa y nuevo estado:  $r_i$ ,  $s'$ 
        Actualizar  $Q_i(s, a_c)$  y  $\pi_i(s, a_i)$ :
        
$$Q_{t+1}^j = (1 - \alpha) \cdot Q_t^j(a) + \alpha \cdot \left[ r^j + \gamma \cdot \max_{a'} Q_t^j(a') \right]$$

        
$$\pi_i(s, a_i) = \pi_i(s, a_i) + \Delta_{sa_i}$$

    
```

Regla de actualización 5. 3 (Win or Learn Fast (WoLF-IGA)):

$$Q_{t+1}^j \leftarrow Q_t^j = (1 - \alpha) \cdot Q_t^j(a) + \alpha \cdot \left[r^j + \gamma \cdot \max_{a'} Q_t^j(a') \right]$$

Algoritmo Minimax-Q El algoritmo propuesto por Littman (1994) [105] se detalla a continuación.

$$V_i^*(s) = \max_{\pi_i(s,\cdot)} \min_{a_{-i} \in \mathcal{A}_{-i}} \sum_{a_i \in \mathcal{A}_i} Q_i^*(s, a_i, a_{-i}) \cdot \pi_i(s, a_i) \quad i = 1, 2$$

Algoritmo Friend or Foe Q-Learning El algoritmo propuesto por Littman, M. L. (2001) [106].

Algoritmo 11 Friend or Foe Q-Learning

begin

 Fijar α (tasa de aprendizaje) y γ (factor de descuento)
 Sea (a_1, \dots, a_{n_1}) las acciones del jugador i y sus amigos
 Sea (o_1, \dots, o_{n_2}) las acciones de los oponentes del jugador i
 Inicializar $V_i(s)$ y $Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2})$

for $iteracion$ **do**

 Observar: s

 Tomar decisión el jugador i : a_i

 Observar recompensa y nuevo estado de amigos y oponentes: r_i, s'

 Actualizar $Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2})$:

$$Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2}) = (1 - \alpha) \cdot Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2}) + \alpha \cdot [r_i + \gamma \cdot V_i(s')]$$

 Actualizar mediante programación dinámica $V_i(S)$

$$V_i(s) = \max_{\pi_i(s,\cdot), \dots, \pi_{n_1}(s,\cdot)} \min_{o_1, \dots, o_{n_2}} \sum_{a_1, \dots, a_{n_1} \in \mathcal{A}_1 \times \dots \times \mathcal{A}_{n_1}} Q_i(s, a_1, \dots, a_{n_1}, o_1, \dots, o_{n_2}) \cdot \\ \cdot \pi_i(s, a_1) \cdot \dots \cdot \pi_{n_1}(s, a_{n_1})$$

Estudios sobre algoritmos de aprendizaje en casos con información imperfecta, tenemos para el caso de suma cero Lakshmivarahan, S. & Narendra, K. S. (1981) [95]. También han tratado el tema Sastry, P. S. et al (1994) [159].

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

6.7. Algoritmos mixtos recientes

En los apartados anteriores se han analizado los principales algoritmos clásicos en los que los agentes interactúan tanto cooperando como enfrentándose entre sí. El principal problema que surge en muchos de ellos es que sólo es aplicable en el caso de un número reducido de agentes. Algunas técnicas más recientes intentan salvar este obstáculo.

6.7.1. Mean Field Q-Learning

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

$$\begin{aligned} Q^j(s, \mathbf{a}) &= \frac{1}{N^j} \sum_k Q^j(s, a^j, a^k) \approx Q^j(s, a^j, \bar{a}^j) \\ Q^j_{t+1}(s, a^j, \bar{a}^j) &= (1 - \alpha) \cdot Q^j_t(s, a^j, \bar{a}^j) + \alpha \cdot [r^j + \gamma v_t^j(s')] \\ v_t^j(s') &= \sum_{a^j} \pi_t^j(a^j | s, \bar{a}^j) \cdot \mathbb{E}_{\bar{a}^j(a^{-j}) \sim \pi_t^{-j}} [Q^j_t(s', a^j, \bar{a}^j)] \end{aligned}$$

TEOREMA 6.1: Convergencia de la regla de actualización MF-Q

Dado un juego estocástico con espacio de estados finito, los valores Q_t calculados con la regla de actualización MF-Q (Mean Field Q-Learning) convergen a los Q-valores de Nash $Q_* = [Q_*^1, \dots, Q_*^N]$ si se cumplen los siguientes supuestos.

- a) **Supuesto 1.** Cada par estado-acción es visitado con probabilidad 1 (i.o. - *infinitely often*) y las recompensas están acotadas por una cierta constante K .
- b) **Supuesto 2.** La estrategia del agente es 'Greedy' en el límite con exploración infinita (GLIE - *Greedy in the Limit with Infinite Exploration*).

Tenemos que en el caso de la estrategia de Boltzmann, se convierte en greedy con respecto a la función Q en el límite, según la temperatura (β) decrece asintóticamente a cero. La estrategia de Boltzmann (Boltzmann policy) adopta la siguiente expresión:

$$\pi_t^j(a^j | s, \bar{a}^j) = \frac{e^{\beta \cdot Q_t^j(s, a^j, \bar{a}^j)}}{\sum_{a^{j'} \in \mathcal{A}^j} e^{\beta \cdot Q_t^j(s, a^{j'}, \bar{a}^j)}}$$

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal,

como cuando se incluye todo el texto en el archivo principal.

Regla de actualización 5. 4 (Mean Field Q-Learning):

$$Q^j(s, a^j, \bar{a}^j) \leftarrow Q^j(s, a^j, \bar{a}^j) + \alpha \cdot [r^j + \gamma \cdot v^j(s') - Q^j(s, a^j, \bar{a}^j)]$$

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

El algoritmo *L_{R-I} Lagging Anchor* aborda algunos de los puntos débiles de los algoritmos basados en el gradiente (Schwartz, H. M. (1994) [166]).

Respecto a algunos otros algoritmos como Team-Q podemos encontrar en algunos textos el detalle de su funcionamiento (Konar, A. & Sadhu, A. K. (2020) [156]).

$$\begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}$$

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$\alpha \quad 1 - \alpha \quad \beta \quad 1 - \beta$$

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir

introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

En la literatura es normal encontrar el planteamiento anterior adaptado para trabajar dentro del marco propuesto por Littman (1994) [105], que considera el problema multiagente como un Juego de Markov.

ARTÍCULOS:

De entre los primeros análisis destaca el realizado sobre el aprendizaje por refuerzo y el Dilema del Prisionero iterado (Sandholm, T. W. & Crites R. H. (1996) [157]). La aportación consiste en que, a diferencia de análisis previos, aborda la situación en la que los pagos no están ni totalmente correlados positivamente (team problems) ni negativamente correlados (juegos de suma cero). Presenta el algoritmo Q-Learning como adecuado para su empleo con juegos repetidos contra un oponente desconocido, utilizando en este caso el Dilema del Prisionero Iterado. Como señala el autor: los supuestos que garantizan la convergencia del algoritmo Q-Learning no se cumplen, ya que el aprendizaje simultáneo de todos los agentes, convierte el ambiente en no estacionario. En su artículo analiza diversos tipos de oponente (jugadores Tit-for-Tat, o PAVLOV por ejemplo). Emplea una Red Neuronal Recurrente para mantener información de las jugadas pasadas.

Lo que sigue sería el capítulo dedicado a Aprendizaje por Refuerzo Multi-agente. La creciente aparición de artículos y algunas monografías sobre el tema refleja el interés por el tema. Algunos de los reviews destacan como el de (Nguyen et al. 2018) [125] (2020 [126]). O libros como Swartz (2014) [166] o Sadhu (2020) [156].

Resumen - Ideas principales

Esto son un conjunto de ideas resumidas que permiten hacerse una idea de todo aquello que se ha analizado en el capítulo.

1. Las redes neuronales que emplearemos serán fundamentalmente perceptrones.
2. También se emplearán redes recurrentes.
3. Además utilizaremos un tipo especial de redes recurrentes, como son las redes LSTM.
Su uso es muy extendido en este tipo de modelizaciones.

Capítulo 7

Diseño de un agente dotado de emociones

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Trabajos previos estudiando la evolución de la cooperación generalmente utilizan la disposición de los agentes en cuadrícula (tanto si se estudia evolutivamente como con los mismos agentes). Algunos otros han estudiado la evolución de la cooperación en una red (Ohtsuki, H. et al. (2006) [132] y Santos, F. C. & Pacheco, J. M. (2006) [158]).

7.1. Teorías sobre emociones

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

7.2. Influencia de las emociones en el flujo de información

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Modelos de Difusión de Información

-Independent Cascade Model.

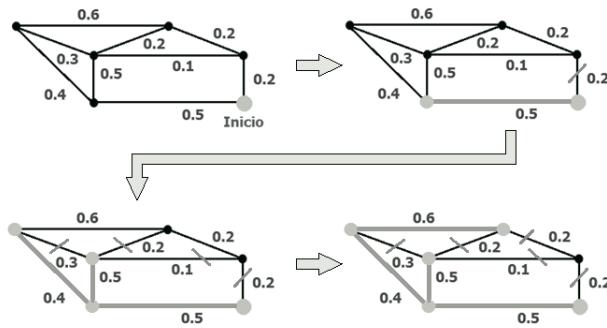
-Threshold Model.

(O quizás poner aquí algo de teoría, y luego en un cuadro de ejemplo poner estas figuras en blanco y negro)

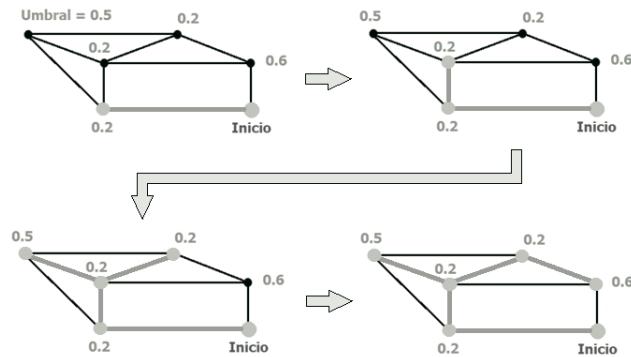
Algunos ejemplos permitirán aclarar el funcionamiento de los diversos modelos para la difusión de información.

EJEMPLO 7.1: Modelos de Difusión de Información

Imaginemos que nos encontramos ante el problema de determinar la secuencia de decisiones óptimas en una serie finita de etapas. La siguiente figura representa un sencillo caso.



La forma de plantear el problema desde la perspectiva de la Programación Dinámica consiste en dividir el problema principal en problemas más sencillos que impliquen un sólo paso o etapa y resolverlo olvidándose del resto del problema. Tendríamos un sistema de ecuaciones para el ejemplo que aparece a continuación.



La resolución del sistema de ecuaciones nos proporciona una solución, que es el valor óptimo de cada estado cuando se sigue la estrategia en cuestión.

7.3. Agente emocional

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

-Artículo de Robert Plutchik sobre las emociones, donde viene la rueda con las principales de ellas. Puedo coger la referencia de las 4 que pretendo enfrentar (Plutchik, R. (2001) [142]).

Algunos artículos de Anna Bazzan y Rafael Bordini:

-Sobre el diseño de agentes con emociones, y la realización de experimentos con el Dilema del Prisionero Iterado (Bazzan, A. & Bordini, R. H. (2001) [9]) y sobre la evolución de los agentes si los dotamos de sentimientos morales (Bazzan, A. & Bordini, R. H. (2002) [10]).

-Mencionan en uno de los artículos un análisis del dilema del prisionero y las emociones, desde la perspectiva de la Teoría de Juegos (O'Neill, B. (2000) [131]).

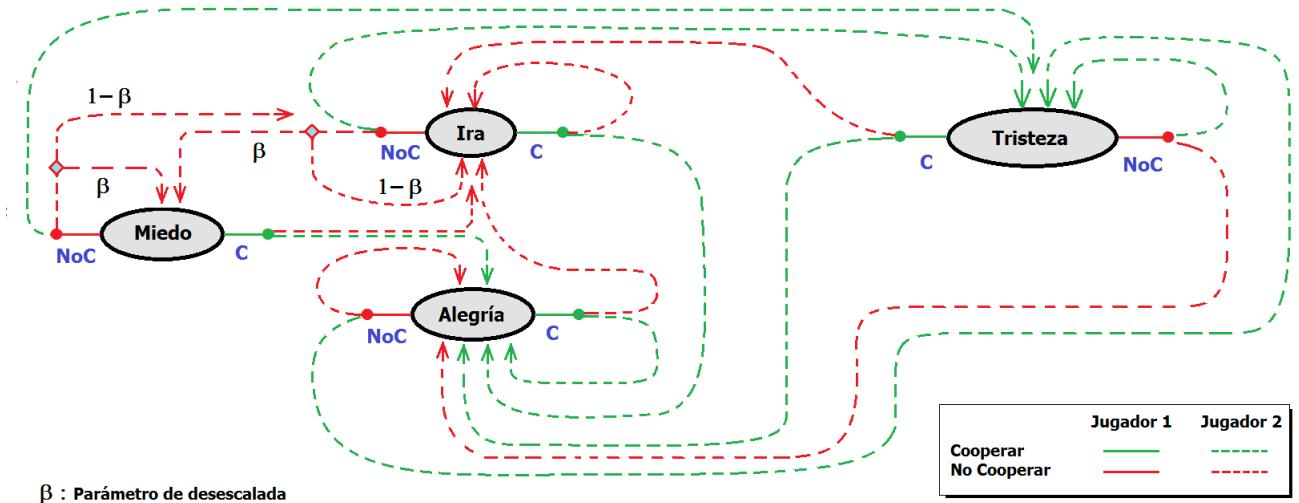


Figura 7.1: Prueba de modelo de las transiciones emocionales de un agente.

7.4. Integración en el esquema de RL

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Lo habitual consiste en añadir un componente a la función de recompensa que permite añadir el efecto de las emociones. En la literatura generalmente se trata de maximizar las emociones positivas y minimizar las negativas (survey: Moerland et al. (2018) [116]).

7.5. Representación del conocimiento

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir

introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

7.6. Transferencia de conocimiento y *feature representation*

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Construcción de una función de recompensa.

Algoritmos tradicionales de RL muestran un rendimiento deficiente en ciertas situaciones (Mataric, M. J. (1994) [110]). El autor propone una metodología para la construcción de una función de recompensa más adecuada, que permita la aceleración del aprendizaje. En este mismo sentido (Ng, Andrew et al. (1999) [124]) se ha estudiado el tipo de modificaciones en la recompensa que conservan la optimalidad de la mejor estrategia en el problema de partida.

Respecto a la naturaleza de la recompensa podemos distinguir entre recompensa intrínseca y la recompensa externa que se recibe del entorno. Respecto a la primera el trabajo pionero establece el marco de trabajo para profundizar en un agente guiado por motivaciones internas (Singh, S. et al. (2009) [173]). Los mismos autores ahondan (Singh, S. et al. (2010) [174]) en la naturaleza evolutiva de las recompensas intrínsecas.

Sobre el aprendizaje en casos no markovianos (POMDP) tenemos el trabajo de Singh, S. et al. (1994) [175].

Convergencia a la Cooperación en Dilemas Iterados

Los estudios de la difusión de la cooperación en una sociedad son abundantes. Se han empleado diversos mecanismos para promover la aparición de este tipo de estrategias. Podemos resaltar

los estudios clásicos sobre reciprocidad directa (Trivers, R. (1971) [190], Axelrod, R. (1984) [4], Nowak, M. & Sigmund, K. (1992) [129], Nowak, M. & May, R. M. (1992) [128]). Estudios más recientes incorporan el tipo de red que interconecta a los individuos (Ohtsuki, H. et al. (2006) [132], Santos, F. C. & Pacheco, J. M. (2006) [158]).

En cuanto al empleo de redes neuronales entrenadas con retropropagación destaca el trabajo de Leibo, J. L. et al. (2017) [100]. En concreto trabajan con varios tipos de dilemas diseñados específicamente para el estudio de equilibrios en Dilemas Sociales Secuenciales. Aquí el agente se enfrenta a un entorno que sólo puede observar parcialmente. Emplean Deep Q-Networks. El entrenamiento es por lotes, por lo que de cara a diseñar una red social no sea el más apropiado.

Sobre este mismo diseño tenemos la adhesión como forma de promover la cooperación (Yuan, Y. et al. (2022) [206]).

El algoritmo Q-Learning ha sido estudiado en multi-agente. Pese a que las condiciones que garantizan la convergencia del algoritmo no se cumplen, se ha aplicado con éxito en entornos sencillos. Si tratamos de su aplicación en juegos matriciales algunos estudios revelan la dificultad para generar la cooperación (Wunder, M. et al. (2010) [203]). Estudios más generales tenemos el de Bloembergen, D. et al. (2015) [30].

Empleo de recompensas intrínsecas en Dilemas Iterados

El empleo de Teoría de Juegos y, en concreto, el Dilema del Prisionero como mecanismo de interacción entre agentes ha impulsado el estudio de los mecanismos mediante los cuales se promueve la estrategia cooperativa. El empleo de recompensas intrínsecas es uno de dichos mecanismos.

Uno de los trabajos es el de Eccles, T. et al. (2019) [51] que emplea la reciprocidad como recompensa intrínseca. En este estudio coexisten dos tipos de agentes: innovadores e imitadores. Ambos son entrenados mediante A3C (algoritmo *Asynchronous Advantage Actor-Critic*) con una red neuronal profunda. Los primeros aprenden del entorno y los imitadores se ven influidos por el nivel de sociabilidad de los innovadores.

Entre los mecanismos empleados para impulsar la cooperación destacan: la reciprocidad, el cumplimiento de normas, las emociones o el efecto de la red.

Empleo de emociones como recompensa intrínseca

Estudios que emplean emociones en los agentes para jugar al IPD tenemos el de Bazzan, A. et al. (2001) [9], basado en el modelo de emociones de OCC. Es básicamente un modelo evolutivo. Toma como referencia el trabajo de Nowak, M. A. & May, R. M. (1992) [128] y cada celda se sustituye por la que mejores resultados obtuvo entre las vecinas implementando un sistema emocional basado en reglas. Dicho trabajo es expandido posteriormente (Bazzan, A. et al. (2002) [10]) para dotar a algunos de los agentes de sentimientos morales hacia agentes del mismo grupo social. Los agentes altruistas, que inicialmente no tienen un buen desempeño, acaban mejorando sustancialmente.

Empleando idéntico modelo como referencia tenemos el trabajo de Sequeira, P. et al (2011) [168], en el que utilizan 4 dimensiones de evaluación de elementos del entorno (reward features)

que combinan linealmente optimizando los pesos para estudiar la adaptación óptima al entorno. Posteriormente llevan a cabo un estudio más profundo sobre el tipo de diseño de la recompensa intrínseca (Sequeira, P. et al. (2014) [169]).

Algunas observaciones últimas a incorporar:

Distinguimos primero entre si es una teoría de Juegos evolutivos (Ver por ejemplo Perc, M & Szolnoki, A. (2010) [137] o también: Szabo, G. & Fath, G. (2007) [183] sobre los juegos evolutivos en grafos).

Sobre los problemas de coordinación en el aprendizaje, por ejemplo en los métodos que emplean Q-Learning (Fulda, N. & Ventura, D. (2007) [60]).

Sobre el equilibrio correlado (Hines, G. & Larson, K (2012) [74]).

Sobre la detección del tipo de oponentes a los que uno se enfrenta en juegos repetidos (Hernandez-Leal, P. & Kaisers, M (2017) [72]).

Sobre un aprendizaje más rápido de los agentes (Elidrisi, M. et al. (2014) [52]).

Algoritmo Pepper para juegos matriciales (Crandall, J. W. (2012) [48]).

Sobre la diversidad social y la promoción de la cooperación en juegos en el dilema del Prisionero espacial (Perc, M. & Szolnoki, A. (2008) [138]).

Efecto de las emociones en la evolución de la cooperación en dilemas sociales, mediante unos agentes con orientación social y otros más egoistas, empleo de emociones surgidas de la reciprocidad (Chen, W. et al. (2021) [43]).

Efecto de las emociones en dilemas espaciales con participación voluntaria (Wang, L. et al. (2018) [194]).

Hacia la cooperación en PD secuenciales con el enfoque Deep RL multiagente (Wang, W. et al. (2018) [195]).

Sobre la imitación de emociones, del perfil emocional, en lugar de estrategias como vía de elevar el bienestar social en juegos espaciales (Szolnoki, A. et al. (2011) [184]).

Imitación de las emociones en IPD con extorsión (Quan, J. et al. (2021) [148]).

Se estudia el aprendizaje del surgimiento de la cooperación utilizando 2 capas de aprendizaje que dotan al agente de capacidades cognitivas y emocionales (Yu, C. et al. (2015) [205]). Se utilizan diversos algoritmos, y los agentes interactúan en diversos espacios: un grid cuadricular y diferentes redes sencillas.

7.6.1. Representación de la función de valor.

La formulación del problema tal y como se ha presentado en el apartado anterior (ver fig. 21) permite visualizar intuitivamente el proceso de toma de decisiones secuenciales; que en un caso de minimización, se traduce en el proceso de búsqueda del camino con menor coste entre dos puntos. A continuación veremos que operando adecuadamente sobre la expresión de la función de valor podemos plantear una estructura recursiva que nos permite plantear un algoritmo para este tipo de problemas y facilita la obtención de la solución óptima.

Para ello trataremos los siguientes puntos:

- Expresión recurrente de la función de valor.
- Implementación computacional.
- Solución exacta vs solución aproximada.

Expresión recurrente de la función de valor. Si partimos de la expresión general de la función de valor, en el caso no determinista:

$$J_\pi(x_0) = \lim_{N \rightarrow \infty} \mathbb{E} \left[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \underbrace{\mu_k(x_k)}_{u_k}, \omega_k) \right]$$

tomando $g_N(x_N) = 0$ para simplificar, podemos extraer el valor de la primera transición y dejar agrupados el resto de términos de la siguiente manera:

$$J_\pi(x_0) = \lim_{N \rightarrow \infty} \mathbb{E} \left[\underbrace{g(x_0, u_0, w_0)}_{\text{valor transición 1}} + \underbrace{\sum_{k=1}^{N-1} g_k(x_k, u_k, \omega_k)}_{\text{resto de transiciones}} \right]$$

Esta expresión sugiere una forma de abordar la búsqueda de las acciones que conducen al valor óptimo: dividir el problema en optimizaciones a un paso. Primero manejamos explícitamente el coste asociado a la primera transición y el estado destino queda etiquetado con el valor del resto de transiciones que quedan por delante.

Si suponemos para simplificar la notación que nos encontramos en un caso discreto, y estacionario, donde en cada estado disponemos del mismo conjunto de decisiones, tendremos lo siguiente: un conjunto de estados $X = \{1, 2, \dots, i, \dots, j, \dots, n\}$ y un espacio de acciones $U = \{1, 2, \dots, s\}$, tendremos $g(x_k = i, u_k = u, x_{k+1} = j) = g_{ij}(u)$, dada la decisión $u \in U$. Entonces la función de valor para una estrategia π : $J_\pi(i)$ $i \in X$, adoptaría la siguiente forma $\forall i \in X$:

$$J_\pi(i) = \underbrace{\mathbb{E}[g_{ij}(u) + J_\pi(j)]}_{Q(i,u)}$$

Donde hemos introducido la notación $Q(i, u)$ para representar el valor esperado del coste de la transición a un paso más el valor del estado siguiente al que se transiciona.

Este procedimiento nos permite acudir a la conocida como **Ecuación de Bellman**, como veremos más adelante, y que permite resolver este tipo de problemas. La forma que adopta dicha ecuación en un problema de minimización es la siguiente:

$$J^*(i) = \min_u \underbrace{\mathbb{E}[g_{ij}(u) + J^*(j)]}_{Q^*(i,u)}$$

La expresión a minimizar también se le conoce como *factor Q*, notado $Q^*(i, u)$. A partir de la obtención recursiva de los valores óptimos de cada estado ($J^*(i), \forall i \in X$) tendremos que las acciones óptimas son:

$$\mu(i) = \arg \min_u \mathbb{E}[g_{ij}(u) + J^*(j)]$$

Implementación. En el plano de la implementación, cuando la función de valor (también la función de estrategia) se definen sobre un conjunto de acciones y estados de pequeña dimensión, los algoritmos emplean una **representación tabular** (ver fig. 7.2). Cada acción-estado se puede reflejar como las filas y columnas de una matriz. Sin embargo, cuando su número es muy elevado (ver fig. 7.3) se suelen emplear la **representación mediante aproximadores funcionales**, como es el caso de las Redes Neuronales.

$Q(i,u)$	1	s
1				
⋮				
i				
⋮				
n				

Figura 7.2: Si el número de estados-acciones es reducido podemos emplear una Representación tabular.

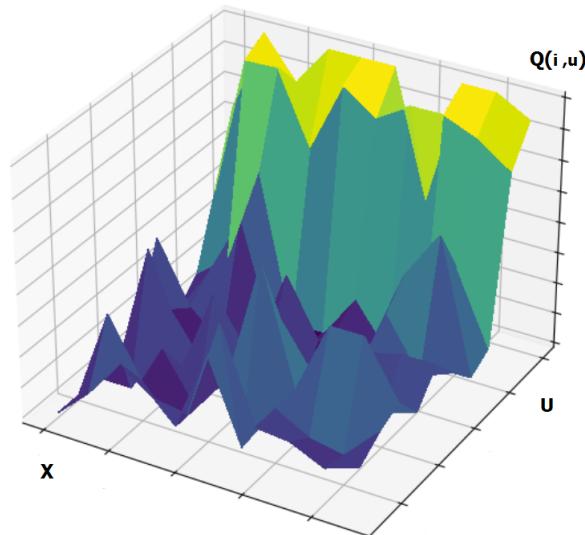


Figura 7.3: Manejar explícitamente el valor de cada par estado acción puede ser muy costoso. Una solución consiste en utilizar una función que aproxime la relación entre los valores.

Una de las herramientas más empleadas en la actualidad son las Redes Neuronales que se emplean como aproximadores funcionales.

Solución exacta vs solución aproximada Cuando el número de estados/acciones crece suficientemente, las exigencias computacionales hacen impracticable el obtener la solución exacta. Por ello se ha de acudir a la obtención de una solución subóptima, mediante la sustitución de $J^*(\cdot)$ por $\tilde{J}(\cdot; \theta)$, donde θ es un vector de parámetros. En la expresión de los costes óptimos el resultado que tendríamos para cada uno de los estados ($i \in X$) sería el que aparece a continuación:

$$\tilde{J}^*(i) = \min_u \underbrace{\mathbb{E} [g_{ij}(u) + \tilde{J}^*(j)]}_{\tilde{Q}^*(i, u)}$$

lo que conducirá a una decisión subóptima: \tilde{u} :

$$\tilde{u}(i) = \arg \min_u \mathbb{E} [g_{ij}(u) + \tilde{J}^*(j)]$$

En la figura 7.4 podemos ver una representación de esta situación. Vemos que la desviación respecto de la decisión óptima puede llegar a ser bastante significativa si el ajuste no es bueno.

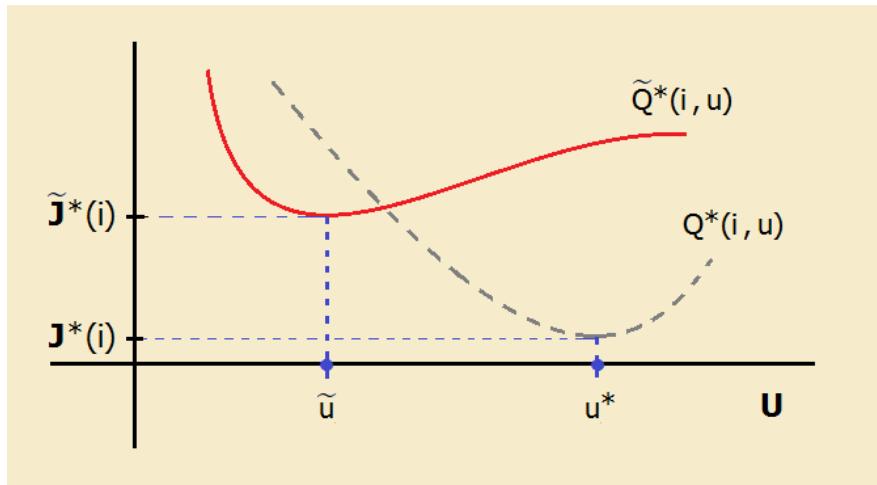


Figura 7.4: Aproximación $\tilde{Q}^*(i, u)$ cuando el sistema se encuentra en un determinado estado $x_k = i$, y la decisión subóptima a la que conduce: \tilde{u} . (Adaptado de Bertsekas, D. (2019) [26])

Si la situación en la que nos encontramos aconseja emplear una aproximación de la función de valor, tendremos que tener en cuenta algunas cuestiones como:

- 1) La estructura de la función \tilde{J} . Usualmente se emplea alguna familia paramétrica de funciones como las Redes Neuronales. Notado: $\tilde{J}(\cdot; \theta)$, donde θ es un vector de parámetros.
- 2) Cómo calcular el vector θ para que el error sea mínimo ($J^*(\cdot) - \tilde{J}(\cdot; \theta)$) y se consiga el mejor ajuste posible al verdadero valor óptimo.

En el siguiente apartado aparece una clasificación de los distintos métodos para encontrar los valores óptimos de los estados. Se empieza por métodos que emplean **Programación Dinámica** y que conducen a una solución exacta. Estos algoritmos también pueden emplearse para obtener una solución aproximada sin más que iterar hasta una cercanía aceptable al verdadero valor. Posteriormente se enumeran algunas técnicas que emplean métodos de **Simulación** para obtener una solución aproximada.

Resumen - Ideas principales

Esto son un conjunto de ideas resumidas que permiten hacerse una idea de todo aquello que se ha analizado en el capítulo.

1. Las redes neuronales que emplearemos serán fundamentalmente perceptrones.
2. También se emplearán redes recurrentes.
3. Además utilizaremos un tipo especial de redes recurrentes, como son las redes LSTM.
Su uso es muy extendido en este tipo de modelizaciones.

Capítulo 8

Estudio de los agentes influyentes en una Red Social

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

8.1. Aplicación de Aprendizaje por refuerzo

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

8.2. Análisis estratégico de un agente emocional

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

8.3. Viralización de información

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Resumen - Ideas principales

Esto son un conjunto de ideas resumidas que permiten hacerse una idea de todo aquello que se ha analizado en el capítulo.

1. Las redes neuronales que emplearemos serán fundamentalmente perceptrones.
2. También se emplearán redes recurrentes.
3. Además utilizaremos un tipo especial de redes recurrentes, como son las redes LSTM.
Su uso es muy extendido en este tipo de modelizaciones.

Capítulo 9

Resultados

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

9.1. Análisis estratégicos

Los siguientes 4 artículos emplean el entorno grid en el que llevan a cabo experimentos con varios tipos de dilemas.

- El siguiente parece ser el primer artículo, de lo que luego son varios que continúan el planteamiento de los dilemas iterados (Leibo, J. et al. (2017) [100]).

- Sobre el aprendizaje de la reciprocidad en dilemas sociales secuenciales tenemos el estudio de Eccles, T. (2019) [51].

- Respecto a otros factores que puedan favorecer la aparición de comportamientos prosociales, tenemos el estudio que relaciona la solución de un dilema social con la aversión a la desigualdad (Hughes, E. (2018) [80]). En este estudio se aborda el efecto de distanciar el impacto negativo del comportamiento egoista.

- Sobre la influencia social como motivación intrínseca tenemos el estudio llevado a cabo por Jaques, N. et al. (2019) [83].

Otro trabajo que también emplea el mismo entorno de entrenamiento:

CAPÍTULO 9. RESULTADOS

- Trata la emergencia de la reciprocidad y la formación de grupos (Baker, B. (2020) [6]).

Otros autores también han creado plataformas para entrenamiento en juegos estratégicos, como el desarrollado por Tian, Y. et al. (2017) [189].

Otro tipo de estudios también intentan analizar los factores que inducen la cooperación en el dilema del prisionero. En cuanto al empleo de emociones tenemos los siguientes estudios interrelacionados.

1. Los sentimientos morales y su efecto en el dilema del prisionero iterado en un sistema multiagente (Bazzan, A. et al. (1998) [11]).
2. También la evolución del comportamiento en agentes dotados de sentimientos morales (Bazzan, A. et al. (2002) [10]).
3. Sobre el papel de los lazos sociales en el desarrollo de la cooperación en el dilema del prisionero iterado tenemos el estudio llevado a cabo por Bazzan, A et al. (2011) [12].
4. Sobre el planteamiento de un marco de trabajo para el desarrollo de agentes dotados de emociones en el Dilema del Prisionero iterado tenemos el trabajo de Bazzan, A. & Bordini, R. (2001) [9].

Algunos otros estudios que relacionan motivaciones intrínsecas al agente con el desarrollo del impulso a la cooperación son los que aparecen a continuación.

1. Sobre la imitación de emociones, en lugar de las estrategias, como vía de elevar el bienestar social en juegos espaciales (Szolnoki, A. (2011) [184]).
2. Sobre las emociones como una motivación intrínseca (Sequeira, P. et al. (2011) [168]), y sobre el diseño de este tipo de señales intrínsecas capaces de guiar el aprendizaje (Sequeira, P. et al. (2014) [169]).
3. Un modelo con algunas emociones (Broekens, J. et al. (2015) [36]).
4. Sobre la extensión en el tiempo de las estrategias cooperadoras (Peysakhovich, A. & Lerer, A. (2017) [140] y (2018) [141]).

9.2. Viralización de información en una red social

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

9.3. Distorsión de la información y consecuencias sobre la difusión

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

9.4. Polarización en una Red Social

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

CAPÍTULO 9. RESULTADOS

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Al ir pulsando ENTER va apareciendo un numerado en los párrafos, por lo que parece que no se necesitan comandos y que tomará de aquí directamente el texto. Posteriormente voy a ir introduciendo órdenes para las diferentes secciones para comprobar si funciona de manera normal, como cuando se incluye todo el texto en el archivo principal.

Esto que aparece aquí es una pequeña muestra de texto para ir haciéndose una idea de la apariencia de la Tesis.

Resumen - Ideas principales

Esto son un conjunto de ideas resumidas que permiten hacerse una idea de todo aquello que se ha analizado en el capítulo.

1. Las redes neuronales que emplearemos serán fundamentalmente perceptrones.
2. También se emplearán redes recurrentes.
3. Además utilizaremos un tipo especial de redes recurrentes, como son las redes LSTM.
Su uso es muy extendido en este tipo de modelizaciones.

Capítulo 10

Conclusiones

Entre los resultados que procedemos a presentar se encuentran algunos y teoremas y resultados que pasaremos a continuación a estudiar pormenorizadamente.

10.1. Efecto de las emociones en las comunicaciones

A continuación se presentan algunos términos que serán de utilidad. Como se ha comentado, en un entorno multiagente, los agentes se ven forzados a interactuar entre sí. Esto provoca que el curso de acción óptimo dependa de las acciones que llevan a cabo el resto de agentes con los que se está interactuando. Por ello este área del Reinforcement Learning está muy influida por la Teoría de Juegos. A continuación se presentan algunos términos que serán de utilidad.

10.2. Viralización de la información

A continuación se presentan algunos términos que serán de utilidad. Como se ha comentado, en un entorno multiagente, los agentes se ven forzados a interactuar entre sí. Esto provoca que el curso de acción óptimo dependa de las acciones que llevan a cabo el resto de agentes con los que se está interactuando. Por ello este área del Reinforcement Learning está muy influida por la Teoría de Juegos. A continuación se presentan algunos términos que serán de utilidad.

A continuación se presentan algunos términos que serán de utilidad. Como se ha comentado, en un entorno multiagente, los agentes se ven forzados a interactuar entre sí. Esto provoca que el curso de acción óptimo dependa de las acciones que llevan a cabo el resto de agentes con los que se está interactuando. Por ello este área del Reinforcement Learning está muy influida por la Teoría de Juegos. A continuación se presentan algunos términos que serán de utilidad.

Resumen - Ideas principales

Esto son un conjunto de ideas resumidas que permiten hacerse una idea de todo aquello que se ha analizado en el capítulo.

1. Las redes neuronales que emplearemos serán fundamentalmente perceptrones.
2. También se emplearán redes recurrentes.
3. Además utilizaremos un tipo especial de redes recurrentes, como son las redes LSTM.
Su uso es muy extendido en este tipo de modelizaciones.

Apéndice A

Estadística y Probabilidad

Apéndice B

Teoría de Juegos

Lo que sigue es una prueba de lo que se pretende llevar a cabo en la tesis. Presentaremos interesantes resultados así como emplearemos una presentación cuidada e impactante. Lo que sigue es una prueba de lo que se pretende llevar a cabo en la tesis. Presentaremos interesantes resultados así como emplearemos una presentación cuidada e impactante. Lo que sigue es una prueba de lo que se pretende llevar a cabo en la tesis. Presentaremos interesantes resultados así como emplearemos una presentación cuidada e impactante.

Apéndice C

Redes Sociales

Dado que el otro paquete da algunos errores e impide la generación del ÍNDICE, pruebo con otro paquete.

Apéndice D

Código PYTHON

D.1. Implementación del algoritmo SARSA

En el código que aparece a continuación se especifican los parámetros para el funcionamiento de la simulación. En primer lugar se importan las librerías. Entre los más importantes para el aprendizaje tenemos el parámetro **alfa** que corresponde a la magnitud del cambio en el proceso de aprendizaje. Se emplea un **factor de descuento** (olvido) gamma igual a 0.1.

```
1 # Importamos las LIBRERIAS
2 import numpy as np
3 import pandas as pd
4 import random
5
6 # Especificamos los parametros que se emplearan a lo largo del
7 # PROGRAMA
8 # -----
9 # PARAMETROS
10 # -----
11 alfa    = 0.1      # TASA DE APRENDIZAJE
12 gamma   = 0.1      # TASA DE DESCUENTO
13 epsilon = 0.2      # probabilidad de EXPLORACION
14 num_estados = 3    # EMOCIONES - Inicialmente 3
15 num_acciones = 2   # Cooperar o NoCooperar
16
17 # Recompensas intr nsecas
18 r_alegria     = 3
19 r_ira          = 2
20 r_culpabilidad = 2
21
22 # TAMA O DE LA CUADRICA (Grid) - (size=10)  =>  (10 filas x 10 columnas)
23 size = 10
```

Podemos ver que inicialmente se introducen 3 emociones. Las recompensas intrínsecas que intervendrán en el aprendizaje aparecen especificadas. El tamaño de prueba del tablero es de 10 filas por 10 columnas.

A continuación, una vez introducidos los parámetros del programa, procedemos a desarrollar

APÉNDICE D. CÓDIGO PYTHON

las clases principales. En primer lugar aparece el código de la clase AGENTE.

```
1 # CLASES PRINCIPALES
2 # =====
3
4 # 1) Clase AGENTE >>> (Agente con exploracion epsilon-greedy)
5 # -----
6
7 class AgenteEmocional(object):
8     def __init__(self, alfa=0.1, gamma=0.1):
9         self.alfa      = alfa
10        self.gamma    = gamma
11        self.estados  = num_estados
12        self.acciones = num_acciones
13        self.num_vecinos = 0
14
15
16     # RECOMPENSAS y GANANCIAS
17     # -----
18     self.recompensa      = 0
19     self.recompensa_intrinseca = 0
20     self.ganancias       = []
21
22     # ESTADOS EMOCIONALES (S y S')
23     # -----
24     # Emocion inicial
25     self.emocion          = np.random.randint(num_estados)
26     # Un valor cualquiera, se modificara de inmediato
27     self.emocion_tras_jugada = 0
28
29     # DECISIONES
30     # -----
31     # Un valor cualquiera, se modificara de inmediato
32     self.accion_propia   = 0
33     # Un valor cualquiera, se modificara de inmediato
34     self.accion_vecindario = 0
35     self.numero_cooperadores = 0
36
37     # PROCESO DE APRENDIZAJE
38     # -----
39     self.matriz_Q = np.zeros((num_estados, num_acciones))
40     self.ERRORRES = []
41
42
43     def Fijar_vecinos(self, vecinos):
44         self.num_vecinos = vecinos
45
46
47     def Vecinos(self):
48         return self.num_vecinos
49
50
51     def Fijar_num_cooperadores(self, numero):
52         self.numero_cooperadores = numero
53
54
55     def Fijar_intensidad_emocional(self, intensidad):
56         self.intensidad_emocional = intensidad
```

```

57
58
59     def Q(self):
60         return self.matriz_Q
61
62
63 # DECISIONES: epsilon-greedy
64 def Tomar_decision(self, estado):
65     valor_aleatorio = np.random.uniform(0,1)
66
67     if valor_aleatorio < epsilon:
68         accion = np.random.randint(num_acciones)
69         self.accion_propia = accion
70     else:
71         Qs = self.matriz_Q[estado]
72         accion = np.argmax(Qs)
73         self.accion_propia = accion
74
75     return accion
76
77
78 def Accion(self):
79     return self.accion_propia
80
81
82 def Accion_vecindario(self, accion_vecinos):
83     self.accion_vecindario = accion_vecinos
84
85
86 def Recompensa(self, recompensa):
87     self.recompensa = recompensa
88
89
90 def nuevo_estado(self, accion_vecindario):
91     ALEGRIA      = 0
92     IRA          = 1
93     CULPABILIDAD = 2
94
95     COOPERAR     = 1
96     No_COOPERAR = 0
97
98     # Partimos de Estado inicial: ALEGRIA ('0')
99     # -----
100    if ((self.emocion == ALEGRIA) and (self.accion_propia == COOPERAR)):
101        # Est ALEGRE y COOPERA
102        if (self.accion_vecindario == 1):                      # <-- Vecinos COOPERAN
103            self.emocion_tras_jugada = ALEGRIA
104            self.recompensa_intrinseca = r_alegria
105        else:                                              # <-- Vecinos No COOPERAN
106            self.emocion_tras_jugada = IRA
107            self.recompensa_intrinseca = 0
108    elif ((self.emocion == ALEGRIA) and (self.accion_propia == No_COOPERAR)):
109        # Est ALEGRE y NoCOOPERA
110        if (self.accion_vecindario == COOPERAR):
111            self.emocion_tras_jugada = CULPABILIDAD
112            self.recompensa_intrinseca = 0
113        else:
114            self.emocion_tras_jugada = ALEGRIA

```

APÉNDICE D. CÓDIGO PYTHON

```
113         self.recompensa_intrinseca = 0
114
115     # Partimos de Estado inicial: IRA ('1')
116     # -----
117     if (self.emocion == IRA) and (self.accion_propia == COOPERAR):
118         # Est IRACUNDO y COOPERA
119         if (self.accion_vecindario == COOPERAR):      # <-- Vecinos COOPERAN
120             self.emocion_tras_jugada = IRA
121             self.recompensa_intrinseca = 0
122         else:                                         # <-- Vecinos No COOPERAN
123             self.emocion_tras_jugada = IRA
124             self.recompensa_intrinseca = 0
125     elif (self.emocion == IRA) and (self.accion_propia == No_COOPERAR):
126         # Est IRACUNDO y NoCOOPERA
127         if (self.accion_vecindario == COOPERAR):      # <-- Vecinos COOPERAN
128             self.emocion_tras_jugada = CULPABILIDAD
129             self.recompensa_intrinseca = 0
130         else:                                         # <-- Vecinos No COOPERAN
131             self.emocion_tras_jugada = IRA
132             self.recompensa_intrinseca = r_ira
133
134     # Partimos de Estado inicial: CULPABILIDAD ('2')
135     # -----
136     if (self.emocion == CULPABILIDAD) and (self.accion_propia == COOPERAR):
137         # Siente CULPABILIDAD y COOPERA
138         if (self.accion_vecindario == COOPERAR):      # <-- Vecinos COOPERAN
139             self.emocion_tras_jugada = ALEGRIA
140             self.recompensa_intrinseca = r_culpabilidad
141         else:                                         # <-- Vecinos No COOPERAN
142             self.emocion_tras_jugada = IRA
143             self.recompensa_intrinseca = 0
144     elif (self.emocion == CULPABILIDAD) and (self.accion_propia == No_COOPERAR):
145         # Siente CULPABILIDAD y NoCOOPERA
146         if (self.accion_vecindario == COOPERAR):      # <-- Vecinos COOPERAN
147             self.emocion_tras_jugada = CULPABILIDAD
148             self.recompensa_intrinseca = 0
149         else:                                         # <-- Vecinos COOPERAN
150             self.emocion_tras_jugada = IRA
151             self.recompensa_intrinseca = 0
152
153     return self.emocion_tras_jugada
154
155 def paso_aprendizaje(self):
156     # ESTADOS: s (estado) y s' (estado_nuevo)
157     estado = self.emocion
158     estado_nuevo = self.nuevo_estado(self.accion_vecindario)
159
160     # RECOMPENSA: r
161     r = self.recompensa + self.recompensa_intrinseca
162
163     # ACCION: a (accion) y a' (accion_nueva)
164     accion = self.accion_propia
165     accion_nueva = self.Tomar_decision(estado_nuevo)
166
167     # Seguimos el rastro de las recompensas que recibe
168     self.ganancias.append(r)
```

```

167
168
169 # -----
170 # Algoritmo SARSA
171 # -----
172 # APRENDIZAJE:
173 target = float(r) + gamma * self.matriz_Q[estado_nuevo][accion_nueva]
174 error = target - self.matriz_Q[estado][accion]
175 self.ERRORES.append(error)
176 self.matriz_Q[estado][accion] = self.matriz_Q[estado][accion] + (alfa*
177 error)
178
179 self.emocion = estado_nuevo
180 self.accion_propia = accion_nueva
180

```

La implementación anterior emplea una representación tabular de la función Q. A continuación representamos los valores que intervienen en el proceso de aprendizaje en el algoritmo SARSA.

	a ₁	...	a	...	a'	...
s ₁						
:	:	:				
s			Q(s,a)			
:	:	:				
s'					Q(s',a')	
:						

Cuadro D.1: *Representación tabular de la función $Q(s, a)$.*

A continuación aparece la implementación de la clase ENTORNO, que será la que nos permita reproducir las interacciones entre los agentes.

```

1
2 # 2) Clase AMBIENTE
3 #
4 # Esta clase debe enviar: (s,r)
5 # Es decir: el estado siguiente (s) - o la informacion para determinarlo -
6 # (en este caso las estrategias de los otros jugadores)
7 # y ademas la RECOMPENSA DEL DILEMA (r)
8
9 class Environment(object):
10     def __init__(self, size_):
11         self.matriz_pagos_J1 = np.zeros((2,2))
12         self.matriz_pagos_J2 = np.zeros((2,2))
13
14     # MATRIZ DE PAGOS del Jugador 1 ( ndice : 0/1 = NoCooperar/Cooperar )
15     self.matriz_pagos_J1[0][0] = 0
16     self.matriz_pagos_J1[0][1] = 5
17     self.matriz_pagos_J1[1][0] = 0
18     self.matriz_pagos_J1[1][1] = 3
19
20     # MATRIZ DE PAGOS del Jugador 2 ( ndice : 0/1 = NoCooperar/Cooperar )
21     self.matriz_pagos_J2[0][0] = 0

```

APÉNDICE D. CÓDIGO PYTHON

```
22     self.matriz_pagos_J2[0][1] = 0
23     self.matriz_pagos_J2[1][0] = 5
24     self.matriz_pagos_J2[1][1] = 3
25
26     self.size = size_
27     self.Grid = np.full((size + 2, size + 2), AgenteEmocional)
28
29 # INCLUIDO EN EL __init__. Antes: def inicializar_tablero(self):
30 # ASIGNAR VECINOS al crear OBJETOS (Para graduar la intensidad emocional)
31 #
32 #   AL INTERIOR DEL TABLERO:           <----- 8 VECINOS
33 for i in range(size+2):
34     for j in range(size+2):
35         self.Grid[i][j] = AgenteEmocional()
36         if ((i>1 and i<size) and (j>1 and j<size)):
37             self.Grid[i][j].Fijar_vecinos(8)
38         else:
39             self.Grid[i][j].Fijar_vecinos(0)
40
41 #   A LAS ESQUINAS                  <----- 3 VECINOS
42 self.Grid[1][1].Fijar_vecinos(3)
43 self.Grid[size][size].Fijar_vecinos(3)
44 self.Grid[size][1].Fijar_vecinos(3)
45 self.Grid[1][size].Fijar_vecinos(3)
46
47 #   AL BORDE                      <----- 5 VECINOS
48 for columna in range(2,size,+1): self.Grid[1][columna].Fijar_vecinos(5)
49 for columna in range(2,size,+1): self.Grid[size][columna].Fijar_vecinos(5)
50 for fila in range(2,size,+1):    self.Grid[fila][1].Fijar_vecinos(5)
51 for fila in range(2,size,+1):    self.Grid[fila][size].Fijar_vecinos(5)
52
53
54 def Devolver_Grid(self):
55     return self.Grid
56
57
58 def step(self):
59     num_cooperadores = 0
60     reward            = 0
61     Vecinos_Cooperan = 0
62     filas             = size
63     columnas          = size
64     recompensa        = 0
65
66     acciones          = np.zeros((size+2, size+2), dtype=int)
67     emociones         = np.zeros((size+2, size+2), dtype=int)
68     intensidad        = np.zeros((size+2, size+2))
69
70     # Aquí cada AGENTE toma su DECISION:
71     for i in range(1,filas+1,+1):
72         for j in range(1,columnas+1,+1):
73             # ESTO CONSTITUIRA EL 'FRAME' con las acciones en un paso
74             estado            = self.Grid[i][j].emocion
75             #acciones[i][j]    = self.Grid[i][j].Tomar_decision(estado)
76             acciones[i][j]    = self.Grid[i][j].Accion()
77                 # (A)
78             emociones[i][j]   = self.Grid[i][j].emocion
79                 # (E)
```

```

78
79
80     # Recopilamos todos los resultados del ENTORNO para enviar al AGENTE
81     for i in range(1,size+1,+1):
82         for j in range(1,size+1,+1):
83             cooperadores = self.Grid[i-1][j-1].Accion() + self.Grid[i-1][j].
84             Accion() + self.Grid[i-1][j+1].Accion() + \
85                 self.Grid[i][j-1].Accion() + \
86                 self.Grid[i][j+1].Accion() + \
87                 self.Grid[i+1][j-1].Accion() + self.Grid[i+1][j].
88             Accion() + self.Grid[i+1][j+1].Accion()
89
90             grado_unanimidad_cooperacion = (cooperadores / self.Grid[i][j].
91             Vecinos())
92
93             if (grado_unanimidad_cooperacion > 0.5):
94                 Vecinos_Cooperan = 1
95                 self.Grid[i][j].Accion_vecindario(Vecinos_Cooperan)
96                 self.Grid[i][j].Fijar_intensidad_emocional(
97                     grado_unanimidad_cooperacion)
98
99             # LA OTRA MATRIZ QUE EMPLEAMOS
100            intensidad[i][j] = grado_unanimidad_cooperacion
101                # (I)
102            a_J1 = acciones[i][j]
103            a_J2 = Vecinos_Cooperan
104            self.Grid[i][j].Fijar_num_cooperadores(cooperadores)
105            self.Grid[i][j].Recompensa(self.matriз_pagos_J1[a_J1][a_J2])
106            self.Grid[i][j].paso_aprendizaje()
107
108        return acciones, emociones, intensidad
109
110
111
112
113
114
115
116

```

D.2. Implementación del algoritmo Q-Learning

En el código que aparece a continuación se implementa el algoritmo Q-Learning, lo que permite al agente aprender. El resto del código es igual al que aparece en la sección anterior dedicada al algoritmo SARSA.

```

1
2     def paso_aprendizaje(self):
3
4         # ESTADOS: s (estado) y s' (estado_nuevo)
5         estado = self.emocion
6         estado_nuevo = self.nuevo_estado(self.accion_vecindario)
7
8         # RECOMPENSA: r
9         r = self.recompensa + self.recompensa_intrinsic
10
11        # ACCION: a (accion) y a' (accion_nueva)
12        accion = self.accion_propia
13        accion_nueva = self.Tomar_decision(estado_nuevo)
14
15        # Seguimos el rastro de las recompensas que recibe
16        self.ganancias.append(r)

```

```

17
18
19     # -----
20     # Algoritmo Q-Learning
21     # -----
22     # APRENDIZAJE:
23     target = float(r) + gamma * self.matriz_Q[estado_nuevo].max()
24     error = target - self.matriz_Q[estado][accion]
25     self.ERRORES.append(error)
26
27     self.matriz_Q[estado][accion] = self.matriz_Q[estado][accion] +
28                                     (alfa*error)
29
30     self.emocion = estado_nuevo
31     self.accion_propia = accion_nueva
32

```

La forma de actualizar los valores durante el aprendizaje emplea la representación tabular de la función $Q(s, a)$ tal y como aparece en la siguiente figura.

	a ₁	...	a	...	a'	...
s ₁						
:	:	:				
s			Q(s,a)			
:	:	:				
s'	m	á	x	i	m	o
:						

Cuadro D.2: Representación tabular de la función $Q(s, a)$. Algoritmo Q-Learning. Actualizamos mediante el máximos valor de la función para el estado s' .

D.3. Análisis Geométrico del Operador de Bellman J_μ

El operador de Bellman nos permite expresar la aplicación reiterada del algoritmo de Programación Dinámica. Esto facilita tanto el análisis del mismo como la comprensión del mecanismo interno del algoritmo. A continuación se lleva a cabo el análisis gráfico del operador T_μ , que es lineal. A continuación aparecen las gráficas que permite seguir el análisis del texto principal con el operador T que no es lineal.

```

1
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # -----
6 def T_mu_J_2(J_1, J_2):                      # Estado (2), pero
7     elegimos rama de acci n 1: u_1
8     alpha = 0.9
9     J_mu_u1 = 0.4*(0 + alpha*J_1) + 0.6*( 6 + alpha*J_2)
10    return J_mu_u1
# -----

```

```

11 x = np.linspace(0, 60, 200)
12 y = np.linspace(0, 60, 200)
13
14 X, Y = np.meshgrid(x, y)
15 Z = T_mu_J_2(X, Y)
16
17
18 # DIBUJAMOS LA CURVA DE COSTES (TJ(2))
19 # -----
20 fig = plt.figure(figsize=(10,6))
21 ax = plt.axes(projection='3d')
22 ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
23                  cmap='viridis', edgecolor='none')
24 ax.set_title('surface');
25 ax.set_xlabel('J(1)')
26 ax.set_ylabel('J(2)')
27 ax.set_zlabel('T_\mu J (2)');
28
29
30 # Fijamos el punto j_1
31 j_1 = 50
32
33 # DIBUJAMOS EL PLANO
34 # -----
35 yy, zz = np.meshgrid(range(60), range(60))
36 xx = yy*0 + j_1
37 ax.plot_surface(xx, yy, zz, alpha=0.2)
38
39 # DIBUJAMOS LA INTERSECCIÓN DE PLANO y CURVA
40 # (Ya que el plano pasa por  $J_1 = 30$ , basta con dar valores a  $J_2$  y calcular  $TJ$ )
41 #
42 -----
43 num_puntos = 200
44 # Fijamos 200 puntos a valor  $J_1 = 30$ 
45 j_1s = np.repeat(j_1, num_puntos)
46 # Fijamos 200 puntos equiespaciados en el sentido de los valores de  $J_2$ 
47 limite_inf = 0
48 limite_sup = 60
49 j_2s = np.linspace(limite_inf, limite_sup, num_puntos)
50 # Obtenemos el valor de  $TJ$  para la combinación ( $T_1$ ,  $T_2$ )
51 valores_z = T_mu_J_2(j_1s, j_2s)
52 # DIBUJAMOS LA LINEA
53 ax.plot(j_1s, j_2s, valores_z, linewidth=2.4)
54 # Si se quiere pintar la bisectriz
55 ax.plot(j_1s, j_2s, j_2s, 'b--', linewidth=0.8)
56

```

Ahora procedemos a presentar el código que realiza la proyección en el plano.

```

1
2 # DIBUJAMOS LA PROYECCIÓN DEL PLANO
3 # -----
4
5 import matplotlib.pyplot as plt
6

```

```
7 # Utilizamos ax en lugar de plt, ya que el color del fondo s lo se puede
8 # cambiar accediendo a ax.
9 ax.set_facecolor('antiquewhite')
10
11 valores_z_J2 = T_mu_J_2(j_1s, j_2s)
12 ax.plot(j_2s, valores_z_J2)
13 puntos = np.linspace(0,60,200)
14 ax.plot(puntos, puntos, 'k--')
15
16 # Buscamos el ndice a partir del cual se cortan las curvas y TJ queda
17 # por debajo de la recta de 45
18 # -----
19 indices_debajo_recta_45_grados = np.where((valores_z - puntos) < 0)
20 indice = indices_debajo_recta_45_grados[0][0]
21 puntos[indice]
22 ax.plot(puntos[indice], puntos[indice], 'ro')
23 ax.plot((puntos[indice], puntos[indice]),(puntos[indice],0), 'r--', linewidth
=0.8)
24
25 #plt.text(38, 10, 'Punto Fijo J(2)*')
26
27 plt.xlabel('J(2)')
28 plt.ylabel('T_\mu J (2)')
29
30 plt.axis([0,60,0,60])
31
```

D.4. Resolución de Sistemas Lineales - Cálculo de J_{μ^k}

En el código que aparece a continuación se especifican los parámetros para el funcionamiento de la simulación. En primer lugar se importan las librerías. Entre los más importantes para el aprendizaje tenemos el parámetro **alfa** que corresponde a la magnitud del cambio en el proceso de aprendizaje. Se emplea un **factor de descuento** (olvido) gamma igual a 0.1.

```
1
2 # RESOLUCI N DE SISTEMAS - Policy Iteration
3 # =====
4
5 # EJERCICIO POLICY ITERATION - Enunciado Libro de Dimitri Bertsekas
6 # =====
7
8 # [ k = 0 ]
9 # -----
10
11 import numpy as np
12
13 a = np.array([[ 0.325, -0.225],
14               [-0.225,  0.325]])
15 b = np.array([2,3])
16
17 J_mu = np.linalg.solve(a,b)
18 J_mu
19
20 >>> array([24.09090909, 25.90909091])
21
```

```

22 # [ k = 1 ]
23 # -----
24
25
26 a = np.array([[ 0.775, -0.675],
27               [-0.675 ,  0.775]])
28 b = np.array([0.5,1])
29
30 J_mu = np.linalg.solve(a,b)
31 J_mu
32
33 >>> array([7.32758621,  7.67241379])
34

```

Respecto al algoritmo VALUE ITERATION, a continuación se expone el fragmento de algoritmo que genera la salida.

```

1 # GENERACION DE VALUE ITERATION
2 # -----
3
4
5 import pandas as pd
6
7 # Procedimientos Operador de Bellman
8 # -----
9
10 def TJ_1(J_1, J_2):
11     alpha = 0.9
12     J_u1 = np.minimum(0.3*(3 + alpha*J_1) + 0.7*(10 + alpha*J_2),
13                       0.6*(7 + alpha*J_1) + 0.4*( 5 + alpha*J_2))
14     return J_u1
15
16 def TJ_2(J_1, J_2):
17     alpha = 0.9
18     J_u2 = np.minimum(0.4*(0 + alpha*J_1) + 0.6*( 6 + alpha*J_2),
19                       0.9*(3 + alpha*J_1) + 0.1*(12 + alpha*J_2))
20     return J_u2
21
22 # -----
23
24
25 J_1, J_2 = [], []
26
27
28 # Valores iniciales
29 # -----
30 j1_o = 0
31 j2_o = 0
32
33
34 # Iniciaremos las iteraciones del algoritmo VALUE ITERATION con
35 # los valores iniciales
36 # -----
37 J_1.append(j1_o)
38 J_2.append(j2_o)
39
40 for iteracion in range(120):
41     tj1 = TJ_1(J_1[iteracion], J_2[iteracion])

```

APÉNDICE D. CÓDIGO PYTHON

```
42 tj2 = TJ_2(J_1[iteracion], J_2[iteracion])
43 J_1.append(tj1)
44 J_2.append(tj2)
45
46 # Convertimos en un Data Frame
47 # -----
48 df_T = pd.DataFrame(np.array([J_1, J_2])).T
49 df_T.columns = ['TJ(1)', 'TJ(2)']
50 df_T
51
```

Referencias

- [1] Anagnostopoulos, A., Kumar, R. y Mahdian, M. “Influence and correlation in social networks”. En: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, págs. 7-15. URL: https://www.ccs.neu.edu/home/yzsun/classes/2014Spring_CS7280/Papers/Diffusion/influence.pdf (vid. pág. 5).
- [2] Aumann, R. J. “Correlated equilibrium as an expression of Bayesian rationality”. En: *Econometrica: Journal of the Econometric Society* (1987), págs. 1-18. URL: <http://www.ma.huji.ac.il/~raumann/pdf/Correlated%20Equilibrium%20as%20an%20Expression%20of%20Bayesian%20Rationality.pdf> (vid. pág. 131).
- [3] Axelrod, R. “The dissemination of culture: A model with local convergence and global polarization”. En: *Journal of conflict resolution* 41.2 (1997), págs. 203-226. URL: https://deepblue.lib.umich.edu/bitstream/handle/2027.42/67489/10.1177_0022002797041002001.pdf?sequence=2 (vid. pág. 8).
- [4] Axelrod, R. y Hamilton, W. D. “The evolution of cooperation”. En: *science* 211.4489 (1981), págs. 1390-1396. URL: <http://www-personal.umich.edu/~axe/research/Axelrod%20and%20Hamilton%20EC%201981.pdf> (vid. págs. 7, 155).
- [5] Baird III, L. C. *Advantage updating*. Inf. téc. WRIGHT LAB WRIGHT-PATTERSON AFB OH, 1993. URL: <https://apps.dtic.mil/sti/pdfs/ADA280862.pdf> (vid. pág. 75).
- [6] Baker, B. “Emergent reciprocity and team formation from randomized uncertain social preferences”. En: *Advances in Neural Information Processing Systems* 33 (2020), págs. 15786-15799. URL: <https://arxiv.org/pdf/2011.05373.pdf> (vid. págs. 8, 164).
- [7] Banerjee, B. y Peng, J. “Adaptive policy gradient in multiagent learning”. En: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. 2003, págs. 686-692. URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.5911&rep=rep1&type=pdf> (vid. págs. 139, 140).
- [8] Barabási, A.-L. y col. *Network science*. Cambridge university press, 2016 (vid. pág. 5).
- [9] Bazzan, A. L. y Bordini, R. H. “A framework for the simulation of agents with emotions”. En: *Proceedings of the fifth international conference on Autonomous agents*. 2001, págs. 292-299. URL: https://web.archive.org/web/20070901220903id_/http://www.vhml.org/theses/wijayat/sources/writings/papers/p292-bazzan.pdf (vid. págs. 8, 153, 155, 164).
- [10] Bazzan, A. L., Bordini, R. H. y Campbell, J. A. “Evolution of agents with moral sentiments in an iterated prisoner’s Dilemma exercise”. En: *Game theory and decision theory in agent-based systems*. Springer, 2002, págs. 43-64. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.5305&rep=rep1&type=pdf> (vid. págs. 8, 153, 155, 164).

REFERENCIAS

- [11] Bazzan, A. L., Bordini, R. H. y Campbell, J. A. “Moral sentiments in multi-agent systems”. En: *International Workshop on Agent Theories, Architectures, and Languages*. Springer. 1998, págs. 113-131 (vid. págs. 8, 164).
- [12] Bazzan, A. L., Peleteiro, A. y Burguillo, J. C. “Learning to cooperate in the Iterated Prisoner’s Dilemma by means of social attachments”. En: *Journal of the Brazilian computer society* 17 (2011), págs. 163-174. URL: <https://link.springer.com/content/pdf/10.1007/s13173-011-0038-2.pdf> (vid. págs. 8, 164).
- [13] Bellemare, M. G., Naddaf, Y., Veness, J. y Bowling, M. “The arcade learning environment: An evaluation platform for general agents”. En: *Journal of Artificial Intelligence Research* 47 (2013), págs. 253-279. URL: <https://arxiv.org/pdf/1207.4708.pdf> (vid. pág. 86).
- [14] Bellman, R. “A Markovian decision process”. En: *Journal of mathematics and mechanics* (1957), págs. 679-684. URL: <https://apps.dtic.mil/sti/pdfs/AD0606367.pdf> (vid. pág. 85).
- [15] Bellman, R. “On the theory of dynamic programming”. En: *Proceedings of the National Academy of Sciences of the United States of America* 38.8 (1952), pág. 716. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1063639/pdf/pnas01581-0064.pdf> (vid. págs. 27, 85).
- [16] Bellman, R. y Kalaba, R. E. *Dynamic programming and modern control theory*. Vol. 81. Citeseer, 1965 (vid. pág. 27).
- [17] Bengio, Y., Frasconi, P. y Simard, P. “The problem of learning long-term dependencies in recurrent networks”. En: *IEEE international conference on neural networks*. IEEE. 1993, págs. 1183-1188 (vid. pág. 116).
- [18] Berger, J. “Arousal increases social transmission of information”. En: *Psychological science* 22.7 (2011), págs. 891-893. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.989.4895&rep=rep1&type=pdf> (vid. pág. 10).
- [19] Berger, J. *Contagious: Why things catch on*. Simon y Schuster, 2016 (vid. pág. 10).
- [20] Berger, J. y Milkman, K. L. “What makes online content viral?” En: *Journal of marketing research* 49.2 (2012), págs. 192-205. URL: <https://jonahberger.com/wp-content/uploads/2013/02/ViralityB.pdf> (vid. pág. 10).
- [21] Bertsekas, D. *Abstract dynamic programming*. Athena Scientific, 2022 (vid. págs. 41, 53, 54, 58, 80).
- [22] Bertsekas, D. *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*. Athena Scientific, 2022 (vid. pág. 56).
- [23] Bertsekas, D. *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific, 2021 (vid. págs. 74, 134).
- [24] Bertsekas, D. P. *Dynamic programming and optimal control: Vol. 1*. 4th. Athena scientific Belmont, 2017 (vid. págs. 17, 74, 75).
- [25] Bertsekas, D. P. *Dynamic programming and optimal control: Vol. 2*. 4th. Athena scientific Belmont, 2012 (vid. págs. 17, 59, 74).
- [26] Bertsekas, D. P. *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019 (vid. págs. 22, 74, 81, 83, 87, 159).
- [27] Bertsekas, D. P. y Tsitsiklis, J. N. *Neuro-dynamic programming*. Athena Scientific, 1996 (vid. págs. 64, 69, 70, 76, 80, 101).

- [28] Bianchi, R. A., Martins, M. F., Ribeiro, C. H. y Costa, A. H. “Heuristically-accelerated multiagent reinforcement learning”. En: *IEEE transactions on cybernetics* 44.2 (2013), págs. 252-265. URL: <https://fei.edu.br/~rbianchi/publications/ecai2012031.pdf> (vid. págs. 138, 140).
- [29] Bjorn, P. A. y Vuong, Q. H. “Econometric modeling of a Stackelberg game with an application to labor force participation”. En: (1985). URL: <https://authors.library.caltech.edu/81492/1/sswp577.pdf> (vid. pág. 131).
- [30] Bloembergen, D., Tuyls, K., Hennes, D. y Kaisers, M. “Evolutionary dynamics of multi-agent learning: A survey”. En: *Journal of Artificial Intelligence Research* 53 (2015), págs. 659-697. URL: <https://asset-pdf.scinapse.io/prod/1192553058/1192553058.pdf> (vid. págs. 130, 140, 155).
- [31] Bond, R. M., Fariss, C. J., Jones, J. J., Kramer, A. D., Marlow, C., Settle, J. E. y Fowler, J. H. “A 61-million-person experiment in social influence and political mobilization”. En: *Nature* 489.7415 (2012), págs. 295-298. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3834737/pdf/nihms524815.pdf> (vid. págs. 1, 6).
- [32] Borkar, V. S. y Meyn, S. P. “The ODE method for convergence of stochastic approximation and reinforcement learning”. En: *SIAM Journal on Control and Optimization* 38.2 (2000), págs. 447-469. URL: <http://repository.ias.ac.in/5333/1/351.pdf> (vid. pág. 80).
- [33] Bourlard, H. y Kamp, Y. “Auto-association by multilayer perceptrons and singular value decomposition”. En: *Biological cybernetics* 59.4 (1988), págs. 291-294. URL: https://www.researchgate.net/profile/Herve-Bourlard/publication/19959069_Auto-Association_by_Multilayer_Perceptrons_and_Singular_Value_Decomposition/links/57600aaa08aeeada5bc2b4cc/Auto-Association-by-Multilayer-Perceptrons-and-Singular-Value-Decomposition.pdf (vid. pág. 116).
- [34] Bowling, M. “Convergence and no-regret in multiagent learning”. En: *Advances in neural information processing systems* 17 (2005), págs. 209-216. URL: <https://era.library.ualberta.ca/items/2225eedc-947a-4ff0-bba4-8c3bd49ddc86/view/2baeb07b-0225-402a-a400-c765a9f316f7/TR04-11.pdf> (vid. págs. 139, 140).
- [35] Bowling, M. y Veloso, M. “Multiagent learning using a variable learning rate”. En: *Artificial Intelligence* 136.2 (2002), págs. 215-250. URL: <http://www.cs.cmu.edu/~mmv/papers/02aij-mike.pdf> (vid. págs. 139, 140, 142-144).
- [36] Broekens, J., Jacobs, E. y Jonker, C. M. “A reinforcement learning model of joy, distress, hope and fear”. En: *Connection Science* 27.3 (2015), págs. 215-233. URL: <https://www.tandfonline.com/doi/pdf/10.1080/09540091.2015.1031081> (vid. págs. 9, 164).
- [37] Brown, G. W. “Iterative solution of games by fictitious play”. En: *Activity analysis of production and allocation* 13.1 (1951), págs. 374-376 (vid. págs. 139, 140).
- [38] Brunton, S. L. y Kutz, J. N. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022 (vid. pág. 117).
- [39] Busoniu, L., Babuska, R. y De Schutter, B. “A comprehensive survey of multiagent reinforcement learning”. En: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), págs. 156-172. URL: https://www.dcsc.tudelft.nl/~bdeschutter/pub/rep/07_019.pdf (vid. pág. 138).
- [40] Calvo, J. A. y Dusparic, I. “Heterogeneous Multi-Agent Deep Reinforcement Learning for Traffic Lights Control.” En: *AICS*. 2018, págs. 2-13. URL: http://ceur-ws.org/Vol-2259/aics_2.pdf (vid. pág. 103).

REFERENCIAS

- [41] Canese, L., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M. y Spanò, S. “Multi-agent reinforcement learning: A review of challenges and applications”. En: *Applied Sciences* 11.11 (2021), pág. 4948. URL: <https://www.mdpi.com/2076-3417/11/11/4948> (vid. pág. 140).
- [42] Cassandra, A. R., Kaelbling, L. P. y Littman, M. L. “Acting optimally in partially observable stochastic domains”. En: *Aaai*. Vol. 94. 1994, págs. 1023-1028. URL: <https://aaai.org/Papers/AAAI/1994/AAAI94-157.pdf> (vid. págs. 91, 107).
- [43] Chen, W., Wang, J., Yu, F., He, J., Xu, W. y Wang, R. “Effects of emotion on the evolution of cooperation in a spatial prisoner’s dilemma game”. En: *Applied Mathematics and Computation* 411 (2021), pág. 126497 (vid. pág. 156).
- [44] Claus, C. y Boutilier, C. “The dynamics of reinforcement learning in cooperative multiagent systems”. En: *AAAI/IAAI* 1998.746-752 (1998), pág. 2. URL: <https://www.aaai.org/Papers/AAAI/1998/AAAI98-106.pdf> (vid. págs. 138, 140).
- [45] Collins, A., Sokolowski, J. y Banks, C. “Applying reinforcement learning to an insurgency Agent-based Simulation”. En: *The Journal of Defense Modeling and Simulation* 11.4 (2014), págs. 353-364. URL: https://www.researchgate.net/publication/261704876_Applying_Reinforcement_Learning_to_an_Insurgency_Agent-based_Simulation (vid. pág. 103).
- [46] Conitzer, V. y Sandholm, T. “AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents”. En: *Machine Learning* 67.1-2 (2007), págs. 23-43. URL: <https://link.springer.com/content/pdf/10.1007/s10994-006-0143-1.pdf> (vid. págs. 138, 140).
- [47] Crandall, D., Cosley, D., Huttenlocher, D., Kleinberg, J. y Suri, S. “Feedback effects between similarity and social influence in online communities”. En: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, págs. 160-168. URL: <https://www.cs.bgu.ac.il/~snean151/wiki.files/6-FeedbackEffectsbetweenSimilarityandSocialInfluence.pdf> (vid. pág. 5).
- [48] Crandall, J. W. “Just add Pepper: extending learning algorithms for repeated matrix games to repeated markov games”. En: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems- Volume 1*. Citeseer. 2012, págs. 399-406. URL: https://faculty.cs.byu.edu/~crandall/papers/Crandall_AAMAS2012.pdf (vid. pág. 156).
- [49] Cybenko, G. “Approximation by superpositions of a sigmoidal function”. En: *Mathematics of control, signals and systems* 2.4 (1989), págs. 303-314. URL: <http://www.vision.jhu.edu/teaching/learning/deeplearning18/assets/Cybenko-89.pdf> (vid. pág. 113).
- [50] Domingos, P. y Richardson, M. “Mining the network value of customers”. En: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001, págs. 57-66. URL: <http://snap.stanford.edu/class/cs224w-readings/domingos01networkvalue.pdf> (vid. pág. 1).
- [51] Eccles, T., Hughes, E., Kramár, J., Wheelwright, S. y Leibo, J. Z. “Learning reciprocity in complex sequential social dilemmas”. En: *arXiv preprint arXiv:1903.08082* (2019). URL: <https://arxiv.org/pdf/1903.08082.pdf> (vid. págs. 8, 155, 163).
- [52] Elidrisi, M., Johnson, N., Gini, M. L. y Crandall, J. W. “Fast adaptive learning in repeated stochastic games by game abstraction.” En: *AAMAS*. 2014, págs. 1141-1148. URL: <https://www.ifaamas.org/Proceedings/aamas2014/aamas/p1141.pdf> (vid. pág. 156).

- [53] Everett, M., Chen, Y. F. y How, J. P. “Collision avoidance in pedestrian-rich environments with deep reinforcement learning”. En: *IEEE Access* 9 (2021), págs. 10357-10377. URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9317723> (vid. pág. 103).
- [54] Fan, R., Xu, K. y Zhao, J. “An agent-based model for emotion contagion and competition in online social media”. En: *Physica a: statistical mechanics and its applications* 495 (2018), págs. 245-259. URL: <https://arxiv.org/pdf/1706.02676.pdf> (vid. pág. 9).
- [55] Ferrara, E. y Yang, Z. “Quantifying the effect of sentiment on information diffusion in social media”. En: *PeerJ Computer Science* 1 (2015), e26. URL: <https://peerj.com/articles/cs-26/> (vid. pág. 10).
- [56] Fowler, J. H. y Christakis, N. A. “Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the Framingham Heart Study”. En: *Bmj* 337 (2008). URL: <https://www.bmjjournals.org/content/bmjj/337/bmj.a2338.full.pdf> (vid. págs. 2, 10).
- [57] Fujimoto, S., Van Hoof, H. y Meger, D. “Addressing function approximation error in actor-critic methods”. En: *arXiv preprint arXiv:1802.09477* (2018). URL: <https://arxiv.org/pdf/1802.09477.pdf> (vid. págs. 124, 126).
- [58] Fukushima, K. “Neocognitron: A hierarchical neural network capable of visual pattern recognition”. En: *Neural networks* 1.2 (1988), págs. 119-130. URL: http://vision.stanford.edu/teaching/cs131_fall1415/lectures/Fukushima1988.pdf (vid. pág. 116).
- [59] Fukushima, K. y Miyake, S. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. En: *Competition and cooperation in neural nets*. Springer, 1982, págs. 267-285. URL: <https://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf> (vid. pág. 116).
- [60] Fulda, N. y Ventura, D. “Predicting and Preventing Coordination Problems in Cooperative Q-learning Systems.” En: *IJCAI*. Vol. 2007. 2007, págs. 780-785. URL: <https://www.aaai.org/Papers/IJCAI/2007/IJCAI07-125.pdf> (vid. pág. 156).
- [61] Games, M. “The fantastic combinations of John Conway’s new solitaire game “life” by Martin Gardner”. En: *Scientific American* 223 (1970), págs. 120-123. URL: <https://web.stanford.edu/class/sts145/Library/life.pdf> (vid. pág. 7).
- [62] Goodfellow, I., Bengio, Y. y Courville, A. *Deep learning*. MIT press, 2016. URL: <https://www.deeplearningbook.org/> (vid. págs. 115, 117).
- [63] Gosavi, A. *Simulation-based optimization*. 2th Ed. Springer, 2015 (vid. págs. 17, 80).
- [64] Greenwald, A., Hall, K. y Serrano, R. “Correlated Q-learning”. En: *ICML*. Vol. 3. 2003, págs. 242-249. URL: <https://www.aaai.org/Papers/Symposia/Spring/2002/SS-02-02/SS02-02-012.pdf> (vid. págs. 139, 140).
- [65] Gupta, J. K., Egorov, M. y Kochenderfer, M. “Cooperative multi-agent control using deep reinforcement learning”. En: *International conference on autonomous agents and multiagent systems*. Springer. 2017, págs. 66-83. URL: https://ala2017.it.nuigalway.ie/papers/ALA2017_Gupta.pdf (vid. pág. 137).
- [66] Haarnoja, T., Zhou, A., Abbeel, P. y Levine, S. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. En: *arXiv preprint arXiv:1801.01290* (2018). URL: <https://arxiv.org/pdf/1801.01290.pdf> (vid. pág. 126).

- [67] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P. y col. “Soft actor-critic algorithms and applications”. En: *arXiv preprint arXiv:1812.05905* (2018). URL: <https://arxiv.org/pdf/1812.05905.pdf> (vid. pág. 126).
- [68] Harmon, M. E., Baird, L. y Klopff, A. H. “Advantage updating applied to a differential game”. En: *Advances in neural information processing systems* 7 (1994). URL: <https://proceedings.neurips.cc/paper/1994/file/2a9d121cd9c3a1832bb6d2cc6bd7a8a7-Paper.pdf> (vid. pág. 75).
- [69] Hasselt, H. “Double Q-learning”. En: *Advances in neural information processing systems* 23 (2010). URL: <https://proceedings.neurips.cc/paper/2010/file/091d584fcfed301b442654dd8c-Paper.pdf> (vid. pág. 120).
- [70] Hasselt, H. P. van. *Insights in reinforcement learning*. Hado van Hasselt, 2011 (vid. pág. 120).
- [71] Hausknecht, M. y Stone, P. “Deep recurrent q-learning for partially observable mdps”. En: *arXiv preprint arXiv:1507.06527* (2015). URL: <https://arxiv.org/pdf/1507.06527.pdf> (vid. pág. 122).
- [72] Hernandez-Leal, P. y Kaisers, M. “Towards a fast detection of opponents in repeated stochastic games”. En: *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers* 16. Springer. 2017, págs. 239-257. URL: https://ir.cwi.nl/pub/27237/2017_TIRL_Hernandez.pdf (vid. pág. 156).
- [73] Hernandez-Leal, P., Kartal, B. y Taylor, M. E. “A survey and critique of multiagent deep reinforcement learning”. En: *Autonomous Agents and Multi-Agent Systems* 33.6 (2019), págs. 750-797. URL: <https://arxiv.org/pdf/1810.05587.pdf> (vid. pág. 126).
- [74] Hines, G. y Larson, K. “Learning when to take advice: A statistical test for achieving a correlated equilibrium”. En: *arXiv preprint arXiv:1206.3261* (2012). URL: <https://arxiv.org/ftp/arxiv/papers/1206/1206.3261.pdf> (vid. pág. 156).
- [75] Hochreiter, S. y Schmidhuber, J. “Long short-term memory”. En: *Neural computation* 9.8 (1997), págs. 1735-1780. URL: <https://blog.xpgreat.com/file/lstm.pdf> (vid. pág. 116).
- [76] Hornik, K., Stinchcombe, M. y White, H. “Multilayer feedforward networks are universal approximators”. En: *Neural networks* 2.5 (1989), págs. 359-366. URL: https://www.cs.cmu.edu/~epxing/Class/10715/reading/Kornick_et_al.pdf (vid. pág. 113).
- [77] Hu, J., Wellman, M. P. y col. “Multiagent reinforcement learning: theoretical framework and an algorithm.” En: *ICML*. Vol. 98. Citeseer. 1998, págs. 242-250. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.2589&rep=rep1&type=pdf> (vid. págs. 139, 140).
- [78] Hu, J. y Wellman, M. P. “Nash Q-learning for general-sum stochastic games”. En: *Journal of machine learning research* 4.Nov (2003), págs. 1039-1069. URL: <https://www.jmlr.org/papers/volume4/hu03a/hu03a.pdf> (vid. pág. 135).
- [79] Hubel, D. H. y Wiesel, T. N. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. En: *The Journal of physiology* 160.1 (1962), pág. 106. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/pdf/jphysiol01247-0121.pdf> (vid. pág. 116).

- [80] Hughes, E., Leibo, J. Z., Philips, M. G., Tuyls, K., Duéñez-Guzmán, E. A., Castañeda, A. G., Dunning, I., Zhu, T., McKee, K. R., Koster, R. y col. “Inequity aversion resolves intertemporal social dilemmas. CoRR abs/1803.08884 (2018)”. En: *arXiv preprint arXiv:1803.08884* (2018). URL: <https://arxiv.org/pdf/1803.08884.pdf> (vid. págs. 8, 163).
- [81] Jaakkola, T., Jordan, M. y Singh, S. “Convergence of stochastic iterative dynamic programming algorithms”. En: *Advances in neural information processing systems* 6 (1993). URL: <https://proceedings.neurips.cc/paper/1993/file/5807a685d1a9ab3b599035bc566ce2b9-Paper.pdf> (vid. pág. 80).
- [82] Jaakkola, T., Singh, S. y Jordan, M. “Reinforcement learning algorithm for partially observable Markov decision problems”. En: *Advances in neural information processing systems* 7 (1994). URL: <https://proceedings.neurips.cc/paper/1994/file/1c1d4df596d01da60385f0bb17a4a9e0-Paper.pdf> (vid. págs. 91, 107).
- [83] Jaques, N., Lazaridou, A., Hughes, E., Gulcehre, C., Ortega, P., Strouse, D., Leibo, J. Z. y De Freitas, N. “Social influence as intrinsic motivation for multi-agent deep reinforcement learning”. En: *International Conference on Machine Learning*. PMLR. 2019, págs. 3040-3049. URL: <https://arxiv.org/pdf/1810.08647.pdf> (vid. págs. 8, 163).
- [84] Kaelbling, L. P., Littman, M. L. y Cassandra, A. R. “Planning and acting in partially observable stochastic domains”. En: *Artificial intelligence* 101.1-2 (1998), págs. 99-134. URL: <https://people.csail.mit.edu/lpk/papers/aij98-pomdp.pdf> (vid. págs. 91, 107).
- [85] Kapetanakis, S. y Kudenko, D. “Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems”. En: *Adaptive Agents and Multi-Agent Systems II*. Springer, 2004, págs. 119-131. URL: <https://www.aaai.org/Papers/AAAI/2002/AAAI02-050.pdf> (vid. págs. 138, 140).
- [86] Kempe, D., Kleinberg, J. y Tardos, É. “Influential nodes in a diffusion model for social networks”. En: *International Colloquium on Automata, Languages, and Programming*. Springer. 2005, págs. 1127-1138. URL: <https://www.cs.cornell.edu/home/kleinber/icalp05-inf.pdf> (vid. pág. 1).
- [87] Kempe, D., Kleinberg, J. y Tardos, É. “Maximizing the spread of influence through a social network”. En: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, págs. 137-146. URL: <https://www.cs.cornell.edu/home/kleinber/kdd03-inf.pdf> (vid. pág. 1).
- [88] Kok, J. R. y Vlassis, N. “Sparse cooperative Q-learning”. En: *Proceedings of the twenty-first international conference on Machine learning*. 2004, pág. 61. URL: <https://icml.cc/Conferences/2004/proceedings/papers/267.pdf> (vid. págs. 138, 140).
- [89] Konda, V. R. y Tsitsiklis, J. N. “Actor-critic algorithms”. En: *Advances in neural information processing systems*. 2000, págs. 1008-1014. URL: <https://papers.nips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf> (vid. pág. 124).
- [90] Könönen, V. “Asymmetric multiagent reinforcement learning”. En: *Web Intelligence and Agent Systems: An international journal* 2.2 (2004), págs. 105-121. URL: <http://lib.tkk.fi/Diss/2004/isbn9512273594/article1.pdf> (vid. pág. 139).

- [91] Kramer, A. D. “The spread of emotion via Facebook”. En: *Proceedings of the SIGCHI conference on human factors in computing systems*. 2012, págs. 767-770. URL: <https://research.fb.com/wp-content/uploads/2012/05/the-spread-of-emotion-via-facebook.pdf> (vid. pág. 10).
- [92] Kramer, A. D., Guillory, J. E. y Hancock, J. T. “Experimental evidence of massive-scale emotional contagion through social networks”. En: *Proceedings of the National Academy of Sciences* 111.24 (2014), págs. 8788-8790. URL: <https://www.pnas.org/content/pnas/111/24/8788.full.pdf> (vid. pág. 10).
- [93] Krishnamurthy, V. *Partially observed Markov decision processes*. Cambridge university press, 2016 (vid. pág. 111).
- [94] Kulkarni, V. G. *Modeling and analysis of stochastic systems*. Crc Press, 2016 (vid. pág. 104).
- [95] Lakshminarahan, S. y Narendra, K. S. “Learning algorithms for two-person zero-sum stochastic games with incomplete information”. En: *Math. of Op. R.* 6.3 (1981), págs. 379-386. URL: https://www.researchgate.net/profile/S-Lakshminarahan/publication/238877175_Learning_Algorithms_for_Two-Person_Zero-Sum_Stochastic_Games_with_Incomplete_Information_A_Unified_Approach/links/59316c0c45851553b69445b9/Learning-Algorithms-for-Two-Person-Zero-Sum-Stochastic-Games-with-Incomplete-Information-A-Unified-Approach.pdf (vid. pág. 146).
- [96] Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., Silver, D. y Graepel, T. “A unified game-theoretic approach to multiagent reinforcement learning”. En: *Advances in neural information processing systems* 30 (2017). URL: <https://arxiv.org/pdf/1711.00832.pdf> (vid. pág. 137).
- [97] Lauer, M. y Riedmiller, M. “An algorithm for distributed reinforcement learning in cooperative multi-agent systems”. En: *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer. 2000. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.772> (vid. págs. 138, 140).
- [98] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W. y Jackel, L. “Handwritten digit recognition with a back-propagation network”. En: *Advances in neural information processing systems* 2 (1989). URL: <https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf> (vid. pág. 116).
- [99] LeDoux, J. *The deep history of ourselves: The four-billion-year story of how we got conscious brains*. Penguin, 2020 (vid. pág. 10).
- [100] Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J. y Graepel, T. “Multi-agent reinforcement learning in sequential social dilemmas”. En: *arXiv preprint arXiv:1702.03037* (2017). URL: <https://arxiv.org/pdf/1702.03037.pdf> (vid. págs. 8, 103, 155, 163).
- [101] Leskovec, J., Singh, A. y Kleinberg, J. “Patterns of influence in a recommendation network”. En: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2006, págs. 380-389. URL: <https://www.cs.cornell.edu/info/people/kleinber/pakdd06-cascade.pdf> (vid. pág. 6).
- [102] Li, J., Yao, L., Xu, X., Cheng, B. y Ren, J. “Deep reinforcement learning for pedestrian collision avoidance and human-machine cooperative driving”. En: *Information Sciences* 532 (2020), págs. 110-124. URL: <https://www.sciencedirect.com/science/article/abs/pii/S0020025520302851> (vid. pág. 103).

- [103] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. y Wiersstra, D. “Continuous control with deep reinforcement learning”. En: *arXiv preprint arXiv:1509.02971* (2015). URL: <https://arxiv.org/pdf/1509.02971.pdf> (vid. pág. 125).
- [104] Littman, M. L. y col. “Friend-or-foe Q-learning in general-sum games”. En: *ICML*. Vol. 1. 2001, págs. 322-328. URL: https://www.researchgate.net/profile/Michael-Littman/publication/2933305_Friend-or-Foe_Q-learning_in_General-Sum_Games/links/54b66cb80cf24eb34f6d19dc/Friend-or-Foe-Q-learning-in-General-Sum-Games.pdf (vid. pág. 139).
- [105] Littman, M. L. “Markov games as a framework for multi-agent reinforcement learning”. En: *Machine learning proceedings 1994*. Elsevier, 1994, págs. 157-163. URL: <https://www2.cs.duke.edu/courses/spring07/cps296.3/littman94markov.pdf> (vid. págs. 138, 140, 146, 149).
- [106] Littman, M. L. “Value-function reinforcement learning in Markov games”. En: *Cognitive systems research* 2.1 (2001), págs. 55-66. URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.217.7294&rep=rep1&type=pdf> (vid. págs. 138, 140, 146).
- [107] Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X. y Pietikäinen, M. “Deep learning for generic object detection: A survey”. En: *International journal of computer vision* 128 (2020), págs. 261-318. URL: <https://arxiv.org/pdf/1809.02165.pdf> (vid. pág. 3).
- [108] Macy, M. W. y Flache, A. “Learning dynamics in social dilemmas”. En: *Proceedings of the National Academy of Sciences* 99.suppl_3 (2002), págs. 7229-7236. URL: <https://www.pnas.org/doi/full/10.1073/pnas.092080099> (vid. pág. 133).
- [109] Marvin, M. y Seymour, A. P. “Perceptrons”. En: *Cambridge, MA: MIT Press* 6 (1969), págs. 318-362 (vid. pág. 116).
- [110] Mataric, M. J. “Reward functions for accelerated learning”. En: *Machine learning proceedings 1994*. Elsevier, 1994, págs. 181-189. URL: <https://www.sci.brooklyn.cuny.edu/~sklar/teaching/boston-college/s01/mc375/ml94.pdf> (vid. pág. 154).
- [111] McCulloch, W. S. y Pitts, W. “A logical calculus of the ideas immanent in nervous activity”. En: *The bulletin of mathematical biophysics* 5.4 (1943), págs. 115-133. URL: <https://waldirbertazzijr.com/wp-content/uploads/2018/10/mcp.pdf> (vid. págs. 116, 126).
- [112] Miller, M., Sathi, C., Wiesenthal, D., Leskovec, J. y Potts, C. “Sentiment flow through hyperlink networks”. En: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 5. 1. 2011. URL: https://cs.stanford.edu/~jure/pubs/sentiflow_icwsm11.pdf (vid. págs. 2, 10).
- [113] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. y Kavukcuoglu, K. “Asynchronous methods for deep reinforcement learning”. En: *International conference on machine learning*. 2016, págs. 1928-1937. URL: <http://proceedings.mlr.press/v48/mnih16.pdf> (vid. pág. 125).
- [114] Mnih, V., Kavukcuoglu, K., Silver, D. y col. “Human-level control through deep reinforcement learning”. En: *nature* 518.7540 (2015), págs. 529-533. URL: <https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassabis15NatureControlDeepRL.pdf> (vid. pág. 86).

REFERENCIAS

- [115] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. y Riedmiller, M. “Playing atari with deep reinforcement learning”. En: *arXiv preprint arXiv:1312.5602* (2013). URL: <https://arxiv.org/pdf/1312.5602.pdf> (vid. págs. 86, 118).
- [116] Moerland, T. M., Broekens, J. y Jonker, C. M. “Emotion in reinforcement learning agents and robots: a survey”. En: *Machine Learning* 107.2 (2018), págs. 443-480. URL: <https://link.springer.com/content/pdf/10.1007/s10994-017-5666-0.pdf> (vid. pág. 153).
- [117] Monahan, G. E. “State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms”. En: *Management science* 28.1 (1982), págs. 1-16. URL: <https://pubsonline.informs.org/doi/pdf/10.1287/mnsc.28.1.1> (vid. págs. 91, 107, 134).
- [118] Moore, A. W. y Atkeson, C. G. “Prioritized sweeping: Reinforcement learning with less data and less time”. En: *Machine learning* 13.1 (1993), págs. 103-130. URL: <https://link.springer.com/content/pdf/10.1007/BF00993104.pdf> (vid. págs. 90, 105, 214).
- [119] Morgenstern, O. y Von Neumann, J. *Theory of games and economic behavior*. Princeton university press, 1953 (vid. pág. 130).
- [120] Murphy, K. P. *Probabilistic machine learning: an introduction*. MIT press, 2022 (vid. pág. 113).
- [121] Nash Jr, J. F. “Equilibrium points in n-person games”. En: *Proceedings of the national academy of sciences* 36.1 (1950), págs. 48-49. URL: <https://www.pnas.org/doi/full/10.1073/pnas.36.1.48> (vid. pág. 131).
- [122] Nash Jr, J. F. “Non-cooperative games”. En: *Annals of Mathematics* 54.2 (1951), págs. 286-295. URL: <https://www.cs.upc.edu/~ia/nash51.pdf> (vid. pág. 131).
- [123] Neumann, J., Burks, A. W. y col. *Theory of self-reproducing automata*. Vol. 1102024. University of Illinois Press Urbana, 1966. URL: <https://cdn.patentlyo.com/media/docs/2012/04/VonNeumann.pdf> (vid. pág. 7).
- [124] Ng, A. Y., Harada, D. y Russell, S. “Policy invariance under reward transformations: Theory and application to reward shaping”. En: *Icml*. Vol. 99. 1999, págs. 278-287. URL: <http://luthuli.cs.uiuc.edu/~daf/courses/games/AIpapers/ml99-shaping.pdf> (vid. pág. 154).
- [125] Nguyen, T. T., Nguyen, N. D. y Nahavandi, S. “Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications”. En: *arXiv preprint arXiv:1812.11794* (2018). URL: <https://arxiv.org/pdf/1812.11794.pdf> (vid. pág. 149).
- [126] Nguyen, T. T., Nguyen, N. D. y Nahavandi, S. “Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications”. En: *IEEE transactions on cybernetics* 50.9 (2020), págs. 3826-3839. URL: <https://ieeexplore.ieee.org/document/9043893> (vid. pág. 149).
- [127] Nowak, M. A. “Five rules for the evolution of cooperation”. En: *science* 314.5805 (2006), págs. 1560-1563. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3279745/pdf/nihms49939.pdf> (vid. pág. 7).
- [128] Nowak, M. A. y May, R. M. “Evolutionary games and spatial chaos”. En: *Nature* 359.6398 (1992), págs. 826-829. URL: https://www.researchgate.net/profile/Martin-Nowak/publication/216634494_Evolutionary_Games_and_Spatial_Chaos/links/54217b730cf274a67f/Evolutionary-Games-and-Spatial-Chaos.pdf (vid. pág. 155).

- [129] Nowak, M. A. y Sigmund, K. “Tit for tat in heterogeneous populations”. En: *Nature* 355.6357 (1992), págs. 250-253. URL: https://abel.math.harvard.edu/archive/153_fall_04/Additional_reading_material/tit_for_tat_in_heterogenous_populations.pdf (vid. pág. 155).
- [130] Nowé, A., Vrancx, P. y De Hauwere, Y.-M. “Game theory and multi-agent reinforcement learning”. En: *Reinforcement Learning*. Springer, 2012, págs. 441-470. URL: https://perso.liris.cnrs.fr/laetitia.matignon/index/14.%20Game_Theory_and_Multi-agent_Reinforcement.pdf (vid. pág. 129).
- [131] O'Neill, B. “Approaches to modelling emotions in game theory”. En: *Proceedings of the Conference on Cognition, Emotion and Rational Choice, UCLA, April*. 2000. URL: <http://www.sscnet.ucla.edu/polisci/faculty/boneill/emotions.html> (vid. pág. 153).
- [132] Ohtsuki, H., Hauert, C., Lieberman, E. y Nowak, M. A. “A simple rule for the evolution of cooperation on graphs and social networks”. En: *Nature* 441.7092 (2006), págs. 502-505. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2430087/pdf/nihms51831.pdf> (vid. págs. 151, 155).
- [133] OroojlooyJadid, A. y Hajinezhad, D. “A review of cooperative multi-agent deep reinforcement learning”. En: *arXiv preprint arXiv:1908.03963* (2019). URL: <https://arxiv.org/pdf/1908.03963.pdf> (vid. pág. 137).
- [134] Papoudakis, G., Christianos, F., Rahman, A. y Albrecht, S. V. “Dealing with non-stationarity in multi-agent deep reinforcement learning”. En: *arXiv preprint arXiv:1906.04737* (2019). URL: <https://arxiv.org/pdf/1906.04737.pdf> (vid. pág. 137).
- [135] Papoudakis, G., Christianos, F., Schäfer, L. y Albrecht, S. V. “Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks”. En: *arXiv preprint arXiv:2006.07869* (2020). URL: <https://arxiv.org/pdf/2006.07869.pdf> (vid. pág. 140).
- [136] Papoudakis, G., Christianos, F., Schäfer, L. y Albrecht, S. V. “Comparative evaluation of cooperative multi-agent deep reinforcement learning algorithms”. En: *arXiv: 2006.07869* (2020). URL: <https://arxiv.org/pdf/2006.07869.pdf> (vid. pág. 126).
- [137] Perc, M. y Szolnoki, A. “Coevolutionary games—a mini review”. En: *BioSystems* 99.2 (2010), págs. 109-125. URL: <https://arxiv.org/ftp/arxiv/papers/1206/1206.3261.pdf> (vid. pág. 156).
- [138] Perc, M. y Szolnoki, A. “Social diversity and promotion of cooperation in the spatial prisoner’s dilemma game”. En: *Physical Review E* 77.1 (2008), pág. 011904. URL: <https://arxiv.org/pdf/0708.1746.pdf> (vid. pág. 156).
- [139] Perolat, J., Leibo, J. Z., Zambaldi, V., Beattie, C., Tuyls, K. y Graepel, T. “A multi-agent reinforcement learning model of common-pool resource appropriation”. En: *arXiv preprint arXiv:1707.06600* (2017). URL: <https://arxiv.org/pdf/1707.06600.pdf> (vid. pág. 103).
- [140] Peysakhovich, A. y Lerer, A. “Consequentialist conditional cooperation in social dilemmas with imperfect information”. En: *arXiv preprint arXiv:1710.06975* (2017). URL: <https://arxiv.org/pdf/1710.06975.pdf> (vid. págs. 9, 164).
- [141] Peysakhovich, A. y Lerer, A. “Maintaining cooperation in complex social dilemmas using deep reinforcement learning”. En: (2018). URL: <https://arxiv.org/pdf/1707.01068.pdf> (vid. págs. 9, 164).

REFERENCIAS

- [142] Plutchik, R. “The nature of emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice”. En: *American scientist* 89.4 (2001), págs. 344-350. URL: https://www.academia.edu/43620307/The_Nature_of_Emotions_Plutchik_2001_ (vid. págs. 10, 152).
- [143] Poundstone, W. *Prisoner’s Dilemma/John Von Neumann, game theory and the puzzle of the bomb*. Anchor, 1993 (vid. pág. 133).
- [144] Powell, W. B. “A unified framework for stochastic optimization”. En: *European Journal of Operational Research* 275.3 (2019), págs. 795-821. URL: <https://castlelab.princeton.edu/wp-content/uploads/2018/08/Powell-EJOR-Unified-Framework-for-Stoch-Opt-Aug-16-2018.pdf> (vid. pág. 69).
- [145] Powell, W. B. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. 1st Ed. Wiley, 2022 (vid. págs. 69, 80).
- [146] Principe, J. C., Euliano, N. R. y Lefebvre, W. C. *Neural and adaptive systems: fundamentals through simulations with CD-ROM*. John Wiley & Sons, Inc., 1999 (vid. pág. 117).
- [147] Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014 (vid. pág. 105).
- [148] Quan, J., Zhou, Y., Ma, X., Wang, X. y Yang, J.-B. “Integrating emotion-imitating into strategy learning improves cooperation in social dilemmas with extortion”. En: *Knowledge-Based Systems* 233 (2021), pág. 107550. URL: https://personalpages.manchester.ac.uk/staff/jian-bo.yang/JB%20Yang%20Journal_Papers/QuanZhouMaWangYang-KBS-Emotion-Imitating%20.pdf (vid. pág. 156).
- [149] Richardson, M. y Domingos, P. “Mining knowledge-sharing sites for viral marketing”. En: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2002, págs. 61-70. URL: <https://homes.cs.washington.edu/~pedrod/papers/kdd02b.pdf> (vid. pág. 1).
- [150] Robbins, H. y Monro, S. “A Stochastic Approximation Method”. En: *The Annals of Mathematical Statistics* 22.3 (1951), págs. 400-407. URL: <https://doi.org/10.1214/aoms/1177729586> (vid. págs. 67, 71).
- [151] Rolls, E. T. *Emotion and decision-making explained*. OUP Oxford, 2014 (vid. pág. 10).
- [152] Romero, D. M., Meeder, B. y Kleinberg, J. “Differences in the mechanics of information diffusion across topics: idioms, political hashtags, and complex contagion on twitter”. En: *Proceedings of the 20th international conference on World wide web*. 2011, págs. 695-704. URL: <https://www.cs.cornell.edu/home/kleinber/www11-hashtags.pdf> (vid. pág. 10).
- [153] Rosenblatt, F. “The perceptron: a probabilistic model for information storage and organization in the brain.” En: *Psy. r.* 65.6 (1958), pág. 386. URL: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=65a83117cbcc4e6eb7c6ac5be8e61195dc84b9fc> (vid. págs. 116, 126).
- [154] Rumelhart, D. E., Hinton, G. E. y Williams, R. J. *Learning internal representations by error propagation*. Inf. téc. California Univ San Diego La Jolla Inst for Cognitive Science, 1985. URL: <https://apps.dtic.mil/sti/pdfs/ADA164453.pdf> (vid. pág. 116).

- [155] Rummery, G. A. y Niranjan, M. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994. URL: https://www.researchgate.net/profile/Mahesan_Niranjan/publication/2500611_On-Line_Q-Learning_Using_Connectionist_Systems/links/5438d5db0cf204cab1d6db0f/On-Line-Q-Learning-Using-Connectionist-Systems.pdf (vid. pág. 97).
- [156] Sadhu, A. K. y Konar, A. *Multi-Agent Coordination: A Reinforcement Learning Approach*. John Wiley & Sons, 2020 (vid. págs. 132, 148, 149).
- [157] Sandholm, T. W. y Crites, R. H. “Multiagent reinforcement learning in the iterated prisoner’s dilemma”. En: *Biosystems* 37.1-2 (1996), págs. 147-166. URL: <http://ww2.odu.edu/~jsokolow/projects/files/Multiagent%20Reinforcement%20Learning%20in%20the%20Iterated%20Prisoners%20Dilemma.pdf> (vid. pág. 149).
- [158] Santos, F. C. y Pacheco, J. M. “A new route to the evolution of cooperation”. En: *Journal of evolutionary biology* 19.3 (2006), págs. 726-733. URL: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1420-9101.2005.01063.x> (vid. págs. 151, 155).
- [159] Sastry, P. S., Phansalkar, V. V. y Thathachar, M. “Decentralized learning of Nash equilibria in multi-person stochastic games with incomplete information”. En: *IEEE Transactions on systems, man, and cybernetics* 24.5 (1994), págs. 769-777. URL: https://web.archive.org/web/20171109184644id_/https://core.ac.uk/download/pdf/11606194.pdf (vid. pág. 146).
- [160] Schaul, T., Quan, J., Antonoglou, I. y Silver, D. “Prioritized experience replay”. En: *arXiv preprint arXiv:1511.05952* (2015). URL: <https://arxiv.org/pdf/1511.05952.pdf> (vid. pág. 121).
- [161] Schelling, T. C. “Dynamic models of segregation”. En: *Journal of mathematical sociology* 1.2 (1971), págs. 143-186. URL: https://www.stat.berkeley.edu/~aldous/157/Papers/Schelling_Seg_Models.pdf (vid. pág. 7).
- [162] Schelling, T. C. *Micromotives and macrobehavior*. WW Norton & Company, 2006 (vid. pág. 7).
- [163] Schmid, K., Belzner, L., Gabor, T. y Phan, T. “Action markets in deep multi-agent reinforcement learning”. En: *International Conference on Artificial Neural Networks*. Springer. 2018, págs. 240-249. URL: https://link.springer.com/chapter/10.1007/978-3-030-01421-6_24 (vid. pág. 103).
- [164] Schulman, J., Levine, S., Abbeel, P., Jordan, M. y Moritz, P. “Trust region policy optimization”. En: *International conference on machine learning*. 2015, págs. 1889-1897. URL: <http://proceedings.mlr.press/v37/schulman15.pdf> (vid. pág. 123).
- [165] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. y Klimov, O. “Proximal policy optimization algorithms”. En: *arXiv preprint arXiv:1707.06347* (2017). URL: <https://arxiv.org/pdf/1707.06347.pdf> (vid. pág. 123).
- [166] Schwartz, H. M. *Multi-agent machine learning: A reinforcement approach*. John Wiley & Sons, 2014 (vid. págs. 148, 149).
- [167] Sen, S., Sekaran, M., Hale, J. y col. “Learning to coordinate without sharing information”. En: *AAAI*. Vol. 94. 1994, págs. 426-431. URL: <https://www.aaai.org/Papers/AAAI/1994/AAAI94-065.pdf> (vid. págs. 139, 140).

REFERENCIAS

- [168] Sequeira, P., Melo, F. S. y Paiva, A. “Emotion-based intrinsic motivation for reinforcement learning agents”. En: *International conference on affective computing and intelligent interaction*. Springer. 2011, págs. 326-336. URL: <https://facsmelo.github.io/publications/sequeirailacii.pdf> (vid. págs. 9, 155, 164).
- [169] Sequeira, P., Melo, F. S. y Paiva, A. “Learning by appraising: an emotion-based approach to intrinsic reward design”. En: *Adaptive Behavior* 22.5 (2014), págs. 330-349. URL: <https://journals.sagepub.com/doi/pdf/10.1177/1059712314543837> (vid. págs. 9, 156, 164).
- [170] Shoham, Y., Powers, R. y Grenager, T. “If multi-agent learning is the answer, what is the question?” En: *Artificial intelligence* 171.7 (2007), págs. 365-377. URL: <http://www.dklevine.com/archive/refs4122247000000001156.pdf> (vid. pág. 130).
- [171] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. y col. “Mastering the game of Go with deep neural networks and tree search”. En: *nature* 529.7587 (2016), págs. 484-489. URL: <http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15780-s16/www/AlphaGo.nature16961.pdf> (vid. pág. 86).
- [172] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. y col. “Mastering the game of go without human knowledge”. En: *nature* 550.7676 (2017), págs. 354-359. URL: https://discovery.ucl.ac.uk/id/eprint/10045895/1/agz_unformatted_nature.pdf (vid. pág. 86).
- [173] Singh, S., Lewis, R. L. y Barto, A. G. “Where do rewards come from”. En: *Proceedings of the annual conference of the cognitive science society*. Cognitive Science Society. 2009, págs. 2601-2606. URL: https://all.cs.umass.edu/pubs/2009/singh_l_b_09.pdf (vid. pág. 154).
- [174] Singh, S., Lewis, R. L., Barto, A. G. y Sorg, J. “Intrinsically motivated reinforcement learning: An evolutionary perspective”. En: *IEEE Transactions on Autonomous Mental Development* 2.2 (2010), págs. 70-82. URL: <http://www-personal.umich.edu/~rickl/pubs/singh-lewis-barto-2010-ieee.pdf> (vid. pág. 154).
- [175] Singh, S. P., Jaakkola, T. y Jordan, M. I. “Learning without state-estimation in partially observable Markovian decision processes”. En: *Machine Learning Proceedings 1994*. Elsevier, 1994, págs. 284-292. URL: <https://web.eecs.umich.edu/~baveja/Papers/ML94.pdf> (vid. pág. 154).
- [176] Singh, S. P., Kearns, M. J. y Mansour, Y. “Nash Convergence of Gradient Dynamics in General-Sum Games.” En: *UAI*. 2000, págs. 541-548. URL: https://www.researchgate.net/profile/Satinder-Singh-3/publication/234140138_Nash_Convergence_of_Gradient_Dynamics_in_Iterated_General-Sum_Games/links/55ad059508ae98e661a2ade1/Nash-Convergence-of-Gradient-Dynamics-in-Iterated-General-Sum-Games.pdf (vid. págs. 139-141).
- [177] Smith, J. M. “Game theory and the evolution of behaviour”. En: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 205.1161 (1979), págs. 475-488 (vid. pág. 7).
- [178] Stella, M., Ferrara, E. y De Domenico, M. “Bots increase exposure to negative and inflammatory content in online social systems”. En: *Proceedings of the National Academy of Sciences* 115.49 (2018), págs. 12435-12440. URL: <https://www.pnas.org/content/pnas/115/49/12435.full.pdf> (vid. pág. 1).

- [179] Suematsu, N. y Hayashi, A. “A multiagent reinforcement learning algorithm using extended optimal response”. En: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: Part 1*. 2002, págs. 370-377. URL: https://www.researchgate.net/profile/Nobuo-Suematsu/publication/221454885_A_multiagent_reinforcement_learning_algorithm_using_extended_optimal_response/links/00b495331616ee577e000000/A-multiagent-reinforcement-learning-algorithm-using-extended-optimal-response.pdf (vid. págs. 139, 140).
- [180] Sutton, R. S. “Learning to predict by the methods of temporal differences”. En: *Machine learning* 3.1 (1988), págs. 9-44. URL: <https://link.springer.com/content/pdf/10.1007/BF00115009.pdf> (vid. pág. 98).
- [181] Sutton, R. S. y Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018. URL: <http://incompleteideas.net/book/RLbook2020.pdf> (vid. págs. 53, 71, 86).
- [182] Sutton, R. S., McAllester, D. A., Singh, S. P. y Mansour, Y. “Policy gradient methods for reinforcement learning with function approximation”. En: *Advances in neural inf. proc. systems*. 2000, págs. 1057-1063. URL: <https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf> (vid. págs. 99, 123).
- [183] Szabó, G. y Fath, G. “Evolutionary games on graphs”. En: *Physics reports* 446.4-6 (2007), págs. 97-216. URL: <https://arxiv.org/pdf/cond-mat/0607344.pdf> (vid. pág. 156).
- [184] Szolnoki, A., Xie, N.-G., Wang, C. y Perc, M. “Imitating emotions instead of strategies in spatial games elevates social welfare”. En: *Europhysics Letters* 96.3 (2011), pág. 38002. URL: <https://arxiv.org/pdf/1109.1712.pdf> (vid. págs. 9, 156, 164).
- [185] Tan, M. “Multi-agent reinforcement learning: Independent vs. cooperative agents”. En: *Proceedings of the tenth international conference on machine learning*. 1993, págs. 330-337. URL: <https://web.media.mit.edu/~cynthiab/Readings/tan-MAS-reinfLearn.pdf> (vid. pág. 137).
- [186] Tesauro, G. “Extending Q-learning to general adaptive multi-agent systems”. En: *Advances in neural information processing systems* 16 (2003), págs. 871-878. URL: <https://proceedings.neurips.cc/paper/2003/file/e71e5cd119bbc5797164fb0cd7fd94a4-Paper.pdf> (vid. págs. 139, 140).
- [187] Tesauro, G. “TD-Gammon, a self-teaching backgammon program, achieves master-level play”. En: *Neural computation* 6.2 (1994), págs. 215-219. URL: <https://www.aaai.org/Papers/Symposia/Fall/1993/FS-93-02/FS93-02-003.pdf> (vid. pág. 86).
- [188] Thrun, S. y Schwartz, A. “Issues in using function approximation for reinforcement learning”. En: *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*. Vol. 6. 1993, págs. 1-9. URL: https://www.ri.cmu.edu/pub_files/pub1/thrun_sebastian_1993_1/thrun_sebastian_1993_1.pdf (vid. pág. 120).
- [189] Tian, Y., Gong, Q., Shang, W., Wu, Y. y Zitnick, C. L. “Elf: An extensive, lightweight and flexible research platform for real-time strategy games”. En: *Advances in Neural Information Processing Systems* 30 (2017). URL: <https://arxiv.org/pdf/1707.01067.pdf> (vid. pág. 164).
- [190] Trivers, R. L. “The evolution of reciprocal altruism”. En: *The Quarterly review of biology* 46.1 (1971), págs. 35-57. URL: <https://pdodds.w3.uvm.edu/files/papers/others/1971/trivers1971a.pdf> (vid. pág. 155).

REFERENCIAS

- [191] Tsitsiklis, J. N. “Asynchronous stochastic approximation and Q-learning”. En: *Machine learning* 16.3 (1994), págs. 185-202. URL: <https://www.mit.edu/~jnt/Papers/J052-94-jnt-q.pdf> (vid. págs. 69, 79, 80, 117).
- [192] Van Hasselt, H., Guez, A. y Silver, D. “Deep reinforcement learning with double q-learning”. En: *arXiv preprint arXiv:1509.06461* (2015). URL: <https://arxiv.org/pdf/1509.06461.pdf> (vid. pág. 120).
- [193] Vosoughi, S., Roy, D. y Aral, S. “The spread of true and false news online”. En: *Science* 359.6380 (2018), págs. 1146-1151. URL: <https://science.sciencemag.org/content/359/6380/1146/tab-pdf> (vid. pág. 1).
- [194] Wang, L., Ye, S.-Q., Cheong, K. H., Bao, W. y Xie, N.-g. “The role of emotions in spatial prisoner’s dilemma game with voluntary participation”. En: *Physica A: Statistical Mechanics and its Applications* 490 (2018), págs. 1396-1407. URL: <https://www.sciencedirect.com/science/article/abs/pii/S0378437117307665> (vid. pág. 156).
- [195] Wang, W., Hao, J., Wang, Y. y Taylor, M. “Towards cooperation in sequential prisoner’s dilemmas: a deep multiagent reinforcement learning approach”. En: *arXiv preprint arXiv:1803.00162* (2018). URL: <https://arxiv.org/pdf/1803.00162.pdf> (vid. pág. 156).
- [196] Wang, X. y Sandholm, T. “Reinforcement learning to play an optimal Nash equilibrium in team Markov games”. En: *Advances in neural inf. processing systems* 15 (2002), págs. 1603-1610. URL: <https://homes.luddy.indiana.edu/xw7/papers/wang2002reinforcement.pdf> (vid. págs. 138, 140).
- [197] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M. y Freitas, N. “Dueling network architectures for deep reinforcement learning”. En: *International conference on machine learning*. 2016, págs. 1995-2003. URL: <http://proceedings.mlr.press/v48/wangf16.pdf> (vid. pág. 122).
- [198] Watkins, C. J. y Dayan, P. “Q-learning”. En: *Machine learning* 8.3 (1992), págs. 279-292. URL: <https://link.springer.com/content/pdf/10.1007/BF00992698.pdf> (vid. pág. 75).
- [199] Weinberg, M. y Rosenschein, J. S. “Best-response multiagent learning in non-stationary environments”. En: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems- Volume 2*. 2004, págs. 506-513. URL: <http://www2.odu.edu/~jsokolow/projects/files/Best-Response%20Multiagent%20Learning%20in%20Non-Stationary%20Environments.pdf> (vid. págs. 139, 140).
- [200] Williams, R. J. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. En: *Machine learning* 8.3-4 (1992), págs. 229-256. URL: <https://link.springer.com/content/pdf/10.1007/BF00992696.pdf> (vid. pág. 99).
- [201] Wu, S., Tan, C., Kleinberg, J. y Macy, M. W. “Does bad news go away faster?” En: *Fifth International AAAI Conference on Weblogs and Social Media*. 2011. URL: <https://www.cs.cornell.edu/home/kleinber/icwsm11-longevity.pdf> (vid. pág. 10).
- [202] Wu, Y., Mansimov, E., Grosse, R. B., Liao, S. y Ba, J. “Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation”. En: *Advances in neural information processing systems*. 2017, págs. 5279-5288. URL: <https://arxiv.org/pdf/1708.05144.pdf> (vid. pág. 123).
- [203] Wunder, M., Littman, M. L. y Babes, M. “Classes of multiagent q-learning dynamics with epsilon-greedy exploration”. En: *ICML*. 2010. URL: <https://icml.cc/Conferences/2010/papers/191.pdf> (vid. pág. 155).

- [204] Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W. y Wang, J. “Mean field multi-agent reinforcement learning”. En: *International Conference on Machine Learning*. PMLR. 2018, págs. 5571-5580. URL: <http://proceedings.mlr.press/v80/yang18d/yang18d.pdf> (vid. pág. 135).
- [205] Yu, C., Zhang, M., Ren, F. y Tan, G. “Emotional multiagent reinforcement learning in spatial social dilemmas”. En: *IEEE transactions on neural networks and learning systems* 26.12 (2015), págs. 3083-3096. URL: https://www.researchgate.net/profile/Minjie-Zhang-4/publication/273463080_Emotiona...Multiagent_Reinforcement_Learning_in_Spatial_Social_Dilemmas/links/552309bc0cf29dcabb0ee0c7/Emotional-Multiagent-Reinforcement-Learning-in-Spatial-Social-Dilemmas.pdf (vid. págs. 9, 156).
- [206] Yuan, Y., Guo, T., Zhao, P. y Jiang, H. “Adherence Improves Cooperation in Sequential Social Dilemmas”. En: *Applied Sciences* 12.16 (2022), pág. 8004. URL: <https://www.mdpi.com/2076-3417/12/16/8004> (vid. pág. 155).
- [207] Zafarani, R., Cole, W. D. y Liu, H. “Sentiment propagation in social networks: a case study in livejournal”. En: *International Conference on Social Computing, Behavioral Modeling, and Prediction*. Springer. 2010, págs. 413-420. URL: https://www.researchgate.net/profile/Huan-Liu-51/publication/221118290_Sentiment_Propagation_in_Social_Networks_A_Case_Study_in_LiveJournal/links/00b495350968dcc61000000/Sentiment-Propagation - in - Social - Networks - A - Case - Study - in - LiveJournal .pdf (vid. pág. 10).
- [208] Zhang, Z., Zhao, D., Gao, J., Wang, D. y Dai, Y. “FMRQ—A multiagent reinforcement learning algorithm for fully cooperative tasks”. En: *IEEE transactions on cybernetics* 47.6 (2016), págs. 1367-1379 (vid. pág. 140).
- [209] Zinkevich, M. “Online convex programming and generalized infinitesimal gradient ascent”. En: *Proceedings of the 20th international conference on machine learning (icml-03)*. 2003, págs. 928-936. URL: <https://www.aaai.org/Papers/ICML/2003/ICML03-120.pdf> (vid. págs. 139, 140).

REFERENCIAS

Índice alfabético

A

- advantage 75
Algoritmos estocásticos iterativos 67
Algoritmos estocásticos iterativos II .. 68, 69

B

- Bellman, Richard E. 27
operador de Bellman, 31

C

- coeficientes de elegibilidad 73
convergencia
 algoritmo de Iteración de Estrategias, 57
 algoritmo de Iteración de Valor, 52
 algoritmo de Programación Dinámica, 37

D

- diferencia temporal 70
Diferencias Temporales 94

F

- funciones
 representación mediante aproximadores
 funcionales, 21, 158

representación tabular, 21, 158

J

- Juegos matriciales 133

M

- Modelos de Difusión de Información 151

N

- norma 38
norma máxima 67
norma máxima ponderada 67

P

- pseudocontracción 67
punto fijo 38

T

- TD véase *Diferencias Temporales*
Teoría de Juegos 131

V

- ventaja véase advantage

ÍNDICE ALFABÉTICO

Listado de Símbolos y Acrónimos

Conjuntos

\mathbb{R}	Números Reales
\mathbb{N}^+	Números Naturales positivos
\mathcal{S}	Conjunto de Estados
\mathcal{A}	Conjunto de acciones
Π	Conjunto de estrategias
\mathcal{M}	Conjunto de estrategias estacionarias
\mathcal{N}	Conjunto de agentes
\mathbf{S}	Conjunto de estados en un problema multiagente ($S = S_1 \times S_2 \times \dots \times S_n$)
\mathbf{A}	Conjunto de acciones en un problema multiagente ($A = A_1 \times A_2 \times \dots \times A_n$)

Símbolos

\leftarrow	Asignación de valor a una variable
$:=$	Asignación de valor en un proceso iterativo
s	Estado de partida en un Proceso de Decisiones de Markov ($s \in \mathcal{S}$)
s'	Estado siguiente en una transición de un Proceso de Decisiones de Markov ($s' \in \mathcal{S}$)
a	Acción ($a \in \mathcal{A}$)
r	Recompensa
\mathcal{D}	<i>Replay Buffer</i> (Colección de observaciones, generalmente en la forma: (s,a,r,s'))
X	Variable aleatoria (Con índice temporal: X_t)
$X \sim f(x)$	Variable aleatoria X con distribución de probabilidad $f(x)$
$\mathbb{E}[X]$	Esperanza matemática de la variable aleatoria X
S_t	Variable aleatoria que representa el estado ocupado en un cierto instante temporal 't' ($S_t = s$, $s \in \mathcal{S}$).
A_t	Variable aleatoria que representa la acción llevada a cabo por el agente en un instante 't' ($A_t = a$, $a \in \mathcal{A}$).

LISTA DE SÍMBOLOS Y ACRÓNIMOS

R_t	Variable aleatoria que representa la recompensa recibida por el agente en un instante 't' ($R_t = r$, $r \in \mathbb{R}$).
γ	Factor de descuento ($\gamma \in [0, 1]$)
α	Tasa de aprendizaje ($\alpha \in [0, 1]$)
$p(s, a, r, s')$	Probabilidad de que en el estado 's' se lleve a cabo la acción 'a' y se reciba como recompensa 'r', produciéndose una transición a otro estado 's'.
$\mathcal{P}_{s,s'}^a$	Probabilidad de pasar de un estado 's' a otro 's'' tras llevar a cabo la acción 'a'
$\arg \min_x f(x)$	Valor de x para el que la función $f(x)$ obtiene el menor valor
$\arg \max_x f(x)$	Valor de x para el que la función $f(x)$ obtiene el mayor valor
\mathcal{F}_t	Historia pasada hasta el instante t en un Algoritmo Estocástico Iterativo. Es decir: $\mathcal{F}_t = \{r_0(i), \dots, r_t(i), w_0(i), \dots, w_{t-1}(i), \gamma_0(i), \dots, \gamma_t(i), \quad i = 1, \dots, n\}$
$\ \cdot\ _\xi$	Norma máxima ponderada
$\ \cdot\ _\infty$	Norma máxima
θ	Vector de parámetros (de una Red Neuronal)
π	Política o estrategia ('policy')
π^*	Política o estrategia ('policy') óptima
$\pi(s)$	Función de estrategia determinista
$\pi(a s)$	Función de estrategia estocástica
$v_\pi(s)$	Función de estado. Proporciona el valor verdadero de los estados ($s, s \in \mathcal{S}$) cuando se sigue la estrategia π .
$\hat{v}_\pi(s, \mathbf{w})$	Función de estado empleando un estimador funcional (por ejemplo una Red Neuronal). Proporciona el valor aproximado de los estados ($s, s \in \mathcal{S}$) cuando se sigue la estrategia π . Los parámetros se denotan por: \mathbf{w} .
$V^\pi(s)$	Función de estado estimada [Algunos autores utilizan la notación: $\hat{V}^\pi(s)$ para la estimación y $V^\pi(s)$ para el valor verdadero de la función]
$q_\pi(s, a)$	Función de estado-acción. Proporciona el valor verdadero que corresponde a estar en un determinado estado ($s, s \in \mathcal{S}$) y tomar una decisión ($a, a \in \mathcal{A}$), y a partir de ahí actuar en base a lo que determina la estrategia en consideración π .
$Q^\pi(s, a)$	Función de estado-acción estimada [Algunos autores utilizan la notación: $\hat{Q}^\pi(s, a)$ para la estimación y $Q^\pi(s, a)$ para el valor verdadero de la función]
$Q_\theta^\pi(s, a)$	Función de estado-acción estimada (aproximación mediante una Red Neuronal con vector de parámetros θ)
$A^\pi(s, a)$	Ventaja ('advantage'). Ventaja de tomar la decisión 'a' en el estado 's' cuando se sigue la estrategia π . Representa la mejora obtenida mediante la elección de

una acción determinada ($a, a \in \mathcal{A}$) en un cierto estado ($s, s \in \mathcal{S}$) respecto del promedio obtenido cuando se tienen en cuenta todas las acciones posibles y se sigue una estrategia π .

$\phi(s)$	Características ('features') asociadas al estado en cuestión ($s, s \in \mathcal{S}$) $\phi(s) = (\phi_1(s), \phi_2(s), \dots, \phi_d(s)) \in \mathbb{R}^d$
$\psi^\pi(s, a)$	'Successor features' asociadas al par estado-acción: s y a ($s \in \mathcal{S}, a \in \mathcal{A}$), cuando se sigue la estrategia π

ACRÓNIMOS

<i>A2C</i>	Advantage actor-critic
<i>A3C</i>	Asynchronous advantage actor-critic
<i>AC</i>	Actor-Critic
<i>ACKTR</i>	Actor-critic using Kronecker-factored trust region
<i>AI</i>	Artificial Intelligence
<i>DDPG</i>	Deep Deterministic Policy Gradient
<i>DPG</i>	Deterministic policy gradient
<i>DQN</i>	Deep Q-Network
<i>GAN</i>	Generative Adversarial Network
<i>MC</i>	Monte Carlo
<i>MCTS</i>	Monte Carlo Tree Search
<i>MDP</i>	Markov Decision Process
<i>ML</i>	Machine Learning
<i>PE</i>	Policy Evaluation
<i>PI</i>	Policy Improvement
<i>PPO</i>	Proximal Policy Optimization
<i>QED</i>	Quod erat demonstrandum (<i>que era lo que se quería demostrar</i>)
<i>RL</i>	Reinforcement Learning
<i>SAC</i>	Soft Actor-Critic
<i>SPG</i>	Stochastic Policy Gradient
<i>TD3</i>	Twin Delayed Deep Deterministic Policy Gradient
<i>TD</i>	Temporal Difference
<i>TRPO</i>	Trust Region Policy Optimization

LISTA DE SÍMBOLOS Y ACRÓNIMOS

Glosario

Advantage Representa la mejora obtenida mediante la elección de una determinada acción en un cierto estado respecto del promedio obtenido cuando se tienen en cuenta todas las acciones posibles.

Asíncrono vs Síncrono Asíncrono (en contraposición a síncrono) alude a que los pasos en las estimaciones no es necesario que se mantengan en un orden fijo de iteración. Esto permite paralelizar en diferentes máquinas el algoritmo más fácilmente. Ver libro de Sutton para otra forma de explicarlo: pag. 191.

Baseline Es el punto de referencia que se toma en algunos algoritmos.

Batch (lote) Conjunto de datos utilizados en cada pasada del algoritmo. En el proceso de entrenamiento es la unidad mínima de datos en que quedan divididos los datos de entrenamiento, y que se pasan al algoritmo para el ajuste del modelo en cada pasada.

Behavior policy Ver: *Target Policy vs Behavior policy*.

Bootstrapping Consiste en el procedimiento iterativo de actualizar una estimación en base a otra estimación previa. (Ver página 89 del libro de Barto y Sutton).

Control Ver: *Prediction vs Control* En un proceso de Aprendizaje por Refuerzo alude al proceso de optimización que conduce al descubrimiento de la estrategia óptima de acción.

Diferencias Temporales (TD) Es un método de aprendizaje de las funciones de valor, que utiliza como regla de actualización la diferencia entre estimaciones en dos momentos consecutivos.

Eligibility traces Procedimiento que funciona como un mecanismo de memoria a corto plazo, que permite introducir una secuencia de pasos anteriores en los algoritmos. Suele contribuir a llevar a cabo un aprendizaje más eficiente. Generalmente proporciona mejores resultados que otros métodos con utilidad parecida, como las Diferencias Temporales a 'n' pasos (*n-step TD*).

Entorno Estacionario En un MDP (*Markov Decision Process*) alude a que las probabilidades de transición entre los estados se mantienen estables. En caso contrario la convergencia (de los algoritmos de aprendizaje de la estrategia óptima) no queda garantizada.

Entrenamiento (Training) Proceso mediante el cual un modelo se ajusta a unos datos, a partir de la estimación de los parámetros óptimos (aquellos que conducen a un menor error en la predicción).

Epoch Alude a cada pasada completa por los datos de entrenamiento. En un proceso de aprendizaje la base de datos se encuentra dividida en '*lotes*', y se le pasan en estos bloques al modelo para el aprendizaje.

Estrategia de Boltzman Forma de caracterizar la distribución de probabilidad con la que un agente reacciona a su entorno. A cada par estado-acción (s, a) , dado s , se le asigna la probabilidad de que la acción seleccionada sea a . Obedece a la siguiente expresión:

$$\pi(s, a) = \frac{e^{\frac{Q(s, a)}{T}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s, a')}{T}}}$$

T es un parámetro llamado '*temperatura*' ($T > 0$) que controla la aleatoriedad de la exploración del entorno. Cuando $T \rightarrow 0$ se acerca a la selección de acciones mediante el método '*greedy*'. Cuando $T \rightarrow \infty$ la selección de acciones por parte del agente es puramente aleatoria. En el rango $T \in (0, \infty)$ los valores altos de $Q(s, a)$ tienen más probabilidad de ser seleccionados frente a los bajos .

Estrategia Epsilon-greedy (ϵ -greedy) Consiste en elegir con una pequeña probabilidad (ϵ -greedy) una acción que no es la óptima de una manera aleatoria uniforme, y con la probabilidad complementaria la acción que conduce a un mayor valor de $Q(s, a)$. Es decir:

$$\pi(s, a) = \begin{cases} 1 - \epsilon & a = \arg \max_{a'} Q(s, a') \\ \epsilon & a \neq \arg \max_{a'} Q(s, a') \end{cases}$$

El valor de ϵ vendría a representar la probabilidad de exploración del agente. El resultado con este método es que '*explota*' el conocimiento acumulado que viene recogido en $Q(s, a)$ la mayor parte del tiempo, pero con una pequeña probabilidad ϵ '*explora*' o tantea nuevos cursos de acción al azar .

Estrategia greedy (Voraz) Consiste en seleccionar en un determinado estado s , sólo aquella acción que conduce al mejor resultado en la función de valor (por ejemplo: $Q(s, a)$ si empleamos la función de estado-acción).

Exploration vs Explotation Explotación consiste en comportarse eligiendo las acciones que al punto actual tienen un valor estimado mayor (las acciones con mayor valor reciben en nombre de acciones '*greedy*'). Cuando elegimos una acción que no-greedy decimos que estamos Explorando, ya que esto nos permite trabajar más la estimación de estas acciones (pag. 26 Sutton).

Juego Certo estereotipo de interacción que modeliza de manera formal las relaciones entre decisores, y con los que se pretende analizar matemáticamente la forma óptima de tomar decisiones. Empleados en Teoría de Juegos.

Juego Estático vs Juego Dinámico Juego estático es aquel en el que se analizan exclusivamente las estrategias de los jugadores si entrar en valoración la existencia de un '*estado*'. En los juegos dinámicos el comportamiento del agente está muy ligado a un estado particular.

Juegos de Suma Cero Son juegos en los que las ganancias de un jugador suponen las pérdidas de su oponente.

Juegos de Suma General En los juegos de suma general no se impone ninguna restricción sobre la forma de las recompensas del juego.

Maldición de la Dimensionalidad (*Curse of Dimensionality*) Alude al crecimiento exponencial de opciones a valorar en un problema de Aprendizaje por Refuerzo conforme crece el número de estados-acciones a tener en cuenta. El problema surge de tener que arrastrar una representación explícita de cada uno de estos valores (*representación tabular*). Este obstáculo puede ser sorteado empleando aproximadores funcionales (*representación compacta*) como las redes neuronales.

Monte Carlo Tree Search (MCTS) Ver el libro de Sutton pag. 185.

Montecarlo (MC) En el ámbito del Aprendizaje por Refuerzo (*Reinforcement Learning*) alude al método de aprendizaje, para la estimación de las funciones de valor, que no requiere conocimiento del entorno, y sólo se basa en secuencias muestrales generadas por simulación del tipo: estado, acción y recomensa.

Off-line Alude al proceso en el que los datos se disponen de antemano, como la secuencia de ganancias de todo el episodio que se va a analizar.

On-line Alude al proceso en el que los datos se generan sobre la marcha. [Libro de Bertsekas (Rollout) pag. 54].

On-policy vs off-policy Un método de búsqueda de la estrategia óptima se dice '*on-policy*' cuando el comportamiento del agente se genera empleando la misma estrategia cuya mejora se está realizando iterativamente. En los métodos '*off-policy*' la estrategia que se mejora es distinta de la que emplea el agente para tomar decisiones. [Ver también: Target policy vs Behavior policy] pag. 202 Ravichandiran.

Overfitting (Sobreajuste) Ocurre cuando la capacidad de ajuste del modelo es muy superior a la complejidad de los datos que se le proporcionan para el entrenamiento. Esto provoca que el error en el entrenamiento sea muy bajo, y sin embargo cuando se le proporcionan datos que no ha visto el modelo durante el entrenamiento (*datos de validación*) el error sea muy alto. En este caso la capacidad de generalización del modelo es muy baja.

Policy Evaluation Texto de prueba.

Policy Improvement Texto de prueba.

Prediction vs Control Predicción alude a la utilización de los algoritmos para encontrar el valor asociado a una determinada estrategia. Control alude a la utilización de los algoritmos para encontrar la estrategia óptima (la que tiene el mayor valor de todas).

Programación Dinámica Texto de prueba.

Rollout (Ver libro de Bertsekas), ver tb libro de Sutton pag. 183.

Sweep (*barrido*) En Aprendizaje por Refuerzo (*Reinforcement Learning*) consiste en la aplicación del operador de Bellman a todos los estados. En algunos algoritmos se puede priorizar el proceso en unos estados sobre otros ('*prioritized sweeping*' - ver Moore, A. & Atkeson, C. (1993) [118]).

Target móvil Alude a los problemas de aprendizaje que surgen especialmente en entornos multiagente (MARL - *Multi-agent Reinforcement Learning*), debido a que la estrategia óptima va cambiando según cambia la estrategia de cada agente, al encontrarse todos ellos en situación de aprendizaje. Esto se traduce en que el entorno se vuelve no estacionario, con lo que las garantías de convergencia de muchos algoritmos no se mantienen.

Target Policy vs Behavior policy La estrategia que está siendo aprendida recibe el nombre de *Target policy*, mientras que la estrategia que se emplea para generar comportamiento recibe en la literatura el nombre de *Behavior policy*. Los métodos en que ambas coinciden reciben el nombre de '*on-policy*', mientras que si son distintas se llaman '*off-policy*'.

Underfitting Ocurre cuando la capacidad explicativa de un modelo ajustado es insuficiente para explicar y predecir correctamente el fenómeno que estamos estudiando.