VNIVERSITAT DE VALÈNCIA

Aprendizaje por refuerzo en espacios continuos

Algoritmos y aplicación al tratamiento de la anemia renal

Pablo Escandell Montero

2014 | Tesis doctoral

TESIS DOCTORAL

Aprendizaje por refuerzo en espacios continuos: algoritmos y aplicación al tratamiento de la anemia renal

Autor

Pablo Escandell Montero

Directores

Dr. José David Martín Guerrero
Dr. Emilio Soria Olivas

Departament d'Enginyeria Electrònica
UNIVERSITAT DE VALÈNCIA – ESTUDI GENERAL
Valencia – Abril, 2014

Aprendizaje por refuerzo en espacios continuos: algoritmos y aplicación al tratamiento de la anemia renal

Pablo Escandell Montero, abril de 2014

VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria

Departament d´Enginyeria Electrònica

Departament d'Enginyeria Electrònica
Escola Tècnica Superior d'Enginyeria

D. JOSÉ DAVID MARTÍN GUERRERO, Doctor por la Universitat de València, Profesor Titular del Departamento de Ingeniería Electrónica de la Escola Tècnica Superior d'Enginyeria de la Universitat de València, y

D. EMILIO SORIA OLIVAS, Doctor en Ingeniería Electrónica por la Universitat de València, Profesor Titular del Departamento de Ingeniería Electrónica de la Escola Tècnica Superior d'Enginyeria de la Universitat de València

HACEN CONSTAR QUE:

El Ingeniero en Electrónica D. Pablo Escandell Montero ha realizado bajo nuestra dirección el trabajo titulado "Aprendizaje por refuerzo en espacios continuos: algoritmos y aplicación al tratamiento de la anemia renal", que se presenta en esta memoria para optar al grado de Doctor.

Y para que así conste a los efectos oportunos, firmamos el presente certificado, en Valencia, a _____ .

José D. Martín Guerrero	Emilio Soria Olivas	J. Rafael Magdalena Benedito Dir. del Departamento

Tesis Doctoral: APRENDIZAJE POR REFUERZO EN ESPACIOS CONTINUOS: ALGORITMOS Y APLICACIÓN AL TRATAMIENTO DE LA ANEMIA RENAL

Autor: PABLO ESCANDELL MONTERO

Directores: Dr. JOSÉ DAVID MARTÍN GUERRERO
Dr. EMILIO SORIA OLIVAS

El tribunal nombrado para juzgar la Tesis Doctoral arriba citada, compuesto por los doctores:

Presidente: _____

Vocal: _____

Secretario: _____

Acuerda otorgarle la calificación de _____

Y para que así conste a los efectos oportunos, firmamos el presente certificado.

Valencia, a

Esta tesis doctoral ha sido parcialmente financiada por *Fresenius Medical Care (FME)* a través del proyecto de investigación titulado *"Intelligent analysis of Fresenius Medical Care data"*.

Agradecimientos

En primer lugar, me gustaría dar las gracias a mis directores, José D. Martín Guerrero y Emilio Soria Olivas. Sin ninguna duda su aportación ha sido de vital importancia para la realización de esta tesis. Desde el comienzo han depositado una gran confianza en mí y me han demostrado su apoyo día tras día. Gracias por vuestro excelente trabajo como directores, por vuestros consejos e ideas y, en definitiva, por todo el tiempo que habéis invertido en mí.

En algunas partes de la tesis han colaborado activamente varios miembros la empresa Fresenius Medical Care. El interés mostrado por los resultados alcanzados así como la posibilidad de aplicar parte de la metodología desarrollada en pacientes reales han sido sin duda una motivación adicional. Concretamente quiero darle las gracias a Milena Chermisi, Flavio Mari y Carlo Barbieri, pertenencientes al departamento *Healthcare and Business Advanced Modeling*, por aportar su conocimiento sobre la enfermedad renal crónica y por permitirme trabajar con los datos recogidos en las clínicas de Fresenius Medical Care.

Quisiera agradecer también la ayuda que me han proporcionado el resto de compañeros del grupo de investigación IDAL (*Intelligent Data Analysis Laboratory*): Juan Gómez, Rafa Magdalena, José María Martínez, Marcelino Martínez, Antonio Serrano y Joan Vila. A lo largo de este tiempo me han acogido como uno más del grupo, haciéndome partícipe de un fantástico e inspirador ambiente de trabajo. Especialmente a José María, con el que he tenido la suerte de trabajar más estrechamente día a día y quien siempre ha estado ahí para echarme una mano.

Gracias a todas las personas de mi alrededor que en algún momento han mostrado interés y preocupación por el desarrollo de esta tesis. Esos pequeños detalles y palabras de ánimo significan mucho para mí.

Por último, y no por ello menos importante, quiero expresar mi más profundo agradecimiento a Delia Lorente por su constante apoyo y comprensión. Además de su inmensa ayuda en el plano personal, su aportación en el plano técnico ha sido totalmente desinteresada y de un valor incalculable.

A todos vosotros, ¡muchísimas gracias!

Pablo Escandell Montero
Valencia, abril de 2014

A mis padres
y a mi hermano

Índice general

Resumen

El aprendizaje por refuerzo es un paradigma de aprendizaje automático orientado a la resolución de problemas de decisión secuenciales. Este tipo de problemas aparece en aplicaciones pertenecientes a campos tan diversos como control automático, medicina, investigación operativa o economía. Los algoritmos clásicos de aprendizaje por refuerzo están fundamentados en la teoría matemática de la programación dinámica, donde se asume que el espacio de estados es discreto y se compone de un número manejable de estados. Desafortunadamente, en la mayoría de aplicaciones de interés práctico el espacio de estados es continuo, por lo que los algoritmos clásicos dejan de ser útiles. Para poder aplicar el aprendizaje por refuerzo en espacios continuos se requiere, por una parte, generalizar el comportamiento aprendido a partir de un conjunto limitado de experiencias a casos que no se hayan experimentado previamente y, por otra parte, representar las políticas de forma compacta. Ambos requisitos han sido ampliamente estudiados en el campo del aprendizaje supervisado, donde a menudo se necesita aproximar una función continua a partir de un conjunto de puntos discretos. La combinación de algoritmos de aprendizaje por refuerzo con técnicas de aproximación de funciones es actualmente un área de investigación activa. A pesar de los avances logrados en los últimos años, todavía hay aspectos que limitan la capacidad del aprendizaje por refuerzo en problemas complejos. Entre ellos destacan la escasa capacidad de escalabilidad a espacios definidos por un número elevado de dimensiones y la elevada cantidad de datos necesarios para aprender políticas útiles. En esta tesis doctoral se proponen algoritmos de aprendizaje por refuerzo enfocados a mejorar estos dos aspectos. Los resultados obtenidos en diversos experimentos demuestran que los algoritmos propuestos suponen un avance hacia métodos de aprendizaje por refuerzo más prácticos y efectivos en problemas complejos. Además de las aportaciones teóricas se ha desarrollado un sistema basado en aprendizaje por refuerzo para la optimización del tratamiento de la anemia asociada a la enfermedad renal crónica.

Abstract

Reinforcement learning is a machine learning paradigm aimed at solving sequential decision making problems. This kind of problems is commonly encountered in areas such as automatic control, medicine, operative research or economy. Classical reinforcement learning algorithms rely on the mathematical theory of dynamic programming, where it is assumed that the state space is discrete and it is composed by a reduced number of states. Unfortunately, in most of the practical applications, the classical algorithms are not useful because the state space is continuous. In order to apply reinforcement learning in continuous spaces is necessary, on the one hand, to generalize the behaviour learned from a limited set of experiences to previously unseen cases and, on the other hand, to represent the policies in a compact way. Both requirements have been widely studied in the supervised learning field, where it is common to approximate a continuous function from a set of discrete points. The combination of reinforcement learning algorithms with function approximation is currently an active field of research. In spite of significant advances over the last years, there are still many issues that limit the ability of reinforcement learning methods in complex domains. Prominent among them are the poor scalability and the high amount of data required to learn useful policies. This thesis proposes several reinforcement learning algorithms intended for improving those two issues. The results obtained in the experiments show that the proposed algorithms represent an important step forward toward more practical and effective methods in complex domains. In addition to the theoretical contributions, this thesis also shows a system based on reinforcement learning aimed to optimize the treatment of patients with secondary anemia to chronic kidney disease.

Acrónimos y abreviaturas

AEE Agente estimulante de la eritropoyesis

AHC *Adaptive heuristic critic*

CERA *Continuous erythropoietin receptor activator*

CMAC *Cerebellar model articulation controller*

DC *Direct current*

DP *Dynamic programming*

ELM *Extreme learning machine*

EPO Eritropoyetina

ER *Experience replay*

ERC Enfermedad renal crónica

ERT Enfermedad renal terminal

FIFO *First-input first-output*

FNAC *Fitted natural actor-critic*

FQI *Fitted Q-iteration*

FVI *Fitted value iteration*

GPI *Generalized policy iteration*

GTD *Gradient temporal difference*

Hb Hemoglobina

HVI Hipertrofia ventricular izquierda

i.i.d. *Independiente e idénticamente distribuido*

IVAO Iteración de funciones valor ajustadas para aprendizaje *online*

KADP *Kernel-based approximate dynamic programming*

k-NN *k-nearest neighbours*

LSPI *Least squares policy iteration*

LSTD *Least squares temporal difference*

MAE *Mean absolute error*

MDP *Markov decision process*

MPC *Model predictive control*

NFTD *Neural-fitted temporal difference*

PI *Policy iteration*

RBF *Radial basis function*

rHuEPO Eritropoyetina recombinante humana

RL *Reinforcement learning*

RLSTD *Recursive least squares temporal difference*

RMSE *Root mean squared error*

RNA Red neuronal artificial

SLFN *Single layer feedforward neural network*

STC *Search-then-converge*

TD *Temporal difference*

VI *Value iteration*

Capítulo 1

Introducción

1.1. Aprendizaje por refuerzo

El aprendizaje por refuerzo (RL, *reinforcement learning*) proporciona un conjunto de técnicas para resolver *problemas de decisión secuenciales*. Este tipo de problemas aparece en una amplia variedad de campos como control automático, inteligencia artificial, investigación operativa, economía y medicina.

Supongamos un sistema físico cuyo estado en cualquier instante temporal k está descrito por una magnitud s. Dicha magnitud puede ser un vector cuyas componentes contienen información de diversa índole como, por ejemplo, coordenadas cartesianas, volumen y temperatura, o si estamos considerando un sistema económico, oferta y demanda, o cantidad de productos en *stock* y capacidad de producción. Durante el transcurso del tiempo el estado del sistema puede evolucionar, es decir, el valor de s puede sufrir cambios. Ahora supongamos que dichos cambios están influenciados en parte por acciones que se pueden realizar sobre el sistema y que el objetivo es alcanzar un determinado estado. Cuando se tiene que decidir qué acción aplicar en un único instante temporal, se dice que el problema de decisión es monoetapa, si se requiere una secuencia de decisiones entonces se trata de un problema de decisión multietapa o secuencial.

Existen diferentes formas de abordar un problema de decisión secuencial. La más obvia posiblemente consista en *programar* un sistema que sea capaz de manejar todos los estados posibles. Para ello es necesario que el programador contemple todos los estados y describa la acción que se debe realizar en cada uno de ellos. Esta solución puede ser viable en problemas sencillos, pero no resulta útil en los problemas donde el número de estados es elevado. Una segunda opción consiste en usar técnicas de *búsqueda* y *planificación* (Russell y Norvig, 1995). Cuando se conoce de antemano la forma en la que evoluciona el estado del sistema en función de las acciones aplicadas,

entonces es posible buscar, o planificar, la mejor secuencia de acciones desde el estado actual (Otterlo, 2009). Sin embargo, en el caso de que exista cierta incertidumbre sobre el efecto que pueden producir las acciones, es decir, cuando el sistema es estocástico, los algoritmos estándar de búsqueda y planificación dejan de ser válidos. La tercera opción es la solución basada en *aprendizaje*. Esta opción, que es la planteada en RL, no requiere información previa del sistema sino que emplea un conjunto de datos (estados y acciones) para obtener la solución. Esta característica es crucial en la mayoría de problemas ya que, a menudo, resulta extremadamente difícil (o imposible) obtener un modelo que describa de forma precisa la evolución del sistema. Además la solución basada en aprendizaje permite manejar de forma eficaz la incertidumbre asociada con los sistemas estocásticos.

La solución planteada en RL suele describirse conceptualmente mediante los elementos de la Figura 1.1 (Busoniu et al., 2010a). La figura muestra a un *agente* que interactúa con un *entorno* mediante tres señales: una señal de estado que describe el estado del entorno, una señal de acción que permite al agente influenciar el estado del entorno y una señal escalar de recompensa, la cual proporciona al agente información sobre la calidad de la acción que acaba de realizar en el estado actual. En cada instante temporal, el agente recibe una medida del estado y realiza una acción. Como consecuencia, en parte, de la acción realizada, se produce una transición del entorno a un nuevo estado. Además se genera una señal de recompensa que evalúa la calidad de dicha transición. Entonces el agente recibe el nuevo estado y el ciclo completo se repite (Sutton y Barto, 1998).

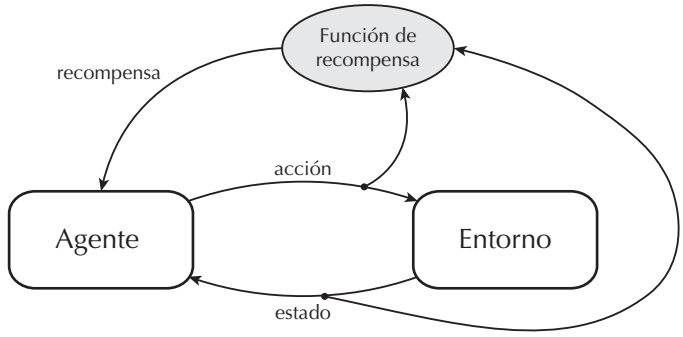


Figura 1.1
Elementos que forman el problema de aprendizaje por refuerzo y su flujo de interacción.

El concepto de agente hace referencia a "algo" que es capaz de percibir y actuar (Russell y Norvig, 1995). De acuerdo con esta definición, un agente puede ser desde un sofisticado robot con decenas de sensores que le proporcionan la capacidad de percibir el estado del entorno y complejos actuadores (por ejemplo un brazo robótico industrial) hasta una sencilla función que toma como argumento de entrada un valor numérico que representa una percepción y devuelve un valor 0 o 1 (como la función de

un microcontrolador encargada de activar el modo de bajo consumo cuando la batería está cerca de agotarse). Por otro lado, se suele considerar que el entorno está formado por todos aquellos elementos que no son parte del agente.

El agente selecciona la acción realizada en cada estado de acuerdo a una *política*. La política es una función que mapea estados a acciones. El objetivo del agente es aprender una política que maximice la cantidad total de recompensa recibida, es decir, la recompensa acumulada a largo plazo. Una característica importante es que el agente no recibe información sobre qué acciones debe realizar para lograr su objetivo, ya que las recompensas únicamente indican cuál es el objetivo. Por ejemplo, supongamos un problema donde un agente debe aprender a jugar al ajedrez. El estado en cada instante está definido por la posición de todas las piezas en el tablero, mientras que las acciones se corresponden con los movimientos permitidos de cada pieza. En este problema, una posible función de recompensa otorgaría una recompensa positiva cuando se alcanza un estado en el que se haya eliminado al rey oponente, es decir, cuando se gana la partida; una recompensa negativa en caso de perderla y una recompensa neutra en el resto de estados. Por tanto, la información proporcionada por las recompensas es insuficiente para que el agente determine qué acción debe realizar en cada estado.

El proceso de aprendizaje se basa en los datos que adquiere el agente mediante la interacción con el entorno. Cada vez que el agente realiza una acción, la información que recibe por parte del entorno (el estado siguiente y la recompensa) es utilizada para modificar la política. Habitualmente se utiliza el término *experiencia* para referirse a los datos adquiridos por el agente. Durante las primeras etapas del aprendizaje, cuando el agente todavía no tiene suficiente experiencia, las acciones realizadas se seleccionan de forma aleatoria y, conforme aumenta la experiencia, la política mejora de forma que la acciones proporcionan al agente una mayor cantidad de recompensa a largo plazo.

Teniendo el cuenta la cantidad de información que recibe el agente, el aprendizaje por refuerzo puede situarse entre el aprendizaje supervisado y el aprendizaje no supervisado (Wiering y van Otterlo, 2012) (ver Figura 1.2). En el aprendizaje no supervisado el objetivo es encontrar la estructura de un conjunto de datos sin etiquetar, es decir, no hay ninguna señal que permita evaluar la solución obtenida. Por el contrario, en el aprendizaje supervisado el conjunto de datos contiene ejemplos de la solución deseada para cada uno de los patrones de entrada. Por ello se puede determinar perfectamente si la solución propuesta es similar a la solución óptima o en qué zonas del espacio de entrada la solución es errónea y en qué cantidad. En el aprendizaje por refuerzo la señal de recompensa proporciona cierta información sobre la solución propuesta, pero la información es mucho más incompleta que en el caso del aprendizaje supervisado. Esta característica resulta vital en los problemas donde se desconoce la solución óptima y el agente debe aprenderla basándose en su propia experiencia.

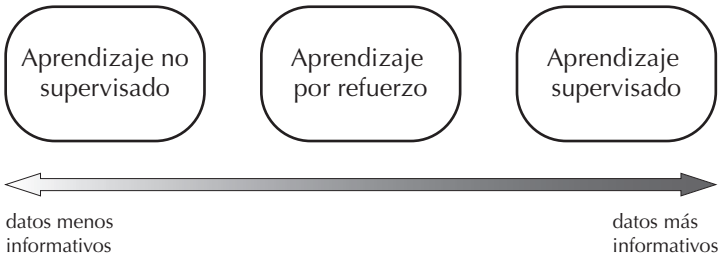


Figura 1.2
Comparación de los distintos tipos de aprendizaje en función de la información que contienen los datos.

1.2. Motivación y objetivos

Durante las dos últimas décadas, el interés de la comunidad científica por el aprendizaje por refuerzo ha aumentado de forma espectacular, lo que ha dado lugar a un gran número de algoritmos nuevos y aplicaciones. Uno de los principales atractivos de este campo es la sólida base teórica sobre la que se asienta. Los problemas tratados se suelen formular empleando el marco matemático de los procesos de decisión de Markov, mientras que muchos de los algoritmos están basados en los principios de funcionamiento de la programación dinámica. Esta base matemática ha permitido el desarrollo de análisis teóricos que demuestran bajo qué condiciones, y a qué velocidad, convergen los algoritmos hacia políticas óptimas.

Los métodos de aprendizaje por refuerzo han proporcionado algunas de las mejores soluciones conocidas en diversos problemas prácticos. Algunos casos populares incluyen el desarrollo de un helicóptero autónomo (Ng et al., 2004), la gestión de un grupo de ascensores (Crites y Barto, 1996) o un agente capaz de jugar al juego de mesa *backgammon* (Tesauro, 1995). Sin embargo, a pesar del gran número de problemas que potencialmente se pueden resolver mediante RL, su uso todavía está mucho menos extendido que el de otras técnicas de aprendizaje automático. Esta situación se debe, principalmente, a dos motivos:

- El número de estados y/o acciones que definen un problema puede ser muy grande o incluso infinito (cuando las variables de estado son continuas). Muchos de los análisis teóricos de convergencia de los algoritmos de RL asumen el uso de tablas para representar de forma exacta las funciones valor y las políticas. Si el número de estados es muy elevado no es posible representar dichas estructuras en tablas, sino que se hace necesario emplear aproximadores para representarlas de una forma más compacta y manejable. Desde el punto de vista teórico, los aproximadores lineales en los parámetros son más adecuados, ya que permiten conservar algunas propiedades de convergencia. Por el contrario, desde el punto de vista práctico, los aproximadores no lineales proporcionan mejores resultados,

siendo la única opción posible en determinados problemas donde el espacio de estados está definido por un número elevado de dimensiones.

- El proceso de aprendizaje es demasiado lento. Los algoritmos clásicos de RL suelen ser computacionalmente muy eficientes, sin embargo, el número de datos que requieren para obtener políticas adecuadas suele ser muy elevado, incluso en problemas relativamente sencillos. Cuando se dispone de algún tipo de simulador capaz de generar los datos, la velocidad de convergencia no supone un inconveniente. En caso contrario, adquirir un número elevado de datos a partir de un entorno real puede ser excesivamente costoso, tanto en tiempo como en dinero.

Como consecuencia de ambos motivos la aplicación de los métodos de RL en problemas con un cierto grado de complejidad dista de ser trivial. La combinación del algoritmo de RL y el método de aproximación debe seleccionarse cuidadosamente ya que, de lo contrario, es probable que el proceso de aprendizaje no converja hacia una política óptima. Aún suponiendo que el algoritmo se comporte de forma estable, si la cantidad de datos disponible es limitada, puede ocurrir que sea insuficiente para obtener una solución útil. Además hay otras etapas de diseño que pueden resultar complejas y para las cuales no existe un procedimiento sistemático como, por ejemplo, el diseño de la función de recompensa o la definición de las variables de estado. Por tanto, para obtener una solución satisfactoria mediante RL, habitualmente se requiere que el usuario posea cierta experiencia tanto en la aplicación de las técnicas como en el problema que se desea resolver.

En términos generales, el objetivo que se persigue en esta tesis es doble. Por una parte se pretende desarrollar nuevos algoritmos orientados a reducir las limitaciones de los métodos actuales de RL, es decir, algoritmos que puedan ser aplicados de forma estable en problemas con espacios continuos y un número elevado de dimensiones así como algoritmos que hagan un uso más eficiente de los datos. Por otra parte, el segundo objetivo es diseñar un sistema basado en RL para resolver un problema real perteneciente al ámbito médico, concretamente se pretende optimizar el tratamiento con darbepoetina alfa de pacientes anémicos en hemodiálisis.

1.3. Organización de la tesis y contribuciones

La presente tesis se ha organizado en siete capítulos. Se ha intentado que los capítulos principales sean autocontenidos, motivo por el cual los capítulos del 3 al 6 contienen las secciones típicas de cualquier artículo científico (introducción, métodos, experimentos, resultados y conclusiones). Exceptuando el capítulo 3, que junto con el capítulo 2 presenta los fundamentos teóricos y el contexto de la tesis, el resto de capítulos (4, 5 y 6) pueden leerse de forma independiente y en cualquier orden, aunque, si es posible, se recomienda leerlos en el orden presentado. A continuación se describe brevemente cada uno de los capítulos indicando las contribuciones realizadas.

- **Capítulo 2. Fundamentos de programación dinámica y aprendizaje por refuerzo**. Este capítulo proporciona los fundamentos teóricos necesarios para describir el resto de conceptos introducidos a lo largo de la tesis. Se presenta el marco matemático de los procesos de decisión de Markov, considerado como un estándar para describir formalmente los elementos del aprendizaje por refuerzo. Posteriormente se introducen los algoritmos básicos de programación dinámica ya que son la base de muchos de los algoritmos de RL. Finalmente, se introducen los algoritmos clásicos de RL asumiendo que el espacio de estados y de acciones es discreto y suficientemente pequeño como para ser enumerado.

- **Capítulo 3. Aprendizaje por refuerzo en espacios continuos**. La primera parte de este capítulo extiende los algoritmos clásicos de RL, introducidos en el capítulo anterior, al caso más general de tener un espacio de estados continuo, lo cual resulta básico para abordar problemas reales. La segunda parte del capítulo se centra en una clase de algoritmos caracterizados por hacer un uso eficiente de los datos. Además de presentar los algoritmos que constituyen el estado del arte, el capítulo contribuye con un estudio experimental en el que se compara y analizan las propiedades de los algoritmos empleando un problema comúnmente utilizado en la evaluación de estas técnicas (el coche en la montaña).

- **Capítulo 4. Iteración de funciones valor ajustadas para problemas de aprendizaje *online***. En este capítulo se introduce un nuevo algoritmo enfocado al uso eficiente de los datos en problemas de aprendizaje *online*. El desarrollo de algoritmos eficientes tradicionalmente se ha dirigido hacia problemas de aprendizaje *offline*, sin embargo dicha eficiencia también es una característica de vital importancia en los problemas *online*. El algoritmo propuesto en este capítulo reduce la cantidad de datos necesaria para aprender políticas óptimas haciendo un uso más intensivo de los recursos computacionales. Además permite emplear ciertos aproximadores no lineales garantizando la convergencia hacia la solución óptima.

- **Capítulo 5. *Least-squares temporal-difference* basado en máquinas de aprendizaje extremo**. El capítulo introduce un algoritmo para predicción de funciones valor que emplea las máquinas de aprendizaje extremo como aproximador de funciones. La predicción de funciones valor es una de las etapas clave en todos los algoritmos basados en el principio de iteración de políticas. Las máquinas de aprendizaje extremo proporcionan un aproximador caracterizado por ser lineal en los parámetros y realizar una aproximación de tipo global. La combinación de ambas características, como se demuestra en los experimentos realizados, permiten que el algoritmo desarrollado conserve las propiedades teóricas de convergencia al mismo tiempo que mejora la escalabilidad a espacios de elevada dimensionalidad.

- **Capítulo 6. Optimización del tratamiento de la anemia en pacientes en hemodiálisis mediante aprendizaje por refuerzo**. En este capítulo se desarrolla un sistema basado en RL para optimizar el tratamiento de la anemia

en pacientes en hemodiálisis. La anemia es una complicación que sufren más del 90 % de los pacientes sometidos a hemodiálisis. El tratamiento estándar consiste en la administración de darbepoetina alfa (u otros fármacos del mismo tipo). La respuesta de los pacientes a la darbepoetina alfa es altamente heterogénea, por lo que es necesario realizar un ajuste individualizado de la dosis. Sin embargo, este proceso de ajuste resulta complicado por varios motivos: (i) los efectos de cada administración del fármaco pueden durar hasta tres meses: cuando se administra regularmente es difícil determinar a qué dosis corresponde el efecto observado en el paciente; (ii) la dosis óptima para un mismo paciente puede variar en función de su estado (inflamación, administración de otros fármacos, variaciones de peso, etc.). Estas dificultades, junto con el estrecho rango terapéutico de la darbepoetina alfa y su elevado precio, justifican la necesidad de optimizar los protocolos actuales de administración.

- **Capítulo 7. Conclusiones y proyección futura**. Este capítulo cierra la tesis describiendo las conclusiones globales que se derivan del trabajo realizado. En el capítulo también se identifican oportunidades de investigación futura y se incluyen los logros científicos alcanzados.

Capítulo 2

Fundamentos de programación dinámica y aprendizaje por refuerzo

2.1. Introducción

El origen del término programación dinámica (*dynamic programming*, DP) no tiene ninguna relación con el concepto actual de programación, entendido éste como el proceso de diseñar, codificar y depurar el código fuente de programas computacionales. La programación dinámica fue desarrollada por Richard Bellman en la década de 1950 (Bellman, 1957), época en la que las computadoras eran extremadamente raras y el concepto actual de "escribir código" apenas existía. Así pues, el significado de la palabra *programación* más bien puede considerarse como un sinónimo de planificación, mientras que el término *dinámica* enfatiza la importancia del tiempo, el carácter multietapa del proceso (Dreyfus, 2002). La programación dinámica es una parte fundamental de la teoría de control óptimo. En un problema de control óptimo, el objetivo es desarrollar un controlador que minimice una medida del comportamiento de un sistema dinámico a lo largo del tiempo (Bertsekas, 2005). Cuando el sistema es estocástico y no lineal, habitualmente se considera que la única forma viable de encontrar una solución es mediante DP (Sutton y Barto, 1998). El campo de la programación dinámica fue desarrollado originalmente para resolver problemas discretos de control óptimo estocástico. Posteriormente las técnicas de DP se han aplicado en otros campos como investigación operativa y economía, donde han sido ampliamente estudiadas (Puterman, 2005).

Por otra parte, el aprendizaje por refuerzo (*reinforcement learning*, RL) tiene sus orígenes en el campo de la inteligencia artificial. Como tantos otros métodos de

inteligencia artificial, RL está inspirado en los mecanismos de aprendizaje biológico. Concretamente, tiene sus raíces en el condicionamiento operante (o instrumental), una forma de aprendizaje en la que el comportamiento de un sujeto se modifica mediante sus antecedentes y consecuencias. De acuerdo con el condicionamiento operante, entre las diferentes formas que un individuo puede responder ante una misma situación, aquellas que estén acompañadas de una satisfacción (refuerzo positivo), estarán más firmemente conectadas a dicha situación. Cuando el individuo se encuentre de nuevo en esa situación, la respuesta con una conexión más fuerte tendrá más posibilidades de repetirse. Por el contrario, la conexión de aquellas respuestas acompañadas por una sensación de malestar (refuerzo negativo) tienden a debilitarse, por lo que son menos probables que se repitan (Skinner, 1938; Thorndike, 1905). Este mecanismo incorpora dos conceptos clave del aprendizaje mediante prueba y error utilizado en RL: *selección* y *asociación*. El concepto de *selección* significa que es necesario intentar varias alternativas y seleccionar entre ellas comparando sus consecuencias. Mientras que el concepto de *asociación* hace referencia a la idea de que las alternativas encontradas en proceso de selección se asocian a situaciones concretas. El aprendizaje supervisado, por ejemplo, es asociativo pero no selectivo (Sutton y Barto, 1998).

Aunque DP y RL inicialmente se desarrollaron de forma independiente pronto comenzaron a observase similitudes entre ambas técnicas. Desde la perspectiva del control automático, tanto DP como RL son equivalentes a un problema de control óptimo estocástico y no lineal (Bertsekas, 2005). Además, RL puede ser visto como una técnica de control óptimo adaptativo (Sutton et al., 1992; Vrabie et al., 2009). Actualmente ambos campos están fuertemente relacionados, tanto es así que a menudo se presentan de forma conjunta (Berenji, 1994; Lewis y Vrabie, 2009). La mayoría de algoritmos de RL pueden considerarse como un intento de dotar a la programación dinámica de una mayor aplicabilidad práctica eliminando la necesidad de tener un conocimiento profundo del entorno (Szepesvári, 2010).

Cuando se dispone de un modelo que describe el comportamiento del entorno es posible aplicar los algoritmos de DP. Comparado con otras técnicas de control clásico que también se basan en un modelo, una de las ventajas de DP es que apenas realiza asunciones sobre el entorno, como suele ser habitual en otros casos (Busoniu et al., 2010a). Además, el modelo requerido por DP puede ser un simulador, y no necesariamente una expresión analítica. Aunque *a priori* esta diferencia pueda parecer trivial, en muchas aplicaciones prácticas resulta más sencillo encontrar un modelo de simulación que una expresión analítica, especialmente cuando se trata de entornos estocásticos (Sutton y Barto, 1998).

La principal diferencia entre DP y RL es que la solución planteada por las técnicas de RL no requieren ningún tipo de modelo, sino que funcionan empleando únicamente los datos obtenidos del entorno. Esta característica es importante porque permite aplicar RL en entornos cuya dinámica es desconocida o cuando construir un modelo de ella resulta demasiado costoso. Los datos se pueden obtener del entorno al mismo tiempo que el agente aprende una política de control o, alternativamente, pueden ser almacenados y procesados posteriormente. Los algoritmos aplicados en el primer

caso son de tipo *online* mientras que en el segundo caso son de tipo *offline*. Como es lógico, cuando se dispone de un modelo también es posible aplicar las técnicas de RL, para ello, la opción más sencilla consiste en utilizar el modelo para generar los datos requeridos por el algoritmo.

Debido a los múltiples orígenes del RL es común encontrar en la literatura los mismos conceptos con diferentes nombres, por ejemplo, los términos programación neuro-dinámica o programación dinámica aproximada se suelen emplear como sinónimos de aprendizaje por refuerzo (Powell, 2011). Las dos nomenclaturas más extendidas son aquellas basadas en la teoría de control y en la inteligencia artificial o aprendizaje automático. En la Figura 2.1 se muestra de nuevo el esquema con los elementos básicos de DP/RL y la equivalencia entre ambas nomenclaturas. A lo largo de esta tesis la información se presenta principalmente desde el punto de vista del aprendizaje automático, por lo que se empleará la notación y terminología propia de este campo. Nótese, sin embargo, que ello no supone un impedimento para emplear resultados y ejemplos procedentes del ámbito del control automático. Obviamente, la metodología descrita en los siguientes capítulos es también válida para los problemas secuenciales de decisión que puedan aparecer en otros ámbitos diferentes al aprendizaje automático.

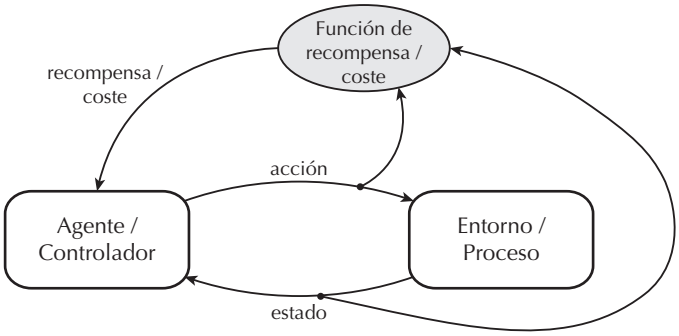


Figura 2.1
Elementos que forman el problema de aprendizaje por refuerzo y equivalencia entre las nomenclaturas del ámbito de la inteligencia artificial y de la teoría de control.

El resto del capítulo está organizado como sigue. En la Sección 2.2 se introduce el marco matemático empleado en DP/RL: los procesos de decisión de Markov. Posteriormente, el concepto de política y los criterios de optimalidad necesarios para definir las políticas óptimas se describen en las Secciones 2.3 y 2.4, respectivamente. La mayor parte de algoritmos de DP y RL están basados en funciones valor y las ecuaciones de Bellman, ambos elementos se describen en la Sección 2.5. Una vez introducidos todos los conceptos previos necesarios, la Sección 2.6 explica los principios de funcionamiento y los algoritmos fundamentales de DP, mientras que en la Sección 2.7 estos

principios se extienden al caso de RL. Finalmente, la Sección 2.8 cierra el capítulo con un resumen y discusión de los aspectos más importantes.

2.2. Procesos de decisión de Markov

Los procesos de decisión de Markov (*Markov decision processes*, MDPs) (Puterman, 2005; Sigaud y Buffet, 2010) proporcionan el marco teórico estándar utilizado para describir formalmente los problemas que resuelven tanto la programación dinámica como el aprendizaje por refuerzo. Un MDP está compuesto por un conjunto de estados, un conjunto de acciones, una función de transición entre estados y una función de recompensa. A continuación se introduce cada uno de estos elementos.

2.2.1. Estados

En cada instante temporal, el entorno se encuentra en un estado determinado s. El conjunto de estados S del entorno se define como el conjunto finito $\{s^1, s^2, \ldots, s^N\}$, siendo N el tamaño del espacio de estados, es decir, $|S| = N$, donde $|\cdot|$ denota cardinalidad. Aunque no existe una definición consensuada sobre qué información debe incluir el estado, en Powell (2011) se define *estado* como una función de la historia (estados anteriores) con la mínima dimensión necesaria y suficiente para tomar decisiones, calcular la función de transición y calcular la función de recompensa. El término "mínima dimensión" hace referencia a que el estado debe ser tan compacto como sea posible, ya que, como se verá posteriormente, el número de dimensiones de los estados condicionará el proceso de aprendizaje. Dicho de otra forma, un estado en el instante k debe contener toda la información necesaria para permitir que el agente actúe de forma óptima en dicho instante.

2.2.2. Acciones

El conjunto de acciones A se define como el conjunto finito $\{a^1, a^2, \ldots, a^M\}$, siendo M el tamaño del espacio de acciones, es decir, $|A| = M$. Las acciones pueden ser utilizadas para controlar las transiciones que realiza el sistema. El conjunto de acciones que se pueden realizar en un estado concreto $s \in S$ es denotado como $A(s)$, siendo $A(s) \subseteq A$ para todo $s \in S$. Aunque existen problemas donde determinados estados únicamente permiten realizar un subconjunto de acciones, en general se asume que $A(s) = A$ para todo $s \in S$.

2.2.3. Función de transición

En el caso de un sistema determinista, cuando se aplica una acción $a \in A$ en un estado $s \in S$, el sistema realiza una transición desde s hasta s' de acuerdo a la función

de transición $f : S \times A \to S$:

$$s' = f(s, a) \tag{2.1}$$

En el caso más general de que el sistema sea estocástico, el estado siguiente es una variable aleatoria y el estado y acción actuales dan lugar a la distribución de probabilidad de dicha variable. La función de transición determinista f es reemplazada por una función de transición probabilística $\bar{f} : S \times A \times S \to [0, 1]$. Tras aplicar una acción a en un estado s la probabilidad de que el estado siguiente sea s' es:

$$P(s'|s, a) = \bar{f}(s, a, s') \tag{2.2}$$

Para cualquier estado s y acción a, la función $\bar{f}$ debe satisfacer $\sum_{s'} \bar{f}(s, a, s') = 1$. Generalmente se define una variable discreta temporal, $k = 1, 2, \ldots$, para distinguir el orden en el que transcurren los estados y las acciones. De acuerdo a esta notación, s_k hace referencia al estado en el instante k. Esto permite comparar diferentes estados y acciones que ocurren de forma ordenada en el tiempo durante la interacción agente-entorno.

El sistema sobre el que se aplican las acciones se dice que es *markoviano* o posee la *propiedad de Markov* si el resultado de aplicar una acción no depende de la historia ni de acciones anteriores, sino que solo depende del estado y la acción actuales. La propiedad de Markov se puede expresar formalmente usando la distribución de probabilidad condicional de los estados futuros:

$$P(s_{k+1}|s_k, a_k, s_{k-1}, a_{k-1}, \ldots) = P(s_{k+1}|s_k, a_k) = \bar{f}(s_k, a_k, s_{k+1}) \tag{2.3}$$

El hecho de que un sistema sea markoviano implica que el estado actual s_k proporciona suficiente información para realizar decisiones óptimas. O lo que es lo mismo, siempre que se aplica la acción a en el estado s, la distribución de probabilidades del estado siguiente es la misma, independientemente del instante temporal k en el que se encuentre el sistema. La propiedad de Markov tiene un papel esencial debido a que proporciona garantías teóricas de convergencia a los algoritmos DP/RL (Sutton y Barto, 1998). Además, esta propiedad es la que distingue los MDPs de otro tipo de modelos más generales como, por ejemplo, los procesos de decisión de Markov parcialmente observables.

2.2.4. Función de recompensa

La *función de recompensa* es el elemento del MDP que especifica implícitamente el objetivo del aprendizaje. En el caso de un sistema determinístico, cada vez que el entorno realiza una transición de un estado a otro, el agente recibe una señal escalar de acuerdo con la función de recompensa $\rho : S \times A \to \mathbb{R}$:

$$r_{k+1} = \rho(s_k, a_k) \tag{2.4}$$

donde se asume que ρ está limitada por una cantidad finita, es decir:

$$\sup_{s,a} |\rho(s, a)| < \infty \tag{2.5}$$

La función de recompensa indica la deseabilidad inmediata de una transición, sin embargo, no indica nada sobre su deseabilidad a largo plazo.

Cuando el sistema es estocástico las transiciones no dependen únicamente del estado y acción actuales. Por lo tanto, dado que la recompensa está asociada a las transiciones, en el caso estocástico la función de recompensa también depende del estado siguiente, $\bar{\rho} : S \times A \times S \to \mathbb{R}$. Así pues, tras una transición hasta el estado s_{k+1}, la recompensa correspondiente viene dada por:

$$r_{k+1} = \bar{\rho}(s_k, a_k, s_{k+1}) \tag{2.6}$$

2.2.5. Proceso de decisión de Markov

Una vez descritos los diferentes elementos que componen los MDPs, se puede definir un *proceso de decisión de Markov* como la tupla $\{S, A, f, \rho\}$ donde S es un conjunto de estados, A es un conjunto de acciones, f es una función de transición y ρ es una función de recompensa.

La función de transición f junto con la función de recompensa ρ forman lo que se conoce como *modelo* del MDP. En ocasiones un MDP puede representarse gráficamente como un diagrama de transición de estados donde cada nodo se corresponde con un estado y los arcos (dirigidos) denotan transiciones. Esta representación gráfica también puede incluir información de la recompensa obtenida en cada una de las transiciones y la probabilidad de cada transición tras realizar una acción determinada. En la Figura 2.2 se muestra un ejemplo de este tipo de diagramas. Debajo de cada arco se muestran dos números entre llaves, el primero de ellos indica la recompensa obtenida por el agente cuando se realiza la transición, mientras que el segundo número indica la probabilidad de dicha transición tras realizar la acción indicada encima del arco. Por ejemplo, si se realiza la acción $a_{1,1}$, un 60 % de las veces el siguiente estado será s_2 y el agente recibirá una recompensa igual a 5, mientras que el restante 40 % el estado siguiente seguirá siendo s_1 y el agente recibirá una recompensa igual a 3.

Existe un gran número de sistemas susceptibles de ser descritos mediante MDPs. Dichos sistemas se pueden dividir en dos grupos según la naturaleza de la interacción entre el agente y el entorno. Supongamos por ejemplo el caso del juego "tres en raya". El objetivo es que el agente aprenda a realizar los movimientos adecuados para ganar. Para ello, se permite que el agente interactúe (juegue) con el entorno (jugador oponente). Los diferentes estados se corresponden con las posibles configuraciones del tablero. Cada vez que alguno de los oponentes gana o se produce un empate, la partida se da por terminada y comienza una partida nueva. Este tipo de problemas, en los que la interacción tiene un final definido, se conocen como *problemas episódicos*. Para referirse a cada una de las secuencias de estados y acciones desde un estado inicial hasta uno final se utiliza el término *episodio* o *trayectoria*. Por contra, cuando la interacción entre agente y entorno no tiene un final definido, se denomina *problema continuo*.

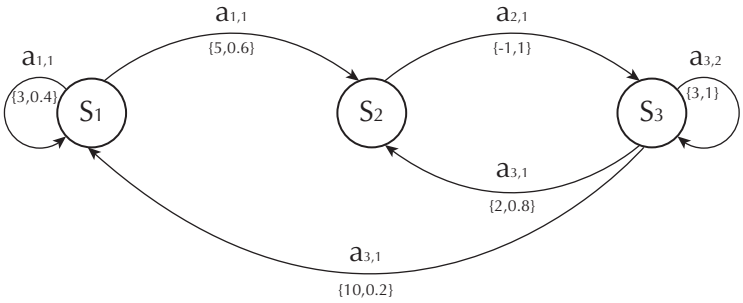


Figura 2.2
Representación esquemática de un proceso de decisión de Markov formado por tres estados.

En los problemas episódicos, el MDP correspondiente contiene uno o varios estados que producen el final del episodio, dichos estados se conocen como *estados terminales* o *absorbentes*. Una vez alcanzado una estado terminal, no es posible abandonarlo; cualquier acción realizada en un estado terminal produce una transición al mismo estado con probabilidad 1 y recompensa 0. Formalmente, para un estado terminal se cumple que $\bar{f}(s,a,s) = 1$ y $\bar{\rho}(s,a,s') = 0$ para todos los estados $s \in S$ y acciones $a \in A$. Esta definición de estado terminal permite utilizar el mismo marco teórico para describir problemas tanto episódicos como continuos.

2.3. Políticas

El agente escoge la acción a realizar en cada estado de acuerdo a su política π. Dado un MDP $\{S, A, f, \rho\}$, una *política* determinista es una función $\pi : S \to A$ que mapea cada estado con una acción. Que un agente siga una política π significa que en cualquier instante k y estado s_k, la acción realizada viene dada por:

$$a_k = \pi(s_k) \tag{2.7}$$

Las políticas también pueden ser estocásticas, en cuyo caso realizan un mapeo entre estados y distribuciones sobre el espacio de acciones, $\pi : S \times A \to [0,1]$, cumpliéndose para cada estado $s \in S$ las condiciones:

$$\pi(s,a) \geq 0 \tag{2.8}$$

$$\sum_{a \in A} \pi(s,a) = 1 \tag{2.9}$$

La política es parte del agente y, en general, el objetivo de los algoritmos de DP y RL es encontrar una política óptima (Busoniu et al., 2010a).

Continuando con la distinción entre problemas episódicos y continuos, supongamos un MDP determinista donde una distribución de estado inicial, $I : S \to [0, 1]$, determina la probabilidad de que el sistema se inicie en el estado s_0. Entonces, si se aplica la política π, la acción realizada por el agente será $a_0 = \pi(s_0)$. Como consecuencia de dicha acción, y de acuerdo con la función de transición f, se producirá una transición hasta el estado $s_1 = f(s_0, a_0)$ y el agente recibirá una recompensa $r_0 = \rho(s, a)$. Este proceso se repite generando la trayectoria $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, \ldots$ En el caso de que se trate de un problema episódico, la trayectoria terminará en un estado terminal $s_{\text{term}} \in S_{\text{term}}$, siendo $S_{\text{term}} \subseteq S$, y comenzará una nueva secuencia con un estado determinado por I. En el caso de un problema continuo, la trayectoria puede prolongarse indefinidamente.

2.4. Criterios de optimalidad

Antes de analizar los diferentes métodos para encontrar políticas óptimas es necesario conocer el significado exacto del término "óptimo", es decir, definir lo que se conoce como modelo, o criterio, de optimalidad. El objetivo de los algoritmos de DP /RL es encontrar una política que maximice la recompensa obtenida por el agente a lo largo del tiempo, cantidad conocida como *retorno*. Dicho de otra forma el retorno es la suma acumulada de las recompensas que recibe el agente cuando sigue una política π en una trayectoria que comienza en el estado s_0. Existen varios tipos de retornos en función del método empleado para acumular las recompensas y el horizonte (o longitud de la trayectoria) que se tiene en cuenta. El retorno con descuento y horizonte infinito viene dado por:

$$R^{\pi}(s_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(s_k, a_k) \tag{2.10}$$

donde $\gamma \in [0, 1)$ es el *factor de descuento*, $a_k = \pi(s_k)$ y $s_{k+1} = f(s_k, a_k))$ para $k \geq 0$. El factor de descuento se puede interpretar intuitivamente como una medida del peso que tienen las recompensas futuras para calcular el retorno, o como una forma de tener en cuenta la incertidumbre creciente sobre las recompensas futuras. Desde el punto de vista matemático γ asegura que el retorno está limitado por una cantidad finita siempre y cuando la función de recompensa también lo esté.

De forma similar es posible definir un retorno con horizonte infinito y sin descuento simplemente fijando γ igual a 1 en la Ecuación (2.10). En este caso el retorno consiste en la suma de todas las recompensas obtenidas en la trayectoria. Emplear un retorno sin descuento y horizonte infinito puede ser problemático debido a que su valor no está acotado y suele causar inestabilidades en el proceso de aprendizaje. Otro tipo de retorno que elimina este problema es el retorno promediado de horizonte infinito, calculado como:

$$\lim_{K \to \infty} \frac{1}{K} \sum_{k=0}^{K} \rho(s_k, a_k) \tag{2.11}$$

Alternativamente también existen retornos de horizonte finito. Este tipo de retornos acumulan las recompensas obtenidas a lo largo de una trayectoria de longitud fija. Por ejemplo, si se fija la longitud del horizonte igual a K, el retorno con descuento de horizonte finito se define como

$$\sum_{k=0}^{K} \gamma^k \rho(s_k, a_k) \tag{2.12}$$

En los retornos con horizonte finito es más fácil no aplicar ninguna tasa de descuento ($\gamma = 1$) debido a que el retorno permanece acotado siempre que la función de recompensa también lo esté.

Los criterios de optimalidad ponen de manifiesto que la meta del agente es optimizar su comportamiento (política) a largo plazo (retorno) a partir de una señal que evalúa el comportamiento de forma inmediata (recompensa). Habitualmente la recompensa que recibe el agente en un instante dado está influenciada por las acciones que realizó en los instantes anteriores. Por tanto, dada una trayectoria donde la recompensa solo se recibe en el estado terminal, el agente debe, en cierto modo, averiguar qué acciones han sido las causantes de dicha recompensa. Este problema se conoce como problema de las recompensas retardadas o asignación temporal de mérito (Sutton, 1992), y será discutido más detalladamente en la Sección 2.7.2. La capacidad de los métodos de DP/RL para resolver eficazmente el problema de la asignación temporal de mérito es una de las características que los diferencian de otras técnicas que, a menudo, tienen dificultades para encontrar una solución adecuada.

La mayor parte de algoritmos de DP y RL emplean el criterio de optimalidad definido en la Ecuación (2.10) debido a que posee propiedades teóricas que lo hacen más adecuado para el análisis matemático (Bertsekas y Tsitsiklis, 1996). Por ejemplo, dicho criterio asegura la existencia de, al menos, una política óptima estacionaria y determinista, mientras que en otros casos las políticas óptimas generalmente dependen del instante temporal k, es decir, no son estacionarias. Por este motivo y salvo que se indique lo contrario, en la presente tesis se asumirá como criterio de optimalidad el retorno con descuento y horizonte infinito.

Frecuentemente, en los análisis teóricos el factor de descuento γ se asume como una parte dada del problema, sin embargo, en la práctica es necesario escoger un valor adecuado en función del problema que se pretende resolver. El funcionamiento de algunos de los algoritmos más utilizados tiene una fuerte dependencia con el factor de descuento, por lo que su rendimiento se deteriora notablemente para ciertos valores de γ. Además, como se verá en secciones posteriores, el factor de descuento puede influir tanto en la calidad de la política obtenida como en la velocidad de convergencia: valores pequeños de γ aceleran la convergencia, pero cuando son demasiado pequeños el horizonte de optimización es corto y puede dar lugar a políticas subóptimas. Así pues, el valor γ debe proporcionar un compromiso entre ambas características. No existe ninguna regla general que permita fijar el valor de γ *a priori*, sino que, en gran medida, dependerá de la experiencia del usuario y del conocimiento que se tenga sobre el

entorno. Aún así, el procedimiento habitual implica experimentar con varios valores de γ y seleccionar aquel que produzca los resultados más satisfactorios.

2.5. Funciones valor y ecuaciones de Bellman

En las secciones anteriores se han introducido de forma independiente los MDPs y los criterios de optimalidad, en esta sección se introducirán las funciones valor, que son el punto de unión entre ambos elementos. Intuitivamente, una *función valor* es una estimación de la bondad que supone para un agente estar en un determinado estado cuando sigue una política fija. El concepto de "cuánto de bueno" se expresa de acuerdo al criterio de optimalidad, es decir, al retorno. Existen dos tipos de funciones valor: función V, que estima la bondad de estar en un estado, y función Q, que estima la bondad de realizar una acción en un estado. Por motivos obvios, la función V también es conocida como *función valor estado* y la función Q como *función valor estado-acción*. Aunque en ocasiones el término "función valor" se emplea como sinónimo de función V, aquí se reserva para referirse de forma conjunta a las funciones V y Q. A lo largo de esta sección se describirán en primer lugar las funciones Q y, posteriormente, las funciones V.

La función Q de una política π, $Q^\pi : S \times A \to \mathbb{R}$, es igual al retorno obtenido cuando, a partir de un estado s y acción a dados, se sigue la política π:

$$Q^\pi(s,a) = \sum_{k=0}^{\infty} \gamma^k \rho(s_k, a_k) \tag{2.13}$$

A partir de esta fórmula se puede obtener otra expresión de la función Q que, aunque es equivalente, resulta más útil para introducir conceptos posteriores. Si $(s_0, a_0) = (s, a)$, $s_{k+1} = f(s_k, a_k)$ para $k \geq 0$ y $a_k = \pi(s_k)$ para $k \geq 1$, entonces se puede separar el primer término del sumatorio, obteniendo:

$$\begin{aligned} Q^\pi(s,a) &= \rho(s,a) + \sum_{k=1}^{\infty} \gamma^k \rho(s_k, a_k) \\ &= \rho(s,a) + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} \rho(s_k, \pi(s_k)) \qquad (2.14) \\ &= \rho(s,a) + \gamma R^\pi(f(s,a)) \qquad (2.15) \end{aligned}$$

donde se ha utilizado la Ecuación (2.10) en el último paso.

Una propiedad fundamental de las funciones valor es que pueden ser caracterizadas de forma recursiva mediante la *ecuación de Bellman* (Bellman, 1957). La ecuación de Bellman establece que, para una función Q^π, el valor de realizar la ación a en el estado s bajo la política π es igual a la suma de la recompensa inmediata y el valor

con descuento logrado por π en el siguiente estado, es decir:

$$\begin{aligned} Q^{\pi}(s,a) &= \rho(s,a) + \gamma Q^{\pi}(f(s,a), \pi(f(s,a))) \\ &= \rho(s,a) + \gamma Q^{\pi}(s', a') \end{aligned} \tag{2.16}$$

La ecuación de Bellman para una función Q se puede obtener a partir de (2.14):

$$\begin{aligned} Q^{\pi}(s,a) &= \rho(s,a) + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} \rho(s_k, \pi(s_k)) \\ &= \rho(s,a) + \gamma \left[\rho(f(s,a), \pi(f(s,a))) + \gamma \sum_{k=2}^{\infty} \gamma^{k-2} \rho(s_k, \pi(s_k)) \right] \\ &= \rho(s,a) + \gamma Q^{\pi}(f(s,a), \pi(f(s,a))) \end{aligned} \tag{2.17}$$

donde $(s_0, a_0) = (s, a)$, $s_{k+1} = f(s_k, a_k)$ para $k \geq 0$ y $a_k = \pi(s_k)$ para $k \geq 1$. Q^{π} es la única solución para el sistema de ecuaciones definido por (2.16). Existen varias políticas que pueden tener la misma función Q pero, para una política π dada, Q^{π} es única.

Generalmente, dado un MDP, el objetivo es encontrar la mejor política posible, aquella que obtenga un retorno mayor. Una función Q óptima, denotada como Q^*, se define como la mejor función Q que se puede obtener con cualquier política:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.18}$$

Una *política óptima* es aquella que en cada estado selecciona la acción con el mayor valor de la función Q óptima y, por tanto, maximiza el retorno obtenido:

$$\pi^*(s) = \arg\max_{a} Q^*(s,a) \tag{2.19}$$

donde $\arg\max_a g(a)$ para una función cualquiera $g(\cdot)$ devuelve el valor de a que proporciona el máximo de la función $g(a)$.

Para cualquier función Q dada, una política π que satisface:

$$\pi(s) = \arg\max_{a} Q(s,a) \tag{2.20}$$

se dice que es voraz o *greedy* respecto a Q. Por tanto, una posible forma de hallar la política óptima es calcular primero Q^* y después simplemente aplicar (2.19) para obtener una política *greedy* respecto Q^*.

Las funciones valor óptimas Q^* también se pueden caracterizar de forma recursiva, en este caso mediante la *ecuación de optimalidad de Bellman*. Dicha ecuación establece que el valor óptimo de realizar una acción a en un estado s es igual a la suma de la recompensa inmediata obtenida más el valor de Q^* obtenido por la acción óptima en el estado siguiente:

$$Q^*(s,a) = \rho(s,a) + \gamma \max_{a'} Q^*(f(s,a), a') \tag{2.21}$$

Todos los conceptos introducidos en esta sección para las funciones Q se pueden extender fácilmente a las funciones V. La función V de una política π, $V^\pi : S \to \mathbb{R}$, es el retorno obtenido cuando, a partir de un estado s, se sigue la política π:

$$V^\pi(s) = \sum_{k=0}^{\infty} \gamma^k \rho(s_k, \pi(s_k)) = R^\pi(s) \tag{2.22}$$

Las funciones V y Q de una misma política están relacionadas de forma que:

$$V^\pi(s) = Q^\pi(s, \pi(s)) \tag{2.23}$$

La función V óptima, denotada con V^*, se define como la mejor función V que se puede obtener con cualquier política y se puede calcular a partir de la función Q óptima:

$$V^*(s) = \max_\pi V^\pi(s) = \max_a Q^*(s, a) \tag{2.24}$$

La funciones V^π y V^* se pueden caracterizar mediante la ecuación de Bellman y la ecuación de optimalidad de Bellman, respectivamente:

$$V^\pi(s) = \rho(s, \pi(s)) + \gamma V^\pi(f(s, \pi(s))) \tag{2.25}$$

$$V^*(s) = \max_a [\rho(s, a) + \gamma V^*(f(s, a))] \tag{2.26}$$

A partir de V^* también es posible encontrar una política óptima haciendo uso de la siguiente relación:

$$\pi^*(s) = \arg\max_a [\rho(s, a) + \gamma V^*(f(s, a))] \tag{2.27}$$

Sin embargo, como se puede observar en (2.27), calcular una política óptima a partir de la función V^* requiere conocer el modelo del MDP, tanto su dinámica f como su función de recompensa ρ. Esto se debe a que la función V solo contiene información sobre la bondad de los estados, por lo que para determinar la acción óptima se debe introducir más información por medio de f y ρ. Por el contrario, la función Q incluye también información sobre las acciones y a partir de ella se puede calcular una política óptima sin conocer el modelo del MDP. Este es el motivo por el cual la mayoría de algoritmos RL, donde se asume que el modelo es desconocido, se basan en funciones Q en lugar de funciones V, a pesar de que estas últimas requieren menos espacio para ser almacenadas.

Nótese que por cuestiones de simplicidad los conceptos introducidos sobre las funciones valor y las ecuaciones de Bellman se han restringido al caso de MPDs deterministas, restricción que también se mantiene en las secciones posteriores. Conceptualmente, el caso estocástico sigue manteniendo los mismos principios de funcionamiento, aunque, obviamente, la expresiones matemáticas resultantes sean más complejas. El lector interesado puede encontrar una descripción detallada para el caso de MDPs estocásticos en (Kaelbling et al., 1996; Bertsekas, 2007; Busoniu et al., 2010a; Szepesvári, 2010).

2.6. Programación dinámica: solución basada en el modelo

El término programación dinámica se refiere a un conjunto de técnicas para resolver MDPs asumiendo que se conoce el modelo del MDP, es decir, la función de transición f y la función de recompensa ρ. En muchas aplicaciones resulta complicado obtener un modelo que describa de forma fiable el comportamiento del MDP, por lo que la utilidad práctica de estás técnicas es bastante limitada. A pesar de ello desde el punto de vista teórico y algorítmico son de vital importancia porque constituyen la base para las técnicas de RL. De hecho, casi cualquier algoritmo de RL puede considerarse como una aproximación para conseguir los mismos resultados que los algoritmos de DP pero sin asumir que se dispone de un modelo del MDP y, además, reduciendo su carga computacional (Sutton y Barto, 1998).

Los métodos estudiados en esta sección asumen, además de la disponibilidad de un modelo del MDP, que los conjuntos de estados y acciones son finitos y suficientemente pequeños como para ser almacenados en tablas y enumerados. Estas restricciones se eliminarán gradualmente en las próximas secciones y en el capítulo siguiente.

2.6.1. Iteración de políticas

El algoritmo iteración de políticas (*policy iteration*, PI) (Howard, 1960) se compone de dos etapas que se repiten alternativamente hasta converger en la política óptima. En la primera etapa, conocida como evaluación de la política, se calcula la función valor de la política actual. En la segunda, conocida como mejora de la política, se calcula una política mejorada utilizando la función valor obtenida en la fase anterior. Generalmente, en la primera iteración se fija una política arbitraria π_0 y el proceso iterativo da lugar a una secuencia de políticas y funciones Q

$$\pi_0 \to Q^{\pi_0} \to \pi_1 \to Q^{\pi_1} \to \pi_\ell \to Q^{\pi_\ell} \to \cdots \to Q^* \to \pi^*$$

que convergen asintóticamente hasta la política óptima conforme $\ell \to \infty$.

A continuación se describen más detalladamente cada una de las dos etapas que forman el algoritmo iteración de políticas.

Evaluación de la política

El primer paso del algoritmo PI es encontrar la función valor de una política fija π. Este subproblema también se conoce como predicción de la función valor y forma parte del problema global de encontrar una política óptima. El proceso para encontrar dicha función se basa en la ecuación de Bellman, por lo que resulta útil introducir el correspondiente operador de Bellman. Si denotamos el conjunto de todas las funciones

Q mediante $\mathcal{Q}$, entonces, el operador de Bellman $T^\pi : \mathcal{Q} \to \mathcal{Q}$ es un mapeado que calcula el lado derecho de la ecuación de Bellman, es decir:

$$[T^\pi(Q)](s,a) = \rho(s,a) + \gamma Q^\pi(f(s,a), \pi(f(s,a))) \tag{2.28}$$

La etapa de evaluación de la política consiste típicamente en aplicar de manera iterativa el operador de Bellman. Partiendo de una estimación arbitraria de la función Q, Q_0^π, en cada iteración τ la estimación se actualiza utilizando:

$$Q_{\tau+1}^\pi = T^\pi(Q_\tau^\pi) \tag{2.29}$$

Nótese que se ha empleado τ como índice de la iteración para distinguirlo del índice ℓ empleado en el algoritmo PI, ya que la etapa de evaluación de la política se realiza dentro de cada iteración del algoritmo.

Se puede demostrar que el operador de Bellman T^π es una contracción con factor $\gamma \leq 1$ (Howard, 1960), por lo tanto, se cumple que para cualquier par de funciones Q y Q':

$$\|T^\pi(Q) - T^\pi(Q')\|_\infty \leq \gamma \|Q - Q'\|_\infty \tag{2.30}$$

Esta expresión implica que cuando se aplica el operador T^π a un par de funciones, las funciones resultantes están más cerca entre sí que las originales. Además, T^π tiene un único punto fijo debido a que es una contracción. La ecuación de Bellman (Ecuación 2.16) se puede reescribir de forma más compacta utilizando T^π como:

$$Q^\pi = T^\pi(Q^\pi) \tag{2.31}$$

de donde se deduce que el único punto fijo de T^π es Q^π (Denardo, 1967). En la práctica esto significa que la secuencia de funciones Q que resulta de aplicar T^π iterativamente

$$Q, T^\pi(Q), T^\pi(T^\pi(Q)), T^\pi(T^\pi(T^\pi(Q))), \ldots$$

converge asintóticamente a Q^π conforme aumenta el número de iteraciones. Para calcular Q^π es necesario aplicar T^π a cada par estado-acción (s,a) en cada una de las iteraciones. La estimación actual $Q_\tau^\pi(s,a)$ es reemplazada por una nueva basándose en la recompensa obtenida y el valor Q del siguiente par (s,a). El Algortimo 2.1 muestra el proceso de evaluación de la política utilizando funciones Q. La convergencia de este algoritmo solo está garantizada cuando $\tau \to \infty$. Por lo tanto, para su uso práctico, se suele plantear alguna condición de parada, como por ejemplo que la distancia entre dos estimaciones consecutivas de la función Q sea menor que una cantidad pequeña previamente fijada ξ, es decir, $\|Q_{\tau+1}^\pi - Q_\tau^\pi\|_\infty \leq \xi$, siendo $\xi > 0$.

El método iterativo mostrado en el Algoritmo 2.1 no es la única forma de encontrar la función Q^π. La ecuación de Bellman es lineal respecto a los valores de Q y, evidentemente, ocurre lo mismo con el operador de Bellman T^π. Si el espacio de estados y acciones es finito y suficientemente pequeño, la ecuación de Bellman da lugar a un sistema de $|S| \times |A|$ ecuaciones lineales con $|S| \times |A|$ incógnitas que es posible resolver directamente y cuyo resultado es Q^π.

Algoritmo 2.1 Evaluación de políticas con funciones Q.

Require: Política π que se quiere evaluar, función de transición f, función de recompensa ρ, factor de descuento γ

1: inicializar la estimación de la función Q, por ejemplo, $Q^{\pi}_0 \leftarrow 0$
2: $\tau = 0$
3: **repeat**
4: **for** todo par (s,a) **do**
5: $Q^{\pi}_{\tau+1}(s,a) = \rho(s,a) + \gamma Q^{\pi}_{\tau}(f(s,a), \pi(f(s,a)))$
6: **end for**
7: $\tau \leftarrow \tau + 1$
8: **until** $Q^{\pi}_{\tau+1} = Q^{\pi}_{\tau}$
9: **return** $Q^{\pi} = Q^{\pi}_{\tau}$

El procedimiento introducido en esta sección para hallar la función Q de una política fija también es válido para el caso de funciones V. De hecho, muchos de los algoritmos disponibles en la literatura basados en evaluación de políticas emplean funciones V debido a que el algoritmo original empleaba este tipo de funciones (Howard, 1960). Sin embargo, cabe recordar que las funciones Q son más útiles en la mayoría de casos prácticos porque permiten encontrar políticas óptimas sin necesidad de utilizar el modelo del MDP.

Mejora de la política

Después de obtener la función Q^{π} como resultado de la etapa de evaluación de la política, el algoritmo PI lleva a cabo la etapa de mejora de la política. En esta etapa el objetivo es, como su propio nombre indica, encontrar una nueva política con un comportamiento más cercano al óptimo que el de la política actual. Esta tarea se puede realizar simplemente escogiendo en cada estado s la acción con mayor valor Q^{π}, es decir, calculando una política *greedy* respecto a Q^{π}. Suponiendo que en la iteración ℓ de PI se ha evaluado la política π_{ℓ}, la política mejorada se obtiene mediante la expresión:

$$\pi_{\ell+1}(s) = \arg\max_{a} Q^{\pi_{\ell}}(s,a) \tag{2.32}$$

Si la nueva política no presenta ninguna mejora respecto a la anterior significa que el algoritmo PI ha convergido y la política actual ya es óptima.

A modo de resumen, en el Algoritmo 2.2 se muestra el pseudo código correspondiente a PI incluyendo las dos etapas del proceso. Cuando los espacios de estados y acciones son finitos, PI converge en un número finito de iteraciones. Cada política $\pi_{\ell+1}$ es estrictamente mejor que π_{ℓ} a menos que $\pi_{\ell} = \pi^*$, en cuyo caso el algoritmo se detiene. Esta mejora monotónica, junto con el hecho de que el número de políticas en un MDP finito es también finito, garantizan que el algoritmo PI converge en un tiempo finito.

Algoritmo 2.2 Iteración de políticas con funciones Q.

Require: Política inicial π_0

1: $\ell = 0$
2: **repeat**
3: Calcular Q^{π_ℓ}, la función Q de π_ℓ {evaluación de la política}
4: $\pi_{\ell+1}(s) = \arg\max_a Q^{\pi_\ell}(s,a)$ {mejora de la política}
5: $\ell \leftarrow \ell + 1$
6: **until** $\pi_{\ell+1} = \pi_\ell$
7: **return** $\pi^* = \pi_\ell$, $Q^* = Q^{\pi_\ell}$

2.6.2. Iteración de funciones valor

El algoritmo iteración de funciones valor (*value iteration*, VI) (Bellman, 1957) genera una secuencia de funciones valor que converge a la función valor óptima:

$$Q_0 \rightarrow Q_1 \rightarrow Q_2 \rightarrow Q_3 \rightarrow Q_4 \rightarrow Q_5 \rightarrow Q_6 \rightarrow \cdots \rightarrow Q^*$$

donde Q_0 es arbitraria.

Mientras que el algoritmo PI se basa en la ecuación de Bellman para evaluar una política que posteriormente es mejorada, el algoritmo VI emplea la ecuación de optimalidad de Bellman para calcular directamente la función valor óptima. En este caso también resulta útil definir el correspondiente operador de optimalidad de Bellman:

$$[T(Q)](s,a) = \rho(s,a) + \gamma \max_{a'} Q^*(f(s,a),a') \tag{2.33}$$

El proceso iterativo de VI comienza con una función Q arbitraria Q_0 que es actualizada en cada iteración ℓ empleando:

$$Q_{\ell+1} = T(Q_\ell) \tag{2.34}$$

El operador T es una contracción con un único punto fijo. Igual que ocurría con el operador T^π, al reescribir la ecuación de optimalidad de Bellman usando T, queda claro que su punto fijo es Q^*:

$$Q^* = T(Q^*) \tag{2.35}$$

Por tanto la aplicación iterativa del operador T a cualquier función Q_0 converge asintóticamente a Q^* conforme $\ell \rightarrow \infty$. Una vez obtenida la función Q óptima, resulta sencillo calcular una política óptima empleando la Ecuación (2.19).

El Algoritmo 2.3 muestra el pseudo código que implementa el método de iteración de funciones valor para el caso de funciones Q. De nuevo cabe recordar que este algoritmo también puede ser aplicado en funciones V.

En el caso del algoritmo PI, el uso del operador T^π daba lugar a un sistema de ecuaciones lineales que podía resolverse directamente para hallar Q^π. Por el contrario, el operador T es altamente no lineal debido a la maximización sobre el conjunto de

Algoritmo 2.3 Iteración de funciones valor usando funciones Q.

Require: Dinámica f, función de recompensa ρ, factor de descuento γ
1: inicializar la estimación de la función Q, por ejemplo, $Q_0 \leftarrow 0$
2: $\ell = 0$
3: **repeat**
4: **for** todo par (s,a) **do**
5: $Q_{\ell+1}(s,a) = \rho(s,a) + \gamma \max_{a'} Q_\ell(f(s,a),a')$
6: **end for**
7: $\ell \leftarrow \ell + 1$
8: **until** $Q_{\ell+1} = Q_\ell$
9: **return** $Q^* = Q_\ell$

acciones por lo que, en general, es más difícil resolver la ecuación de optimalidad de Bellman con métodos directos.

2.6.3. Principio generalizado de iteración de políticas

Los dos algoritmos descritos en las secciones previas pueden considerarse instancias del principio generalizado de iteración de políticas (*generalized policy iteration*, GPI) (Sutton y Barto, 1998). Este principio, que está presente en la mayoría de métodos de DP y RL, consta de dos procesos simultáneos que interaccionan entre sí. Uno de ellos se encarga de hacer la función valor consistente con la política actual (evaluación de la política), y el otro de hacer la política *greedy* respecto a la función valor actual (mejora de la política). El principio GPI hace referencia a la idea general de permitir que ambos procesos interactúen. La Figura 2.3a describe gráficamente este principio (Sutton y Barto, 1998).

La forma exacta de implementar este principio da lugar a diferentes algoritmos. Por ejemplo, en el caso del algoritmo PI, los dos procesos están claramente diferenciados: cada política es evaluada mediante un proceso iterativo para después llevar a cabo el proceso de mejora de la política. La Figura 2.3b muestra de forma esquemática este proceso: la línea superior representa las funciones valor que evalúan perfectamente las políticas, es decir, $V = V^\pi$; mientras que la linea inferior representa las políticas que son *greedy* respecto a las funciones valor. Se puede observar que, dada una política inicial π_0, el algoritmo PI encuentra su función valor V^π, posteriormente una política *greedy* respecto a V^π, y el proceso se repite hasta converger en V^* y π^*.

Otra posibilidad es realizar una evaluación parcial de la política. En la Figura 2.3c se muestra un proceso iterativo similar al que lleva a cabo el algoritmo PI (Figura 2.3b), pero donde la evaluación de la política es parcial: para cada política se halla una función valor que no coincide exactamente con V^π y, aún así, el proceso iterativo converge hacia la solución óptima. En el caso extremo se puede evaluar la política actual en un único estado (o par acción-estado si se trata de funciones Q en

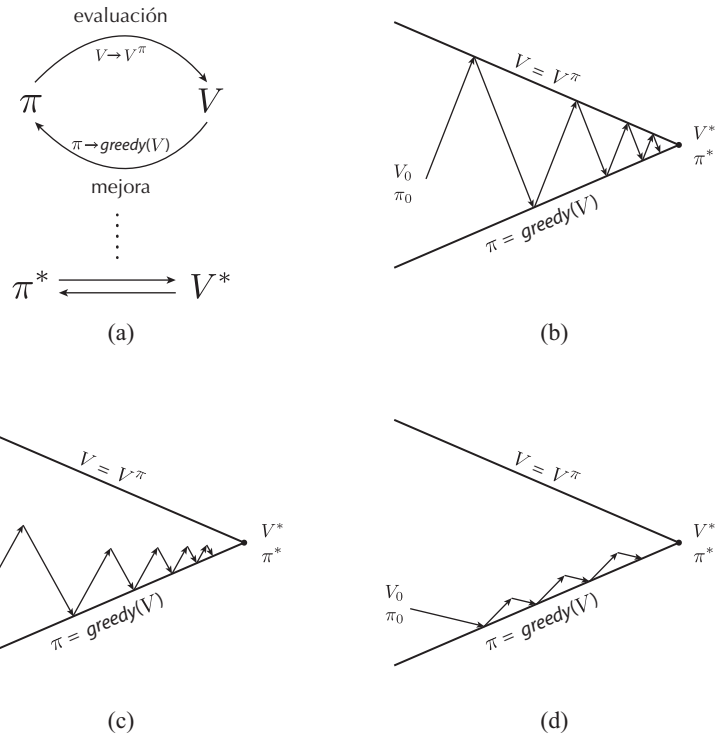


Figura 2.3
(a) Principio generalizado de iteración de políticas: la función valor y la política interaccionan hasta que ambas son óptimas y consistentes entre sí. (b) Proceso de interacción con evaluación completa de la política. (c) Proceso de interacción con evaluación parcial de la política. (d) Proceso de interacción del algoritmo iteración de funciones valor donde únicamente se realiza una iteración del proceso de evaluación.

lugar de V), por lo que la función valor resultante ni siquiera es similar a V^{π}. Como se muestra en la Figura 2.3d esta evaluación es suficiente para que el principio GPI converja. El proceso iterativo que implementa el algoritmo VI se corresponde con este último caso. En dicho algoritmo, además, la política no se representa explícitamente, sino que está implícitamente representada en la función valor y se calcula de forma explícita únicamente para los pares acción-estado actualizados en cada iteración.

Las Figuras 2.3b-d muestran una simplificación de lo que ocurre en el caso real. Las posibles soluciones para cada uno de los procesos, evaluación y mejora, están representados por una línea. Los algoritmos que utilizan el principio GPI hallan soluciones que oscilan entre ambas líneas y el proceso completo únicamente se estabiliza cuando se alcanza la optimalidad. Como puede observarse en las figuras, no es necesario hallar la solución exacta a cada uno de los procesos. Los algoritmos PI y VI pueden considerarse los dos extremos del espectro de algoritmos. Entre ambos extremos hay un gran número de posibles variaciones. En general, mientras ambos procesos se vayan produciendo para todos los pares acción-estado, el resultado final converge hacia la función valor y una política óptimas.

2.6.4. Significado de las ecuaciones de Bellman

Desde un punto de vista intuitivo, la idea fundamental de las ecuaciones de Bellman consiste en, dado un problema, dividirlo en diferentes partes (subproblemas), resolver cada uno de los subproblemas, y combinar las soluciones parciales para formar una solución global. Cuando se utilizan otros métodos más simples, muchos de los subproblemas se generan y resuelven repetidas veces. Por el contrario, los métodos basados en las ecuaciones de Bellman buscan resolver cada subproblema una única vez, reduciendo así la carga computacional: tras resolver un determinado subproblema, su solución es almacenada y reutilizada siempre que es necesario. Esta técnica, que recibe el nombre de *memoization*, no es exclusiva de los métodos de DP, sin embargo, aparece de forma natural en la recursión de las ecuaciones de Bellman. La *memoization* es una característica de vital importancia cuando el número de subproblemas que se repiten aumenta exponencialmente con el número de dimensiones del problema (Skiena, 1998).

A partir del principio de funcionamiento descrito en el párrafo anterior es posible deducir dos características que poseen todos los problemas que pueden ser resueltos mediante DP: subproblemas superpuestos y subestructura óptima (Dasgupta et al., 2008). La primera característica indica que el problema se puede dividir en subproblemas más simples y cuya solución tenga partes comunes. Es decir, la solución de cada subproblema se puede reutilizar varias veces. Por otra parte, la característica de subestructura óptima indica que la solución óptima global se puede construir a partir de las soluciones óptimas de los subproblemas. Ambos conceptos se pueden apreciar más claramente en el problema del camino más corto (Bertsekas, 2005), un ejemplo ilustrativo en el que se pueden aplicar las técnicas de DP. Supongamos que se quiere

viajar entre dos ciudades, representadas por los nodos C1 y C7 en la Figura 2.4. El resto de nodos representan ciudades intermedias, y las líneas que los conectan son las posibles rutas y sus correspondientes distancias. El objetivo es encontrar la ruta más corta; puede apreciarse a simple vista que la solución óptima es la ruta que pasa por las ciudades C1-C3-C5-C7. También se puede observar que el problema de hallar la ruta entre C1 y C7 puede dividirse en los subproblemas de hallar las rutas C5-C7, C6-C7, C2-C5-C7, C3-C5-C7, etc. Asimismo resulta obvio que la solución global óptima puede construirse como la suma de las soluciones óptimas de los subproblemas.

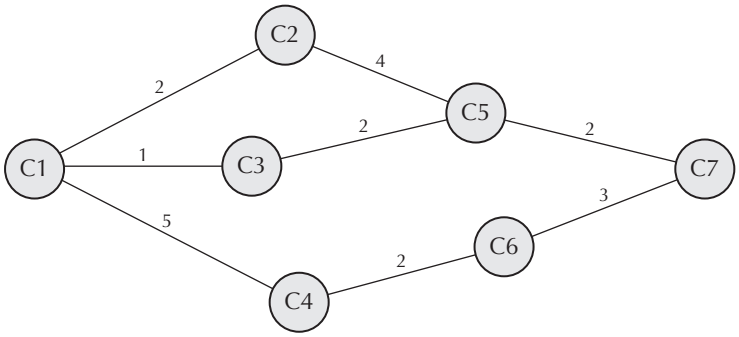


Figura 2.4
Problema del camino más corto. Las líneas representan los posibles caminos entre ciudades, representadas por nodos. El objetivo es encontrar el camino más corto entre C1 y C7.

2.6.5. Búsqueda directa de políticas

Los algoritmos estudiados en las secciones previas estaban basados en funciones valor y las ecuaciones de Bellman. Un enfoque totalmente diferente, pero con el mismo objetivo, es el planteado por los algoritmos de búsqueda directa de políticas. Básicamente todos los algoritmos de DP tienen como objetivo encontrar una política óptima, es decir, se trata de un problema de optimización. En lugar de aplicar las ecuaciones de Bellman, otra opción válida para encontrar políticas óptimas consiste en resolver el problema de optimización en el espacio de políticas, técnica conocida como búsqueda directa de políticas. El criterio de optimización debe tener en cuenta los retornos obtenidos desde cada posible estado inicial. Aunque cualquier técnica de optimización es válida para buscar una política óptima, conviene tener en cuenta que el criterio de optimización puede ser no derivable y contener óptimos locales. Por tanto, para garantizar que la solución obtenida es global, resulta necesario emplear algoritmos de búsqueda global en lugar de aquellos basados en el cálculo de gradientes (Busoniu et al., 2010a). Algunos ejemplos de métodos de optimización global incluyen algoritmos genéticos (Goldberg, 1989), búsqueda tabú (Glover y Laguna, 1997), entropía cruzada (Rubinstein y Kroese, 2004), etc.

La búsqueda directa de políticas ha sido típicamente considerada como un problema más difícil de resolver que el planteamiento basado en funciones valor (Wiering y van Otterlo, 2012). El coste computacional es generalmente mayor además de requerir un elevado número de muestras para converger (Busoniu et al., 2010a).

2.7. Aprendizaje por refuerzo: solución basada en la experiencia

Dentro de los métodos de DP se han descrito los algoritmos PI, VI y búsqueda directa de políticas. Para aplicar cualquiera de ellos es indispensable disponer de un modelo del MDP. En esta sección se presentan algunos de los principales algoritmos de RL, centrándose en aquellos basados en el aprendizaje *temporal difference*, ya que son los algoritmos más extendidos y ampliamente estudiados. En este caso el objetivo sigue siendo exactamente el mismo que en DP: encontrar una política que maximice la recompensa obtenida por el agente. La principal diferencia radica en que los algoritmos de RL están basados en la experiencia en lugar de en el modelo del MDP, una característica que incrementa notablemente sus posibles aplicaciones prácticas. Por otra parte, dado que la política se obtiene a partir de las muestras que adquiere el agente, es necesario que el proceso de muestreo cumpla ciertas propiedades estadísticas para asegurar que la política resultante es óptima.

2.7.1. *Temporal difference*

Temporal difference (TD) hace referencia a una familia de métodos para estimar, o predecir, la función V de una política fija, aunque como veremos en secciones posteriores, el concepto del aprendizaje TD puede ser extendido al caso de funciones Q (Sutton, 1984, 1988). En los métodos TD la función V se estima en base a otras estimaciones previas, técnica que recibe el nombre de *bootstraping* (Sutton y Barto, 1998). Cada vez que el agente realiza una acción el algoritmo TD utiliza la recompensa generada y la estimación actual de V para realizar una nueva estimación de acuerdo a la expresión:

$$V_{k+1}(s_k) = V_k(s_k) + \alpha_k \left[r_{k+1} + \gamma V_k(s_{k+1}) - V_k(s_k) \right] \tag{2.36}$$

donde $\alpha_k \in [0, 1]$ es la secuencia de tasas de aprendizaje que determina la cantidad con la que se actualiza el valor del estado s_k. El término entre corchetes, que se conoce como diferencia temporal y da nombre al método, es la diferencia entre la nueva estimación de la función V, $r_{k+1} + \gamma V_k(s_{k+1})$, y la estimación en el instante temporal anterior, $V_k(s_k)$. Para asegurar la convergencia del algoritmo la secuencia de tasas de aprendizaje debe satisfacer las condiciones de Robbins-Monro (Szepesvári,

2010):

$$\sum_{k=0}^{\infty} \alpha_k = \infty \tag{2.37}$$

y

$$\sum_{k=0}^{\infty} \alpha_k^2 < \infty \tag{2.38}$$

En ocasiones se selecciona una tasa de aprendizaje fija por lo que la segunda condición no se cumple. En dicho caso, si se está estimando una función Q y su correspondiente política, la estimación cambia constantemente cada vez que el agente recibe una nueva recompensa. Esta situación puede resultar útil en problemas no estacionarios ya que permite adaptar la política aprendida a los cambios del entorno. En problemas estacionarios, si el valor fijado de α_k es suficientemente pequeño, es posible que la política aprendida converja hasta la óptima debido a que los cambios producidos en la función valor no llegan a tener ningún efecto sobre la política. En la práctica, para obtener una solución adecuada mediante TD, es necesario realizar un ajuste *ad-hoc* de la tasa de aprendizaje α_k. El pseudo código correspondiente a TD se muestra en el Algoritmo 2.4.

Algoritmo 2.4 *Temporal difference.*

Require: Factor de descuento γ, secuencia de tasas de aprendizaje α_k, política π
1: inicializar la estimación de la función V arbitrariamente, por ejemplo $V_0 \leftarrow 0$
2: **repeat**
3: inicializar episodio s_0
4: **for** cada paso del episodio **do**
5: $a_k = \pi(s_k)$
6: aplicar a_k, observar el estado siguiente s_{k+1} y la recompensa r_{k+1}
7: $V_{k+1}(s_k) = V_k(s_k) + \alpha_k \left[r_{k+1} + \gamma V_k(s_{k+1}) - V_k(s_k)\right]$
8: **end for**
9: **until** cumplir condiciones de convergencia
10: **return** V^π

Igual que muchos otros algoritmos de RL, TD es un método de aproximación estocástica. La ecuación utilizada para actualizar V puede identificarse fácilmente con la expresión general:

$$\text{estimación}_{\text{nueva}} \leftarrow \text{estimación}_{\text{vieja}} + \alpha[\text{objetivo} - \text{estimación}_{\text{vieja}}] \tag{2.39}$$

Al comparar las expresiones (2.39) y (2.36) se observa que el objetivo hacia el que se mueven las estimaciones de la función valor es $r_{k+1} + \gamma V_k(s_{k+1})$. Este objetivo se puede considerar una versión muestreada del operador de Bellman para funciones V:

$$[T^\pi(V)](s_k) = \rho(s_k, \pi(s_k)) + \gamma V^\pi(f(s_k, a_k)) \tag{2.40}$$

donde $\rho(s_k, \pi(s_k))$ se ha sustituido por la recompensa observada r_{k+1} y $f(s_k, a_k)$ por el estado siguiente observado s_{k+1}. Conforme el número de muestras se aproxima a infinito, TD converge asintóticamente a V^π. En el caso determinista, el operador de Bellman se puede calcular exactamente con la recompensa y el estado siguiente observados. En cambio, cuando se trata de un MDP estocástico, el operador de Bellman es el valor esperado de una variable aleatoria (Geist y Pietquin, 2013), por lo que los valores observados únicamente proporcionan una realización de T^π.

Dilema exploración-explotación

El algoritmo TD permite que un agente estime la función valor de una política dada empleando *bootstrapping*. Cuando se aplica el principio de funcionamiento TD a las funciones Q y se combina con el principio GPI, los algoritmos resultantes son capaces de resolver el problema más general de aprender una política óptima. Un requisito que deben cumplir dichos algoritmos es que actualicen todos los pares acción-estado de la función Q de forma indefinida.

Para satisfacer esta condición, la política empleada por el agente debe seleccionar en cada estado todas las acciones posibles con una probabilidad mayor que cero, es decir, debe *explorar* todo el espacio de estados y acciones. En caso contrario, es posible que haya ciertos valores de la función Q que nunca se actualicen, dando lugar a estimaciones incorrectas. Por otra parte, y al mismo tiempo, el agente también debe emplear el conocimiento adquirido para actuar de forma adecuada y obtener la mayor recompensa posible o, dicho de otra forma, debe *explotar* el conocimiento que posee. Ambos requisitos, exploración y explotación, son contrarios y dan lugar a lo que se conoce como dilema exploración-explotación. En la práctica se requiere llegar a un compromiso entre los dos requisitos. La estrategia clásica para balancear el nivel de exploración y explotación consiste en utilizar una política ϵ-*greedy*, la cual selecciona acciones de acuerdo a (Sutton y Barto, 1998):

$$a_k = \begin{cases} a \in \arg\max_a Q_k(s_k, a) & \text{con probabilidad } 1 - \epsilon_k \\ \text{acción uniformemente aleatoria en } A & \text{con probabilidad } \epsilon_k \end{cases} \tag{2.41}$$

siendo $\epsilon_k \in (0, 1)$ la probabilidad de escoger una acción exploratoria en el instante k. Típicamente se escoge un nivel de exploración mayor en las primeras fases del aprendizaje, cuando la estimación de la función Q aún dista de ser óptima y, conforme el agente adquiere mayor conocimiento, el nivel de exploración se suele disminuir. Una forma sencilla de implementar esta estrategia consiste simplemente en escoger $\epsilon_k = 1/k$.

Una posible limitación de la estrategia ϵ-*greedy* es que, cuando se escoge una acción exploratoria, todas las acciones tienen la misma probabilidad de ser elegidas. En cambio, si hay dos acciones subóptimas que parecen mejores que el resto, tal vez una estrategia de exploración más inteligente sería escoger con mayor probabilidad aquellas acciones más prometedoras. Esto es exactamente lo que hace la estrategia de

exploración de Boltzmann (también llamada *softmax*). Para distinguir las acciones más prometedoras emplea su valor Q. La expresión que determina la probabilidad de escoger cada acción es:

$$P(a|s_k) = \frac{e^{Q_k(s_k,a)/\tau_k}}{\sum_{\bar{a}} e^{Q_k(s_k,\bar{a})/\tau_k}} \tag{2.42}$$

donde τ_k es un parámetro positivo llamado temperatura que controla el nivel de exploración. Cuando $\tau_k \to 0$, el proceso de selección de acciones se aproxima a la política *greedy* mientras que, cuando $\tau_k \to \infty$, todas las acciones son seleccionadas de forma uniformemente aleatoria. Para valores intermedios de τ_k, cada acción es seleccionada con una probabilidad proporcional a su valor Q. Igual que ocurría con la estrategia ϵ-*greedy*, el nivel de exploración habitualmente disminuye conforme el agente aprende.

Una tercera aproximación para balancear el nivel de exploración con un enfoque diferente es la técnica conocida como "optimismo frente a la incertidumbre" (Sutton y Barto, 1998). La idea consiste en asignar un valor Q inicial mayor que su valor real. De este modo, cada vez que se visite un par (s_k, a_k), la función Q será ajustada disminuyendo la estimación inicial, por lo que una política *greedy* respecto a Q escogerá las acciones que todavía no se han actualizado, lo que implica un alto nivel de exploración. Aunque el valor exacto de la función Q es desconocido *a priori*, sí que se puede hallar fácilmente el retorno máximo, así que es suficiente con inicializar Q empleando un valor mayor que dicho máximo.

Las tres estrategias de exploración comentadas en esta sección están muy extendidas debido a que son fáciles de implementar y generalmente ofrecen buenos resultados. Sin embargo conviene tener en cuenta que existen muchas otras formas de balancear el nivel de exploración-explotación. En Ratitch (2005), por ejemplo, se puede encontrar un análisis detallado de algunas de ellas.

SARSA

El algoritmo SARSA, propuesto por Rummery y Niranjan (1994), se basa en la regla de actualización TD para estimar la función Q óptima a partir de la cual extrae una política óptima. El nombre del algoritmo se debe a los elementos utilizados para actualizar la función Q (en inglés): estado actual (S), acción actual (A), recompensa (R), estado siguiente (S) y acción siguiente (A), denotados como $(s_k, a_k, r_{k+1}, s_{k+1}, a_{k+1})$. De forma similar al algoritmo TD, SARSA comienza con una estimación arbitraria Q_0 que es actualizada después de cada interacción agente-entorno mediante la expresión:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left[r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1}) - Q_k(s_k, a_k)\right] \tag{2.43}$$

donde α_k es la secuencia de tasas de aprendizaje. El término entre corchetes es la diferencia temporal entre la estimación actual $Q_k(s_k, a_k)$ y la nueva estimación $r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1})$. Se puede observar que la nueva estimación se calcula mediante una operación equivalente al operador de Bellman muestreado. El Algoritmo 2.5 muestra

una posible implementación del algoritmo SARSA combinado con la estrategia de exploración ϵ-*greedy*.

Algoritmo 2.5 SARSA con exploración ϵ-*greedy*.

Require: Factor de descuento γ, secuencia de tasas de aprendizaje α_k, secuencia de exploración ϵ_k

1: inicializar la estimación de la función Q arbitrariamente, por ejemplo $Q_0 \leftarrow 0$
2: **repeat**
3: medir estado inicial s_0
4: $a_0 = \begin{cases} a \in \arg\max_a Q_0(s_0, a) & \text{con probabilidad } 1 - \epsilon_0 \\ \text{acción aleatoria en } A & \text{con probabilidad } \epsilon_0 \end{cases}$
5: **for** cada paso del episodio **do**
6: aplicar a_k, observar el estado siguiente s_{k+1} y la recompensa r_{k+1}
7: $a_{k+1} = \begin{cases} a \in \arg\max_a Q_k(s_k, a) & \text{con probabilidad } 1 - \epsilon_k \\ \text{acción aleatoria en } A & \text{con probabilidad } \epsilon_k \end{cases}$
8: $Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left[r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1}) - Q_k(s_k, a_k) \right]$
9: **end for**
10: **until** cumplir condiciones de convergencia
11: **return** π^*

El principio de funcionamiento de SARSA consiste en evaluar la política actual (línea 8) para, posteriormente, hallar una política mejorada que sea *greedy* respecto a la nueva estimación de Q (línea 7), proceso que se repite de forma iterativa hasta obtener una función Q y una política óptimas. Este principio de funcionamiento es el mismo que realizaba el algoritmo PI perteneciente a la programación dinámica. En general el término PI, además de ser un algoritmo concreto, se utiliza para referirse a una clase de algoritmos basados en este principio. A diferencia del algoritmo PI (Algoritmo 2.2), SARSA no requiere conocer el modelo del MDP, sino que aprende una política de forma *online*, es decir, mientras el agente interactúa con el entorno. Otra diferencia notable es que SARSA únicamente evalúa la política actual en un par estado-acción antes de obtener una política mejorada. Esta variante de PI recibe el nombre de "totalmente optimista" (Bertsekas y Tsitsiklis, 1996; Sutton y Barto, 1998), y permite que la política mejore desde el comienzo del aprendizaje; una característica que resulta de especial interés en los algoritmos de tipo *on-policy*. Se dice que un algoritmo es de tipo *on-policy* cuando requiere que la política que está aprendiendo el agente sea la misma que utiliza para interactuar con el entorno (Sutton y Barto, 1998). Si en el algoritmo SARSA se tuviese que evaluar completamente cada política, al ser de tipo *on-policy*, el agente debería de actuar con el entorno empleando políticas potencialmente erróneas durante largos periodos de tiempo, situación que debe evitarse.

Para asegurar la convergencia hacia la función Q óptima, SARSA debe satisfacer, además de las condiciones de Robbins-Monro sobre el parámetro α, ciertas condiciones de exploración. Tal y como se introdujo en el apartado anterior, los algoritmos que aprenden a partir de datos adquiridos del MDP necesitan muestrear el espacio

de estados y acciones por completo. Para ello la política empleada por el agente debe escoger todas las acciones en todos los estados con una probabilidad mayor que cero. Esto se puede lograr mediante técnicas de exploración como ϵ-*greedy* o Boltzmann. En el caso de los algoritmos *on-policy*, la política aprendida es la misma que se emplea para muestrear el MDP, por lo que es necesario que dicha política se haga gradualmente *greedy* y deje de incluir exploración (Singh et al., 2000). Esta condición se puede cumplir reduciendo asintóticamente hasta cero el parámetro ϵ_k en el caso de exploración ϵ-*greedy* o el parámetro τ_0 para la exploración de Boltzmann.

Q-learning

Otra posible extensión del algoritmo TD al problema general de aprender políticas óptimas es el algoritmo *Q-learning* (Watkins, 1989; Watkins y Dayan, 1992). A partir de una estimación arbitraria de la función Q, el algoritmo *Q-learning* emplea la tupla $(s_k, a_k, s_{k+1}, r_{k+1})$ para actualizar la estimación de Q mediante la regla:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left[r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k) \right] \quad (2.44)$$

siendo $\alpha_k \in (0, 1]$ la secuencia de tasas de aprendizaje. El término entre corchetes es la diferencia temporal entre la nueva estimación de la función Q y la estimación actual. La nueva estimación se calcula con el operador de optimalidad de Bellman muestreado, es decir, sustituyendo en la Ecuación (2.21) la función de recompensa $\rho(s_k, a_k)$ por su valor observado r_{k+1} y el estado siguiente $f(s_k, a_k)$ por el estado siguiente observado s_{k+1}. El pseudo código del algoritmo *Q-learning* con exploración ϵ-*greedy* se muestra en el Algoritmo 2.6.

Algoritmo 2.6 *Q-learning* con exploración ϵ-*greedy*.

Require: Factor de descuento γ, secuencia de tasas de aprendizaje α_k, secuencia de exploración ϵ_k

1: inicializar la estimación de la función Q arbitrariamente, por ejemplo $Q_0 \leftarrow 0$
2: **repeat**
3: medir estado inicial s_0
4: **for** cada paso del episodio **do**
5: $a_{k+1} = \begin{cases} a \in \arg\max_a Q_k(s_k, a) & \text{con probabilidad } 1 - \epsilon_k \\ \text{acción aleatoria en } A & \text{con probabilidad } \epsilon_k \end{cases}$
6: aplicar a_k, observar el estado siguiente s_{k+1} y la recompensa r_{k+1}
7: $Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left[r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k) \right]$
8: **end for**
9: **until** cumplir condiciones de convergencia
10: **return** π^*

El principio de funcionamiento de *Q-learning* consiste en estimar de forma iterativa la función Q óptima (línea 7), un procedimiento similar al empleado por el

algoritmo clásico VI de programación dinámica. Igual que ocurría con el algoritmo PI, el algoritmo VI da nombre a una clase general de algoritmos basados en estimar directamente la función Q óptima, siendo *Q-learning* un ejemplo de esta clase de algoritmos. Al contrario que SARSA, el algoritmo *Q-learning* es de tipo *off-policy*, ya que converge hacia una política óptima independientemente de la política que emplee el agente para interactuar con el entorno. Nótese que en la implementación de *Q-learning* mostrada en el Algoritmo 2.6 la política que emplea el agente sí que depende de la política que está siendo aprendida. Sin embargo, aunque ésta es la implementación más habitual, no es necesario que exista dicha dependencia.

Para converger, *Q-learning* requiere, por una parte, que todos los pares estado-acción de la función Q sean actualizados indefinidamente, es decir, una política exploratoria y, por otra parte, que la secuencia de tasas de aprendizaje satisfaga las condiciones de Robbins-Monro (Szepesvári, 2010). Al contrario que SARSA, al tratarse de un algoritmo *off-policy*, no es necesario que el nivel de exploración se reduzca conforme aumenta el número de iteraciones.

Métodos actor-crítico

Tanto en el algoritmo SARSA como en *Q-learning* la función valor se almacena de forma explícita, mientras que la política únicamente se calcula cuando es necesario a partir de la función valor. Los métodos actor-crítico son una clase de algoritmos caracterizados por representar de forma explícita e independiente la política y la función valor (Barto et al., 1983; Konda, 2002; Grondman et al., 2012). La política es referida como el *actor* debido a que se encarga de seleccionar acciones, y la función valor como el *crítico* porque su tarea es evaluar, o criticar, las acciones seleccionadas por el actor. De acuerdo con este criterio, los algoritmos que únicamente emplean funciones valor reciben el nombre de métodos solo-crítico; mientras que aquellos que únicamente emplean políticas, como los algoritmos de búsqueda directa de políticas, reciben el nombre de métodos solo-actor (Konda y Tsitsiklis, 2003).

En los algoritmos basados en una arquitectura actor-crítico las políticas evaluadas siempre son las mismas que se usan para interactuar con el entorno, se trata, por tanto, de algoritmos *on-policy*. Después de realizar la acción seleccionada por el actor, el crítico evalúa el nuevo estado utilizando típicamente el error TD:

$$\delta_k = r_{k+1} + \gamma V_k(s_{k+1}) - V_k(s_k) \tag{2.45}$$

siendo V_k la función valor actual implementada por el crítico. El error TD es utilizado para reforzar, o debilitar, la preferencia de seleccionar la acción que se acaba de realizar en el estado evaluado. Si la preferencia de la acción a en el estado s se denota como $p(s, a)$, ésta puede ser actualizada usando

$$p_{k+1}(s, a) = p_k(s, a) + \alpha_k \delta_k \tag{2.46}$$

donde α es la secuencia de tasas de aprendizaje. Los valores positivos del error TD indican que la acción evaluada debe elegirse más frecuentemente y, por el contrario, los

valores negativos tienden a debilitar la preferencia de dicha acción. En la Figura 2.5 se muestra de forma esquemática la arquitectura de los algoritmos actor-crítico. Se puede observar que el agente se ha separado en los bloques actor y crítico. La línea punteada indica que el crítico es el responsable de actualizarse a sí mismo (función valor) y al actor (política). Existen otras posibilidades para implementar algoritmos actor-crítico, por ejemplo, variando la forma en la que se actualizan las preferencias $p(s, a)$ o el modo de utilizar la experiencia. También es posible emplear otras técnicas para evaluar la política que no estén basadas en TD.

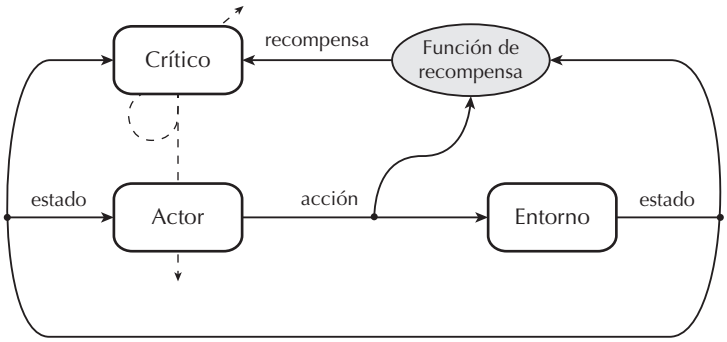


Figura 2.5
Arquitectura de un algoritmo actor-crítico. La línea punteada indica que el crítico es el responsable de actualizarse a sí mismo (función valor) y al actor (política).

Comparado con los algoritmos basados en PI y VI, el hecho de representar la política de manera explícita supone una ventaja cuando el número de acciones es elevado, ya que no es necesario calcular el valor de la función Q de todas las acciones cada vez que se selecciona una acción. Otra ventaja de almacenar directamente la política es que permite aprender políticas estocásticas de manera natural, aprendiendo directamente las probabilidades óptimas de seleccionar cada acción. Estas características han convertido a los métodos actor-crítico en una opción muy popular en el ámbito de la robótica, especialmente los algoritmos basados en gradientes (Peters y Schaal, 2008a,b; Thomas et al., 2013). Por otra parte, en problemas con espacios de estados continuos (ver Capítulo 3), no es posible almacenar la función valor ni la política de forma exacta sino que se tiene que usar algún tipo de aproximador. En estos casos, una desventaja de almacenar por separado la política y la función valor es que el error de aproximación se introduce dos veces (Lagoudakis y Parr, 2003).

2.7.2. Asignación temporal de mérito

Uno de los aspectos más diferenciadores de los métodos introducidos en este capítulo es que permiten resolver de forma eficiente el conocido como problema de asignación de mérito (Sutton, 1992). Este problema, que aparece en la mayoría de sistemas de

aprendizaje con un cierto grado de complejidad, consiste en asignar mérito a cada una de las decisiones internas del sistema que producen una determinada salida, es decir, determinar cuánto ha contribuido cada parte del sistema a dicha salida (Minsky, 1961, 1963). En muchos casos las salidas dependen de las decisiones internas a través de una secuencia de acciones generadas por el sistema de aprendizaje. En otras palabras, las decisiones internas afectan a las acciones realizadas, y son éstas las que tienen una influencia directa sobre las salidas. En estos casos resulta útil separar el problema en dos subproblemas (Sutton, 1984; Haykin, 2008):

- Asignación del mérito que tienen las decisiones internas sobre las acciones (*problema estructural de asignación de mérito*): implica asignar mérito a la estructura interna del sistema que genera las acciones.
- Asignación del mérito que tienen las acciones sobre las salidas (*problema temporal de asignación de mérito*): en este caso es necesario tener en cuenta el instante temporal en el que se realizaron las acciones.

El problema estructural de asignación de mérito es relevante en los métodos de aprendizaje supervisado cuya estructura está formada por diferentes componentes, ya que, a menudo, se debe determinar el componente que debe modificar su comportamiento y en qué cantidad para obtener una mejora en la salida del sistema. Un ejemplo típico es el que aparece en las redes neuronales artificiales, donde cada nodo oculto y cada nodo de la capa de salida puede influir sobre la salida de la red. Durante la etapa de aprendizaje, o entrenamiento, para cada patrón de entrada una de las mayores dificultades consiste en determinar la influencia de cada nodo en la salida de la red y cuánto se deben modificar sus pesos sinápticos para mejorarla, un problema que resuelve el algoritmo *backpropagation* (Rumelhart et al., 1986).

Por otra parte, el problema temporal de asignación de mérito es relevante cuando un sistema de aprendizaje realiza diversas acciones que producen un cierto resultado y se debe determinar cuáles de esas acciones son las responsables del resultado y en qué proporción. Supongamos, por ejemplo, un sistema que debe aprender a jugar al ajedrez. Al final de cada partida el sistema recibe información del resultado obtenido (ganar, perder o empatar). Este resultado está asociado a una secuencia larga de acciones, y no resulta obvio determinar cuales de ellas han sido las responsables del resultado final. Los métodos TD (Sutton, 1988) proporcionan una forma de resolver el problema temporal de asignación de mérito a partir de la experiencia.

2.8. Resumen y discusión

En la primera parte de este capítulo se ha descrito cada uno de los elementos que da lugar al marco matemático de los MPDs sobre el que se apoya la teoría de DP y RL. Posteriormente, en la segunda parte, se han presentado tres tipos de algoritmos para encontrar políticas óptimas cuando se dispone del modelo del MDP. Por un lado,

iteración de políticas (basado en la ecuación de Bellman) e iteración de funciones valor (basado en la ecuación de optimalidad de Bellman), así como una visión global de ambos bajo el principio GPI. Por otro lado, los algoritmos basados en búsqueda directa de políticas que emplean métodos convencionales de optimización global. Por último, en la tercera parte del capítulo se han introducido los métodos TD, los cuales proporcionan una forma de encontrar políticas óptimas a partir de la experiencia.

El objetivo del capítulo es proporcionar una introducción a los principios de funcionamiento de DP y RL. Se ha intentado enfatizar en los conceptos manteniendo una descripción matemática mínima dentro de la rigurosidad necesaria. Como consecuencia de este enfoque, las ecuaciones presentadas son válidas únicamente para MDPs deterministas, a pesar de que una de las principales ventajas de DP/RL es que permiten manejar las incertidumbres asociadas con los sistemas estocásticos.

Debido a que el capítulo no ofrece una revisión exhaustiva, algunos conceptos que pueden resultar importantes en la práctica se han omitido como, por ejemplo, las trazas de elegibilidad (Sutton, 1988), una técnica empleada en los métodos TD que permite acelerar la velocidad de convergencia haciendo un uso más eficiente de la experiencia adquirida por el agente. Tampoco se ha descrito el diseño de la función de recompensa, una etapa que, a menudo, no resulta trivial y puede ser crítica en la aplicación de los algoritmos de RL. Algunos textos clásicos recomiendan mantener la función de recompensa tan simple como sea posible, de forma que el agente únicamente reciba recompensa cuando alcance el objetivo final (Sutton y Barto, 1998). Por el contrario, en algunos trabajos más recientes se destaca que este tipo de recompensas suele dar lugar a tiempos de aprendizaje excesivamente grandes por lo que, en aplicaciones con un cierto grado de complejidad, recomiendan diseñar funciones de recompensa que aporten más información al agente (Busoniu et al., 2010a). En cualquier caso, se debe tener especial cuidado a la hora de incluir información adicional en la función de recompensa porque si se hace incorrectamente puede limitar la capacidad del agente para aprender políticas óptimas.

Los métodos de RL se caracterizan por no requerir un modelo completo del MDP, sin embargo, también pueden beneficiarse de la información previa que se tenga sobre el MDP (Shapiro et al., 2001; Dixon et al., 2000). Una de las posibles formas de incluir dicha información es codificándola en la función de recompensa (Randløv y Alstrøm, 1998; Ng et al., 1999). Los algoritmos presentados aquí se distinguen claramente en función de si emplean, o no, el modelo del MDP. En otros casos esta distinción no es tan evidente, ya que ciertos algoritmos como DYNA (Sutton, 1990) o *prioritized sweeping* (Moore y Atkeson, 1993) aprenden un modelo que posteriormente utilizan junto con la experiencia adquirida por el agente para actualizar la función valor.

Otras áreas de RL que extienden los algoritmos introducidos en este capítulo son, por ejemplo, los MDP parcialmente observables (Kaelbling et al., 1998; Pineau et al., 2006), el uso de la estructura jerárquica que aparece en algunos problemas (Hengst, 2002; Barto y Mahadevan, 2003) y la combinación de diversos agentes en los conocidos como sistemas multi-agente (Panait y Luke, 2005; Busoniu et al., 2008a).

Capítulo 3

Aprendizaje por refuerzo en espacios continuos

3.1. Introducción

En el Capítulo 2 se introdujeron los algoritmos clásicos de programación dinámica (DP) y aprendizaje por refuerzo (RL). En ambos casos se asumía que las funciones valor y las políticas se podían representar de forma exacta. Esto significa que, para cada par estado-acción de una función Q, se tiene que almacenar un valor $Q(s, a)$, o en el caso de una política determinista, para cada estado se tiene que almacenar la acción $\pi(s)$. Un requisito indispensable en dichos casos es que el número de estados y acciones tiene que ser discreto y suficientemente pequeño como para ser almacenado en tablas. Esta clase de algoritmos a menudo reciben el nombre de *métodos exactos*[1] porque permiten calcular la solución exacta al problema tratado o, al menos, proporcionan métodos que convergen a la solución exacta de forma asintótica. Desafortunadamente, en muchas de las aplicaciones reales de DP/RL los estados están definidos por variables que pueden tomar un número muy elevado de valores, o incluso infinito si se trata de variables continuas. En tal caso, las funciones valor y las políticas no pueden ser almacenadas de forma exacta y se hace necesario utilizar algún tipo de aproximación. Cuando los métodos introducidos en el Capítulo 2 se combinan con técnicas de aproximación dan lugar a lo que se conoce como *métodos aproximados*.

La necesidad de los métodos aproximados en DP/RL no surge únicamente debido al problema de almacenar funciones con un número elevado de valores o continuas. Supongamos el caso del algoritmo de DP iteración de funciones valor (Algoritmo 2.3),

[1] Como las funciones valor y las políticas se almacenan en tablas, en parte de la literatura también son referidos como *métodos tabulares*.

donde es necesario aplicar la siguiente regla de actualización:

$$Q_{\ell+1}(s,a) = \rho(s,a) + \gamma \max_{a} Q_{\ell}(f(s,a),a') \tag{3.1}$$

en cada par estado-acción. Obviamente cuando el espacio de estados contiene un número infinito de elementos no es viable actualizar la función Q para todos los estados. Una posible alternativa consiste en seleccionar un conjunto discreto suficientemente representativo de pares estado-acción, aplicar la regla de actualización, y utilizar los valores discretos de $Q_{\ell+1}$ junto con algún método de regresión para calcular una estimación aproximada de la función completa. Una explicación similar también es válida para la etapa de evaluación de la política perteneciente al algoritmo iteración de políticas (Algoritmo 2.2).

Aún en el hipotético caso de disponer de un ordenador capaz de almacenar un número infinito de elementos y siendo capaz de recorrer dichos elementos para aplicar una regla de actualización similar a la de la Ecuación (3.1), no sería posible utilizar los métodos exactos de DP/RL en problemas con espacios continuos. Cabe recordar que una de las condiciones de convergencia es que todos los elementos de la función valor deben ser actualizados indefinidamente. Sin embargo, cuando el espacio de estados es continuo, lo más probable es que el entorno nunca se encuentre exactamente en el mismo estado en más de una ocasión, por lo que el agente difícilmente podrá actualizar toda la función valor. Esto pone de manifiesto otra de las motivaciones de emplear métodos aproximados: el agente debe ser capaz de generalizar su comportamiento a estados que no haya visitado previamente.

Muchos de los algoritmos basados en funciones valor requieren realizar una maximización sobre las acciones cada vez que se actualiza la función valor como, por ejemplo, ocurre en la regla de actualización mostrada en la Ecuación (3.1). Si el espacio de acciones es continuo, dicha maximización es un problema de optimización potencialmente no convexo cuya solución dista de ser trivial. Una alternativa empleada por la mayoría de algoritmos para simplificar el problema de optimización consiste en discretizar el espacio de acciones en un número pequeño de valores. De esta forma es posible calcular el valor de la función Q para todo el conjunto de acciones y encontrar el máximo fácilmente mediante enumeración (Busoniu et al., 2010b). Si el hecho de discretizar el espacio de acciones supone un decremento importante en la calidad de la política óptima, otra opción para trabajar con acciones continuas y evitar tener que resolver un problema de optimización no convexo en cada actualización es emplear alguno de los algoritmos que representan de forma explícita la política. Por ejemplo, aquellos basados en una arquitectura actor-crítico (Konda y Tsitsiklis, 2003) o en búsqueda directa de políticas (Deisenroth et al., 2013b).

En este capítulo se introducen las nociones básicas sobre aprendizaje por refuerzo en espacios continuos, extendiendo las capacidades de los algoritmos clásicos mediante el uso de aproximadores. Dada la amplitud que supondría hacer una revisión exhaustiva de todos los métodos aproximados de DP/RL, resulta inevitable reducir el alcance del capítulo. Los conceptos presentados se restringen a los métodos de RL, es decir,

se asume que no se conoce el modelo completo del MDP. Más específicamente, se van a introducir diversos algoritmos basados en funciones valor, tanto de la clase iteración de políticas como de iteración de funciones valor. Así pues, se asume de forma implícita que en los problemas tratados es posible discretizar el espacio de acciones sin que ello suponga un perjuicio considerable para controlar el MDP. La última parte del capítulo analiza de forma más detallada una subclase de algoritmos caracterizados por hacer un uso eficiente de los datos, una característica indispensable en muchas de las aplicaciones reales.

A continuación se describe la organización del resto del capítulo. En la Sección 3.2 se analizan los distintos tipos de aproximadores y las ventajas e inconvenientes que ofrecen cuando se utilizan para aproximar funciones valor en el contexto del RL. La Sección 3.3 introduce los algoritmos en espacios continuos basados en iteración de políticas, mostrando el caso particular del algoritmo SARSA. Análogamente, la Sección 3.4 realiza una introducción similar para los algoritmos basados en iteración de funciones valor empleando como ejemplo el algoritmo *Q-learning*. Posteriormente, en la Sección 3.5, se estudian los algoritmos de tipo *batch*, describiendo los más populares y realizando un estudio experimental en el que se compara su funcionamiento. El capítulo finaliza con la Sección 3.6 donde se discuten las conclusiones obtenidas.

3.2. Aproximación de funciones en aprendizaje por refuerzo

La aproximación de funciones es un problema que ha sido ampliamente estudiado en el campo del aprendizaje supervisado, por lo que existe una gran variedad de métodos que se pueden combinar con los algoritmos de RL. En general cualquier tipo de aproximador puede considerarse como un mapeado entre el espacio de parámetros y el espacio de funciones que se pretende representar (funciones valor, o políticas, en el caso de RL). Cuando se quiere aproximar una determinada función los parámetros del aproximador se deben ajustar para conseguir tal efecto, proceso conocido como *entrenamiento* o *aprendizaje* (Bertsekas y Tsitsiklis, 1996). Consideremos, por ejemplo, que se quiere aproximar una función Q mediante un aproximador parametrizado por un vector n-dimensional $\boldsymbol{\theta}$. Si denotamos al aproximador como el mapeado $F : \mathbb{R}^n \rightarrow \mathcal{Q}$, siendo $\mathbb{R}^n$ el espacio de parámetros y $\mathcal{Q}$ el espacio de funciones Q; entonces, cada posible vector de parámetros $\boldsymbol{\theta}$ proporciona la representación aproximada de una función Q (Busoniu et al., 2010a):

$$\widehat{Q} = F(\boldsymbol{\theta}) \tag{3.2}$$

donde el símbolo $\widehat{\cdot}$ indica que la función es una aproximación. Otra notación equivalente es:

$$\widehat{Q}(s, a) = [F(\boldsymbol{\theta})](s, a) \tag{3.3}$$

donde se indica explícitamente que la función $F(\boldsymbol{\theta})$ es evaluada en el par estado-acción (s, a). Cuando las funciones Q se representan de forma aproximada una de las

ventajas obtenidas es que, en lugar de almacenar un valor para cada par (s, a), solo es necesario almacenar n parámetros. Lógicamente, cuando el espacio de estados es discreto, el número de parámetros debe ser mucho menor que $|S| \times |A|$, de forma que la representación aproximada sea más compacta. Por otra parte una desventaja es que el espacio de funciones Q que puede representar F es un subconjunto del espacio $\mathcal{Q}$, lo que significa que, al representar una función arbitraria Q, se asume un cierto error de aproximación.

Hay dos aspectos importantes que se deben tener en cuenta para el desarrollo de un aproximador efectivo. En primer lugar el espacio de parámetros debe ser suficientemente completo como para proporcionar una aproximación aceptablemente cercana a la función que se está intentando aproximar. En este sentido escoger un aproximador adecuado suele requerir tener experiencia en el problema tratado o realizar un análisis teórico que proporcione información de la forma de la función a aproximar (aunque sea *a grosso modo*). En segundo lugar es necesario emplear algoritmos efectivos para ajustar los parámetros del aproximador. Normalmente, estos dos aspectos suelen estar en conflicto. Que un aproximador posea una buena capacidad de aproximación suele implicar que contenga un elevado número de parámetros o que la dependencia con los parámetros sea no lineal, lo cual incrementa la complejidad del proceso de entrenamiento (Bertsekas y Tsitsiklis, 1996).

En la mayoría de aplicaciones de aprendizaje supervisado se asume que se dispone de un conjunto de datos de entrenamiento formado por pares del estilo (x_i, y_i), siendo el objetivo construir una función

$$y = f(x) \tag{3.4}$$

que explique lo mejor posible el conjunto de entrenamiento. En el contexto del RL, cuando el objetivo es aproximar la función Q óptima, dicho conjunto de entrenamiento idealmente estaría formado por los pares $[(s, a), Q^*(s, a)]$. Sin embargo la función Q^* es desconocida *a priori*, por lo que el proceso de aproximación tiene que realizarse al mismo tiempo que el algoritmo de RL intenta calcular Q^*. En consecuencia, aplicar métodos de aproximación de funciones en RL a menudo es más difícil que hacerlo en el contexto estándar de aprendizaje supervisado.

En general el mapeado F realizado por el aproximador puede ser no lineal en los parámetros. Un ejemplo de aproximador no lineal ampliamente utilizado es el perceptrón multicapa. En DP y RL resulta de especial interés un subgrupo de aproximadores caracterizados por estar linealmente parametrizados, ya que simplifican el análisis de las propiedades teóricas de los algoritmos. La salida de un aproximador lineal se puede calcular como una combinación lineal del vector de parámetros. En el caso de aproximar una función valor, el valor Q para un determinado par (s, a) es:

$$[F(\boldsymbol{\theta})](s, a) = \sum_{l=1}^{n} \phi_l(s, a)\theta_l = \boldsymbol{\phi}^\top(s, a)\boldsymbol{\theta} \tag{3.5}$$

donde $\boldsymbol{\theta}$ es el vector n-dimensional de parámetros y $\boldsymbol{\phi}(s, a) = [\phi_1(s, a), \ldots, \phi_n(s, a)]^\top$ es un vector formado por las funciones fijas $\phi_1, \ldots, \phi_n : S \times A \to \mathbb{R}$, habitualmente

conocidas como funciones base o características (Busoniu et al., 2010a). Las funciones base pueden ser construidas de diversas formas, algunos ejemplos populares en RL son las funciones base binarias utilizadas en la codificación en baldosas, método también conocido originalmente como CMAC (*cerebellar model articulation controller*) (Albus, 1975, 1981) y las funciones de base radial (gausianas principalmente) empleadas en las redes RBF (*radial basis function*) de base fija, las cuales se analizarán más detalladamente en la Sección 3.2.3.

3.2.1. Maldición de la dimensionalidad

Una limitación de los aproximadores lineales comúnmente utilizados en RL es que habitualmente se vuelven intratables cuando el espacio de estados está definido por un número elevado de dimensiones (Szepesvári, 2010). Dado que las características usadas suelen ser de ámbito local, el número de características necesarias para aproximar una determinada función aumenta de forma exponencial con la dimensionalidad de la misma, efecto conocido como "maldición de la dimensionalidad" (Bellman, 1957). En ciertas aplicaciones prácticas con decenas, o incluso cientos de variables de estado, el aumento exponencial del número requerido de características puede suponer una restricción importante desde el punto de vista computacional. Por otra parte, conviene tener en cuenta que el espacio de estados debe ser escogido por el usuario, por lo que habitualmente se escoge de una manera excesivamente conservadora, incluyendo variables de estado que son irrelevantes. Aún suponiendo que el número de variables sea adecuado *a priori*, puede ocurrir que los estados muestreados por el agente caigan dentro (o cerca) de un *submanifold* de baja dimensionalidad dentro del espacio de estados escogido (Szepesvári, 2010).

Supongamos, como ejemplo ilustrativo de este último caso, un brazo robótico industrial con 6 grados de libertad. Asumiendo que la dinámica del brazo es de segundo orden, el estado del brazo se puede describir exactamente con 12 variables (6 ángulos y sus correspondientes velocidades angulares), es decir, la dimensionalidad intrínsica del espacio de estados es 12. Una posible representación alternativa del estado del brazo robótico consiste en tomar imágenes con una cámara de alta resolución en diferentes instantes temporales (para tener en cuenta la dinámica) y desde diferentes ángulos (para evitar zonas ocultas). Esta posible representación del estado, basada en imágenes, puede tener fácilmente una dimensionalidad del orden de millones de variables, a pesar de que la dimensionalidad intrínseca del problema sigue siendo 12. Así pues, en estos casos, para poder aproximar las funciones valor, es necesario emplear métodos de aproximación que sean capaces de descubrir y aprovechar la dimensionalidad intrínseca (normalmente baja) de los problemas de alta dimensionalidad. Algunos métodos prometedores en este aspecto son los aproximadores no lineales (como las redes neuronales y las redes RBF con funciones base variables), los métodos basados en proyecciones aleatorias (Dasgupta y Freund, 2008; Ghavamzadeh et al., 2010) y los métodos no paramétricos (como los árboles de decisión y las máquinas de vectores soporte) (Szepesvári, 2010).

3.2.2. Métodos no paramétricos

En los métodos de aproximación descritos hasta ahora se asumía que el usuario debía fijar el espacio de parámetros de antemano seleccionando, por ejemplo, el número de neuronas y capas ocultas en el caso de una red neuronal. Dichos métodos se pueden englobar bajo el nombre de *métodos paramétricos*. Por otra parte existe otra clase de métodos conocidos como *métodos no paramétricos*. A pesar de su nombre, estos métodos también emplean un conjunto de parámetros para realizar la aproximación, la diferencia radica en que el número de parámetros y la estructura del aproximador no la escoge el usuario, sino que se obtiene a partir de los datos del conjunto de entrenamiento y puede cambiar y adaptarse en función de los datos disponibles (Alpaydin, 2004). Por ejemplo, en el método de regresión de los k vecinos más cercanos (*k-nearest neighbours*, k-NN), dado un conjunto de m datos $D = [((s,a)_1, Q_1), \ldots, ((s,a)_m, Q_m)]$, el valor del par (s,a) es aproximado como:

$$\widehat{Q}(s,a) = \frac{1}{k} \sum_{\substack{(s,a)_i \in \\ N_k(s,a)}} Q_i \tag{3.6}$$

donde $N_k(s,a)$ es la vecindad del par (s,a) definida por los k pares más cercanos a (s,a) en el conjunto D. El concepto de "cercano" necesariamente implica el uso de alguna métrica, típicamente la distancia euclidea (Hastie et al., 2009). Es decir, el valor aproximado $\widehat{Q}(s,a)$ se calcula como un promedio de los valores Q pertenecientes a los k pares más cercanos a (s,a) contenidos en el conjunto de datos. Nótese, que aunque en los métodos no paramétricos la estructura del aproximador se determina en función de los datos, siempre existe algún hiperparámetro que necesita ser ajustado manualmente y cuyo valor puede influir en la calidad de la aproximación como, por ejemplo, el número de vecinos k en el caso del algoritmo k-NN.

Otros métodos no paramétricos ampliamente utilizados son los métodos *kernel* (Shawe-Taylor y Cristianini, 2004), entre los que destacan las máquinas de vectores soporte (Scholkopf et al., 1999) y los procesos Gaussianos (Rasmussen y Williams, 2006), y los árboles de regresión (Breiman et al., 1984). Estos métodos de aproximación han sido combinados con diferentes algoritmos de RL, como por ejemplo métodos *kernel* con iteración de funciones valor (Farahmand et al., 2009) y con iteración de políticas (Bethke et al., 2008) o árboles de regresión con evaluación de políticas (Jodogne et al., 2006).

Los aproximadores no paramétricos se caracterizan por su elevada flexibilidad. Esta propiedad, que *a priori* es deseable, puede causar problemas de estabilidad cuando se combina con ciertos algoritmos de RL ya que, al modificarse la estructura del aproximador durante el aprendizaje de la política óptima, resulta difícil obtener garantías teóricas de convergencia (Busoniu et al., 2010a). Otra característica de los métodos no paramétricos es su capacidad para adaptar la complejidad del aproximador en función de la cantidad de datos disponibles. Esto suele ser una ventaja en las aplicaciones donde resulta costoso obtener muestras del MDP, bien sea en tiempo o en recursos.

Por el contrario, si la cantidad de datos adquiridos aumenta rápidamente, la carga computacional y la cantidad de memoria requerida por el aproximador también suele aumentar de manera proporcional. Por tanto, los métodos no paramétricos resultan inapropiados para ser empleados junto con algoritmos de RL cuyo aprendizaje sea *online*, ya que en esta clase de algoritmos la cantidad de datos muestreada por el agente puede aumentar de forma indefinida.

Por otra parte, si se comparan lo métodos no paramétricos con los paramétricos, estos últimos tienen que ser suficientemente flexibles como para aproximar, modificando únicamente sus parámetros, todas las funciones valor que se producen durante el proceso de aprendizaje. Aunque dicha flexibilidad habitualmente requiere que el aproximador sea no lineal, los algoritmos de RL combinados con aproximadores no lineales son difíciles de tratar desde el punto de vista teórico y en la práctica a menudo se comportan de forma inestable. Por este motivo, y a pesar de sus limitaciones, en la mayor parte de la literatura se emplean aproximadores lineales. También existen aproximadores lineales que incrementan su flexibilidad al permitir la introducción de nuevas funciones base según sea necesario (Munos y Moore, 2002; Szepesvári y Smart, 2004; Waldock y Carse, 2008). En esta clase de algoritmos, el método de aproximación no se puede considerar puramente paramétrico, ya que la estructura del aproximador cambia en función de los datos, una de las propiedades que caracterizan a los métodos no paramétricos.

3.2.3. Aproximación de funciones Q con redes RBF de base fija

Los aproximadores lineales poseen varias características que los hacen especialmente atractivos para emplearlos junto con los algoritmos de RL. En esta sección se describe más detalladamente esta clase de aproximadores debido a que aparecerá de forma recurrente en capítulos posteriores, concretamente en el caso de que el espacio de acciones pueda ser discretizado en un conjunto manejable de acciones.

Supongamos un problema con un espacio de acciones continuo denotado por A en el que es posible seleccionar un conjunto finito de acciones $a_1, a_2, \ldots, a_M$ que permita obtener políticas adecuadas. El espacio de acciones discreto resultante de dicho proceso se denota como $A_d = \{a_1, a_2, \ldots, a_M\}$. Una posible opción para aproximar la función $Q(s, a)$ consiste en definir un conjunto de N funciones base que únicamente dependan del estado s, $\bar{\phi}_1, \ldots, \bar{\phi}_N : S \to \mathbb{R}$ replicadas para cada una de las acciones contenidas en A_d. El valor aproximado de Q para cualquier par estado-acción se puede calcular como (Busoniu et al., 2010a):

$$[F(\boldsymbol{\theta})](s, a_j) = \boldsymbol{\phi}^\top(s, a_j)\boldsymbol{\theta} \tag{3.7}$$

donde el vector de funciones base $\boldsymbol{\phi}^\top(s, a_j)$ es la combinación de las funciones base para cada una de las acciones y cuyos valores son igual a 0 para todas las acciones

diferentes a la acción actual, es decir:

$$\boldsymbol{\phi}(s, a_j) = [\underbrace{0, \ldots, 0}_{a_1}, \ldots, 0, \underbrace{\bar{\phi}_1, \ldots, \bar{\phi}_N}_{a_j}, 0, \ldots, \underbrace{0, \ldots, 0}_{a_M}]^\top \in \mathbb{R}^{NM} \tag{3.8}$$

Nótese que en las Ecuaciones (3.7) y (3.8), el subíndice j de las acciones indica que la acción debe pertenecer al conjunto A_d, ya que en otro caso la salida del aproximador es siempre igual a 0. Una aproximación de este tipo puede considerarse como una forma de representar M porciones de la función Q, una por cada posible acción.

Las funciones base $\bar{\phi}(s)$ pueden definirse de múltiples formas, una de ellas es mediante RBFs. Una RBF típica es la función gaussiana, cuya salida depende de la distancia entre el estado s y el centro de la función, siendo máxima cuando s coincide con el centro. La velocidad a la que disminuye la salida de la función conforme s se aleja del centro viene determinada por el parámetro σ, que define el "ancho" de la función gaussiana. La distancia se puede calcular mediante la métrica que se considere más oportuno, siendo la más habitual la distancia Euclidea. En dicho caso, el valor de la i-ésima función base gaussina se calcula como:

$$\bar{\phi}_i(s) = \exp\left(-\frac{||s - c_i||^2}{2\sigma_i^2}\right) \tag{3.9}$$

donde el vector $\mathbf{c}_i = [c_{i,1}, c_{i,2}, \ldots, c_{i,d}]^\top \in \mathbb{R}^d$ indica la posición de su centro y σ el ancho de la misma. En la Figura 3.1 se muestran las funciones gaussianas correspondientes a un aproximador formado por 4 funciones cuando el espacio de estados es unidimensional. Lógicamente, la dimensionalidad de las gaussianas tiene que ser igual a la del espacio de estados. Por ejemplo, en la Figura 3.2 se muestra el caso de $d = 2$ con 16 funciones, 4 por cada dimensión. Al comparar ambas figuras se puede observar que, a medida que aumenta el número de dimensiones, para cubrir el espacio de estados con una determinada densidad es necesario incrementar el número de funciones base exponencialmente, tal y como describe el efecto de la "maldición de la dimensionalidad".

Las redes RBF de base fija no son los únicos aproximadores lineales usados en RL. Además de la ya comentada codificación en baldosas, otros aproximadores que comparten una estructura similar pero que emplean métodos diferentes para generar las funciones base son, por ejemplo, *state aggregation* (Singh et al., 1995; Bertsekas y Tsitsiklis, 1996), interpolación multilineal (Davies, 1997) y triangulación de Kuhn (Munos y Moore, 2002).

3.3. Iteración de políticas en espacios continuos

Generalmente todos los algoritmos basados en iteración de políticas comparten el mismo principio de funcionamiento: parten de una política inicial arbitraria para la que estiman su función valor, posteriormente, esta función valor se emplea para

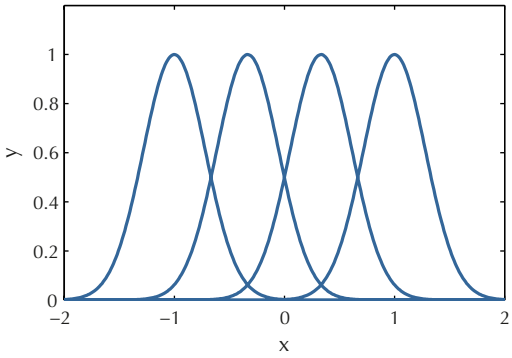


Figura 3.1
Funciones base gaussianas de una red RBF cuando el espacio de entrada es unidimensional.

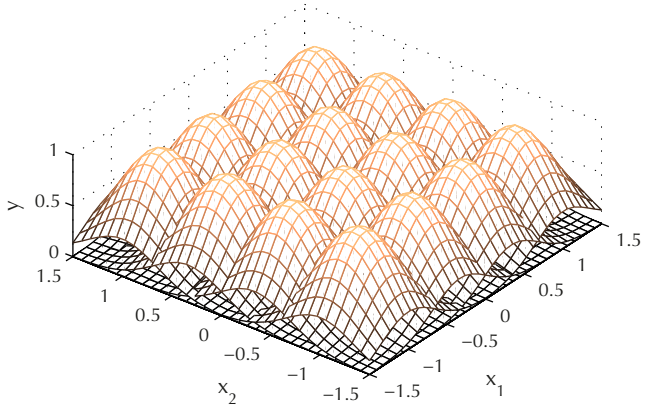


Figura 3.2
Funciones base gaussianas de una red RBF cuando el espacio de entrada es bidimensional.

encontrar una nueva política mejor que la anterior. Este proceso, introducido en la Sección 2.6.1, se repite hasta converger en la política óptima. Cuando el espacio de estados es grande, o continuo, tanto las funciones valor como las políticas no pueden ser representadas de forma exacta, por lo que es necesario usar una representación aproximada. En el Algoritmo 3.1 se muestra el pseudo código de un algoritmo genérico basado en iteración de políticas con aproximación de funciones.

La etapa de evaluación de la política básicamente consiste en resolver la ecuación de Bellman de forma aproximada. Normalmente se puede evitar representar explícitamente la política, por lo que únicamente es necesario aproximar la función Q. La maximización llevada a cabo sobre las acciones en la línea 4 puede resolverse fácilmente cuando el espacio de acciones es discreto, en caso contrario puede suponer una dificultad añadida.

Algoritmo 3.1 Iteración de políticas con aproximación de funciones.

Require: Política inicial $\widehat{\pi}_0$
1: $\ell = 0$
2: **repeat**
3: calcular $\widehat{Q}^{\widehat{\pi}_\ell}$, la función Q de $\widehat{\pi}_l$ {evaluación de la política}
4: $\widehat{\pi}_{\ell+1}(s) \approx \arg\max_a \widehat{Q}^{\widehat{\pi}_\ell}(s,a)$ {mejora de la política}
5: $\ell \leftarrow \ell + 1$
6: **until** $\widehat{\pi}_{\ell+1} = \widehat{\pi}_\ell$
7: **return** $\widehat{\pi}^* = \widehat{\pi}_\ell$

El pseudo código mostrado en el Algoritmo 3.1 es de carácter general, lo cual significa que la forma de implementar las etapas de evaluación y mejora de la política dará lugar a varios algoritmos concretos dependiendo de diversos factores como, por ejemplo, si se conoce el modelo del MDP, el tipo de aproximador empleado, si el aprendizaje se realiza *online*, etc. A continuación se muestra un ejemplo representativo consistente en una versión aproximada del algoritmo SARSA. En este ejemplo el aproximador empleado es lineal en los parámetros y para realizar el ajuste de dichos parámetros se utiliza la técnica de descenso por gradiente estocástico.

La etapa que conlleva una mayor dificultad en el algoritmo SARSA es la que se encarga de evaluar la política actual, es decir, obtener la función Q^π dada una política fija π. Cuando se emplea aproximación de funciones, el objetivo en esta etapa es minimizar la siguiente función de coste teórica:

$$J_{Q^\pi}(\boldsymbol{\theta}) = ||Q^\pi - \widehat{Q}|| \tag{3.10}$$

Nótese que, por cuestiones de simplicidad, la función Q aproximada se ha denotado como $\widehat{Q}(s,a) = [F(\boldsymbol{\theta})](s,a)$, dejando la dependencia con el vector de parámetros implícita. Una forma de encontrar los parámetros $\boldsymbol{\theta}$ que minimicen dicha función es aplicar descenso por gradiente estocástico. Supongamos que, en el instante k, tenemos acceso a una observación (posiblemente ruidosa, siempre y cuando el ruido sea aditivo y blanco) de Q^π denotada como q_k^π, entonces la función de coste empírica es (Geist y Pietquin, 2013):

$$\widehat{J}_{Q^\pi}(\boldsymbol{\theta}) = \sum_k \left(q_k^\pi - \widehat{Q}(s_k, a_k)\right)^2 \tag{3.11}$$

Cuando el agente realiza una transición desde el estado s_k realizando la acción a_k, es posible ajustar los parámetros del aproximador con una cantidad proporcional al gradiente de la función de coste empírica, evaluada únicamente en el par (s_k, a_k). Para ello, se aplica la regla de Widrow-Hoff (Haykin, 2008):

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{1}{2}\alpha_k \frac{\partial}{\partial \boldsymbol{\theta}_k}\left[q_k^\pi - \widehat{Q}_k(s_k, a_k)\right]^2 \tag{3.12}$$

$$= \boldsymbol{\theta}_k + \alpha_k \left[q_k^\pi - \widehat{Q}_k(s_k, a_k)\right] \frac{\partial}{\partial \boldsymbol{\theta}_k}\widehat{Q}_k(s_k, a_k) \tag{3.13}$$

donde α_k es una secuencia de tasas de aprendizaje que satisface las condiciones de Robbins-Monro (Ecuación (2.37)). Una de las condiciones que debe cumplir el aproximador es que sea derivable respecto a los parámetros.

Sin embargo, la regla de actualización definida en la Ecuación (3.12) no puede ser aplicada en la práctica puesto que no es posible observar el valor q_k^π. En su lugar, se puede calcular una estimación de dicho valor aplicando el operador de Bellman muestreado:

$$r_{k+1} + \gamma \widehat{Q}_k(s_{k+1}, a_{k+1})$$

Al incluir esta estimación en la regla de actualización anterior se obtiene la ecuación utilizada por el algoritmo SARSA:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \left[r_{k+1} + \gamma \widehat{Q}_k(s_{k+1}, a_{k+1}) - \widehat{Q}_k(s_k, a_k) \right] \frac{\partial}{\partial \boldsymbol{\theta}_k} \widehat{Q}_k(s_k, a_k) \tag{3.14}$$

Donde el término entre corchetes se corresponde con una aproximación de la diferencia temporal empleada en el algoritmo TD. Cuando el aproximador es lineal en los parámetros, resulta mucho más sencillo calcular el gradiente. Aplicando la Ecuación (3.5) sobre la regla de actualización se obtiene:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \left[r_{k+1} + \gamma \boldsymbol{\phi}^\top(s_{k+1}, a_{k+1}) \boldsymbol{\theta}_k - \boldsymbol{\phi}^\top(s_k, a_k) \boldsymbol{\theta}_k \right] \boldsymbol{\phi}(s_k, a_k) \tag{3.15}$$

Además, el hecho de que el aproximador sea lineal implica que la función $\widehat{J}$ tiene un único mínimo global, lo cual mejora las propiedades de convergencia. Igual que ocurría con la versión exacta del algoritmo SARSA, en la versión aproximada también es necesario que el agente incorpore alguna técnica de exploración para obtener muestras (s, a) con $a \neq \pi(s)$. El Algoritmo 3.2 muestra el pseudo código de SARSA con aproximación de funciones y exploración ϵ-*greedy*. En (Tsitsiklis y Van Roy, 1997) puede encontrarse un análisis detallado sobre las propiedades de convergencia del algoritmo TD con aproximación lineal, las cuales pueden extenderse al algoritmo SARSA (Szepesvári, 2010).

3.4. Iteración de funciones valor en espacios continuos

Cuando se aplica un algoritmo basado en iteración de funciones valor a un problema cuyo espacio de estados es continuo, aparecen las mismas dificultades que en el caso de los algoritmos basados en iteración de políticas. Las técnicas para hacer frente a estas dificultades también consisten en representar las funciones valor mediante aproximadores. Recordemos que el principio de funcionamiento de los algoritmos de iteración de funciones valor consistía en calcular una secuencia de funciones valor que convergía hacia la función valor óptima, a partir de la cual se obtenía una política óptima (ver Sección 2.6.2). De entre todos los algoritmos basados en este principio, el

Algoritmo 3.2 SARSA con aproximación de funciones y exploración ϵ-*greedy*.

Require: Factor de descuento γ, secuencia de tasas de aprendizaje α_k, secuencia de exploración ϵ_k, funciones base $\phi_1, \ldots, \phi_n : S \times A \to \mathbb{R}$

1: inicializar el vector de parámetros arbitrariamente, por ejemplo $\boldsymbol{\theta}_0 \leftarrow 0$
2: **repeat**
3: medir estado inicial s_0
4: $a_0 = \begin{cases} a \in \arg\max_a \widehat{Q}_0(s_0, a) & \text{con probabilidad } 1 - \epsilon_0 \\ \text{acción aleatoria en } A & \text{con probabilidad } \epsilon_0 \end{cases}$
5: **for** cada paso del episodio **do**
6: aplicar a_k, observar el estado siguiente s_{k+1} y la recompensa r_{k+1}
7: $a_{k+1} = \begin{cases} a \in \arg\max_a \widehat{Q}_k(s_k, a) & \text{con probabilidad } 1 - \epsilon_k \\ \text{acción aleatoria en } A & \text{con probabilidad } \epsilon_k \end{cases}$
8: $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \left[r_{k+1} + \gamma \boldsymbol{\phi}^\top(s_{k+1}, a_{k+1})\boldsymbol{\theta}_k - \boldsymbol{\phi}^\top(s_k, a_k)\boldsymbol{\theta}_k \right] \boldsymbol{\phi}(s_k, a_k)$
9: **end for**
10: **until** cumplir condiciones de convergencia
11: **return** π^*

más estudiado posiblemente sea el algoritmo *Q-learning* (Horiuchi et al., 1996; Jouffe, 1998; Fernández y Borrajo, 2000; Glorennec, 2000; Murphy, 2005; Sherstov y Stone, 2005), por lo que se empleará dicho algoritmo como ejemplo ilustrativo de los métodos de iteración de funciones valor en espacios continuos.

El algoritmo *Q-learning* con aproximación de funciones puede obtenerse siguiendo un procedimiento similar al llevado a cabo previamente con el algoritmo SARSA. En este caso el objetivo es estimar directamente Q^*, por lo que la función de coste teórica a minimizar viene dada por la diferencia entre la función valor óptima y la función valor estimada:

$$J_{Q^*}(\boldsymbol{\theta}) = ||Q^* - \widehat{Q}|| \tag{3.16}$$

Esta función de coste teórica da lugar a una función de coste empírica donde se emplea, en cada instante k, las observaciones q_k^* y $\widehat{Q}(s_k, a_k)$. Evidentemente, el valor q_k^* no es directamente observable, sino que se estima mediante el operador de optimalidad de Bellman muestreado:

$$r_{k+1} + \gamma \max_{a'} \widehat{Q}_k(s_{k+1}, a')$$

Al combinar esta estimación con la regla de Widrow-Hoff se obtiene la ecuación que actualiza los parámetros del aproximador en el algoritmo *Q-learning*. Además, si el aproximador es lineal en los parámetros, dicha ecuación se puede simplificar, obteniendo:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \left[r_{k+1} + \gamma \max_{a'} \boldsymbol{\phi}^\top(s_{k+1}, a')\boldsymbol{\theta}_k - \boldsymbol{\phi}^\top(s_k, a_k)\boldsymbol{\theta}_k \right] \boldsymbol{\phi}(s_k, a_k) \tag{3.17}$$

Cuando un agente emplea la Ecuación (3.17) para actualizar la estimación de Q^* es necesario que la política seguida sea exploratoria. Una de las diversas formas de

asegurar la exploración consiste en aplicar una política ϵ-*greedy*, tal y como se muestra en el pseudo código del Algoritmo 3.3.

Algoritmo 3.3 *Q-learning* con aproximación de funciones y exploración ϵ-*greedy*.

Require: Factor de descuento γ, secuencia de tasas de aprendizaje α_k, secuencia de exploración ϵ_k, funciones base $\phi_1, \ldots, \phi_n : S \times A \to \mathbb{R}$

1: inicializar la estimación de la función Q arbitrariamente, por ejemplo $\boldsymbol{\theta}_0 \leftarrow 0$
2: **repeat**
3: medir estado inicial s_0
4: **for** cada paso del episodio **do**
5: $a_k = \begin{cases} a \in \arg\max_a Q_k(s_k, a) & \text{con probabilidad } 1 - \epsilon_k \\ \text{acción aleatoria en } A & \text{con probabilidad } \epsilon_k \end{cases}$
6: aplicar a_k, observar el estado siguiente s_{k+1} y la recompensa r_{k+1}
7: $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \left[r_{k+1} + \gamma \max_{a'} \boldsymbol{\phi}^\top(s_{k+1}, a')\boldsymbol{\theta}_k - \boldsymbol{\phi}^\top(s_k, a_k)\boldsymbol{\theta}_k \right] \boldsymbol{\phi}(s_k, a_k)$
8: **end for**
9: **until** cumplir condiciones de convergencia
10: **return** π^*

En esta ocasión, al tratarse de un algoritmo *off-policy*, las propiedades de convergencia de TD no pueden extenderse al algoritmo *Q-learning*. De hecho, a pesar de ser uno de los algoritmos más utilizados en la práctica, se sabe que puede diverger cuando no se le aplica ninguna restricción. La única prueba de convergencia es la proporcionada por Melo et al. (2008), donde se asumen fuertes restricciones sobre la distribución de los pares estado-acción muestreados. Más recientemente, la aparición de la familia de algoritmos *gradient temporal difference* (GTD) (Sutton et al., 2008, 2009) ha dado lugar a una nueva versión de *Q-learning*, conocida como *greedy* GD (Maei et al., 2010), cuya convergencia sí está asegurada. Sin embargo, aún en el caso de utilizar un aproximador lineal, la función de coste J no es convexa por lo que puede converger hacia mínimos locales (Maei, 2011).

3.5. *Batch* RL: uso eficiente de los datos

Los dos algoritmos introducidos en las secciones previas (SARSA y *Q-learning* combinados con descenso por gradiente estocástico) son solo una pequeña representación de la amplia gama de métodos de RL que permiten resolver problemas con espacios continuos. El motivo de incluir estos dos algoritmos y no otros es doble. Por una parte suponen una extensión natural de los algoritmos introducidos en el Capítulo 2 para problemas discretos. Y, por otra parte, fueron los primeros algoritmos de la familia TD que permitieron encontrar políticas óptimas. Como consecuencia de ello, y junto con su facilidad de implementación y la calidad de los resultados ofrecidos, son dos de los algoritmos más representativos de RL. De hecho, SARSA y *Q-learning* suelen ser los únicos algoritmos incluidos en la mayoría de libros sobre aprendizaje

automático e inteligencia artificial donde se dedica algún capítulo al RL (Russell y Norvig, 1995; Alpaydin, 2004; Haykin, 2008).

A pesar de su popularidad, existen alternativas que pueden mejorar ciertos aspectos de los algoritmos SARSA y *Q-learning*. Esta sección se centra en una clase específica de algoritmos conocida como *batch* RL. Estos algoritmos se caracterizan porque la política no se aprende al mismo tiempo que el agente interactúa con el entorno para adquirir experiencia, sino que ésta es adquirida y almacenada en una etapa previa y, posteriormente, procesada. Se trata, por tanto, de algoritmos *offline*. La experiencia habitualmente se almacena en tuplas del estilo $(s_k, a_k, r_{k+1}, s_{k+1})$, de forma que cada tupla contiene toda la información relacionada con una interacción agente-entorno. La política empleada para adquirir la experiencia se puede elegir de forma arbitraria, pero debe contener un cierto grado de exploración para que las muestras adquiridas sean representativas de todo el espacio de estados. En el Algoritmo 3.4 se muestra la estructura genérica de esta clase de algoritmos. La función ALGORITMOBATCH() es la encargada de obtener la política a partir del conjunto de experiencias D. En las siguientes secciones se estudiarán tres métodos diferentes para realizar esta tarea.

Algoritmo 3.4 Estructura genérica de los algoritmos *batch* RL.

Require: Política π empleada para adquirir la experiencia, número de episodios m
1: // Adquisición de la experiencia
2: $D \leftarrow 0$
3: $episodio \leftarrow 0$
4: **repeat**
5: medir estado inicial s_0
6: **for** cada paso del episodio **do**
7: $a_k = \pi(s_k)$
8: aplicar a_k, observar el estado siguiente s_{k+1} y la recompensa r_{k+1}
9: $experiencia \leftarrow (s_k, a_k, r_{k+1}, s_{k+1})$
10: $D \leftarrow$ AGREGAR($experiencia$)
11: **end for**
12: **until** realizar m episodios
13: // Procesado de la experiencia
14: $\pi^* =$ ALGORITMOBATCH(D)
15: **return** π^*

Una ventaja de esta clase de algoritmos es que hacen un uso eficiente de la experiencia. En otros métodos, como por ejemplo *Q-learning*, cada vez que el agente realiza una acción en un estado determinado y recibe la recompensa correspondiente, los datos de la transición realizada se emplean para realizar una nueva estimación de la función Q y, posteriormente, esos datos nunca más se vuelven a utilizar, a pesar de que son útiles para realizar futuras estimaciones de Q (Lin, 1992). Este procedimiento conlleva un uso ineficiente de la experiencia adquirida por el agente, lo que en la práctica se traduce en largos tiempos de interacción agente-entorno hasta lograr que

la política aprendida converja.

Desafortunadamente, en ciertas aplicaciones prácticas, adquirir experiencias puede ser un proceso muy costoso, tanto en recursos como en tiempo. Supongamos, por ejemplo, el caso de un problema médico donde el objetivo es que un agente obtenga una política óptima para la administración de un fármaco determinado. En este caso el paciente puede ser modelado como el entorno, y las acciones se corresponderían con las diferentes dosis de fármaco que recibe el paciente. Para adquirir una experiencia es necesario evaluar el estado del paciente mediante un análisis clínico (s), administrar la dosis de medicamento que se estime oportuno (a), esperar el tiempo necesario para que el medicamento haga efecto, volver a medir el nuevo estado del paciente (s') y calcular la recompensa en función de los criterios clínicos (r). Es decir, para adquirir cada una de las experiencias se necesita emplear una cantidad considerable de recursos.

En este tipo de aplicaciones no es viable usar algoritmos de RL que requieran una cantidad de datos muy elevada. Los algoritmos de tipo *batch* almacenan todos los datos adquiridos para reutilizarlos posteriormente múltiples veces. Este enfoque les permite extraer más conocimiento de cada una de las experiencias. Por otro lado, dicho proceso requiere una mayor capacidad computacional y de memoria, lo cual no suele suponer ningún inconveniente debido a que los algoritmos de tipo *batch* trabajan de forma *offline* y, por tanto, no requieren procesar las experiencias en tiempo real (Wiering y van Otterlo, 2012).

En ocasiones, los algoritmos *batch* son considerados como métodos basados en el modelo del MDP (Kalyanakrishnan, 2011). Nótese que una de las posibles estrategias para resolver un MDP cuyo modelo es desconocido consiste en utilizar las transiciones adquiridas por el agente para estimar el modelo y, seguidamente, aplicar un algoritmo perteneciente a la programación dinámica basándose en el modelo estimado. A diferencia de otros métodos que explícitamente aprenden la función de transiciones f y la recompensa ρ del MDP, los algoritmos *batch* almacenan esta información de forma implícita en el conjunto de experiencias D. Una ventaja de esta aproximación es que evitan los posibles errores introducidos en la estimación de f y ρ.

3.5.1. *Experience replay*

Experience replay (ER) (Lin, 1992; Adam et al., 2012) es una forma conceptualmente sencilla de aplicar el principio de funcionamiento de los métodos *batch* a otros algoritmos estándar, como por ejemplo *Q-learning*. Aunque esta técnica originalmente fue aplicada a los algoritmos *adaptive heuristic critic* (AHC) (Barto et al., 1989) y *Q-learning*, se puede extender fácilmente a otros casos. ER consiste en almacenar las transiciones adquiridas y presentarlas múltiples veces al algoritmo estándar, como si el aprendizaje fuese *online* y el agente estuviese realizando las mismas transiciones una y otra vez. El orden en el que se presentan las transiciones puede afectar a la velocidad de convergencia del algoritmo. Generalmente se recomienda que se pre-

senten ordenadas de forma temporalmente inversa, es decir, las últimas transiciones muestreadas deben presentarse primero, de esta forma se acelera la propagación de las recompensas. Dependiendo del algoritmo concreto con el que se emplee ER tal vez sea necesario tener en cuenta otras consideraciones sobre como presentar las transiciones (Lin, 1992). En el Algoritmo 3.5 se muestra el pseudo código que implementa ER con *Q-learning*. Además de esta implementación básica, es posible utilizar las transiciones almacenadas de forma más sofisticada para incrementar la estabilidad del algoritmo cuando se combina con determinados aproximadores (Lin, 1993). Por ejemplo, al emplear redes neuronales artificiales es más conveniente ajustar los parámetros con un entrenamiento por lotes.

Algoritmo 3.5 *Experience replay* basado en *Q-learning* con aproximación de funciones.

Require: Conjunto de experiencias $D = \{(s_{l_s}, a_{l_s}, r_{l_s}, s'_{l_s}) | l_s = 1, \ldots, n_s\}$, factor de descuento γ, secuencia de tasas de aprendizaje α_ℓ, funciones base $\phi_1, \ldots, \phi_n : S \times A \to \mathbb{R}$

1: inicializar la estimación de la función Q arbitrariamente, por ejemplo, $\boldsymbol{\theta}_0 \leftarrow 0$
2: **repeat**
3: **for** cada transición l_s en D **do**
4: $\boldsymbol{\theta}_{\ell+1,l_s} = \boldsymbol{\theta}_\ell + \alpha_\ell \left[r_{l_s} + \gamma \,\text{máx}_{a'}\, \boldsymbol{\phi}^\top(s'_{l_s}, a')\boldsymbol{\theta}_\ell - \boldsymbol{\phi}^\top(s_{l_s}, a_{l_s})\boldsymbol{\theta}_\ell \right] \boldsymbol{\phi}(s_{l_s}, a_{l_s})$
5: **end for**
6: $\ell = \ell + 1$
7: **until** cumplir condiciones de convergencia
8: **return** π^*

3.5.2. *Least squares policy iteration*

El algoritmo *least squares policy iteration* (LSPI) (Lagoudakis y Parr, 2003) está motivado en gran parte por un algoritmo previo diseñado para predecir funciones valor y conocido como *least squares temporal difference* (LSTD) (Bradtke y Barto, 1996; Boyan, 2002). Dada una política fija π y un conjunto de funciones base $\phi_1, \ldots, \phi_n : S \times A \to \mathbb{R}$, LSTD es capaz de encontrar el mismo vector de parámetros $\boldsymbol{\theta}$ al que converge el algoritmo TD, pero en lugar de realizar una búsqueda mediante descenso por gradiente estocástico, construye un sistema de ecuaciones lineales que, posteriormente, soluciona por métodos directos (una discusión más detallada del algoritmo LSTD puede encontrarse en la Sección 5.4). Una condición necesaria para que dicho sistema sea lineal es que la función valor esté representada mediante un aproximador lineal. Dado que el objetivo de LSTD es encontrar funciones V, este algoritmo no resulta útil para que el agente actúe de forma óptima cuando se desconoce el modelo del MDP. En LSPI, la idea de encontrar una función valor resolviendo directamente un sistema de ecuaciones es extendida al caso de funciones Q mediante LSTD-Q (Lagoudakis y Parr, 2001). Cuando este método de evaluar políticas se combina con una etapa de mejora de la política, el algoritmo resultante, conocido como LSPI, converge

asintóticamente hacia la política óptima. El Algoritmo 3.6 contiene el pseudo código de LSPI, mientras que la función que realiza la evaluación de la política se muestra en el Algoritmo 3.7.

Algoritmo 3.6 *Least squares policy iteration.*

Require: Conjunto de experiencias D, factor de descuento γ, funciones base $\phi_1, \ldots, \phi_n : S \times A \to \mathbb{R}$

1: inicializar la estimación de la función Q arbitrariamente, por ejemplo, $\boldsymbol{\theta}_0 \leftarrow 0$
2: **repeat**
3: $\quad \pi_{\ell+1} = \arg\max_a \widehat{Q}^{\pi_\ell}(s, a)$
4: $\quad \boldsymbol{\theta}_{\ell+1} = \text{LSTD-Q}(D, \boldsymbol{\theta}, \gamma, \phi_1, \ldots, \phi_n)$
5: $\quad \ell = \ell + 1$
6: **until** cumplir condiciones de convergencia
7: **return** $\boldsymbol{\theta}$

Algoritmo 3.7 LSTD-Q.

Require: Conjunto de experiencias $D = \{(s_{l_s}, a_{l_s}, r_{l_s}, s'_{l_s}) | l_s = 1, \ldots, n_s\}$, factor de descuento γ, política π, funciones base $\phi_1, \ldots, \phi_n : S \times A \to \mathbb{R}$

1: $\mathbf{A}_0 = 0$ // Matriz de tamaño $n \times n$
2: $\mathbf{b}_0 = 0$ // Vector de tamaño n
3: **for** cada transición l_s en D **do**
4: $\quad \mathbf{A}_{l_s+1} = \mathbf{A}_{l_s} + \boldsymbol{\phi}(s_{l_s}, a_{l_s}) \left(\boldsymbol{\phi}(s_{l_s}, a_{l_s}) - \gamma \boldsymbol{\phi}(s'_{l_s}, \pi(s'_{l_s}))\right)^\top$
5: $\quad \mathbf{b}_{l_s+1} = \mathbf{b}_{l_s} + \boldsymbol{\phi}(s_{l_s}, a_{l_s}) r_{l_s}$
6: **end for**
7: $\boldsymbol{\theta}^\pi = \mathbf{A}^{-1}\mathbf{b}$
8: **return** $\boldsymbol{\theta}^\pi$

El algoritmo LSPI, además de hacer un uso eficiente de los datos, presenta la ventaja de eliminar la secuencia de tasas de aprendizaje α_ℓ. Nótese que en el algoritmo LSTD-Q, para resolver el sistema de ecuaciones lineales $\mathbf{b} = \boldsymbol{\theta}^\pi \mathbf{A}$, es posible usar otros métodos aparte de la matriz inversa. De hecho, cuando el número de funciones base es elevado resulta más conveniente calcular la inversa de forma incremental mediante la fórmula de Sherman-Morrison (Sherman y Morrison, 1950). También existen otras variaciones de LSTD que consiguen una carga computacional menor actualizando únicamente una parte reducida de los parámetros en cada iteración (Geramifard et al., 2006b).

3.5.3. *Fitted Q iteration*

Fitted Q iteration (FQI) (Ernst et al., 2005a) es un algoritmo basado en iteración de funciones valor que, a diferencia de los algoritmos previos (ER y LSPI), no requiere que las funciones valor sean aproximadas mediante un conjunto de funciones base

previamente definidas para garantizar la convergencia: permite emplear algunos tipos de aproximadores no lineales y no paramétricos. FQI puede considerarse una versión muestreada y aproximada del algoritmo VI de programación dinámica. Recordemos que VI empleaba la ecuación de optimalidad de Bellman para calcular las nuevas estimaciones de la función Q (ver Sección 2.6.2):

$$Q_{\ell+1}(s,a) = \rho(s,a) + \gamma \max_{a'} Q_\ell(f(s,a),a')$$

en la que es necesario conocer f y ρ. Al aplicar de forma iterativa dicha ecuación la estimación de Q converge hacia la función Q óptima, a partir de la cual resulta sencillo obtener una política óptima (Sutton y Barto, 1998).

En FQI se emplea el mismo principio de funcionamiento pero con dos variaciones que hacen posible aplicarlo en problemas donde el espacio de estados es continuo y el modelo del MDP desconocido:

- En lugar de aplicar la regla de actualización sobre todos los pares (s,a), únicamente se aplica sobre los pares contenidos en el conjunto de experiencias D. Posteriormente, el conjunto de valores actualizados se emplea como conjunto de entrenamiento de un método de regresión. El resultado de esta regresión es una estimación aproximada de la función Q en el espacio de estados completo.
- Los valores $\rho(s,a)$ y $f(s,a)$ se sustituyen por las observaciones de dichos valores contenidos en el conjunto D.

En el Algoritmo 3.8 se muestra el pseudo código que implementa una versión básica del algoritmo FQI. Se puede observar que la regla de actualización se aplica únicamente en el conjunto D (líneas 3-5) y, posteriormente, la función Q estimada en dicho conjunto se generaliza aplicando un algoritmo de regresión (línea 6).

Algoritmo 3.8 *Fitted Q iteration.*

Require: Conjunto de experiencias $D = \{(s_{l_s}, a_{l_s}, r_{l_s}, s'_{l_s}) | l_s = 1, \ldots, n_s\}$, factor de descuento γ, algoritmo de regresión

1: inicializar la estimación de la función Q arbitrariamente, por ejemplo, $\widehat{Q}_0 \leftarrow 0$
2: **repeat**
3: **for** cada transición l_s en D **do**
4: $Q^{\ddagger}_{\ell+1,l_s} = r_{l_s} + \gamma \max_{a'} \widehat{Q}_\ell(s'_{l_s}, a')$
5: **end for**
6: estimar $\widehat{Q}_{\ell+1}$ utilizando el algoritmo de regresión en $\{((s_{l_s}, a_{l_s}), Q^{\ddagger}_{\ell+1,l_s}) | l_s = 1, \ldots, n_s\}$
7: $\ell = \ell + 1$
8: **until** cumplir condiciones de convergencia
9: **return** $\widehat{Q}^* = \widehat{Q}_{l+1}$

La convergencia de FQI está garantizada únicamente para aquellos aproximadores que realizan un mapeado no expansivo entre su espacio de entrada y el de salida.

Dentro de este tipo de aproximadores se encuentran métodos como la regresión k-NN, interpolación lineal y *kernel averaging*, entre otros. Sin embargo varios autores han obtenido resultados positivos aplicando FQI con aproximadores que violan dicha restricción, como, por ejemplo, redes neuronales (Riedmiller, 2005), árboles de decisión (Ernst et al., 2006), codificación en baldosas (Timmer y Riedmiller, 2007) y regresión lineal (Lizotte et al., 2012).

3.5.4. Estudio comparativo utilizando el problema del coche en la montaña

En esta sección se realiza un estudio empírico con el objetivo de comparar las características de los algoritmos ER, LSPI y FQI. El funcionamiento de cada algoritmo puede variar considerablemente en función de varios factores como el tipo de aproximador empleado, la dimensionalidad del problema que se pretende resolver, la forma de la función de recompensa, la distribución de las muestras adquiridas del MDP, etc. Así pues, realizar una comparación exhaustiva donde se estudie la combinación de todos estos factores puede resultar excesivamente complejo. En lugar de ello, se decidió emplear un único aproximador por algoritmo, concretamente se eligió una red RBF con funciones de base fija para los algoritmos ER y LSPI y un comité (o *ensemble*) de árboles de decisión para el algoritmo FQI. El motivo de dicha elección es que estas combinaciones de algoritmo y aproximador se han empleado de forma habitual en la literatura obteniendo buenos resultados. Esto no implica que, en determinados casos, otro tipo de aproximadores sea más adecuado. Por ejemplo cuando se dispone de información adicional sobre el problema, una opción más recomendable es diseñar un aproximador *ad-hoc*. Para comparar los tres algoritmos se empleó el problema del coche en la montaña (Moore y Atkeson, 1995) descrito en la siguiente sección.

Descripción del problema

El problema del coche en la montaña ha sido utilizado por diversos autores para evaluar algoritmos de RL (Singh et al., 1996; Davies, 1997; Sutton y Barto, 1998; Munos y Moore, 2002; Martín H. et al., 2011). Existen dos versiones diferentes en las que varían ligeramente las ecuaciones de la dinámica y la función de recompensa, aunque desde el punto de vista conceptual el problema es el mismo. La versión utilizada aquí, propuesta originalmente por Moore y Atkeson (1995), consiste en un coche representado de forma simplificada como un punto de masa m que debe ser conducido hasta lo alto de una montaña. El agente puede actuar sobre el coche aplicando una fuerza horizontal (ver Figura 3.3). Se asume que el coeficiente de rozamiento entre el coche y la montaña es despreciable, por lo que, además de la fuerza aplicada por el agente, el único factor que modifica la velocidad del coche es la fuerza de gravedad. Cuando el coche se encuentra en ciertas posiciones cercanas a la zona plana, la máxima fuerza que puede aplicar el agente sobre el coche es insuficiente como para alcanzar la posición de meta, por lo que la política obvia de acelerar hacia la meta no consigue el

objetivo deseado. En su lugar, lo que se debe hacer es acelerar hacia el lado opuesto (izquierda) para acumular energía antes de acelerar hacia la meta (derecha).

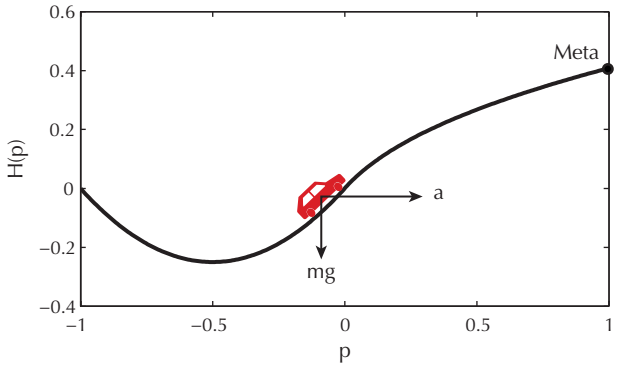


Figura 3.3
Problema del coche en la montaña. El objetivo es conducir el coche hasta la posición de meta en el mínimo tiempo posible.

La ecuación en tiempo continuo que describe la evolución del sistema es (Moore y Atkeson, 1995; Busoniu et al., 2010a):

$$\ddot{p} = \frac{1}{1 + \left(\frac{\mathrm{d}H(p)}{\mathrm{d}p}\right)^2} \left(a - g\frac{\mathrm{d}H(p)}{\mathrm{d}p} - \dot{p}^2 \frac{\mathrm{d}H(p)}{\mathrm{d}p} \frac{\mathrm{d}^2 H(p)}{\mathrm{d}^2 p} \right) \tag{3.18}$$

siendo $p \in [-1, 1]$ m la posición del coche en el eje horizontal, $\dot{p} \in [-3, 3]$ m/s es su velocidad, $a \in [-4, 4]$ N es la fuerza horizontal que aplica el agente sobre el coche, $g = 9.8$ m/s^2 es la aceleración gravitacional, y H describe la forma de la montaña, la cual se define como:

$$H(p) = \begin{cases} p^2 + p & \text{si } p < 0 \\ \frac{p}{\sqrt{1+5p^2}} & \text{si } p \geq 0 \end{cases} \tag{3.19}$$

En la Ecuación (3.18) se ha asumido que la masa del vehículo m tiene valor unitario. Para simular el sistema se ha empleado la instrucción de MATLAB *ode45* con una constante de tiempo $t = 0.1$ s.

El estado del coche en cada instante se define mediante su posición y velocidad, $s = [p, \dot{p}]$, mientras que la acción es la fuerza aplicada a. Por tanto, el espacio de estados es continuo y queda definido por $S = [-1, 1] \times [-3, 3]$. Por otra parte, el espacio de acciones se suele discretizar en sus dos valores extremos, $A = \{-4, 4\}$, de forma que, únicamente, se pueda aplicar una fuerza máxima positiva ($a = 4$) o negativa ($a = -4$). Esta dos acciones son suficientes para obtener una política adecuada (Busoniu et al., 2010a). El factor de descuento empleado en este problema es $\gamma = 0.95$.

El objetivo es aplicar la aceleración necesaria sobre el coche para alcanzar la meta lo más rápido posible sin exceder los límites de posición y velocidad. Si se alcanzan dichos límites sin llegar a la meta, el episodio actual termina considerándose como un fallo. Una de las posibles funciones de recompensa que expresan este objetivo es (Ernst et al., 2005a):

$$r_{k+1} = \rho(s_k, a_k) = \begin{cases} -1 & \text{si } s_{1,k+1} < -1 \text{ o } |s_{2,k+1}| > 3 \\ 1 & \text{si } s_{1,k+1} > 1 \text{ y } |s_{2,k+1}| \leq 3 \\ 0 & \text{en otro caso} \end{cases} \tag{3.20}$$

La Figura 3.4 muestra una porción de la función de recompensa cuando se fija $a = -4$. Como se puede observar la recompensa presenta numerosas zonas donde su valor varía de forma abrupta. Estas variaciones abruptas también estarán presentes en algunas de las funciones valor generadas durante el proceso de aprendizaje, lo cual exige que los aproximadores empleados tengan una elevada capacidad de modelización.

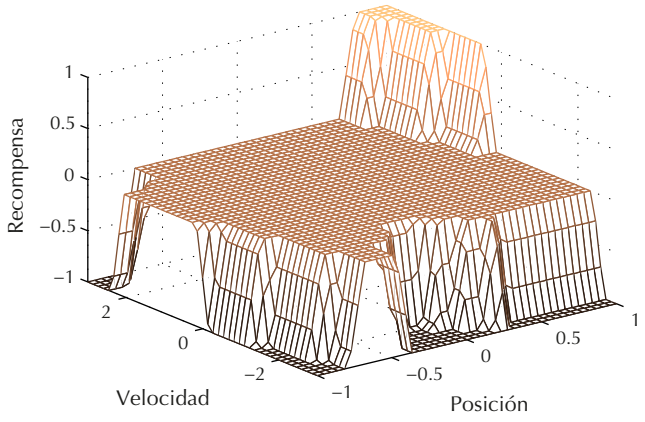


Figura 3.4
Función de recompensa empleada en el problema del coche en la montaña cuando $a = -4$.

Esta dificultad a la hora de aproximar las funciones valor generadas por el problema del coche en la montaña es uno de los motivos por lo que se eligió para realizar la comparación. El otro motivo es que, al tener una estructura relativamente sencilla (espacio de estados bidimensional y espacio de acciones discreto), es posible realizar experimentos de forma intensiva con un coste computacional razonable.

Configuración de los algoritmos

En los algoritmos ER y LSPI se empleó un aproximador formado por un conjunto de RBFs con base fija. Siguiendo el procedimiento descrito en la Sección 3.2.3, se

definió un conjunto de funciones gaussianas distribuidas de forma equidistante en todo el espacio de estados. Las funciones dependen únicamente del estado s, por lo que el conjunto de funciones está duplicado: uno por cada acción permitida. Las funciones gaussianas están definidas por dos parámetros: posición de su centro, c, y ancho, σ. El valor de los centros se puede obtener fácilmente si se conoce el número de gaussianas que formarán el aproximador. Sin embargo, tanto el número de gaussianas como el valor de σ se suele determinar mediante prueba y error. Tras experimentar con varias combinaciones, los mejores resultados se obtuvieron con 81 funciones (9 por cada dimensión) y $\sigma = 0.18$.

El algoritmo FQI se combinó con un comité de árboles de regresión. El comité se obtuvo mediante el algoritmo *tree bagging* (Breiman, 1996), el cual construye M árboles de regresión y combina las salidas de cada árbol para formar un único modelo. *Tree bagging* requiere ajustar dos parámetros: el número de árboles del comité, M, y el número mínimo de muestras por hoja en cada uno de los árboles, n_{min}. Ambos parámetros se seleccionaron siguiendo el procedimiento descrito en Ernst et al. (2005a), es decir, M se fijó igual a 50 y el valor de n_{min} se determinó automáticamente mediante validación cruzada. Para ello se seleccionó aleatoriamente dos tercios del conjunto de entrenamiento y se aplicó *tree bagging*, usando el resto de datos para determinar qué valor de $n_{min} = \{2, 5, 10, 50\}$ minimiza el error cuadrático. De esta forma FQI aprovecha la ventaja de poder adaptar la estructura del aproximador de acuerdo a las características de la función Q que se quiere aproximar. Después de calcular el valor óptimo de n_{min}, el aproximador definitivo se construye usando el conjunto completo de entrenamiento.

Los tres algoritmos (ER, LSPI y FQI) emplearon el mismo conjunto de experiencias D. Para generar dicho conjunto se simularon diversos episodios almacenando por cada transición una tupla con la forma (s, a, r, s'). El estado inicial de cada episodio fue seleccionado mediante dos estrategias diferentes: (i) de forma aleatoria entre todo el espacio de estados, y (ii) fijando un estado inicial común en todos los episodios. En la primera parte de los experimentos se utilizan los datos generados con la estrategia (i), mientras que, en la segunda parte, se repiten los mismos experimentos pero con un conjunto de experiencias generados con la estrategia (ii). Los episodios finalizan únicamente al alcanzar un estado terminal, es decir, al exceder los límites de posición o velocidad permitidos o al llegar a la meta, por lo que cada episodio puede tener una duración diferente. En cada instante temporal k, la acción a_k realizada por el agente se seleccionó con la misma probabilidad entre los dos valores posibles $a = -4$ y $a = 4$. Esta política empleada para muestrear el MDP introduce un cierto nivel de exploración en D. Otro factor que influye en el nivel de exploración de los datos es la estrategia empleada para seleccionar el estado inicial. Cuando se utiliza la estrategia (i), las experiencias están más distribuidas por todo el espacio de estados (mayor nivel de exploración), por el contrario, en la estrategia (ii) la mayoría de experiencias se concentran alrededor del estado inicial.

Se simularon un total de 400 episodios con la estrategia (i), dando lugar a 20442 transiciones o experiencias, de las cuales solo 10 alcanzaron la posición de meta y

recibieron una recompensa positiva $r = 1$. En principio, con un conjunto de experiencias suficientemente grande, cualquiera de los tres algoritmos debe ser capaz de aprender una política adecuada. Para poder apreciar diferencias entre los algoritmos se dividió el conjunto D en varios subconjuntos de distintos tamaños y se repitieron los experimentos con cada uno de los subconjuntos. Concretamente se crearon 8 subconjuntos formados por $50, 100, 150, \ldots, 400$ episodios. El conjunto más pequeño contenía 2260 transiciones y únicamente una de ellas con $r = 1$. El mismo proceso también se llevó a cabo empleando la estrategia (ii) de selección del estado inicial.

La convergencia de los algoritmos se determinó empíricamente analizando la distancia entre estimaciones consecutivas de Q. En la Figura 3.5 se muestra la distancia obtenida por los tres algoritmos en tres conjuntos de datos, el más pequeño ($D = 50$), uno intermedio ($D = 200$), y el más grande ($D = 400$), frente al número de iteraciones ℓ. Las gráficas correspondientes al resto de tamaños de D se han omitido porque el comportamiento es muy similar. Nótese que, en el caso del algoritmo LSPI, no se ha mostrado la distancia entre estimaciones de Q cuando $D = 50$ porque presenta problemas de convergencia y, en algunos puntos, la distancia alcanza valores mucho mayores que 15. En vista de esta figura, las condiciones de convergencia de cada algoritmo se fijaron como $\ell_{max} = 25$ en ER, $\ell_{max} = 20$ en LSPI y FQI.

Resultados

Una métrica comúnmente utilizada para comparar diferentes políticas en el problema del coche en la montaña consiste en contar el número de pasos empleado por cada política para llevar el coche desde la posición de reposo hasta la meta (Martín H. et al., 2011). La posición de reposo es aquella en la que el coche se encuentra en la zona plana de la montaña y con velocidad cero, es decir, $s = [-0.5, 0]$. En la Figura 3.6a se muestra el número de pasos para cada uno de los algoritmos frente al tamaño de D. Los estados iniciales de cada episodio se generaron con la estrategia (i), es decir, de forma aleatoria entre todo el espacio de estados. Dado que el objetivo es alcanzar la meta en el menor tiempo posible, la política será mejor cuanto menor sea el número de pasos. Si después de 50 pasos todavía no se ha alcanzado la meta, el episodio finaliza y se asume que la política es incapaz de lograr el objetivo. De acuerdo con los resultados mostrados en la figura, cuando el número de episodios en D es suficientemente grande (mayor o igual a 200), los tres algoritmos logran resultados similares y cercanos al comportamiento óptimo. Por otra parte, cuando se emplean 50 episodios, ninguno de los algoritmos es capaz de proporcionar una política adecuada. Para el resto de tamaños de D los mejores resultados se obtienen con el algoritmo FQI. En la Figura 3.7 se muestra un ejemplo de la trayectoria seguida por el coche desde la posición de reposo cuando se controla con una política cercana a la óptima. Se puede apreciar que, antes de acercarse hacia la meta, el coche acelera en dirección contraria con el objetivo de acumular la energía necesaria para poder superar la pendiente hasta la meta.

Una limitación de evaluar las políticas contando el número de pasos requeridos

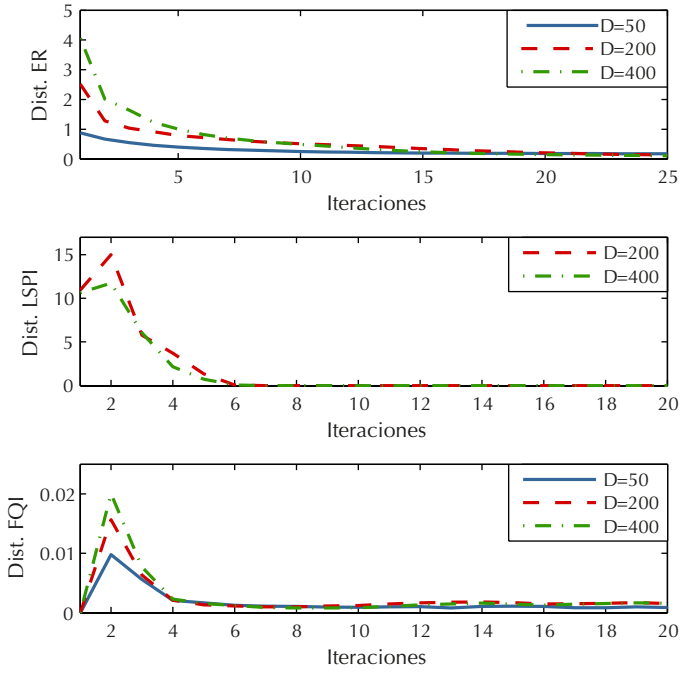
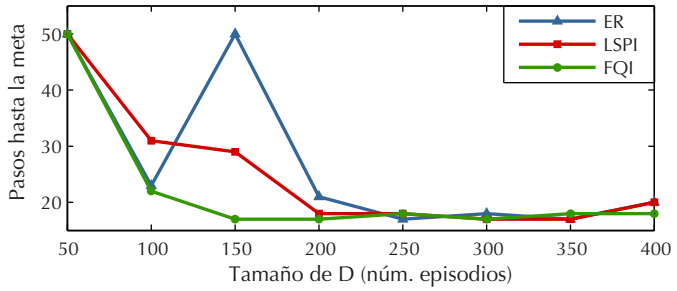


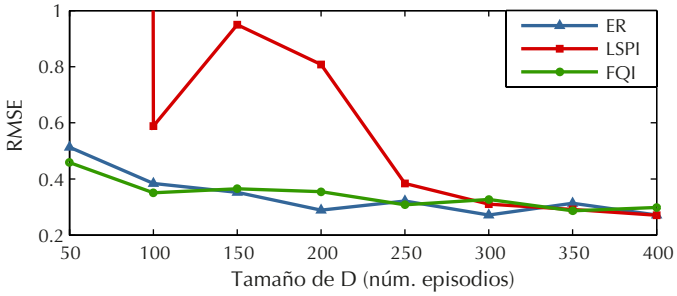
Figura 3.5
Convergencia de los algoritmos ER, LSPI y FQI, respectivamente, medida como la distancia entre estimaciones sucesivas de la función Q frente al número de iteraciones. En cada figura se muestra la convergencia cuando el conjunto de experiencias D está formado por 50, 200 y 400 episodios.

para llevar el coche desde $s_0 = [-0.5, 0]$ hasta la meta es que solo se tiene en cuenta una zona limitada del espacio de estados. Las zonas alejadas de los estados encontrados en la trayectoria desde s_0 hasta la meta son completamente ignoradas. Otra opción que puede resultar más adecuada consiste en calcular la función valor V de cada política y compararla con la función V óptima, V^*. Habitualmente no es posible obtener V^* ya que ese es, precisamente, el objetivo del problema. Sin embargo, debido a la estructura del problema del coche en la montaña es relativamente sencillo obtener $V^*(s)$ en un conjunto discreto de estados S_{test} mediante un procedimiento de búsqueda exhaustiva. Para ello, dado un estado $s \in S_{test}$, se puede obtener $V^*(s)$ determinando mediante sucesivas pruebas el valor de L para el que se cumple alguna de las siguientes condiciones (Ernst et al., 2005a):

1. Al menos una secuencia de acciones de longitud L produce una recompensa igual a 1 cuando el estado inicial $s_0 = s$.



(a)



(b)

FIGURA 3.6
Comparación de los resultados obtenidos por los algoritmos ER, LSPI y FQI en el problema del coche en la montaña cuando el conjunto de experiencias D se ha generado empleando la estrategia (i). En (a) se evalúan las políticas mediante el número de pasos requeridos para conducir el coche desde $s_0 = [-0.5, 0]$ hasta la meta. En (b) se muestra el RMSE entre la función valor óptima $V^*(s)$ y las estimaciones obtenidas con cada algoritmo.

2. Todas las secuencias de acciones de tamaño L producen una recompensa igual a -1 cuando el estado inicial $s_0 = s$.

Si se denota con L_{min} al valor mínimo de L, el valor óptimo $V^*(s)$ es igual a $\gamma^{L_{min}-1}$ si se cumple la primera condición y $-\gamma^{L_{min}-1}$ en otro caso. Este proceso se llevó a cabo para un conjunto S_{test} de 400 estados distribuidos uniformemente en el espacio de estados, por lo que S_{test} proporciona información de prácticamente la función $V^*(s)$ completa.

Una vez obtenida $V^*(s)$ mediante el procedimiento descrito, se calculó la función valor óptima estimada por los algoritmos ER, LSPI y FQI en el conjunto S_{test}. Posteriormente se usó el valor RMSE (*root mean square error*) para calcular la desviación entre V^* y la estimaciones. La Figura 3.6b muestra el valor de RMSE para cada algoritmo empleando diferentes tamaños de D. De acuerdo con este criterio, el

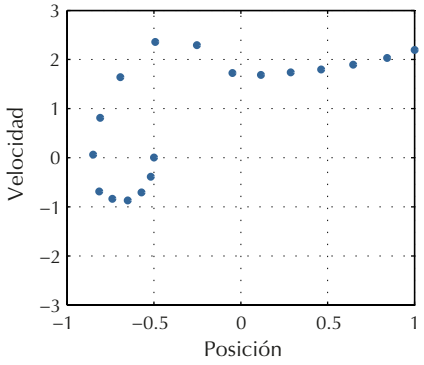


Figura 3.7
Ejemplo de una trayectoria desde $s_0 = [-0.5, 0]$ hasta la meta siguiendo una política aproximadamente óptima. La política de control se obtuvo con el algoritmo ER y un conjunto de experiencias formado por 400 episodios.

comportamiento de los tres algoritmos es bastante similar cuando D contiene 250 (o más) episodios, lo cual coincide con el criterio anterior. Sin embargo, para tamaños menores de D, los dos criterios sugieren conclusiones diferentes. Por ejemplo, cuando se emplean 150 episodios, la estimación de la función V^* obtenida por el algoritmo LSPI es peor que la obtenida por el algoritmo ER y, a pesar de ello, la política obtenida con LSPI consigue llevar el coche hasta la meta y la obtenida con ER no. Aunque este resultado pueda parecer sorprendente existen al menos dos factores que pueden explicarlo:

- El primer criterio está evaluando únicamente una zona limitada del espacio de estados, al contrario que ocurre con el segundo criterio, que tiene en cuenta todo el espacio. Por tanto, es posible que si se evalúan las políticas obtenidas por ER y LSPI desde otros estados iniciales diferentes a $s = [-0.5, 0]$, el resultado obtenido sea favorable al algoritmo ER.
- En la mayoría de métodos basados en funciones valor se asume, de manera implícita, que cuanto más se aproxime la función valor estimada a la función valor óptima, más se aproximará la política derivada de dicha función a la política óptima. Sin embargo, a pesar de que esta asunción es razonable, no siempre es cierta (Bertsekas, 2007). Un algoritmo que obtenga un RMSE menor no implica necesariamente que vaya a ser más efectivo para conducir el coche hasta la meta.

Un ejemplo de este último punto se puede apreciar en la Figura 3.8, donde se ha representado gráficamente V^* y las estimaciones obtenidas con cada algoritmo cuando D está formado por 400 episodios. Se puede observar que, en términos generales y

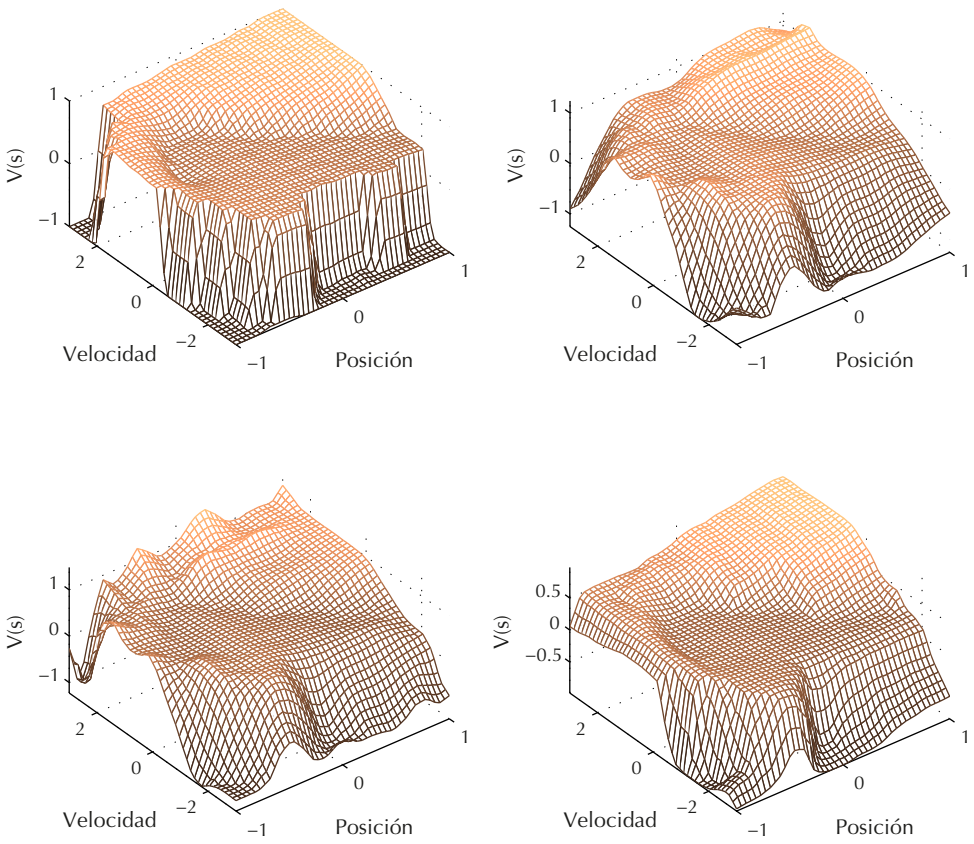


Figura 3.8
Estimación de la función V óptima para el problema del coche en la montaña obtenida mediante los algoritmos (a) búsqueda exhaustiva, (b) ER, (c) LSPI y (d) FQI. La función (a) se puede considerar como una estimación exacta de V^*. Para los algoritmos ER, LSPI y FQI se empleó un conjunto de experiencias D formado por 400 episodios.

de manera visual, la función valor con una forma aparentemente más parecida a V^* (Figura 3.8a) es la obtenida con FQI (Figura 3.8d), por lo que, posiblemente, FQI sea el algoritmo que proporciona una política más próxima a la óptima. Sin embargo, atendiendo únicamente al RMSE, los resultados sugieren que ER (Figura 3.8b) y LSPI (Figura 3.8c) obtienen aproximaciones de la función valor ligeramente mejores. Si se presta atención al eje z se observa que la única función que no alcanza el valor 1 es la estimada con el algoritmo FQI por lo que es lógico que, al calcular el valor RMSE, obtenga el peor resultado. Este ejemplo pone de manifiesto que medir la distancia entre la función óptima y la estimación no siempre es una medida adecuada para determinar la calidad de la aproximación en el caso de las funciones valor.

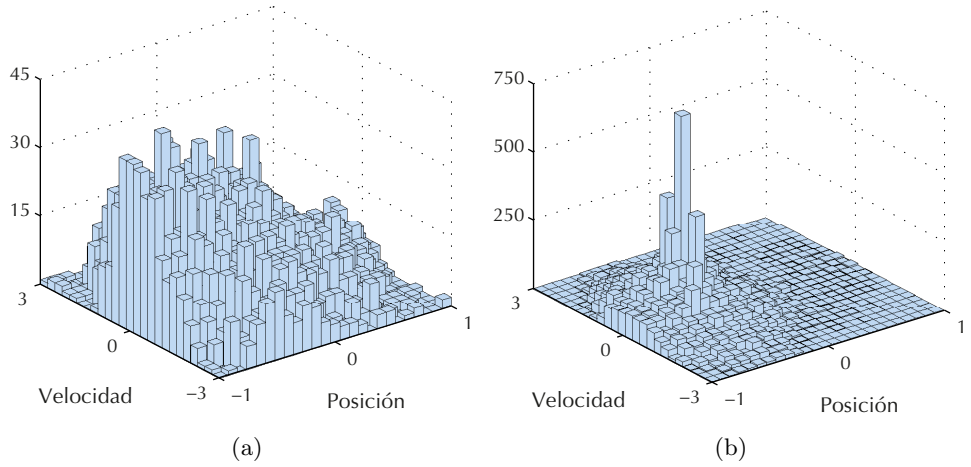
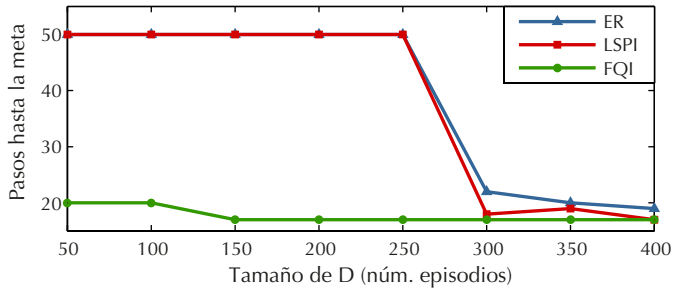


Figura 3.9
Distribución de los estados que forman el conjunto de transiciones D. (a) corresponde al conjunto de transiciones generado con la estrategia (i), mientras que (b) al conjunto generado con la estrategia (ii).

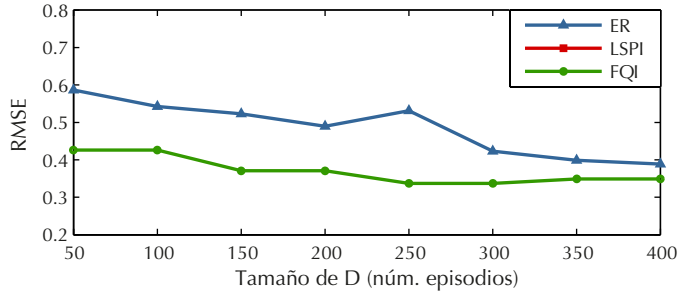
En los resultados mostrados hasta ahora el conjunto de experiencias D se ha generado utilizando la estrategia (i). Como ya se ha comentado previamente, esta estrategia facilita el aprendizaje porque proporciona información de prácticamente todo el espacio de estados, es decir, contiene un elevado nivel de exploración. Sin embargo es probable que en un problema real no sea posible iniciar las trayectorias aleatoriamente en cualquier estado sino que, de forma natural, todas las trayectorias comiencen desde un estado (o un conjunto de estados) de reposo. A continuación se va a repetir la comparación de los tres algoritmos pero empleando un conjunto de experiencias generado con la estrategia (ii), donde todas las trayectorias tienen el mismo estado inicial. En la Figura 3.9 se han representado los histogramas de los estados correspondientes al conjunto de experiencias generado con la estrategia (i) (Figura 3.9a) y (ii) (Figura 3.9b). En dichas figuras se puede observar que, cuando el estado inicial de las trayectorias es fijo, la mayoría de estados en D se concentra

en una pequeña zona alrededor del estado inicial. Esta distribución de los datos tan poco uniforme supone una dificultad añadida para la mayoría de aproximadores.

Igual que en el caso anterior, se han empleado dos métricas para evaluar los algoritmos. En la Figura 3.10a se muestra el número de pasos que requieren las políticas para alcanzar la posición de meta desde el estado inicial $s_0 = [-0.5, 0]$. Cabe recordar que, cuando el número de pasos es igual o mayor que 50, el episodio finaliza y se considera que dicha política es incapaz de lograr el objetivo. La figura indica que el funcionamiento de ER y LSPI ha empeorado considerablemente comparado con el experimento anterior (Figura 3.6a). Ahora ambos algoritmos requieren un mínimo de 300 episodios para aprender a conducir el coche hasta la meta. Por el contrario, FQI ha mejorado: con tan solo 50 episodios es capaz de aprender una política cercana a la óptima. A partir de 150 episodios ofrece un comportamiento óptimo (17 pasos) desde el estado $s_0 = [-0.5, 0]$ hasta la meta.



(a)



(b)

Figura 3.10
Comparación de los resultados obtenidos por los algoritmos ER, LSPI y FQI en el problema del coche en la montaña cuando el conjunto de experiencias D se ha generado empleando la estrategia (ii). En (a) se evalúan las políticas mediante el número de pasos requeridos para conducir el coche desde $s_0 = [-0.5, 0]$ hasta la meta. En (b) se muestra el RMSE entre la función valor óptima $V^*(s)$ y las estimaciones obtenidas con cada algoritmo.

El motivo de que algunos algoritmos empeoren y otros mejoren está relacionado con la capacidad que tienen los aproximadores de trabajar con datos distribuidos de forma no uniforme. En principio, si todos los episodios que forman el conjunto D comienzan en el estado $s_0 = [-0.5, 0]$, es lógico que los algoritmos tengan una mayor facilidad para aprender una política óptima desde dicho estado hasta la meta, por lo que todos los algoritmos deberían de mejorar respecto el criterio de evaluación basado en el número de pasos. Por otra parte, dado que las experiencias están concentradas en una región específica, todos los algoritmos deberían presentar más dificultades para estimar la función valor óptima. En la Figura 3.10b se muestra el valor RMSE para los algoritmos ER y FQI (LSPI no aparece porque el error es mucho mayor que 0.8 para todos los tamaños de D). Si se comparan estos valores con los del experimento anterior (Figura 3.6b) se observa que, aunque el RMSE ha aumentado en todos los casos, el incremento en los algoritmos ER y LSPI es notablemente mayor que en el algoritmo FQI. Es decir, aunque D contiene más información sobre la trayectoria desde $s_0 = [-0.5, 0]$ hasta la meta, el hecho de que los datos no estén distribuidos en todo el espacio de estados puede afectar negativamente a ciertos aproximadores, dando lugar a políticas de peor calidad.

3.6. Conclusiones

En este capítulo se ha introducido un conjunto de métodos de RL cuyo objetivo principal es resolver problemas con un número N de estados muy elevado. En tales problemas, las operaciones ordinarias de álgebra como un producto escalar requieren tiempos de computación prohibitivos, incluso el hecho de almacenar un vector de tamaño N puede ser imposible debido a las restricciones de memoria. La estrategia básica para afrontar estos problemas consiste en el uso de aproximadores, de tal forma que un vector de longitud N sea representado mediante un vector de parámetros de longitud M, siendo $M \ll N$. Los aproximadores proporcionan una representación mucho más compacta y manejable, sin embargo, también introducen un error de aproximación. El impacto de dicho error en los algoritmos de RL puede dar lugar a problemas de convergencia.

En general los aproximadores cuya salida se puede calcular como una combinación lineal del vector de parámetros son más adecuados desde el punto de vista teórico. En la práctica, los aproximadores lineales presentan dificultades cuando se quieren aplicar en espacios de estados de elevada dimensionalidad debido al efecto conocido como "maldición de la dimensionalidad". Además de la escalabilidad a espacios de alta dimensionalidad, la otra gran dificultad que aparece un muchas aplicaciones reales es el elevado número de experiencias necesarias para aprender políticas óptimas. Una clase de métodos enfocada a resolver esta limitación son los algoritmos de tipo *batch* RL.

A lo largo del capítulo se ha descrito una versión aproximada de los algoritmos SARSA y *Q-learning* basada en descenso por gradiente estocástico y aproximación

lineal. También se han presentado los tres algoritmos de tipo *batch* RL más representativos: ER, LSPI y FQI; incluyendo un estudio experimental en el que se compara su funcionamiento cuando la cantidad de experiencias disponible es pequeña y están distribuidas de forma no uniforme. Los resultados obtenidos indican que el algoritmo FQI basado en árboles de decisión proporciona los resultados más robustos.

Capítulo 4

Iteración de funciones valor ajustadas para problemas de aprendizaje *online*

4.1. Introducción

En el Capítulo 3 se estudió una clase de algoritmos que se caracterizan por procesar los datos en modo *batch*. El objetivo de estos algoritmos es encontrar políticas óptimas cuando la única información disponible sobre el entorno viene dada como un conjunto de datos con interacciones entre el agente y el entorno. Las interacciones, típicamente formadas por tuplas del estilo (s, a, s', r), son almacenadas previamente, por lo que el aprendizaje se realiza de forma *offline*. El desarrollo de este campo, denominado *batch* RL, ha permitido aplicar las técnicas de RL en ciertos ámbitos donde hace apenas una década era inviable, como por ejemplo en el ámbito médico (Ernst et al., 2006; Pineau et al., 2007; Guez et al., 2008). Uno de los principales atractivos de los algoritmos de tipo *batch* es su eficiencia respecto al uso de los datos. Esta clase de algoritmos permite extraer más conocimiento de cada interacción agente-entorno que los métodos clásicos debido, en gran parte, a que las interacciones se procesan varias veces en lugar de una única vez.

La eficiencia en el procesamiento de los datos también es una característica de vital importancia en los problemas de RL estándar, donde el aprendizaje de la política se realiza al mismo tiempo que el agente interactúa con el entorno, es decir, de forma *online*. De hecho, el número de datos requeridos por muchos algoritmos de RL para aprender políticas óptimas sigue siendo hoy en día una de las principales limitaciones prácticas en problemas con un cierto grado de complejidad (Deisenroth et al., 2013a). Durante las décadas de los 80 y 90, cuando se desarrollaron algunos de los algoritmos

más extendidos (por ejemplo, *Q-learning* y SARSA), la mayoría de aplicaciones del aprendizaje por refuerzo estaban relacionadas con la robótica. En dichas aplicaciones el objetivo era desarrollar robots capaces de aprender una determinada tarea en tiempo real. Este enfoque, junto con las limitaciones impuestas por los procesadores de aquella época, hizo que en los algoritmos primara la eficiencia computacional frente a otros aspectos como el uso eficiente de los datos (Busoniu, 2008).

El desarrollo tecnológico actual ha dado lugar a procesadores con una capacidad notablemente mayor que los de hace unas décadas. Por otra parte, el campo de aplicación de las técnicas del RL se ha extendido a otros ámbitos en los cuales las restricciones de un algoritmo que debe funcionar en tiempo real son mucho menores que en robótica. Los tiempos de muestreo en problemas relacionados con la robótica suelen ser inferiores a una décima de segundo. En cambio, otras aplicaciones de RL que implican controlar, por ejemplo, una planta de tratamiento de aguas residuales (Hernández-del Olmo et al., 2012; Hernández-del Olmo et al., 2012), el sistema de calefacción, ventilación y aire acondicionado de un edificio (Liu y Henze, 2006a,b), un sistema de *trading* (Peters et al., 2013) o una central hidroeléctrica (Castelletti et al., 2010), requieren que se actúe en intervalos de tiempo mucho mayores, del orden de minutos. Por tanto, es posible la aplicación de algoritmos *online* y en tiempo real a pesar de que su carga computacional sea mucho mayor que la de *Q-learning*, SARSA u otros algoritmos similares.

En este capítulo se propone un algoritmo de aprendizaje *online* caracterizado por hacer un uso eficiente de los datos. El algoritmo está basado en un esquema de iteración de funciones valor: partiendo de una estimación inicial arbitraria de la función valor, ésta se va mejorando de forma iterativa hasta alcanzar la función valor (aproximadamente) óptima. A partir de dicha función se obtiene una política óptima simplemente eligiendo la acción con mayor valor en cada estado. En el algoritmo propuesto la información obtenida en cada interacción agente-entorno es almacenada y utilizada diferentes veces para actualizar la estimación de la función valor. Además, se utilizan funciones valor "ajustadas" que hacen posible emplear un mayor rango de aproximadores sin comprometer la estabilidad del algoritmo. La idea fundamental consiste en combinar las ventajas del aprendizaje *batch* en un método *online*. En la Figura 4.1 se muestra un esquema donde se clasifican los algoritmos de RL en función de dos posibles criterios: el tipo de interacción entre el agente y el entorno, y la forma de procesar los datos. El primer criterio da lugar a algoritmos *online* y *offline*, y el segundo criterio a algoritmos inmediatos (o incrementales) y *batch* (Wiering y van Otterlo, 2012). El algoritmo desarrollado en este capítulo interactúa con el entorno mientras calcula la política óptima, por lo que el agente aprende de forma *online* y, al mismo tiempo, almacena los datos recogidos, por lo que puede considerarse de tipo *batch* o, más concretamente, *growing batch*.

El resto del capítulo comienza con una introducción al concepto de funciones valor ajustadas en la Sección 4.2. Posteriormente, el algoritmo propuesto se describe en la Sección 4.3, donde también se analizan las ventajas de reutilizar los datos y se discute la relación del algoritmo propuesto con otros algoritmos existentes. La Sección 4.4

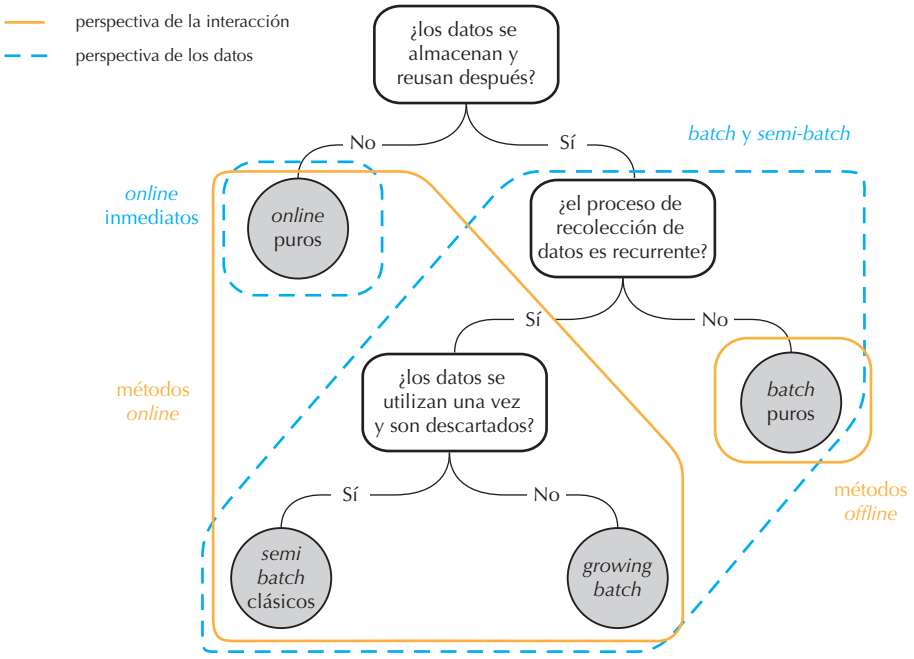


Figura 4.1
Clasificación de los algoritmos de aprendizaje por refuerzo desde dos posibles perspectivas: modo de interacción entre agente y entorno (línea naranja continua) y forma en la que se procesan los datos (línea azul discontinua).

consta de un estudio experimental en el que se ha evaluado el funcionamiento del algoritmo propuesto en tres problemas diferentes. Además, contiene un apartado que investiga la influencia de modificar algunos de los parámetros de los que depende el algoritmo. La Sección 4.5 cierra el capítulo con las conclusiones y algunas posibles líneas de investigación futura.

4.2. Funciones valor ajustadas

La mayoría de algoritmos de RL basados en funciones valor usan actualizaciones asíncronas. Dada una transición desde el estado s hasta s' tras realizar la acción a, una actualización asíncrona es aquella que calcula la nueva función valor inmediatamente después de la transición y únicamente para el estado s (o el par (s, a) si se trata de funciones Q). En el caso de que el número de estados sea discreto y la función $V(s)$ esté representada mediante una tabla, el hecho de actualizar $V(s)$ de forma asíncrona significa que el valor almacenado en la celda correspondiente a s es sobreescrito cada

vez que el agente se encuentra en dicho estado (Wiering y van Otterlo, 2012). Una ventaja de las actualizaciones asíncronas es que, tras visitar un estado determinado, su valor actualizado está disponible de forma instantánea para realizar los cálculos correspondientes a los valores de los estados siguientes. Esta misma idea también se puede extender al caso más general de representar $V(s)$ mediante un aproximador de funciones. Por ejemplo, supongamos que el agente realiza una transición desde s hasta s', generando la recompensa correspondiente r. Entonces, una posible regla de actualización asíncrona consistiría en utilizar la expresión (Wiering y van Otterlo, 2012)

$$v_s = r + \gamma F(s') \tag{4.1}$$

para recalcular la estimación de la función valor como la suma de la recompensa obtenida más el valor del estado siguiente, donde F denota un aproximador genérico que almacena la función valor[1]. A continuación la nueva estimación v_s se actualizaría inmediatamente en el aproximador moviendo $F(s)$ ligeramente hacia v_s. Para ello una opción habitual es emplear la regla:

$$F(s) \leftarrow F(s) + \alpha(v_s - F(s)) \tag{4.2}$$

donde α es la tasa de aprendizaje.

Las actualizaciones asíncronas son especialmente útiles cuando la función valor se representa de forma exacta debido a que proporcionan mayor velocidad de convergencia sin modificar las propiedades teóricas de los algoritmos (Sutton y Barto, 1998). Por el contrario, cuando la función valor se representa de forma aproximada únicamente se han obtenido garantías teóricas de convergencia para ciertas combinaciones de aproximadores y reglas de actualización. Además, dichas garantías a menudo se basan en imponer restricciones sobre la estructura del MDP y la recompensa (Schoknecht y Merke, 2003). Diversos autores han demostrado que, en general, combinar funciones valor aproximadas con actualizaciones asíncronas suele dar lugar a algoritmos que tienden a comportarse de forma inestable o incluso diverger con total seguridad (Baird, 1995; Gordon, 1996).

Los problemas de estabilidad de los algoritmos asíncronos surgen como consecuencia de la interdependencia entre el error introducido por el aproximador y la diferencia entre la verdadera función valor $V(s)$ y las estimaciones v_s. Por un lado, la regla de actualización (4.1) (u otras similares) intenta minimizar la distancia entre v_s y la función valor pero, por otro lado, el hecho de almacenar v_s de forma aproximada puede introducir nuevas desviaciones. El error introducido por el aproximador también afecta a las siguientes actualizaciones, por lo que se puede acumular y aumentar hasta provocar un comportamiento inestable. El problema se ve acentuado cuando el aproximador es de tipo global (como es el caso, por ejemplo, de un perceptrón multicapa (Werbos, 1974; Rumelhart et al., 1986)), ya que el error introducido al actualizar el valor de un único estado puede afectar a la estimación de toda la función $V(s)$ (Wiering y van Otterlo, 2012).

[1] Nótese que por cuestiones de sencillez se ha omitido la dependencia del aproximador con el vector de parámetros.

Ante esta situación, Gordon (Gordon, 1995a, 1999) propuso una solución sencilla que consiste en modificar el orden en el que se calculan las nuevas estimaciones de la función valor y se actualizan en el aproximador. El algoritmo de Gordon, llamado *fitted value iteration* (FVI), requiere conocer el modelo completo del MDP debido a que fue propuesto en el contexto de la programación dinámica. También asume que se dispone de un subconjunto discreto de estados $S_c \in S$ suficientemente pequeño como para examinarlo repetidamente y, al mismo tiempo, suficientemente grande como para ser representativo de todo el espacio de estados. En el Algoritmo 4.1 se muestra el pseudo código del algoritmo FVI. La idea principal consiste en calcular primero las nuevas estimaciones v_s para todos los estados de S_c. Esta parte se corresponde con las líneas 5-7 del pseudo código, donde se observa que es necesario conocer tanto la función de transición f como la recompensa ρ del MDP. Nótese que, en el Algoritmo 4.1, v_s se ha denotado como o. Con estas estimaciones se construye un conjunto de entrenamiento que se emplea para "ajustar" la función valor con un método de aprendizaje supervisado (línea 9). Al plantear la actualización de $V(s)$ como un problema de aprendizaje supervisado tradicional las actualizaciones realizadas son de tipo síncrono, ya que $V(s)$ se calcula para todo el espacio de estados o, más concretamente, para un conjunto de estados que se supone representativo de todo el espacio. Desde el punto de vista teórico, el algorimto FVI converge hacia una política óptima cuando se combina con cierto tipo de aproximadores. Si se considera al aproximador como un mapeado entre su espacio de entrada y de salida, entonces el algoritmo FVI requiere que dicho mapeado sea no expansivo (Gordon, 1995a). Esta clase de aproximadores incluye métodos como *k-nearest neighbors*, interpolación lineal y multilineal, *locally weighted averaging* o *kernel averaging* (Ernst et al., 2005a). A pesar de esta restricción, la convergencia del algorimto FVI sigue siendo de carácter más general que la existente para los algoritmos basados en actualizaciones asíncronas.

Como ya se ha comentado previamente, la utilidad práctica del algoritmo FVI es limitada debido a que forma parte de los métodos de programación dinámica y requiere disponer de un modelo completo del entorno. Sin embargo, su principio de funcionamiento también se ha aplicado en el aprendizaje por refuerzo. De hecho, constituye la base de prácticamente todos los algoritmos modernos de tipo *batch* (Wiering y van Otterlo, 2012). Cuando se utilizan funciones valor ajustadas en algoritmos de RL aparecen dos dificultades fundamentales que no están presentes en el algoritmo original FVI:

- No es posible disponer de un subconjunto de estados S_c prefijados *a priori*.
- No se puede calcular las nuevas estimaciones v_s con el operador de Bellman debido a que no se conoce la función de transiciones del MDP.

La solución consiste en utilizar los estados por los que pasa el agente y una versión "muestreada" del operador de Bellman. El procedimiento habitual para implementar esta solución consta de dos etapas. En la primera el agente interactúa con el entorno para recoger un conjunto de transiciones (s, a, r, s') suficientemente grande como para

Algoritmo 4.1 *Fitted value iteration.*

Require: Función de transición f, función de recompensa ρ, factor de descuento γ, subconjunto de estados $S_c = \{(s_c^t), t = 1, \ldots, \#S_c\}$, método de aprendizaje supervisado AS

1: Inicializar la estimación de la función V arbitrariamente, por ejemplo $\widehat{V}_0 \leftarrow 0 \ \forall S$
2: j = 0
3: **repeat**
4: j = j + 1
5: **for** $t = 1, 2, \ldots, \#S_c$ **do**
6: $i^t = s_c^t$
7: $o^t = \text{máx}_a \, E\{\rho(s_c^t, a) + \gamma \widehat{V}_{j-1}(f(s_c^t, a))\}$
8: **end for**
9: Estimar $\widehat{V}_j$ entrenando el método de aprendizaje supervisado AS usando i^t como entradas y o^t como salidas deseadas
10: **until** alcanzar convergencia
11: **return** estimación de la función valor óptima V^*

ser representativo de todo el espacio de estados. En la segunda etapa las transiciones son procesadas de forma *offline* con el objetivo de obtener una política óptima. Algunos ejemplos de algoritmos basados en este principio son KADP (*kernel-based approximate dynamic programming*) (Ormoneit y Sen, 2002), FQI (*fitted Q iteration*) (Ernst et al., 2005a), FNAC (*fitted natural actor-critic*) (Melo y Lopes, 2008) o NFTD (*neural-fitted temporal difference learning*) (Dries y Wiering, 2012). Las garantías teóricas de convergencia en estos métodos suele depender en gran medida de que el mapeado realizado por el aproximador sea no expansivo, igual que ocurría en el algoritmo original FVI. Aún así, existen diversos casos prácticos en los que se han obtenido políticas cercanas a la óptima a pesar de emplear aproximadores que no cumplen esta condición como, por ejemplo, árboles de decisión (Ernst et al., 2005b), *ensembles* de árboles (Ernst et al., 2006), redes neuronales (Riedmiller, 2005; Kalyanakrishnan y Stone, 2007) o *tile coding* (Timmer y Riedmiller, 2007).

4.3. Iteración de funciones valor ajustadas para aprendizaje *online*

La mayoría de algoritmos basados en el concepto de funciones valor ajustadas trabajan en modo *offline* (Ormoneit y Sen, 2002; Ernst et al., 2005a; Melo y Lopes, 2008; Dries y Wiering, 2012), es decir, durante el proceso de aprendizaje no se interactúa con el entorno, sino que los datos son almacenados y posteriormente procesados. En esta sección se propone un algoritmo basado en la **i**teración de funciones **v**alor **a**justadas para aprendizaje ***o**nline*, denominado con el acrónimo IVAO. Este algoritmo extiende los beneficios de las funciones valor ajustadas al aprendizaje *online*. Los

algoritmos *online* deben poseer al menos dos características que no están presentes en los algoritmos *offline*:

1. Deben proporcionar políticas de actuación desde el comienzo del proceso de aprendizaje en lugar de hacerlo únicamente al final, como en el caso de los algoritmos *offline*.
2. Tienen que incorporar algún sistema que les permita realizar acciones exploratorias durante la interacción agente-entorno, aunque ello suponga un decremento temporal en la calidad de la política seguida.

Un algoritmo que necesita interactuar con el entorno con una política de baja calidad durante un periodo largo de tiempo no resulta útil en la mayoría de aplicaciones reales, incluso aún cuando se tiene la certeza de que el algoritmo puede encontrar una política óptima al final de la etapa de aprendizaje. Por este motivo, la eficiencia respecto al uso de los datos es vital en los algoritmos *online* (Adam et al., 2012). Respecto a la exploración, dado que los algoritmos *offline* no tienen la posibilidad de aplicar acciones sobre el entorno, no es necesario que incluyan ninguna estrategia de exploración. Nótese, sin embargo, que los datos procesados de forma *offline* sí que deben de contener acciones exploratorias a pesar de que el algoritmo en sí no debe encargarse de ello. En cambio, cuando la política se aprende al mismo tiempo que se interactúa con el sistema, la exploración es necesaria para evitar que el proceso iterativo quede atascado en un mínimo local. Si el algoritmo solamente elige las acciones indicadas por la política en cada estado, es probable que haya zonas del espacio de estados que nunca sean visitadas. Esta situación puede dar lugar a estimaciones erróneas de la función Q y, en consecuencia, a políticas subóptimas. El segundo requisito va en contra del primero, y la combinación de ambos es lo que se conoce como dilema *exploración-explotación* (ver Sección 2.7). Así pues, en la práctica es necesario llegar a una solución de compromiso que satisfaga ambas condiciones.

En el algoritmo propuesto en este capítulo se ha optado por incluir el esquema clásico de exploración ϵ-*greedy* (Sutton y Barto, 1998): en cada paso, o instante temporal k, se elige una acción uniformemente aleatoria entre el conjunto A de todas las acciones posibles con probabilidad $\epsilon_k \in [0, 1]$, y la acción indicada por la política con probabilidad $1 - \epsilon_k$, o lo que es lo mismo:

$$a_k = \begin{cases} a \in \arg\max_a Q_0(s_0, a) & \text{con probabilidad } 1 - \epsilon_k \\ \text{acción aleatoria en } A & \text{con probabilidad } \epsilon_k \end{cases} \tag{4.3}$$

Normalmente el valor de ϵ_k se suele disminuir conforme aumenta k, de forma que al comienzo del aprendizaje se realizan más acciones exploratorias y, a medida que la política aprendida mejora, se prefieren las acciones que explotan el conocimiento ya adquirido. Existen otras técnicas de exploración más sofisticadas que obtienen mejores resultados en determinadas situaciones (Li et al., 2009), sin embargo, la exploración ϵ-*greedy* proporciona un buen compromiso entre sencillez y eficacia. Evidentemente, el algoritmo propuesto sigue siendo válido al combinarse con otras técnicas de exploración.

En el Algoritmo 4.2 se muestra el pseudo código del algoritmo IVAO con exploración ϵ-*greedy*. La estructura de IVAO es la típica de los algoritmos basados en un esquema de iteración de funciones valor. El objetivo es estimar una función valor óptima a partir de la cual resulta inmediato obtener una política óptima. Concretamente, se pretende estimar la función Q óptima, ya que resulta más sencillo extraer una política de este tipo de funciones que de las funciones V. EI algoritmo IVAO comienza con una estimación arbitraria de Q (línea 1) que se actualiza iterativamente (línea 15) conforme el agente interactúa con el entorno. En lugar de realizar una actualización incremental en cada paso k, el algoritmo almacena un conjunto de transiciones D y realiza varias actualizaciones sobre todos los estados almacenados en D cada cierto número de transiciones. El número de transiciones, denotado como K, es un parámetro que debe ser elegido por el usuario. El otro parámetro que introduce IVAO es el número máximo de transiciones almacenadas, denotado como N. Es necesario fijar un número máximo de transiciones porque en caso contrario los requisitos de memoria aumentarían indefinidamente con el tiempo. Incluso suponiendo que se dispone de memoria ilimitada, la carga computacional que se requiere para procesar D es directamente proporcional a su tamaño. Una de las posibles formas de implementar D consiste simplemente en almacenar las transiciones en una memoria de tipo FIFO (*first-in first-out*), en la cual únicamente se guardan las transiciones más recientes. Esta solución se ha utilizado en muchos otros algoritmos de aprendizaje *online* como, por ejemplo, en el contexto del control adaptativo.

La estimación de Q se actualiza dentro de la función LEARNQ(), cuyo pseudo código se muestra en el Algoritmo 4.3. Como se puede observar, las transiciones almacenadas en D se emplean para generar un conjunto de entrenamiento donde las entradas son los pares (s, a) de cada transición y la señal deseada se calcula aplicando una versión muestreada del operador de optimalidad de Bellman (línea 5). Posteriormente, se ajustan los parámetros del método de aprendizaje supervisado para estimar Q (línea 6). Este proceso se repite de forma iterativa hasta alcanzar la convergencia. Nótese que cada vez que se actualiza Q se parte desde la estimación previa (línea 2) que se le pasa como argumento a la función LEARNQ(). En la mayoría de los casos la nueva estimación y la previa son similares por lo que el proceso iterativo suele converger rápidamente. Para determinar la convergencia se calcula la distancia entre dos aproximaciones consecutivas de $\widehat{Q}_j$, es decir, $||\widehat{Q}_j - \widehat{Q}_{j-1}||$. Se considera que ha convergido cuando dicha distancia es menor que un umbral ξ previamente fijado. También es conveniente fijar un número máximo de iteraciones debido a que, para determinados métodos de aprendizaje supervisado, es posible que la distancia entre aproximaciones nunca sea menor que el umbral fijado (Ernst et al., 2005a).

Para que el algoritmo IVAO proporcione políticas de actuación desde el comienzo de la interacción con el entorno (y así satisfaga el primero de los dos requisitos que deben cumplir los algoritmos de RL *online*), la estimación de la función Q se debe actualizar tras un número K pequeño de transiciones. Durante las primeras actualizaciones, la cantidad de transiciones almacenadas no serán representativas de todo el espacio de estados, pero aún así la política aprendida a partir de dicho conjunto

Algoritmo 4.2 Iteración de funciones valor ajustadas para aprendizaje *online*.

Require: método de aprendizaje supervisado AS, factor de descuento γ, probabilidad de exploración ϵ, número máximo N de transiciones almacenadas, número de transiciones K entre actualizaciones de la función Q

Inicializar la función Q arbitrariamente (por ejemplo, $\widehat{Q}_1 \leftarrow 0 \; \forall \; S \times A$)
$D \leftarrow 0$; $t \leftarrow 0$; $l \leftarrow 1$
for cada episodio **do**
  $k \leftarrow 0$
  observar el estado s_0
  while el episodio no haya finalizado **do**
    $k = k + 1$
    $t = t + 1$
    $a_k = \begin{cases} a \in \arg\max_a Q_l(s_0, a) & \text{con probabilidad } 1 - \epsilon_k \\ \text{acción aleatoria en } A & \text{con probabilidad } \epsilon_k \end{cases}$
    aplicar a_k
    observar el estado siguiente s_{k+1} y la recompensa r_{k+1}
    añadir la transición realizada al conjunto de datos:
      $D = \text{UPDATEDATA}(D, N, s_k, a_k, r_{k+1}, s_{k+1})$
    if $t = l \cdot K$ (una vez cada K transiciones) **then**
      $l = l + 1$
      Actualizar la estimación de la función Q:
        $\widehat{Q}_l = \text{LEARNQ}(D, \gamma, \widehat{Q}_{l-1}, AS)$
    end if
  end while
end for
return $\widehat{Q}^* = \widehat{Q}_l$

reducido puede ser útil en determinadas zonas del espacio. Por otra parte, a la hora de fijar el parámetro K, se debe tener en cuenta que la carga computacional del algoritmo aumenta conforme K disminuye. Además, cuando la función Q se actualiza con demasiada frecuencia, la mejora aportada en cada nueva estimación suele ser muy pequeña, por lo que conviene ajustar el parámetro K para alcanzar un compromiso entre carga computacional y velocidad de convergencia.

4.3.1. Beneficios adicionales de reutilizar las transiciones

Aunque la motivación principal de reutilizar las transiciones en el algoritmo IVAO es aumentar la eficiencia respecto al uso de los datos, este procedimiento también conlleva una serie de beneficios que tal vez no resultan tan obvios. Supongamos un problema ilustrativo que consiste en un agente situado en un laberinto como el que se muestra en la Figura 4.2 (Adam et al., 2012). El objetivo del agente es encontrar el camino más corto desde la posición *inicio* hasta la posición *salida*, indicadas en la

Algoritmo 4.3 LEARNQ$(D, \gamma, \widehat{Q}, AS)$.

Require: método de aprendizaje supervisado AS, factor de descuento γ, estimación actual de la función Q ($\widehat{Q}$), conjunto de transiciones D

1: $j = 0$
2: $\widehat{Q}_j = \widehat{Q}$
3: **repeat**
4: j = j + 1
5: construir un conjunto de entrenamiento $CE = \{(i^t, o^t), t = 1, \ldots, \#D\}$ basado en la estimación actual $\widehat{Q}$ y el conjunto de transiciones D, tal que:

$$i^t = (s_k^t, a_k^t) \tag{4.4}$$

$$o^t = r_k^t + \gamma \max_{a \in A} \widehat{Q}_{j-1}(s_{k+1}^t, a) \tag{4.5}$$

6: calcular la nueva estimación $\widehat{Q}_j$ usando el algoritmo de aprendizaje supervisado AS con el conjunto de entrenamiento CE
7: **until** alcanzar condiciones de convergencia
8: **return** estimación de la función Q, $\widehat{Q}_j$

figura con las letras I y S, respectivamente. Por cuestiones de sencillez el número de estados (posiciones) es discreto, sin embargo, las conclusiones obtenidas a partir de este ejemplo son también válidas en el caso continuo. La recompensa recibida por el agente es siempre 0 excepto cuando alcanza el estado *salida*, donde la recompensa es igual a 1.

Al procesar múltiples veces los datos de una misma trayectoria para actualizar la función Q de forma iterativa, uno de los efectos que se produce es que la información se propaga con cada iteración. Consideremos que en el ejemplo del laberinto el agente se mueve desde el estado inicial hasta la salida siguiendo la trayectoria indicada en la Figura 4.2a. Cuando el agente emplea un algoritmo incremental basado en TD (como por ejemplo *Q-learning* o SARSA), la recompensa recibida tras llegar al estado *salida* únicamente se propaga hasta el estado anterior. La cantidad de recompensa propagada depende de la tasa de aprendizaje α. En la Figura 4.2b se muestra el caso de $\alpha = 1$, donde se propaga toda la recompensa. Cuando el algoritmo TD se combina con trazas de eligibilidad utilizando $\lambda < 1$, la recompensa se propaga a lo largo de toda la trayectoria, pero el valor propagado disminuye exponencialmente conforme se aleja del estado *salida* (ver Figura 4.2c). En el caso de utilizar $\lambda = 1$, se propaga la recompensa al completo (aplicando el factor de descuento γ que se haya fijado) por todos los estados de la trajectoria, tal y como se ilustra en la Figura 4.2d. El hecho de propagar la recompensa suele acelerar el proceso de aprendizaje, especialmente en problemas donde las trayectorias son largas y el agente únicamente recibe una recompensa al finalizar la trayectoria (Sutton y Barto, 1998). Al procesar todos los datos de la trayectoria por primera vez con el algoritmo IVAO, la recompensa se propaga

hasta el estado anterior al de *salida* pero, al volver a procesarlos, la recompensa es propagada hasta dos estados anteriores al de *salida*, y así sucesivamente. Por tanto, conforme aumenta el número de veces que se procesa la información, se consigue un efecto similar al de utilizar trazas de elegibilidad con $\lambda = 1$. Esta característica no es exclusiva del algoritmo IVAO y se ha observado previamente en otros algoritmos que también procesan los datos en modo *batch* repetidas veces (Cichosz, 1999; Adam et al., 2012).

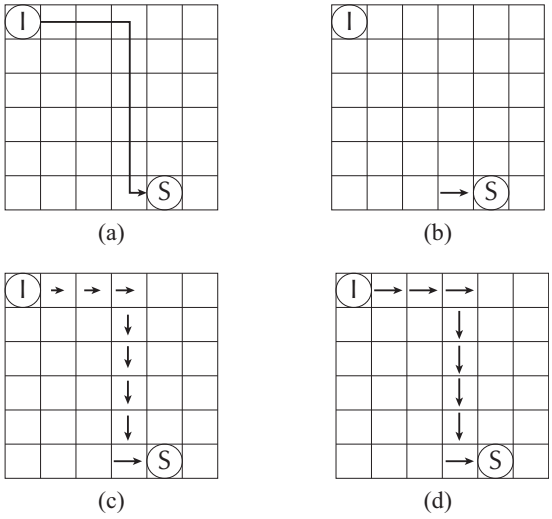


Figura 4.2
Al procesar varias veces los datos de una misma trayectoria se produce un efecto equivalente al de usar trazas de elegibilidad. El tamaño de las flechas en la figura indica la cantidad de recompensa propagada hacia atrás desde el estado *salida*. (a) Trayectoria seguida por el agente. (b) Algoritmos incrementales sin trazas de elegibilidad. (c) Algoritmos incrementales con trazas de elegibilidad y $\lambda < 1$. (d) Algoritmo IVAO y algoritmos incrementales con trazas de elegibilidad y $\lambda = 1$.

Otro beneficio de reutilizar los datos adquiridos del MDP es que hace posible agregar información procedente de diferentes trayectorias. Continuando con el ejemplo del laberinto, supongamos que el agente realiza las dos trayectorias indicadas en la Figura 4.3a: en primer lugar realiza la trayectoria que está indicada con línea punteada, y en segundo lugar la indicada con línea continua. Si se utiliza un algoritmo incremental con trazas de elegibilidad, el estado marcado en gris únicamente se beneficiará de la información procedente de la trayectoria punteada (ver Figura 4.3b). Por otra parte, al procesar los datos de las dos trayectorias a la vez, la información se agrega y el estado marcado en gris también obtendrá conocimiento procedente de la trayectoria continua (ver Figura 4.3c) (Adam et al., 2012). Este efecto y el explicado en el párrafo anterior están relacionados entre sí y explican algunos de los mecanismos que

emplea el algoritmo IVAO para extraer información de las transiciones realizadas por el agente.

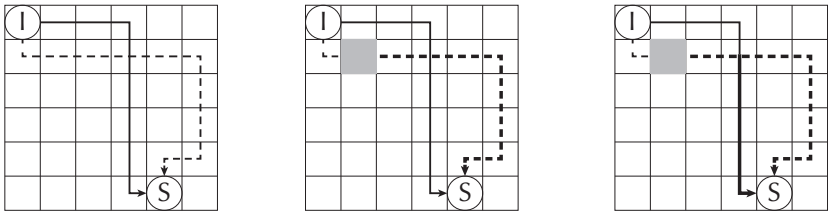


Figura 4.3
IVAO agrega información de las diferentes trayectorias. El estado marcado en gris recibe información de los estados donde la trayectoria está indicada con línea gruesa. (a) Trayectorias realizadas por el agente. (b) Algoritmo incremental con trazas de elegibilidad. (c) Algoritmo IVAO.

4.3.2. Relación con otros algoritmos similares

El algoritmo IVAO está basado, por una parte, en el esquema clásico de iteración de funciones valor (Bellman, 1957) y, por otra, en el concepto de funciones valor ajustadas, introducido originalmente en (Gordon, 1995a). Ambas ideas surgieron en el contexto de la programación dinámica, aunque más tarde han sido adaptadas al aprendizaje por refuerzo. El primer algoritmo de RL basado en funciones valor ajustadas probablemente fue el propuesto por Gordon (Gordon, 1995b). Se trata de una modificación de FVI (ver Algoritmo 4.1) para estimar funciones Q cuando no se dispone de un modelo del MDP. El funcionamiento de dicho algoritmo es muy parecido a FVI: se parte de un subconjunto de estados discretos S_c sobre los que se aplica el operador de Bellman para generar un conjunto de entrenamiento y, posteriormente, se emplea un método de aprendizaje supervisado para ajustar la función valor. Sin embargo, al no disponer del modelo del MDP para evaluar el conjunto S_c, cada vez que el agente realiza una transición desde un estado $s \notin S_c$, se busca el estado $s_c \in S_c$ más próximo a s y se asume que la transición se ha iniciado en s_c.

Más recientemente, el uso de funciones valor ajustadas se ha popularizado con la aparición de los algoritmos de RL en modo *batch*. En el trabajo de Ormoneit y Sen (2002) se propone un algoritmo que estima la función valor para los estados presentes en una trayectoria (o un conjunto de trayectorias). Después de adquirir las transiciones de dicha trayectoria, el algoritmo de Ormoneit actualiza las estimaciones de la función Q repetidas veces, empleando como aproximador de funciones un método *kernel* (Shawe-Taylor y Cristianini, 2004). El autor demuestra que el algoritmo es estadísticamente consistente y converge a una solución única. Es decir, que al aumentar el número de estados en la trayectoria, la estimación de Q siempre mejora hasta

alcanzar la función óptima. Por otra parte, Ernst et al. (2005a) hacen uso de árboles de regresión en su algoritmo denominado FQI. El principio de funcionamiento de FQI es similar al del algoritmo de Ormoneit, de hecho, la convergencia de FQI se basa en asumir que cierto tipo de árboles es equivalente a un método *kernel*. En el trabajo de Riedmiller (Riedmiller, 2005) se proporciona un estudio empírico de un algoritmo que combina redes neuronales con funciones valor ajustadas. Todos estos algoritmos (y otros similares) tienen en común que aprenden de forma *offline*, al contrario que el algoritmo IVAO.

Kalyanakrishnan y Stone (2007) aplican el algortimo FQI en un problema donde el objetivo es que un robot aprenda a jugar al fútbol. En este caso, FQI se utiliza al mismo tiempo que el agente interactúa con el entorno para poder comparar su funcionamiento con el algoritmo *Q-learning*. A pesar de ello, no se trata de una versión *online* de FQI por dos motivos: i) no se incluye ningún tipo de exploración y, ii) se almacenan todas las transiciones realizadas. Por tanto, en un problema que requiera aprendizaje *online*, esta versión del algoritmo FQI se volvería rápidamente intratable conforme aumenta el tiempo de interacción agente-entorno.

Un algoritmo que no está basado en funciones valor ajustadas pero que guarda cierta relación con IVAO es el algoritmo *experience replay* (ER) (Lin, 1992). Aunque la versión descrita en la Sección 3.5.1 funcionaba de forma *offline*, también existen implementaciones del algoritmo ER que estiman la política óptima al mismo tiempo que interactúan con el entorno. En dicho caso, el algoritmo ER almacena las transiciones más recientes y las reutiliza varias veces para actualizar la estimación de la política. Así pues, de acuerdo con la clasificación de algoritmos de RL mostrada en el esquema de la Figura 4.1, tanto el algoritmo ER como IVAO pertenecen a la clase *growing batch*. La principal diferencia entre ambos algoritmos es que ER está basado en métodos TD y requiere utilizar un aproximador de tipo paramétrico, mientras que el algoritmo IVAO permite el uso de aproximadores tanto paramétricos como no paramétricos.

4.4. Estudio experimental

A lo largo de esta sección se evaluará extensivamente el funcionamiento del algoritmo IVAO en tres entornos cuya complejidad (número de dimensiones) aumenta gradualmente. En los tres entornos el objetivo es controlar alguna de las variables de estado hacia un valor fijado previamente. Concretamente, la velocidad de un vehículo submarino en el primer entorno, la posición de un motor de corriente continua en el segundo y el ángulo de incidencia de un avión cuando vuela en modo crucero en el último entorno. Como método de referencia para comparar los resultados obtenidos por el algoritmo IVAO se utilizó el algoritmo *Q-learning* combinado con trazas de elegibilidad. En ambos algoritmos se aproximaron las funciones valor mediante una red de RBFs con base fija. Nótese que una de las ventajas del algoritmo IVAO es que no impone ninguna restricción en cuanto al tipo de aproximador, por lo que también

se puede combinar con aproximadores no paramétricos o, incluso, es posible cambiar el aproximador de una iteración a otra, adaptándolo a las necesidades de la función que se quiere aproximar. A pesar de ello se decidió mantener el mismo aproximador que *Q-learning* para que las condiciones en ambos casos fuesen similares. De este modo, las diferencias observadas en los resultados serán consecuencia de la forma en que procesan los datos ambos algoritmos. Finalmente, en la última parte del estudio experimental, se utiliza el problema del motor de corriente continua para examinar el efecto que se produce en los resultados al variar los parámetros de configuración del algoritmo IVAO.

4.4.1. Vehículo submarino

Descripción del entorno

Se trata de un problema diseñado originalmente para demostrar las propiedades de ciertos algoritmos de control automático. El propósito no es reproducir con fidelidad ningún sistema real, de hecho, la estructura del problema es relativamente simple ya que únicamente contiene una variable de estado. A pesar de ello la dinámica tiene propiedades altamente no lineales, por lo que obtener una política de control no es una tarea trivial. El problema ha sido empleado recientemente para evaluar algoritmos de RL en (Hafner y Riedmiller, 2011), aunque se puede encontrar una descripción más detallada en (Slotine y Li, 1991).

El objetivo es controlar la velocidad de un vehículo submarino de pequeño tamaño que es propulsado por una hélice. La variable de estado es la velocidad, v, del vehículo que se encuentra sumergido en el agua. Tanto la masa, m, como el coeficiente de rozamiento, c, se asumen que no son constantes, sino que se sustituyen por funciones que dependen de la velocidad, denotadas como $m(v)$ y $c(v)$, respectivamente. Estas funciones representan los efectos que aparecen cuando un vehículo se desplaza a través de un fluido. Además se asume que la propulsión producida por la hélice no es directamente proporcional a la acción de control, a. La propulsión efectiva se obtiene a través de un coeficiente, $k(v, a)$, que modela la eficiencia que exhibe la hélice a diferentes velocidades del vehículo y para las distintas acciones de control (Hafner y Riedmiller, 2011). En la Figura 4.4 se muestra la dinámica del vehículo submarino, $\dot{v} = f(v, a)$, para varias combinaciones de acciones de control y velocidades. En esta figura se puede apreciar que $\dot{v}$ es altamente no lineal.

La siguiente ecuación describe cómo evoluciona el estado del vehículo submarino a lo largo del tiempo (Hafner y Riedmiller, 2011):

$$\dot{v} = f(v, a) = \frac{a \cdot k(v, a) - c(v) \cdot v \cdot |v|}{m(v)} \tag{4.6}$$

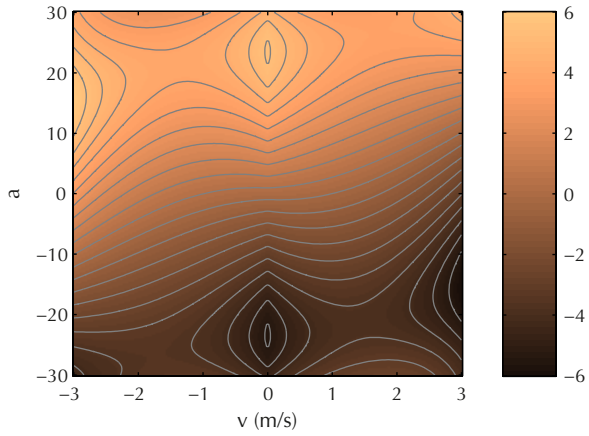


Figura 4.4
Dinámica del vehículo submarino, $\dot{v} = f(v, a)$. Se puede apreciar que la aceleración que sufre el vehículo varía de forma no lineal en función de la velocidad a la que se mueve y la acción de control que se le aplica.

donde

$$c(v) = 1.2 + 0.2 \cdot \sin(|v|)$$
$$m(v) = 3.0 + 1.5 \cdot \sin(|v|)$$
$$k(v, a) = -0.5 \cdot \tanh\left[(|(c(v) \cdot v \cdot |v|) - a| - 30) \cdot 0.1\right] + 0.5$$

Durante los experimentos el sistema fue simulado mediante la instrucción *ode45* de MATLAB, que implementa un método de Runge-Kutta de orden 4. El incremento temporal se seleccionó como $t = 0.03$ s. El vector de estados, $s = [v]$, es unidimensional y solo depende de la velocidad, cuyo rango está limitado a $[-4, 4]$ m/s. Tal y como ocurre en otros problemas similares usados en RL (Lagoudakis y Parr, 2003), el espacio de acciones se discretizó en su valor medio, máximo y mínimo, es decir, $A = \{-30, 0, 30\}$. Cada episodio se dio por concluido tras transcurrir 150 instantes temporales o cuando el vehículo alcanza los límites de velocidad permitidos. La variable de consigna se seleccionó como $v_{\text{set}} = 2$ m/s. El estado inicial de los episodios se eligió de forma aleatoria a partir de una distribución uniforme sobre el rango completo de velocidades permitidas.

La función de recompensa alcanza su valor máximo cuando la variable que se quiere controlar, en este caso la velocidad, se encuentra cerca del valor de consigna, y decrece suavemente hasta cero conforme se aleja de la consigna. Si definimos la desviación de control en el instante k, e_k, como la diferencia entre la variable a controlar, v_k y el valor de consigna, v_{set}:

$$e_k = v_k - v_{\text{set}} \tag{4.7}$$

entonces, la función de recompensa viene dada por la expresión:

$$\begin{aligned} \rho(s,a) &= \rho(e) \\ &= 1 - \tanh^2(|e| \cdot w) \\ w &= \tanh^{-1}\left(\frac{\sqrt{0.95}}{\mu}\right) \end{aligned} \tag{4.8}$$

donde μ, que es el parámetro que controla la pendiente de la función, se ha fijado igual a 0.01. En la Figura 4.5 se muestra la forma de la función de recompensa frente a la variable controlada. El hecho de que la recompensa sea continua y varíe de forma suave aporta dos ventajas comparado con otras recompensas discretas (Deisenroth et al., 2009; Hafner y Riedmiller, 2011):

1. Permite que el agente aprenda políticas de control más precisas, especialmente cuando el espacio de acciones es continuo.
2. Produce funciones valor que también varían de forma suave y, por tanto, pueden aproximarse con mayor exactitud (cuando se emplean ciertos aproximadores).

Además, la función de recompensa definida en la Ecuación (4.8) es de carácter general y se puede aplicar en un amplio rango de problemas con modificaciones mínimas.

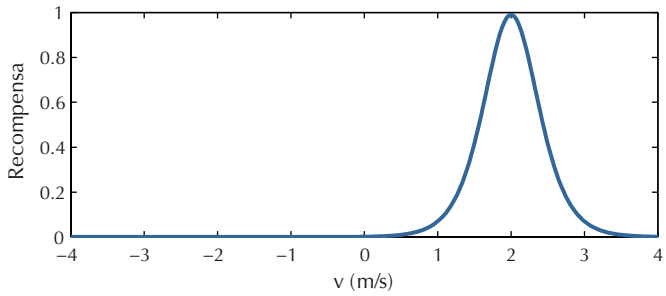


Figura 4.5
Función de recompensa empleada en el problema del vehículo submarino. En este caso el valor de consigna es $v_{\text{set}} = 2$ m/s.

Configuración de los algoritmos

Los parámetros que tienen en común el algoritmo propuesto, IVAO, y el algoritmo empleado como referencia, *Q-learning*, se configuraron con los mismos valores con el fin de que la comparación fuese lo más equitativa posible. La función Q se aproximó de forma independiente para cada una de las tres acciones. Cada red RBF constaba de

50 funciones de base radial (gaussianas) cuyos centros se distribuyeron formando una malla regular en todo el espacio de estados, lo que hace un total de $50 \times 3 = 150$ RBFs. El ancho las gaussianas fue el mismo en todos los casos, $\sigma = 0.2$. Esta configuración del aproximador se seleccionó empíricamente. El parámetro ϵ, que determina el grado de exploración, se mantuvo constante e igual a 0.3.

El número de transiciones muestreadas entre diferentes actualizaciones de la política en el algoritmo IVAO se fijó como $K = 25$. Por otra parte, el número de transiciones que se almacenan y procesan en cada actualización se eligió $N = 800$. Para detener el bucle que calcula la estimación de Q se implementaron las dos condiciones de parada explicadas en la Sección 4.3. Por una parte, se determinó la distancia entre aproximaciones sucesivas de Q y se fijó un umbral, es decir, $||\widehat{Q}_j - \widehat{Q}_{j-1}|| < \xi$, siendo $\xi = 0.07$. De esta forma el bucle se detiene cuando Q permanece aproximadamente constante entre dos estimaciones consecutivas. Dado que las estimaciones de Q son funciones continuas, la distancia entre ellas se calculó de forma aproximada en un conjunto discreto de 25 puntos distribuidos uniformemente en el espacio de estados. Por otra parte, la segunda condición de parada consistió en fijar un número máximo de 40 iteraciones que habitualmente no se suele alcanzar.

Respecto al algoritmo *Q-learning*, es necesario establecer una secuencia de tasas de aprendizaje adecuada. En determinadas circunstancias es posible, e incluso aconsejable, mantener un valor de α_k fijo. Sin embargo, en general, el hecho de modificarlo conforme aumenta el número de interacciones agente-entorno permite disminuir el tiempo de convergencia a la vez que se mejora la estabilidad. Concretamente se utilizó la regla *search-then-converge* (STC) (Darken y Moody, 1992):

$$\alpha_k = \alpha_0 \frac{\left(\frac{m}{n} + n\right)}{\left(\frac{m}{k} + n + k^{\beta}\right)} \tag{4.9}$$

donde $m = 10$, $n = 3$, $\beta = 0.45$ y $\alpha_0 = 0.6$. La secuencia de tasas de aprendizaje generada por la regla STC proporciona un valor de α_k elevado al principio (etapa de búsqueda) que se va reduciendo a medida que k aumenta (etapa de convergencia). La regla STC requiere ajustar más parámetros que otros métodos para definir la secuencia de tasas de aprendizaje pero, por otra parte, ofrece una mayor flexibilidad para adaptar α de acuerdo a las necesidades del algoritmo (Powell, 2011). Otra técnica que puede acelerar la convergencia de *Q-learning* es la inclusión de trazas de elegibilidad (Sutton, 1988). Esta técnica introduce un nuevo parámetro que controla la "longitud" de las trazas y suele denotarse como λ. Igual que ocurre con otros parámetros, para seleccionar un valor adecuado de λ es necesario experimentar con el entorno y ajustar el valor empíricamente. Para el problema del vehículo submarino se utilizó $\lambda = 0.4$.

Resultados

Los resultados obtenidos con los algoritmos IVAO y *Q-learning* se muestran en las Figuras 4.6a y 4.6b, respectivamente. Dichas figuras describen cómo evoluciona el proceso de aprendizaje conforme aumenta la experiencia adquirida por el agente. Para construir las gráficas se ha evaluado la política estimada por cada algoritmo en diferentes instantes temporales. En cada evaluación se detiene el aprendizaje y la tasa de exploración se configura igual a 0, por lo que todas las acciones son *greedy* respecto a la política que se pretende evaluar. A continuación se realizan varias trayectorias desde diferentes estados iniciales y se calcula un promedio de la recompensa obtenida por cada una de las trayectorias. Los valores promedio obtenidos son los que, finalmente, se representan en las gráficas. Concretamente se ha empleado el conjunto de estados iniciales $S_0 = \{-3, -2.14, -1.29, -0.43, 0.43 - 1.29, -2.14, -3\}$ que cubre distintas partes del espacio de estados y proporciona una medida del comportamiento general de las políticas.

Los experimentos se han repetido 24 veces con el objetivo de obtener resultados estadísticamente significativos. En las Figuras 4.6a y 4.6b se muestra el valor medio de los experimentos junto con el intervalo de confianza al 95 %. Se puede observar que ambos algoritmos convergen hacia políticas aproximadamente óptimas. Conforme la calidad de las políticas aumenta, el intervalo de confianza se reduce, lo que muestra la fiabilidad de los algoritmos. Al comparar la velocidad de convergencia, los resultados reflejan una clara superioridad del algoritmo IVAO. Para obtener la misma recompensa media que alcanza el algoritmo IVAO con 100 transiciones ($k = 100$), el algoritmo *Q-learning* requiere unas 200 transiciones, prácticamente el doble.

Adicionalmente, en la Figura 4.7 se muestra un ejemplo de una trayectoria típica controlada por una política aproximadamente óptima aprendida con el algoritmo IVAO. En la parte superior de la figura se muestra la variable de estado, donde puede apreciarse que el estado inicial de la trayectoria es $v = -2.14$ m/s. Dado que el objetivo es alcanzar $v_{\text{set}} = 2$ m/s, la política comienza con una acción (parte inferior de la figura) de propulsión $a = 30$ que se mantiene hasta alcanzar la velocidad deseada, aproximadamente 1.2 s después. A partir de este momento el agente alterna entre dos acciones, $a = 30$ y $a = 0$. El resultado es que la velocidad se mantiene aproximadamente constante en el punto de consigna. La pequeñas oscilaciones que se observan en la variable controlada son consecuencia de utilizar un conjunto de acciones discreto.

4.4.2. Posición de un motor de corriente continua

Descripción del entorno

En el segundo entorno se dispone de un motor de corriente continua (DC) que proporciona movimiento rotacional. El circuito eléctrico equivalente se muestra en la Figura 4.8, mientras que el significado y el valor de cada uno de los parámetros físicos se resumen en la Tabla 4.1. Dichos valores se corresponden con los obtenidos

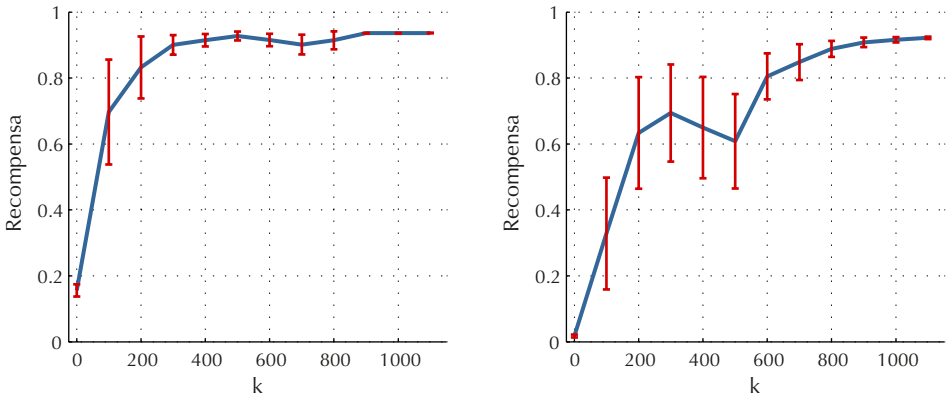


FIGURA 4.6
Evolución de las políticas aprendidas por los algoritmos (a) IVAO y (b) *Q-learning* en el problema del vehículo submarino conforme aumenta la experiencia adquirida por el agente.

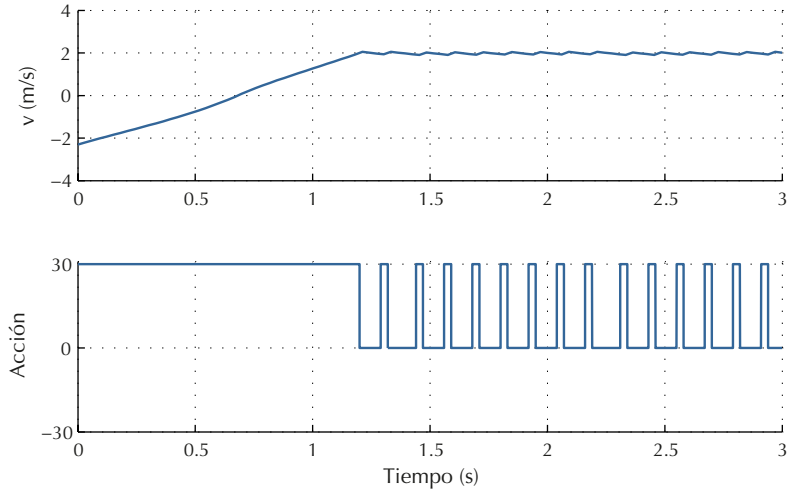


FIGURA 4.7
Trayectoria del vehículo submarino comenzada desde $s_0 = -2.14$ y controlada por una política aprendida con el algoritmo IVAO, siendo $v_{\text{set}} = 2$ m/s. En la parte superior se muestra la variable controlada y en la inferior la acción de control.

experimentalmente a partir de un motor real (Busoniu et al., 2008b, 2010a). Una vez obtenidos los parámetros, se diseñó un modelo en tiempo continuo que posteriormente fue discretizado mediante el método del mantenedor de orden cero, usando como tiempo de muestreo $T_s = 0.005$ s (Busoniu et al., 2010a). El resultado de este proceso es un modelo de segundo orden en tiempo discreto que describe el comportamiento del motor (Busoniu et al., 2008b).

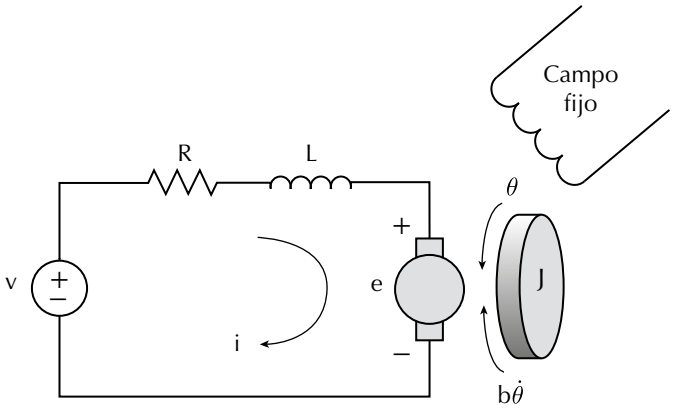


Figura 4.8
Circuito eléctrico equivalente de un motor DC.

Tabla 4.1
Parámetros físicos del motor DC.

Parámetro	Símbolo	Valor	Unidades
Inercia del rotor y de la carga	J	$3.24 \cdot 10^{-5}$	N·m^2
Constante de fricción del motor	b	$3 \cdot 10^{-6}$	N
Par de giro	K	$53.6 \cdot 10^{-3}$	N·m/A
Resistencia del rotor	R	9.5	Ω
Inductancia del rotor	L	$0.84 \cdot 10^{-3}$	H

El modelo discreto se puede expresar mediante la siguiente ecuación en diferencias:

$$s_{k+1} = f(s_k, a_k) = As_k + Ba_k \tag{4.10}$$
$$A = \begin{pmatrix} 1 & 0.0049 \\ 0 & 0.9540 \end{pmatrix}, \quad B = \begin{pmatrix} 0.0021 \\ 0 \end{pmatrix}$$

Las variables que definen el estado del motor en un instante dado son el ángulo del eje, θ, y la velocidad angular, $\dot{\theta}$, por tanto $s = [\theta, \dot{\theta}]$. La variable de control es la tensión

de entrada, cuyo valor puede variar en el rango $[-10, 10]$ V. El objetivo es estabilizar el motor DC en el punto de equilibrio, $\theta_{\text{set}} = 0$. La siguiente función de recompensa cuadrática es una de las posibles formas de expresar este objetivo (Busoniu et al., 2010a):

$$r_{k+1} = \rho(s_k, a_k) = -s_k^\top Q_{\text{rew}} s_k - R_{\text{rew}} a_k^2 \tag{4.11}$$
$$Q_{\text{rew}} = \begin{pmatrix} 5 & 0 \\ 0 & 0.005 \end{pmatrix}, \quad R_{\text{rew}} = 0.01$$

En la Figura 4.9 se ha representado la recompensa en función del ángulo del eje y su velocidad angular cuando $a = 0$. La función de recompensa empleada da lugar a un problema de regulación cuadrática. Una política óptima no llevará únicamente el ángulo del motor θ hasta el valor de consigna sino que también tenderá a minimizar la velocidad angular $\dot{\theta}$ y la señal de control. El factor de descuento fue seleccionado como $\gamma = 0.95$, un valor suficientemente grande como para aprender políticas con un comportamiento adecuado (Busoniu et al., 2008b).

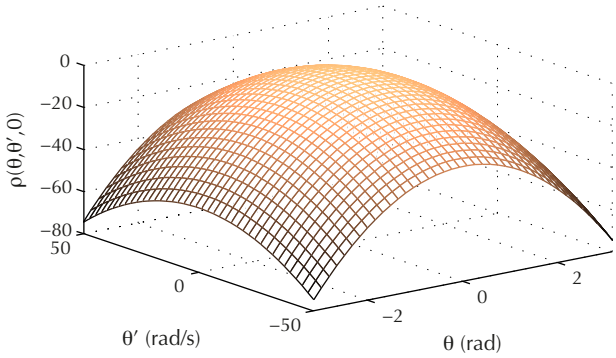


Figura 4.9
Porción de la función de recompensa usada en el problema del motor DC cuando $a = 0$.

Configuración de los algoritmos

Igual que en el caso del vehículo submarino la función Q se aproximó mediante una red de RBFs gaussianas para cada una de las acciones. Los centros de las funciones gaussianas se colocaron formando una malla con 9×9 puntos equidistantes cubriendo todo el espacio de estados. La variables de estado se normalizaron en el rango $[-1, 1]$ y el ancho de cada gaussiana se fijó como $\sigma = 0.18$ en ambas dimensiones. Este valor de σ se seleccionó para que la red de RBFs proporcionara una interpolación suave en todo el espacio de estados. La probabilidad de elegir acciones exploratorias fue fijada como $\epsilon = 0.3$. Tal vez sorprenda que tanto en este entorno como en el anterior se haya empleado un valor de ϵ constante. De acuerdo con lo explicado en la Sección 4.3,

resulta más adecuado que el valor de ϵ varíe conforme aumenta la experiencia del agente. Sin embargo, como la influencia de ϵ es la misma en el algoritmo IVAO y en *Q-learning*, su efecto no se aprecia en la comparación. En consecuencia, por cuestiones de sencillez, se decidió utilizar un valor de ϵ constante en todos los experimentos (excepto cuando se estudia la influencia de dicho parámetro).

El algoritmo IVAO se configuró para actualizar la estimación de Q cada $K = 100$ transiciones y empleando un máximo de $N = 1500$ transiciones. Respecto a las condiciones de parada de la función LEARN(), se estableció un número máximo de 40 iteraciones y un umbral de distancia $\xi = 0.1$. La distancia entre aproximaciones sucesivas de Q se calculó en un conjunto de 36 puntos distribuidos regularmente en el espacio definido por los rangos $\theta = [-\pi, \pi]$ y $\dot{\theta} = [-50, 50]$.

En la secuencia de tasas de aprendizaje de *Q-learning* se empleó la regla STC (ver Ecuación (4.9)) con los valores $m = 25$, $n = 5$, $\beta = 0.3$ y $\alpha_0 = 0.6$. Por último, el parámetro λ relacionado con las trazas de elegibilidad se fijó igual a 0.4.

Resultados

El proceso de aprendizaje tanto del algoritmo propuesto (IVAO) como del algoritmo de referencia (*Q-learning*) se ilustra en la Figura 4.10. Se emplearon un total de 25 trayectorias para calcular la recompensa promedio, cuyos estados iniciales fueron:

$$S_0 = \{-3, -1.5, 0, 1.5, 3\} \times \{-45, -22.5, 0, 22.5, 45\}$$

En este experimento resulta más evidente la diferencia respecto a la velocidad de convergencia entre ambos algoritmos. Después de 150 instantes temporales, el algoritmo *Q-learning* obtiene un resultado ligeramente superior, ya que las políticas aprendidas por el algoritmo IVAO durante este periodo presentan resultados bastante dispares que, en promedio, no mejoran la política aleatoria inicial. A partir de $k = 150$, el algoritmo propuesto converge rápidamente hacia políticas óptimas mientras que la convergencia de *Q-learning*, a pesar de ser constante, es considerablemente más lenta. Tanto es así, que incluso después de 1500 transiciones, *Q-learning* aún está lejos de obtener políticas cercanas a la óptima.

En la Figura 4.11 se muestra una trayectoria típica del motor DC cuando es controlado con una política aprendida con el algoritmo IVAO. El eje del motor comienza con una posición de -2.7 rad y una velocidad angular de 18 rad/s. Se puede observar que, en apenas 0.2 s, el motor ya ha alcanzado una posición prácticamente de equilibrio, con ambas variables de estado con un valor cercano a 0. Cabe mencionar que, en ocasiones, el motor se queda oscilando alrededor del punto de equilibrio indefinidamente, igual que ocurría en el problema del vehículo submarino. De nuevo este efecto se debe a que el espacio de acciones se ha discretizado en un conjunto de acciones que no permiten alcanzar exactamente el valor de consigna desde todos los estados iniciales. En las Figuras 4.12 y 4.13 se muestra la política de control usada en la trayectoria mostrada y una porción de la función Q que da lugar a dicha política.

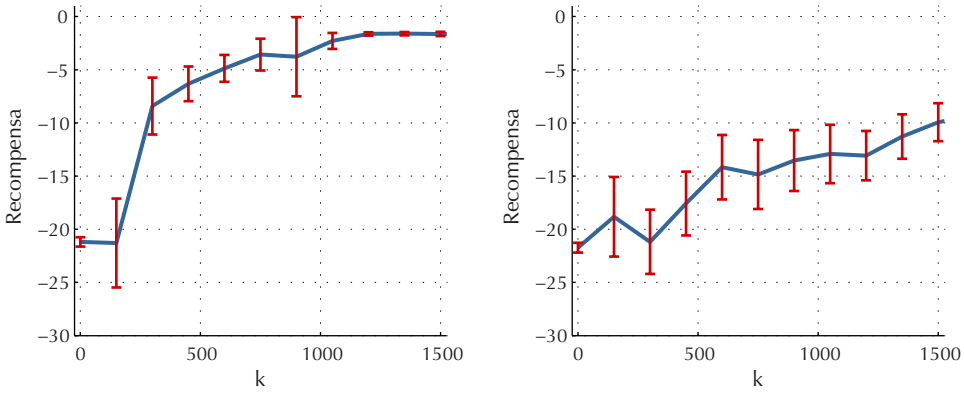


Figura 4.10
Evolución de las políticas aprendidas por los algoritmos (a) IVAO y (b) *Q-learning* en el problema del motor DC conforme aumenta la experiencia adquirida por el agente.

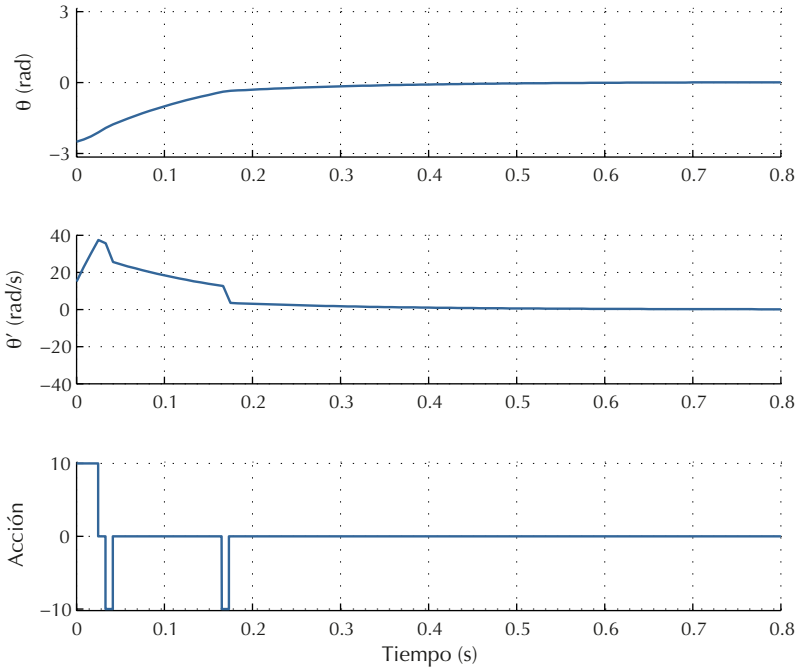


Figura 4.11
Trayectoria del motor DC desde la posición inicial $\theta = -2.7$ rad y $\dot{\theta} = 18$ rad/s. Se ha empleado una política aprendida con el algoritmo IVAO tras 1500 interacciones agente-entorno.

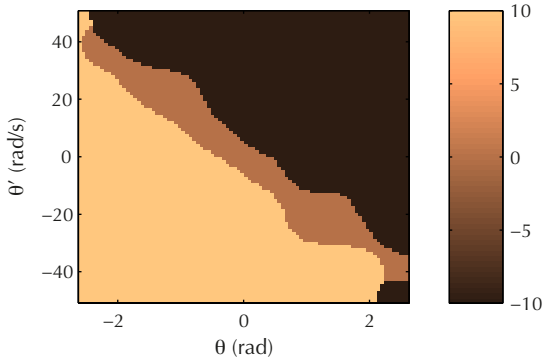


Figura 4.12
Política de control del motor DC obtenida con el algoritmo IVAO.

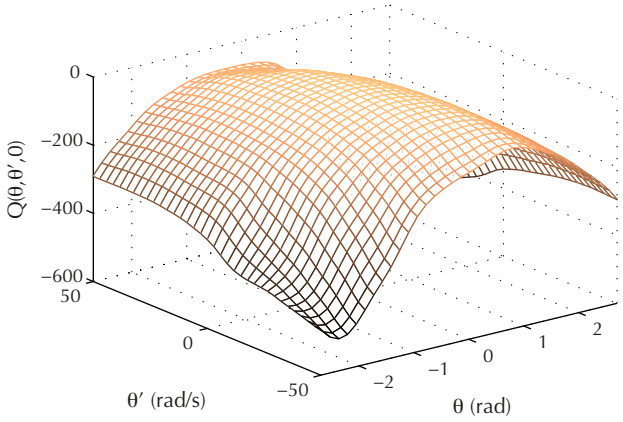


Figura 4.13
Porción de la función Q obtenida con el algoritmo IVAO y $a = 0$.

4.4.3. Control del ángulo de incidencia de un avión

Descripción del entorno

El último entorno utilizado para evaluar el funcionamiento del algoritmo IVAO consistió en un piloto automático que controla el ángulo de incidencia de un avión Boeing. El problema original está descrito por un sistema de seis ecuaciones diferenciales no lineales que, bajo ciertas asunciones, se pueden desacoplar y linealizar (University of Michigan, 1996). En la versión simplificada el objetivo es controlar el ángulo de incidencia del avión cuando éste se encuentra en modo crucero, donde la velocidad y la altitud se mantienen constantes (Hafner y Riedmiller, 2011). Bajo estas condiciones, los efectos de la propulsión, rozamiento y sustentación pueden ser despreciados. Tam-

bién se asume que cualquier cambio en el ángulo de incidencia no provoca variación alguna en la velocidad del avión bajo ninguna circunstancia.

La Figura 4.14 muestra de forma esquemática algunas de las variables de estado del problema. El eje principal del avión, X, se considera la base de coordenadas del sistema. V denota la velocidad del avión que, de acuerdo con las simplificaciones comentadas previamente, se asume constante para cualquier ángulo de incidencia. El ángulo de incidencia, θ, o inclinación del avión es el ángulo que forma el eje principal con el horizonte; mientras que el ángulo de ataque, α, es el formado entre el eje principal y el vector de velocidad. La tarea de control consiste en modificar la posición de los estabilizadores para que el ángulo de incidencia se aproxime tan pronto como sea posible al valor deseado $\theta_{\text{set}} = 0.02$ rad. El ángulo δ indica la posición de los estabilizadores respecto al eje principal. Al modificar δ también se influye en otras variables de estado, concretamente en el ángulo de ataque y en la velocidad de variación de dicho ángulo, denotada por q.

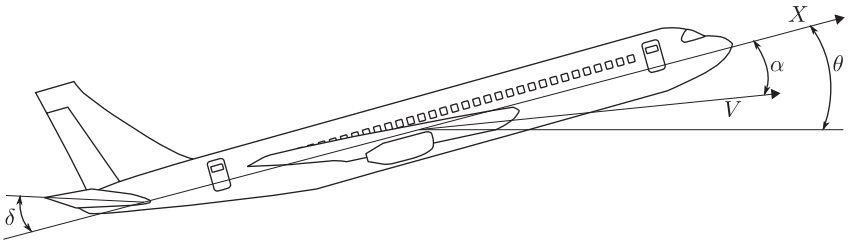


Figura 4.14
Representación esquemática del avión con algunas de las variables de estado.

La variables de estado del avión evolucionan de acuerdo al siguiente sistema de ecuaciones:

$$\begin{aligned} \dot{\alpha} &= -0.313\alpha + 56.7q + 0.232\delta \\ \dot{q} &= -0.0139\alpha - 0.426q + 0.0203\delta \\ \dot{\theta} &= 56.7q \end{aligned} \tag{4.12}$$

donde se asume que $\theta \in [-0.5, 0.5]$ rad, ya que el modelo simplificado únicamente proporciona una aproximación realista del comportamiento del avión para dicho rango.

A pesar de que el sistema de ecuaciones (4.12) es lineal, el proceso tiene una propiedad que dificulta la tarea de control. El ángulo de incidencia se puede modificar rápidamente actuando sobre el estabilizador; sin embargo, el ángulo de ataque tiene una dinámica muy lenta comparado con el ángulo de incidencia. Una vez alcanzado θ_{set}, el ángulo de ataque todavía sigue variando porque necesita más tiempo para

estabilizarse. Por lo tanto, el agente debe actuar de forma continua sobre la posición de los estabilizadores para compensar este efecto y mantener el ángulo de ataque cerca de θ_{set}.

Igual que en el problema del vehículo submarino se empleó un método de Runge-Kutta de orden 4 para simular el sistema; en este caso con una constante temporal $t = 0.05$ s. El espacio de estados está formado por tres variables, $s = [\alpha, q, \theta]$, mientras que el espacio de acciones es unidimensional, $a = [\delta]$, donde δ se ha discretizado en dos valores $\delta = \{-1.4, 1.4\}$. La longitud máxima de cada episodio se fijó igual a $k = 600$. Cada episodio termina cuando se alcanza la longitud máxima o cuando el ángulo de incidencia excede el rango permitido. El estado inicial de los episodios se eligió de forma aleatoria dentro del subespacio definido por $\alpha = [-0.3, 0.3]$, $q = [-0.015, 0.015]$ y $\theta = [-0.5, 0.5]$. La recompensa recibida por el agente se calculó mediante la Ecuación (4.8).

Configuración de los algoritmos

La estructura del aproximador de funciones empleado en este entorno consistió en 125 RBFs que dependen únicamente del estado y las cuales se replicaron para cada una de las acciones. Así pues, la función Q se aproximó con un total de 250 funciones gaussianas. Las tres variables de estado se normalizaron en el rango $[-1, 1]$. La forma de todas las funciones base se mantuvo idéntica con $\sigma = 0.5$ en las tres dimensiones. Los centros se distribuyeron de forma equiespaciada por todo el espacio formando una malla regular de tamaño $5 \times 5 \times 5$. Igual que en los problemas anteriores, la estrategia de exploración se configuró con ϵ constante e igual a 0.3.

Tras diversos experimentos para explorar diferentes configuraciones del algoritmo IVAO, se decidió fijar $N = 3500$ transiciones y actualizar la función Q cada $K = 100$ instantes temporales. Al comparar estos valores con los usados en problemas anteriores, se observa que el parámetro N aumenta paralelamente a la complejidad del problema. Este efecto no supone ninguna sorpresa ya que es de esperar que para aproximar una función en un espacio de mayor dimensionalidad se requiera un mayor número de puntos. Las condiciones de parada del bucle que actualiza la estimación de Q se configuraron con un número máximo de 40 iteraciones y un umbral $\xi = 0.25$. En este caso, la distancia entre aproximaciones de Q se calculó en 216 puntos discretos equidistantes.

En cuanto al algoritmo *Q-learning*, se emplearon los parámetros $m = 20$, $n = 15$, $\beta = 0.4$ y $\alpha_0 = 0.6$ en la regla STC para calcular la secuencia de tasas de aprendizaje α_k. La longitud de las trazas de elegibilidad que obtuvo mejores resultados durante las pruebas fue $\lambda = 0.3$ de entre los valores $\lambda = \{0.1, \ldots, 0.9\}$.

Resultados

Los resultados obtenidos por los algoritmos IVAO y *Q-learning* en este entorno se resumen en las Figuras 4.15a y 4.15b. Ambos métodos son capaces de controlar adecuadamente el ángulo de incidencia del avión tras adquirir unas 2500 transiciones. Las recompensas promedio mostradas en las figuras se obtuvieron a partir de 27 trayectorias con los siguientes estados iniciales:

$$S_0 = \{-0.25, 0, 0.25\} \times \{-0.01, 0, 0.01\} \times \{-0.4, 0, 0.4\} \tag{4.13}$$

De nuevo, los experimentos realizados con este entorno sugieren que el algoritmo IVAO hace un uso más eficiente de los datos: con menos de 1000 transiciones es capaz de proporcionar políticas cercanas a la óptima, mientras que el algoritmo *Q-learning* requiere más de 2000 transiciones para lograr políticas de una calidad similar. Por tanto, se puede considerar que en este entorno la velocidad de convergencia del algoritmo IVAO es el doble que la de *Q-learning*.

La Figura 4.16 muestra un ejemplo de trayectoria controlada por una política aprendida con el algoritmo IVAO. Se muestran las tres variables de estado junto con la acción de control. Se puede apreciar que el ángulo de ataque α tiene una dinámica mucho más lenta que el ángulo de incidencia θ, tal y como se explicó en la descripción del entorno. Después de que el avión haya alcanzando una posición con un ángulo de incidencia adecuado, α todavía no se ha estabilizado y continúa aumentando de forma lenta pero constante. Debido a ello se tiene que modificar constantemente la acción aplicada sobre los estabilizadores del avión para mantener θ cerca del valor de consigna.

4.4.4. Efecto de los parámetros

En esta sección se estudia el efecto de variar algunos de los parámetros del algoritmo IVAO. En concreto se realizó un barrido del número de transiciones almacenadas, N, y de la probabilidad de escoger acciones exploratorias, ϵ. Los experimentos de esta sección se realizaron únicamente con el entorno del motor DC; sin embargo, las conclusiones obtenidas son también válidas para el resto de entornos. La calidad de las políticas en este caso se ha medido utilizando directamente la diferencia entre el ángulo del eje del motor, θ, y el valor de consigna, θ_{set}. Esta medida, aunque es equivalente a la utilizada en las secciones anteriores (basada en la recompensa obtenida), permite apreciar mejor las diferencias entre políticas de baja calidad.

Para estudiar la influencia de N se realizaron experimentos con los siguientes valores, $N = 250, 500, 750, 1000, 1250, 1500, 2000$ y 2500. El resto de parámetros, tanto del algoritmo como aquellos relacionados con el aproximador, se mantuvieron fijos e igual a los empleados en la Sección 4.4.2. Para cada valor de N se evaluó la política estimada en distintos instantes temporales k, proceso que fue repetido de forma independiente 24 veces. Los resultados obtenidos para cada valor de N se muestran

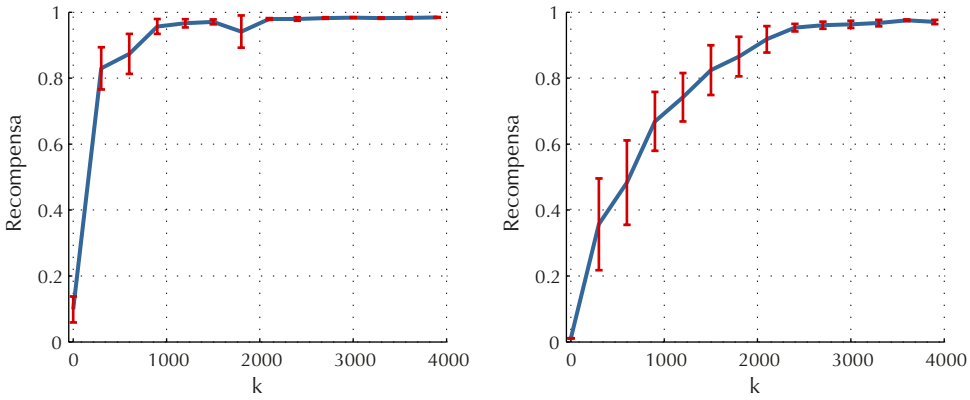


Figura 4.15
Evolución de las políticas aprendidas por los algoritmos (a) IVAO y (b) *Q-learning* conforme aumenta la experiencia adquirida por el agente en la tarea de controlar al ángulo de incidencia de un avión.

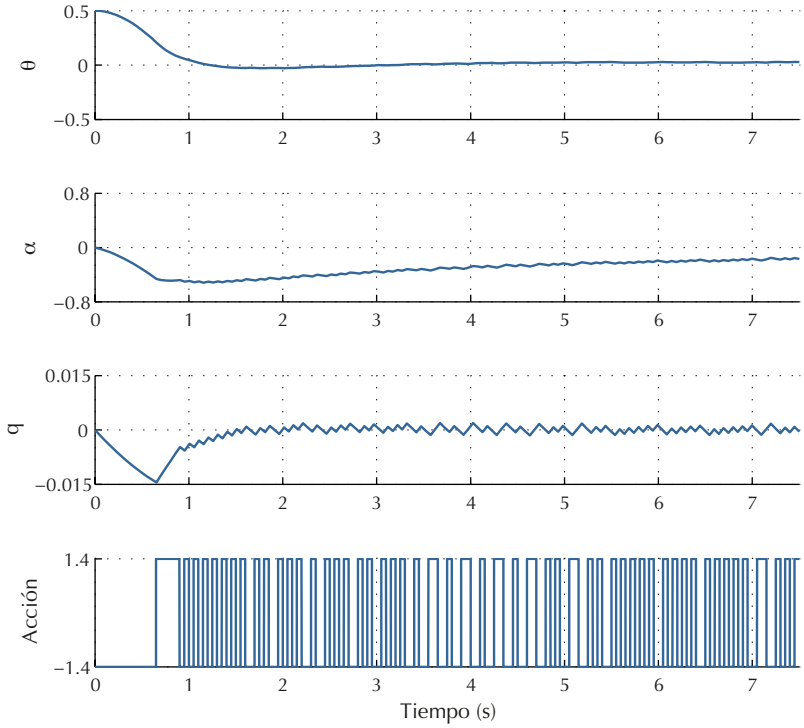
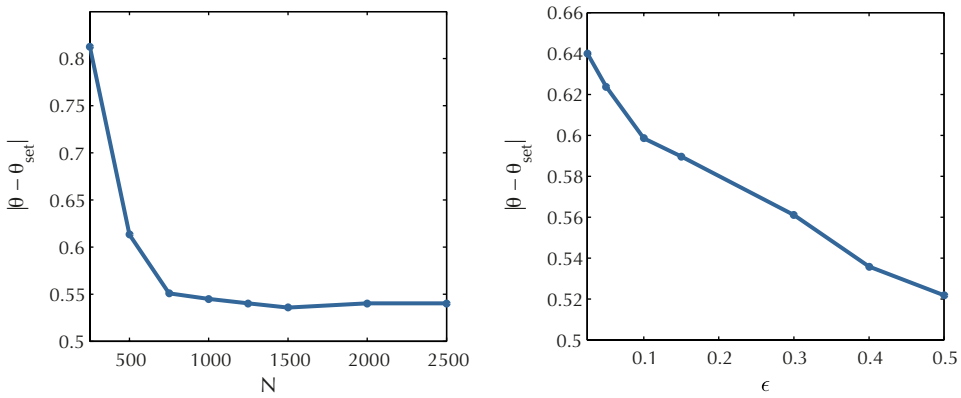


Figura 4.16
Trayectoria típica de las variables de estado del avión cuando se aplica una política de control óptima.

de forma resumida en la Figura 4.17a. Cada punto de la curva se ha calculado promediando la distancia $|\theta - \theta_{\text{set}}|$ en todos los instantes para los que se han evaluado las políticas. La figura confirma el comportamiento que se puede esperar de forma intuitiva: cuando el número de transiciones almacenadas es pequeño y no representa adecuadamente todo el espacio de estados, las estimaciones de Q son erróneas y, por tanto, las políticas obtenidas a partir de dichas estimaciones son subóptimas. A medida que el valor configurado de N crece, la calidad de las políticas obtenidas aumenta hasta ser aproximadamente óptimas. Valores de N excesivamente grandes no suponen un deterioro de las políticas aprendidas, sin embargo, conllevan una mayor carga computacional. Así pues, es importante elegir el valor de N mínimo que produzca políticas adecuadas.



(a) (b)

Figura 4.17
Efectos de variar (a) el número N de transiciones almacenadas y (b) la probabilidad ϵ de escoger una acción exploratoria en el proceso de aprendizaje del algoritmo IVAO aplicado al problema del motor DC. Durante la evaluación de las políticas el aprendizaje se detiene y ϵ se fija igual a 0.

El efecto de variar la probabilidad de escoger una acción aleatoria, ϵ, fue evaluado con un experimento similar. En este caso se dejaron fijos el resto de parámetros y se probaron los valores $\epsilon = 0.025, 0.05, 0.1, 0.15, 0.3, 0.4$ y 0.5. En la Figura 4.17b se muestran los resultados para cada valor de ϵ. La forma decreciente de la curva no supone ninguna sorpresa ya que, dado un número fijo de transiciones, el hecho de que contengan acciones más exploratorias permite al agente descubrir mejores políticas. Es importante notar que el algoritmo IVAO siempre convergerá hacia la misma política con cualquier valor de $\epsilon > 0$ siempre y cuando se permita al agente interaccionar con el entorno suficiente tiempo. Por tanto, el valor de ϵ tiene una influencia directa sobre la velocidad de convergencia, siendo dicha velocidad mayor conforme ϵ aumenta. De acuerdo con este criterio sería conveniente elegir valores de ϵ elevados, incluso igual a 1. Sin embargo, es necesario tener en cuenta al menos un segundo criterio a la hora

de elegir ϵ: cuanto más elevada sea la probabilidad de escoger acciones aleatorias, peor es la calidad de la política con la que el agente interacciona con el entorno. Este efecto no se aprecia en la Figura 4.17b porque durante el proceso de evaluación de las políticas se detiene el aprendizaje y se fija $\epsilon = 0$. Sin embargo, si se evalúa la calidad de la interacción para una política fija y diferentes valores de ϵ, la pendiente de la curva es opuesta, tal y como se ilustra en la Figura 4.18. En este caso se ha evaluado una misma política para diferentes valores de ϵ. Se aprecia que, de acuerdo a este criterio y al contrario que con el criterio anterior, conviene reducir la probabilidad de escoger acciones exploratorias. Este es un ejemplo más del ya comentado dilema exploración-explotación (ver Sección 2.7).

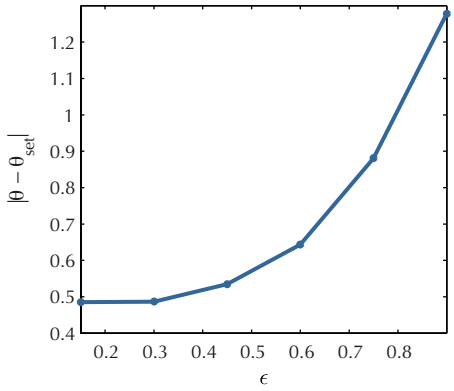


Figura 4.18
Efectos de variar la probabilidad ϵ de escoger una acción exploratoria en la calidad de la política empleada para interactuar con el entorno del motor DC. La política utilizada es aproximadamente óptima y permanece fija para todos los valores de ϵ.

4.5. Conclusiones

Una de las principales limitaciones que aparecen a la hora de aplicar las técnicas de RL en problemas reales es la gran cantidad de datos necesaria para aprender políticas óptimas. Los métodos clásicos se caracterizan por hacer un uso poco eficiente de los datos: las transiciones adquiridas por el agente habitualmente se utilizan una única vez para actualizar la política y después son desechadas. La reciente aparición de los algoritmos de tipo *batch* tiene como objetivo principal el desarrollo de métodos que sean más eficientes respecto al uso de los datos a consta de una mayor carga computacional. Sin embargo, prácticamente la totalidad algoritmos *batch* están centrados en el aprendizaje *offline*. En este capítulo se ha propuesto el algoritmo IVAO, caracterizado por hacer un uso eficiente de los datos y aprender *online*. Este algoritmo puede considerarse como un puente entre los algoritmos incrementales y los algoritmos *batch*,

ya que es posible configurarlo para que se comporte como cualquiera de ellos. Por lo tanto, proporciona las ventajas de ambos tipos de algoritmos, lo cual puede resultar muy útil en determinadas aplicaciones.

Por otra parte el algoritmo IVAO también arrastra algunos inconvenientes. Comparado con los algoritmos incrementales, la mayor limitación del algoritmo IVAO es su carga computacional. Aunque el número de operaciones que requiere depende en gran medida del método de aproximación, el hecho de procesar cada transición varias veces implica una carga computacional mayor. Por tanto, en las aplicaciones donde el tiempo de muestreo sea muy pequeño es más adecuado utilizar algoritmos incrementales como, por ejemplo, el algoritmo *Q-learning*. De forma similar, en el caso contrario de que no se requiera que el agente aprenda mientras interactúa con el entorno, posiblemente los algoritmos *batch* puros resultan más apropiados. Entre estos dos casos extremos existe un gran número de problemas donde el algoritmo IVAO obtiene mejores resultados.

El algoritmo propuesto se ha evaluado empíricamente de forma extensiva en tres entornos con diferente grado de complejidad. Como método de referencia se ha empleado el algoritmo *Q-learning* con trazas de elegibilidad. En todos los casos los resultados sugieren que el algoritmo IVAO es capaz de extraer más conocimiento de los datos adquiridos por el agente, mostrando en los tres entornos una velocidad de convergencia notablemente mayor. Durante los experimentos ambos algoritmos usaron el mismo aproximador de funciones consistente en una red de RBFs con base fija. Sin embargo, conviene tener en cuenta que, a diferencia de *Q-learning*, el algoritmo IVAO no requiere que el aproximador sea de tipo paramétrico, sino que permite el uso de un mayor rango de aproximadores.

Siguiendo con la flexibilidad que ofrece el algoritmo IVAO respecto al tipo de aproximadores con el que se puede combinar, una línea clara trabajo futuro incluye experimentar con otros aproximadores. Por una parte resulta interesante investigar las mejoras que puede aportar el uso de aproximadores más potentes que las redes RBFs. También sería conveniente, al menos desde el punto de vista teórico, evaluar el funcionamiento del algoritmo IVAO con aproximadores que realicen un mapeado no expansivo, ya que aseguran la convergencia hacia políticas óptimas. Una segunda modificación que puede mejorar el funcionamiento del algoritmo IVAO consiste en implementar una estrategia más "inteligente" para almacenar las transiciones. En la versión actual del algoritmo se guardan las transiciones más recientes, sin embargo sería más útil seleccionar únicamente aquellas transiciones que aporten información relevante para estimar la función Q. Esta aproximación combina técnicas de aprendizaje activo con RL, algunos resultados preliminares basados en este principio pueden encontrarse en (Ernst, 2005; Rachelson et al., 2011).

Capítulo 5

Least-squares temporal-difference basado en máquinas de aprendizaje extremo

5.1. Introducción

El presente capítulo se centra en el problema de estimar la función valor V dado un MDP y una política π fija. A este tipo de problemas se le conoce con el nombre de predicción de la función valor o evaluación de la política, y aparecen, por ejemplo, cuando se quiere estimar la probabilidad de un evento futuro o el tiempo esperado hasta que un determinado evento ocurra. Algunas aplicaciones prácticas son la planificación del despliegue de grandes redes de telecomunicaciones (Frank et al., 2008), estimación del *taxi-out* (tiempo que transcurre entre que un avión abandona la puerta de embarque y el momento en que despega) en función de las condiciones del espacio aéreo (Balakrishna et al., 2008, 2010), o la evaluación de diferentes posiciones de los jugadores en el tablero del juego de mesa *Go* (Silver et al., 2007). Además de las situaciones mencionadas, la estimación de una función valor correspondiente a una política fija es un problema que aparece en prácticamente todos los algoritmos de RL basados en iteración de políticas (Sutton y Barto, 1998). Se trata, por tanto, de un problema de elevado interés.

De acuerdo con los conceptos introducidos en el Capítulo 3, cuando el MDP está formado por un conjunto discreto de estados suficientemente pequeño, su función valor se puede almacenar de forma exacta mediante tablas. Sin embargo, en general,

el espacio de estados puede ser continuo y se hace necesario representar la función valor de forma aproximada. La aproximación de funciones valor difiere de los problemas típicos que aparecen en el aprendizaje supervisado debido a la ausencia de una señal deseada explícita (Wiering y van Otterlo, 2012). Aunque muchos de los conceptos del aprendizaje supervisado siguen siendo válidos para aproximar funciones valor, a menudo aparecen nuevas restricciones que se deben tener en cuenta. Una de las técnicas más populares para predecir funciones valor en espacios continuos recibe el nombre de aprendizaje *least squares temporal difference* (LSTD). El algoritmo LSTD fue inicialmente propuesto por Bradtke y Barto (1996), quienes utilizaron un enfoque basado en variables instrumentales (Ljung y Söderström, 1983; Söderström y Stoica, 1983) para probar su convergencia. Más tarde, Boyan propuso una derivación más sencilla e intuitiva al mismo tiempo que extendió el algoritmo LSTD con la incorporación de trazas de elegibilidad (Boyan, 1999, 2002). Desde entonces, las propiedades del algoritmo LSTD y sus variantes han sido estudiadas en diversas ocasiones (Geramifard et al., 2006a; Xu et al., 2007; Ghavamzadeh et al., 2010), especialmente tras el desarrollo del algoritmo *least squares policy iteration* (LSPI) (Lagoudakis y Parr, 2003), el cual adapta LSTD al caso de funciones Q y lo integra en un esquema de iteración de políticas.

Una propiedad fundamental del algoritmo LSTD es que asume que la función valor se representa mediante un aproximador lineal en los parámetros. Por tanto, la salida del aproximador se calcula mediante la suma ponderada de un conjunto de características. Existen varios factores que pueden afectar a la calidad final de la aproximación, pero uno de los más importantes es la selección del conjunto de características. De hecho, el diseño de las características es la etapa más crítica del algoritmo LSTD (Xu et al., 2007; Busoniu et al., 2010a; Lagoudakis y Parr, 2003). Algunos autores han propuesto algoritmos que adaptan automáticamente las características durante el proceso de aprendizaje (Munos y Moore, 2002; Ernst et al., 2005a; Mahadevan y Maggioni, 2007), sin embargo, se ha demostrado que cambiar las propiedades del espacio de características mientras se estima la función valor puede repercutir negativamente en la convergencia del algoritmo (Busoniu et al., 2011). Cuando se dispone de suficiente conocimiento *a priori* de la función valor que se quiere predecir, una opción adecuada consiste en definir manualmente características *ad-hoc*. En la práctica, raramente se dispone de dicho conocimiento, por lo que es habitual emplear un conjunto de características que realizan una partición regular del espacio de entrada, usando, por ejemplo, codificación en baldosas (Sutton, 1995) o funciones RBF con base fija (Busoniu et al., 2010a). Estos métodos suelen ser *aproximadores locales*, es decir, cuando sus parámetros se actualizan debido a un cambio en una determinada región del espacio de entrada, únicamente se ve afectada una parte limitada del espacio de salida. Por el contrario, se dice que un *aproximador es de carácter global* cuando un cambio en el espacio de entrada puede afectar a todo el espacio de salida. Algunos ejemplos de aproximadores globales son el perceptrón multicapa o las máquinas de vectores soporte. Tal y como se describe en secciones posteriores, una limitación potencial de los aproximadores locales frente a los globales es que el número de característi-

cas requeridas para alcanzar una determinada exactitud aumenta rápidamente con la dimensionalidad de las entradas debido a la "maldición de la dimensionalidad". Un mayor número de características no solo incrementa los requisitos de memoria y el tiempo de computación, sino que requiere un mayor número de datos para asegurar que los parámetros el aproximador se determinan correctamente (Bishop, 1995).

En el ámbito del RL, el uso de aproximadores locales va más allá del algoritmo LSTD. Algunos de los éxitos más notables logrados mediante técnicas de RL están basados en aproximadores globales como, por ejemplo, el programa desarrollado por Gerry Tesauro para competir en el juego de mesa *Backgammon* (Tesauro, 1995). A pesar de ello, también se han reportado resultados negativos en diversas aplicaciones que combinan RL y aproximadores globales (Boyan y Moore, 1995). Los problemas de dicha combinación surgen en gran medida debido al aprendizaje *online* en el que se basan muchos algoritmos de RL (Sutton y Whitehead, 1993). Cuando la función valor se aprende mientras el agente interactúa con el entorno, cada vez que el agente realiza una acción y recibe la recompensa correspondiente, se actualizan los parámetros del aproximador. En el caso de emplear un aproximador global, la actualización hecha para estimar el valor de un determinado estado puede repercutir arbitrariamente en las estimaciones de otros estados. La consecuencia de este comportamiento es que el algoritmo puede converger de forma extremadamente lenta o incluso diverger. En el caso del algoritmo LSTD, los datos obtenidos de la interacción agente-entorno son recogidos previamente y procesados de forma *offline*; estimando la función valor para todos los estados al mismo tiempo. Así pues, el algoritmo LSTD es un método adecuado para ser combinado con aproximadores globales.

Este capítulo propone el uso de máquinas de aprendizaje extremo (ELM, *extreme learning machine*) junto con el algoritmo LSTD para reducir las limitaciones de los aproximadores locales en problemas de elevada dimensionalidad. ELM es un algoritmo propuesto por Huang et al. (2006) para entrenar redes neuronales de una sola capa oculta alimentadas hacia delante, más conocidas por su siglas en inglés SLFNs (*single-hidden layer feedforward neural networks*). El principio de funcionamiento de ELM se basa en asignar aleatoriamente los pesos de la capa oculta y optimizar únicamente los pesos de la capa de salida mediante mínimos cuadrados. Este proceso puede ser visto como un mapeado de las entradas a un espacio de características definido por los nodos de la capa oculta, y una combinación de dichas características ponderadas por los pesos de la capa de salida. Como se introducirá a lo largo del capítulo, el algoritmo ELM puede ser integrado en LSTD para resolver problemas de predicción de funciones valor.

El resto del capítulo se ha organizado como sigue. En la Sección 5.2 se analizan las limitaciones de los aproximadores locales en espacios con alta dimensionalidad. La base teórica de las máquinas de aprendizaje extremo y del algoritmo LSTD se introduce en las Secciones 5.3 y 5.4, respectivamente. En la Sección 5.5 se describen los detalles del algoritmo propuesto. Los experimentos llevados a cabo para evaluar dicho algoritmo se detallan en la Sección 5.6. La Sección 5.7 presenta y discute los

resultados obtenidos y, finalmente, la conclusiones del capítulo se encuentran en la Sección 5.8.

5.2. Aproximadores locales en espacios de alta dimensionalidad

En la sección anterior se han expuesto algunos de los motivos que justifican el uso de aproximadores locales en el ámbito del RL. Este tipo de aproximadores suelen ser intuitivos y proporcionan resultados satisfactorios en espacios de baja dimensionalidad. Sin embargo, debido al efecto conocido como "maldición de la dimensionalidad" (Bellman, 1957), dejan de ser efectivos cuando aumenta el número de dimensiones. En problemas de RL con un cierto grado de complejidad es habitual que el espacio de estados contenga decenas de variables. Como se discutirá a continuación, aproximar funciones con un elevado número de variables de entrada conlleva una serie de dificultades que conviene tener en cuenta a la hora de elegir el aproximador más adecuado.

Supongamos que se desea aproximar una función a partir de un conjunto de entrenamiento[1] formado por n datos de entrada d-dimensionales y una salida unidimensional. Un hipotético método de aproximación local muy sencillo consiste en dividir el espacio de entrada en hipercubos regulares de d dimensiones. Cuando llega un nuevo dato de test para el que se quiere predecir la salida, primero se halla el hipercubo al que pertenece el dato, y después se calcula el valor de salida promediando las salidas de todos los datos de entrenamiento que caen en el mismo hipercubo. Intuitivamente, si el espacio de entrada se ha dividido en un número suficientemente grande de hipercubos, esta técnica proporcionará una buena aproximación de la función.

Existen varias limitaciones con este método simplista, pero el más importante aparece cuando el número de variables de entrada es elevado. En la Figura 5.1 se ilustra que ocurre al dividir el espacio de entrada en hipercubos conforme el número de dimensiones aumenta. Como se puede observar, la cantidad de hipercubos necesarios para cubrir el espacio crece exponencialmente con la dimensionalidad. El problema con un número de regiones exponencialmente grande es que también es necesario un número de datos del mismo orden para asegurar que ningún hipercubo está vacío. Si se divide cada dimensión en 10 regiones, para una función con 10 variables de entrada haría falta 10 mil millones de datos, lo cual pone de manifiesto la inviabilidad del método (Bishop, 1995).

La intuición geométrica que desarrollamos día tras día en un mundo tridimensional puede fallar estrepitosamente cuando intentamos trasladarla a espacios con un mayor número de dimensiones. Consideremos un ejemplo sencillo consistente en una esfera

[1] Por cuestiones de sencillez se ha utilizado un ejemplo típico de aprendizaje supervisado; sin embargo, las conclusiones obtenidas son igualmente válidas para el problema de aproximar funciones valor en RL.

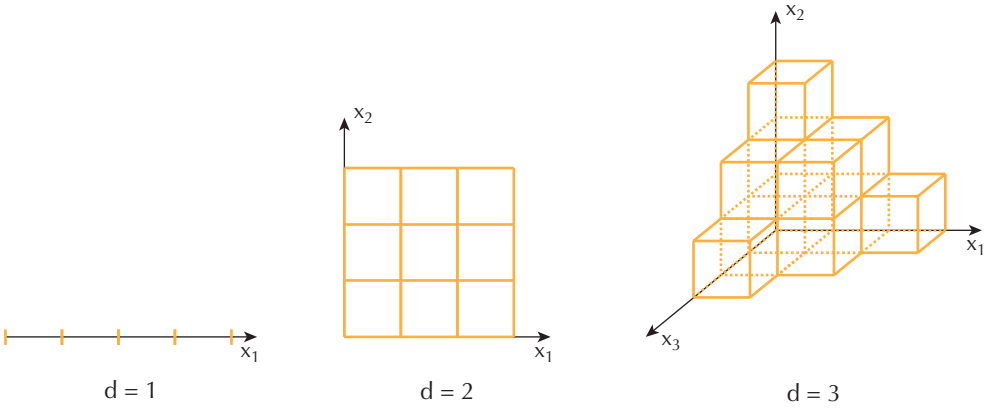


Figura 5.1
Ilustración de la "maldición de la dimensionalidad". El número de regiones o hipercubos regulares necesarios para cubrir un espacio aumenta exponencialmente con la dimensionalidad d de dicho espacio. Por claridad, solo se ha dibujado un subconjunto de las regiones en el caso de $d = 3$.

de radio $r = 1$ en un espacio de d dimensiones. Si nos preguntamos qué fracción del volumen de la esfera se encuentra entre el radio $r = 1 - \epsilon$ y $r = 1$, posiblemente nuestra intuición sea correcta únicamente cuando el número de dimensiones es bajo. La fracción de volumen se puede calcular mediante la expresión (Bishop, 2007):

$$\frac{V_d(1) - V_d(1-\epsilon)}{V_d(1)} = 1 - (1-\epsilon)^d \tag{5.1}$$

donde $V_d(r)$ es el volumen de la esfera d-dimensional de radio r. En la Figura 5.2 se ha representado dicha fracción en función de ϵ para diferentes dimensionalidades. Se puede observar que, para valores grandes de d, la fracción de volumen entre ambas esferas tiende a 1 incluso cuando el parámetro ϵ es pequeño. Es decir, el volumen de una esfera en un espacio de elevada dimensionalidad se concentra en una capa estrecha cercana a la superficie.

El hecho de que la mayoría de las muestras dentro de un espacio de alta dimensionalidad se concentre cerca de los límites supone una dificultad añadida a la hora aproximar una función. En los límites, no hay suficientes datos para interpolar entre ellos, por lo que el aproximador debe extrapolar a partir de datos cercanos, lo cual siempre es una tarea más complicada (Hastie et al., 2009).

Otro ejemplo más directamente relacionado con la aproximación de funciones es el comportamiento de una distribución gaussiana es un espacio multidimensional. La densidad de probabilidad $p(r)$ en función del radio desde el origen viene determinada

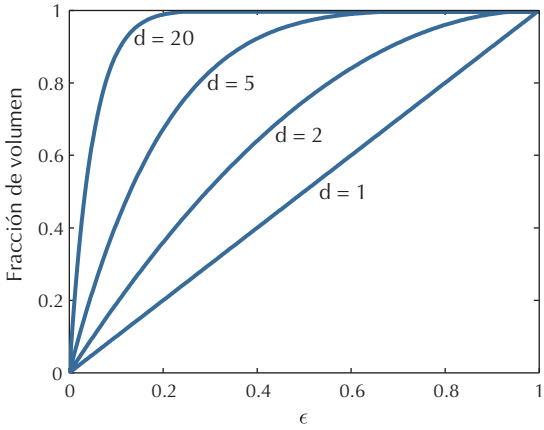


Figura 5.2
Representación de la fracción del volumen de una esfera que cae en el rango de radios entre $r = 1 - \epsilon$ y $r = 1$ para diferentes dimensiones d.

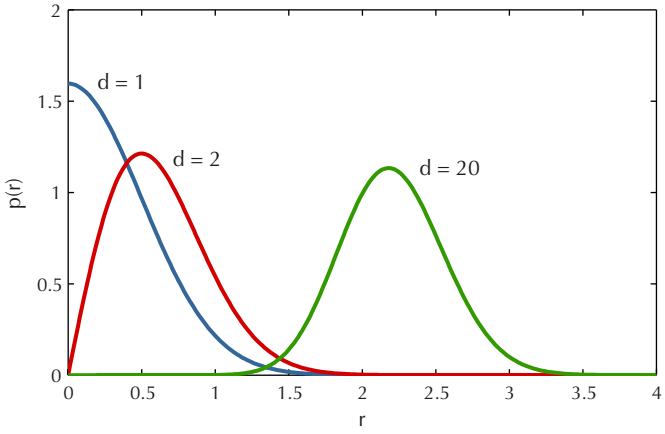


Figura 5.3
Densidad de probabilidad en función del radio r correspondiente a una distribución gaussiana con $\sigma = 0.5$ para varias dimensiones d. Cuando la dimensionalidad es elevada, la mayor parte de la probabilidad se concentra en una capa estrecha y alejada del origen.

por (Bishop, 2007):

$$p(r) = \frac{S_d r^{d-1}}{(2\pi\sigma^2)^{d/2}} \exp(-\frac{r^2}{2\sigma^2}) \tag{5.2}$$

donde S_d es el área de la superficie de una esfera unitaria d-dimensional. En la Figura 5.3 se muestra $p(r)$ frente al valor del radio r para diferentes valores de d y $\sigma = 0.5$. Como se puede observar, cuando la dimensionalidad es elevada, la densidad de probabilidad de la gaussiana se concentra en una capa estrecha y alejada del origen. De nuevo, este resultado es difícil de adivinar a partir de la intuición desarrollada en espacios tridimensionales.

En la práctica, es posible que los efectos de la dimensionalidad no sean tan severos como los presentados en los ejemplos anteriores. Por una parte, la variables de entrada suelen tener algún tipo de correlación, de forma que los datos no se reparten por todo el espacio, sino que tienden a estar restringidos en un subespacio de menor dimensionalidad. Esto da lugar al concepto conocido como dimensionalidad intrínseca, el cual queda fuera del alcance de esta tesis. Por otra parte, el valor de la variable de salida normalmente no cambia arbitrariamente de una región del espacio de entrada a otra, sino que exhibe cierto grado de regularidad, al menos de carácter local. Ambas características, dimensionalidad intrínseca y regularidad, pueden ser aprovechadas por los métodos de aproximación para mejorar los resultados que obtienen. Sin embargo, es importante tener claro que la intuición suele fallar en espacios de alta dimensionalidad y que los aproximadores locales no son adecuados debido a la "maldición de la dimensionalidad".

5.3. Máquinas de aprendizaje extremo (ELM)

Las redes neuronales artificiales se utilizan de forma habitual en diversos campos debido a su habilidad para aproximar funciones altamente no lineales y modelar toda clase de fenómenos complejos en los que es complicado aplicar otras técnicas clásicas. El teorema de aproximación universal establece que una red neuronal de una sola capa oculta y un número finito de nodos puede aproximar cualquier función continua con una exactitud arbitraria (Hornik et al., 1989; Hornik, 1991). Tradicionalmente, se necesita ajustar todos los parámetros del modelo neuronal y existe una relación de dependencia entre los parámetros de las diferentes capas. Prácticamente la totalidad de los algoritmos destinados a optimizar dichos parámetros están basados en métodos iterativos, mayoritariamente en técnicas de descenso por gradiente. Aunque estos métodos ofrecen buenos resultados, el tiempo requerido para calcular los parámetros óptimos suele ser elevado (Huang et al., 2006).

Las máquinas de aprendizaje extremo, o ELM, son redes neuronales de tipo SLFN cuyos parámetros se han determinado mediante el algoritmo ELM. Dicho algoritmo, propuesto por Huang et al. (2006), se caracteriza por calcular los parámetros de forma analítica, a diferencia de las aproximaciones estándar basadas en métodos iterativos.

Se ha demostrado que, cuando se asignan aleatoriamente los pesos y sesgos de la capa oculta en una red SLFN con M nodos, la red es capaz de aprender M observaciones distintas (Tamura y Tateishi, 1997; Huang, 2003). Esto implica que no es necesario ajustar todos los parámetros de la red, sino que parte de ellos se pueden fijar de forma aleatoria. El algoritmo ELM se basa en este principio para ajustar únicamente los parámetros de la capa de salida. El resultado es que el proceso de entrenamiento queda reducido a la resolución de un sistema de ecuaciones lineales, lo cual resulta extremadamente rápido comparado con los métodos de entrenamiento iterativos. Desde su aparición, el número de aplicaciones de las máquinas de aprendizaje extremo no ha dejado de crecer, demostrando en numerosos problemas reales que son capaces de obtener soluciones similares a las de otras redes neuronales pero con una velocidad de aprendizaje mucho mayor (Yeu et al., 2006; Zong y Huang, 2011; Wu et al., 2013; Cao et al., 2013).

Dado un conjunto $\mathcal{D}$ de N patrones u observaciones, $\mathcal{D} = (\mathbf{x}_i, \mathbf{o}_i); i = 1 \dots N$, donde $\{\mathbf{x_i}\} \in \mathbb{R}^{d_1}$ y $\{\mathbf{o_i}\} \in \mathbb{R}^{d_2}$, el objetivo del algoritmo ELM es encontrar una relación entre las variables independientes $\{\mathbf{x_i}\}$ y las dependientes $\{\mathbf{o_i}\}$. Por cuestiones de simplicidad nos centraremos en el caso de aproximar funciones con un única salida, por lo tanto $d_2 = 1$. La salida de una red SLFN con M nodos ocultos para el j-ésimo patrón viene dada por:

$$y_j = \sum_{l=1}^{M} h_l \cdot f\left(\mathbf{w}_l \cdot \mathbf{x}_j + b_l\right) \tag{5.3}$$

donde $1 \leq j \leq N$, $\mathbf{w}_l = [w_{l1}, w_{l2}, \dots, w_{ld_1}]^\top$ es el vector de pesos que conecta el l-ésimo nodo oculto con los nodos de entrada, b_l es el sesgo del nodo oculto l, h_l es el peso que conecta el nodo oculto l con el nodo de salida, y $f(x)$ es la función de activación de la capa oculta. $\mathbf{w}_l \cdot \mathbf{x}_j$ denota el producto escalar entre $\mathbf{w}_l$ y $\mathbf{x}_j$. La función de activación de la capa de salida típicamente se escoge lineal para problemas de regresión. En la Figura 5.4 se muestra la estructura de la red SLFN empleada por el algoritmo ELM y de una de las neuronas.

La Ecuación (5.3) se puede expresar de forma compacta con notación matricial como:

$$\mathbf{y} = \mathbf{G} \cdot \mathbf{h}, \tag{5.4}$$

donde

$$\mathbf{G} = \begin{pmatrix} f\left(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1\right) & \dots & f\left(\mathbf{w}_M \cdot \mathbf{x}_1 + b_M\right) \\ \vdots & \ddots & \vdots \\ f\left(\mathbf{w}_1 \cdot \mathbf{x}_N + b1\right) & \cdots & f\left(\mathbf{w}_M \cdot \mathbf{x}_N + b_M\right) \end{pmatrix}_{N \times M}, \tag{5.5}$$

$$\mathbf{h} = \begin{pmatrix} h_1 \\ \vdots \\ h_M \end{pmatrix}_{M \times 1}, \text{y} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}_{N \times 1} \tag{5.6}$$

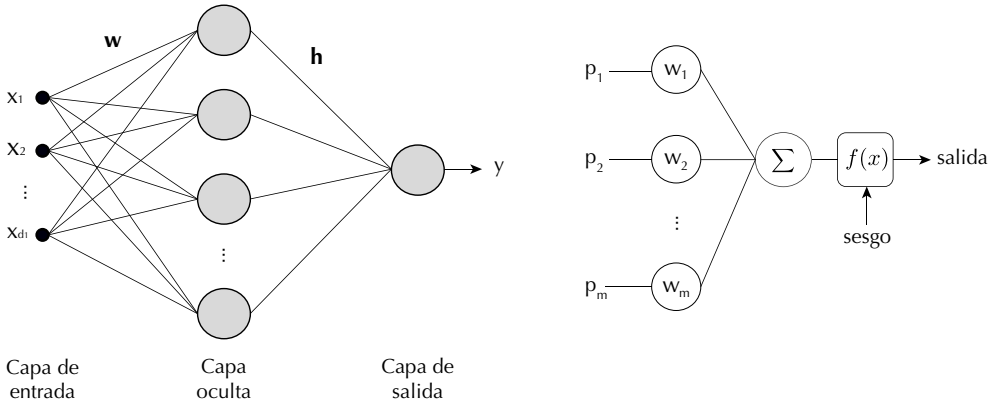


Figura 5.4
Estructura de la red neuronal SFNL empleada en las máquinas de aprendizaje extremo (izquierda) y de una de las neuronas artificiales (derecha).

En la matriz $\mathbf{G}$, que suele denominarse matriz de salida de la capa oculta, la l-ésima columna se corresponde con la salida del nodo oculto l respecto a las entradas $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$.

De acuerdo con la teoría de las máquinas de aprendizaje extremo, tanto los pesos que conectan la capa de entrada con la capa oculta, $\mathbf{w}$, como el sesgo de los nodos ocultos, $\mathbf{b}$, se pueden asignar aleatoriamente. Una vez asignados, la matriz $\mathbf{G}$ permanece fija y únicamente es necesario optimizar la pesos de la capa de salida, $\mathbf{h}$. Así pues, todo el proceso de entrenamiento de la red queda prácticamente reducido al cálculo del vector $\mathbf{h}$, el cual es la solución del sistema de ecuaciones lineales definido por $\mathbf{o} = \mathbf{G} \cdot \mathbf{h}$. Como el número de nodos de la capa oculta suele ser mucho menor que el número de patrones del conjunto de datos, $M \ll N$, $\mathbf{G}$ no es una matriz cuadrada y puede que no exista una solución exacta de dicho sistema lineal. Aún así, es posible obtener una solución aproximada calculando $\mathbf{h}$ como

$$\widehat{\mathbf{h}} = \mathbf{G}^{\dagger} \cdot \mathbf{o} \tag{5.7}$$

donde $\mathbf{G}^{\dagger} = \left(\mathbf{G}^{\top} \cdot \mathbf{G}\right)^{-1} \cdot \mathbf{G}^{\top}$ es la matriz inversa generalizada de Moore-Penrose (Rao y Mitra, 1972).

Resumiendo, el algoritmo ELM consta de los siguientes pasos:

1. Inicializar aletoriamente los pesos $\mathbf{w}$ y segos $\boldsymbol{b}$ de la capa oculta.
2. Calcular la matriz de salida de la capa oculta, $\mathbf{G}$.
3. Calcular los pesos $\mathbf{h}$ de la capa de salida mediante la matriz inversa generalizada de acuerdo a la expresión
$$\mathbf{h} = \mathbf{G}^{\dagger} \cdot \mathbf{o}$$

Este algoritmo es válido para cualquier función de activación $f(x)$ siempre que sea infinitamente diferenciable. Tales funciones incluyen las de base radial, sigmoidal, seno, coseno o exponencial, entre otras. El único requisito respecto al número de nodos de la capa oculta es que debe ser menor o igual que el número de patrones de entrenamiento (diferentes), es decir, $M \leq N$. Igual que ocurre con otros métodos basados en redes neuronales, el tamaño de la capa oculta se debe escoger en función del problema que se quiere resolver. Si la red tiene pocos nodos, es posible que no sea capaz de modelar los datos adecuadamente. Por el contrario, si tiene demasiados nodos, aunque no supere el número máximo permitido, puede que se produzca un sobreajuste. El número óptimo de nodos no se puede conocer *a priori*, debido a ello se han propuesto diversos métodos para optimizar automáticamente el tamaño de la capa oculta durante el proceso de entrenamiento con ELM (Rong et al., 2008; Miche et al., 2010; Martínez-Martínez et al., 2011; Soria-Olivas et al., 2011). Sin embargo, la opción más habitual consiste en estimar el error de generalización para diferentes tamaños de la capa oculta y escoger aquel que obtenga mejores resultados.

Comparado con los métodos estándar (basados en descenso por gradiente) de entrenamiento de redes neuronales, el algoritmo ELM introduce las siguientes ventajas:

- Los algoritmos de descenso por gradiente dependen de un parámetro conocido como tasa de aprendizaje. Valores demasiado pequeños de la tasa de aprendizaje provocan una convergencia lenta mientras que, si son demasiado elevados, el algoritmo se puede volver inestable y diverger, por lo que debe ser seleccionado cuidadosamente; ELM elimina dicho parámetro.

- En ELM, el cálculo de los pesos de la capa de salida se formula como un problema de optimización convexo, es decir, sin mínimos locales. En otros algoritmos, la función de error suele ser multimodal, por lo que la búsqueda basada en descenso por gradiente puede detenerse al caer en un mínimo local.

- El proceso de entrenamiento iterativo puede dar lugar a un sobreentrenamiento de los parámetros si no se detiene a tiempo. Debido a ello se hace necesario emplear alguna técnica de parada como, por ejemplo, *early stopping* (Haykin, 2008).

- El algoritmo ELM es más sencillo de implementar y requiere una carga computacional mucho menor, lo que se traduce en tiempos de aprendizaje que pueden llegar a ser varios órdenes de magnitud menores.

5.3.1. Comité de máquinas de aprendizaje extremo

A pesar de la ventajas descritas en la sección anterior, el proceso de entrenamiento que plantea el algoritmo ELM también presenta algunos inconvenientes. Los parámetros de la capa oculta se asignan aleatoriamente y no se modifican durante el aprendizaje. Dado que el valor de dichos parámetros puede afectar al resultado de

la red, existe el riesgo de que, debido al azar, los parámetros sean fijados en valores subóptimos y la calidad de la aproximación sea baja. El uso de comités permite minimizar las posibilidad de que aparezca este problema.

Los comités, también conocidos como comités de expertos o métodos *ensemble*, básicamente consisten en la combinación de varios modelos con el objetivo de formar un único modelo que suponga una mejora respecto a los miembros del comité. La mejora puede estar relacionada con diversos aspectos, como la exactitud, robustez, estabilidad, etc. En principio, cualquier modelo es susceptible de ser combinado mediante un comité. Una condición deseable de los miembros es que exista cierto grado de diversidad entre ellos, es decir, el error cometido por cada miembro debe ser independiente. El motivo es evidente, ya que si, por ejemplo, se combinan varios modelos que fallan al aproximar una función en la misma región del espacio de entrada, la combinación resultante (o comité) seguirá produciendo el mismo fallo. Algunos métodos basados en comités de uso común son los algoritmos *tree bagging* (Breiman, 1996) y *random forest* (Breiman, 2001). Ambos algoritmos utilizan árboles de regresión/clasificación como miembros del comité, pero se diferencian tanto en la forma de generar los árboles como en la técnica que emplean para combinarlos.

El uso de comités en el contexto de las máquinas de aprendizaje extremo ha sido estudiado por varios autores (Lan et al., 2009; Lu et al., 2014; Wang y Alhamdoosh, 2013). El procedimiento habitual para generar el comité consiste en utilizar un conjunto de redes ELM con la misma arquitectura y función de activación, pero cuyos parámetros se han inicializado de forma independiente. A la hora de combinar los miembros se han planteado diversos enfoques, como *bootstrap aggregating* (Tian y Meng, 2010), *AdaBoost* (Tian y Mao, 2010), algoritmos evolutivos (Wang y Alhamdoosh, 2013) o utilizando la entropía (Zhai et al., 2012). La mayoría de estas técnicas emplean las salidas de los miembros para optimizar la forma de combinarlos. Sin embargo, cuando el comité se combina con el algoritmo LSTD para aproximar funciones valor (ver Sección 5.5), dichas salidas no están disponibles. Una opción viable en dicho caso consiste en calcular la salida del comité promediando las salidas de cada red ELM.

En la Figura 5.5 se muestra la estructura típica de un comité basado en redes ELM. Todas las redes emplean la misma función de activación y tienen el mismo número de nodos en la capa oculta. La mayoría de modelos ELM serán capaces de aproximar la función valor con una exactitud razonable en prácticamente todo el espacio de entrada. Aún así, es probable que aparezcan pequeñas desviaciones en determinadas zonas que variarán entre las distintas redes en función de los parámetros fijados aleatoriamente. También es posible que en ciertos casos puntuales los parámetros de una red sean subóptimos y la exactitud de la aproximación sea baja. Al combinar los resultados de varias redes se minimizan tanto las pequeñas desviaciones de cada red como el efecto debido a los posibles parámetros subóptimos de alguna de las redes. Los comités empleados en los experimentos de las secciones posteriores utilizan la mediana para combinar las salidas de cada miembro. Se ha elegido esta forma de calcular el promedio debido a que, ante la presencia de valores atípicos, proporciona una estimación del

valor central más robusta que la media.

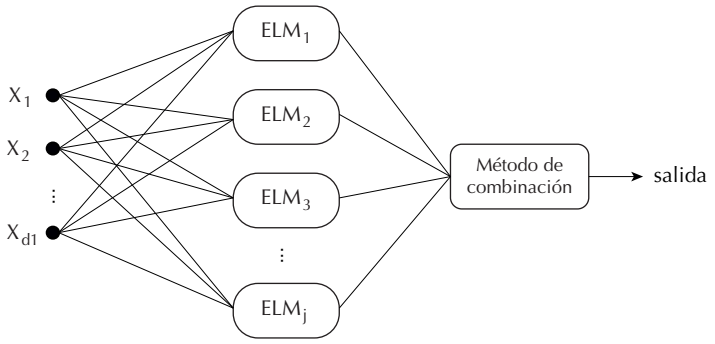


Figura 5.5
Estructura de un comité basado en redes ELM.

5.4. Predicción de la función valor mediante LSTD

Existen diversos métodos para predecir funciones valor, pero los métodos basados en diferencias temporales (TD, *temporal difference*) (Sutton, 1988) son probablemente los más estudiados y extendidos. Aunque el aprendizaje TD se ha introducido previamente en la Sección 2.7.1, aquí se repasa brevemente debido a que es una parte fundamental del algoritmo LSTD. El aprendizaje TD puede considerarse una clase de procedimientos incrementales especializados en problemas de predicción. La característica más representativa de estos métodos es que utilizan lo que se conoce como *bootstrapping*: la estimación de la función que se quiere predecir se actualiza basándose en otras estimaciones (Sutton y Barto, 1998). La implementación de los métodos TD da lugar de forma natural a algoritmos *online* e incrementales.

Tal y como se comentó en la introducción, el presente capítulo se centra en el problema de predecir funciones valor en MDPs con espacios de estados continuos, en los cuales es necesario utilizar aproximadores. Asumiendo un aproximador lineal en los parámetros, se puede considerar que el valor del estado s bajo la política π, $V^\pi(s)$, se aproxima primero mapeando s a un vector de características $\boldsymbol{\phi}(s)$ y después combinando esas características mediante un vector de coeficientes $\boldsymbol{\theta}$, denotado por $V^\pi_{\boldsymbol{\theta}}(s)$. Entonces, para cada estado observado por el agente, TD ajusta los coeficientes de $V^\pi_{\boldsymbol{\theta}}(s)$ incrementalmente hacia los nuevos valores de la función objetivo. Concretamente, si $V^\pi_{\boldsymbol{\theta}_k}(s)$ denota la estimación del estado s en el instante k, en la iteración k-ésima TD realiza las siguientes operaciones (Sutton, 1988):

$$\begin{aligned} \delta_{k+1} &= r_k + \gamma V^\pi_{\boldsymbol{\theta}_k}(s_{k+1}) - V^\pi_{\boldsymbol{\theta}_k}(s_k) \\ \boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k + \alpha_k \delta_{k+1} \end{aligned} \tag{5.8}$$

donde r_{k+1} es la recompensa observada, γ es el factor de descuento, y α_k es la secuencia de tasas de aprendizaje. Se ha demostrado que TD converge (bajo ciertas condiciones técnicas) hacia una buena aproximación de V^π conforme el número de estados observados aumenta (Sutton, 1988). Analizando la Ecuación (5.8) se puede apreciar que, tras observar una trayectoria de estados $(s_0, s_1, \ldots, s_L)$, los cambios hechos por la regla de actualización TD tienen la forma:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_k(\mathbf{d} + \mathbf{C}\boldsymbol{\theta} + \boldsymbol{\omega}), \tag{5.9}$$

siendo

$$\begin{aligned} \mathbf{d} &= \mathrm{E}\left\{ \sum_{i=0}^{L} \boldsymbol{\phi}(s_i) r_i \right\}, \\ \mathbf{C} &= \mathrm{E}\left\{ \sum_{i=0}^{L} \boldsymbol{\phi}(s_i)(\gamma \boldsymbol{\phi}(s_{i+1}) - \boldsymbol{\phi}(s_i))^\top \right\}; \end{aligned} \tag{5.10}$$

y $\boldsymbol{\omega}$ representa una señal aleatoria de media cero (Boyan, 2002). De acuerdo con Bertsekas y Tsitsiklis (1996), el vector de coeficientes $\boldsymbol{\theta}$ converge a un punto fijo $\boldsymbol{\theta}_{\mathrm{TD}}$ que satisface la ecuación:

$$\mathbf{d} + \mathbf{C}\boldsymbol{\theta}_{\mathrm{TD}} = \mathbf{0} \tag{5.11}$$

De hecho, TD resuelve el sistema de ecuaciones definido por (5.11) empleando un método similar al descenso por gradiente. Las matrices $\mathbf{d}$ y $\mathbf{C}$ nunca se representan explícitamente, sino que los cambios realizados en $\boldsymbol{\theta}$ solo dependen de los estados y recompensas más recientes. Una vez actualizado el vector de coeficientes, las observaciones son desechadas. Este método requiere pocos cálculos por iteración, pero hace un uso poco eficiente de los datos (interacciones agente-estado observadas) y suele necesitar un elevado número de estados y recompensas para alcanzar la convergencia.

El algoritmo LSTD propone una solución alternativa para predecir la función valor. LSTD también converge al mismo vector de coefficientes $\boldsymbol{\theta}_{\mathrm{TD}}$, pero en lugar de descenso por gradiente, construye una estimación explícita de la matriz $\mathbf{C}$ y el vector $\mathbf{d}$, para después resolver directamente la Ecuación (5.11). A partir de los estados y recompensas observados, LSTD calcula la matriz $\mathbf{A}$ (de tamaño $n \times n$, donde n es el número de características) y del vector $\mathbf{b}$ (de tamaño n) como (Boyan, 2002):

$$\begin{aligned} \mathbf{A} &= \sum_{i=0}^{L} \boldsymbol{\phi}(s_i)(\gamma \boldsymbol{\phi}(s_{i+1}) - \boldsymbol{\phi}(s_i))^\top, \\ \mathbf{b} &= \sum_{i=0}^{L} \boldsymbol{\phi}(s_i) r_i \end{aligned} \tag{5.12}$$

Después de m trayectorias independientes, $\mathbf{A}$ y $\mathbf{b}$ son estimaciones insesgadas de $m\mathbf{d}$ y $-m\mathbf{C}$ respectivamente. Dado que m es una constante, $\boldsymbol{\theta}_{\mathrm{TD}}$ se puede obtener simplemente como $\boldsymbol{\theta}_{\mathrm{TD}} = \mathbf{A}^{-1}\mathbf{b}$. Cabe mencionar que, por cuestiones de simplicidad,

el capítulo se limita al algoritmo LSTD básico. Sin embargo, este método se puede extender fácilmente con el uso de trazas de elegibilidad, dando lugar a un algoritmo más general y con mayor velocidad de convergencia, denominado LSTD(λ).

Los métodos TD realizan pequeñas actualizaciones del vector de coeficientes siguiendo una señal de gradiente estocástico (Szepesvári, 2010). En dicho proceso, la elección de la tasa de aprendizaje α_k puede ser crucial para obtener unos resultados adecuados. Por otra parte, estos métodos también son sensibles al valor inicial del vector de coeficientes $\boldsymbol{\theta}$. El algoritmo LSTD elimina estas dependencias además de aportar otras ventajas, las cuales de resumen a continuación:

- Desde un punto de vista estadístico, el algoritmo LSTD es más eficiente debido a que extrae más información de cada observación y, por tanto, el número de observaciones (estados y recompensas) necesario para converger es menor.
- La convergencia de TD puede ser extremadamente lenta si la tasa de aprendizaje no se escoge adecuadamente, mientras que el algoritmo LSTD elimina la dependencia con dicho parámetro.
- La inicialización del vector de coeficientes $\boldsymbol{\theta}$ puede afectar al funcionamiento de TD, al contrario de lo que ocurre en el algoritmo LSTD.
- En general, el algoritmo LSTD es más estable y alcanza una aproximación más cercana a $\boldsymbol{\theta}_{\text{TD}}$ que TD.

Como contrapartida a estas ventajas, la complejidad computacional de LSTD es considerablemente mayor. En concreto, para una secuencia de L observaciones, la complejidad de una implementación directa de LSTD es de $\mathcal{O}(Ln^2 + n^3)$, siendo n la dimensionalidad del vector de caracterísctcias. Mientras que los métodos incrementales TD solo requieren $\mathcal{O}(Ln)$, o incluso menos si se selecciona un espacio de características disperso (Szepesvári, 2010).

La implementación directa del algoritmo LSTD requiere invertir la matriz $\mathbf{A}$. En determinadas aplicaciones donde el coste computacional es un factor crítico, o se requiere realizar la inversión de $\mathbf{A}$ frecuentemente, una opción más adecuada es utilizar la versión recursiva de LSTD, conocida como RLSTD (Bradtke y Barto, 1996). RLSTD usa la fórmula de Sherman-Morrison (Sherman y Morrison, 1950) para evitar la inversión de la matriz $\mathbf{A}$, lo cual permite reducir la complejidad del algoritmo hasta $\mathcal{O}(Ln^2)$. A pesar de ello, TD sigue siendo más eficiente en cuanto al número de operaciones por iteración. Una comparación más detallada entre los algoritmos TD y LSTD puede encontrarse en Szepesvári (2010).

5.5. LSTD basado en ELM

La idea principal del método propuesto en este capítulo consiste en la combinación del algoritmo LSTD y el proceso de entrenamiento empleado en las redes ELM. Esta

combinación, como se demostrará en secciones posteriores, proporciona una serie de ventajas en la tarea de aproximar funciones valor. Un requisito fundamental en el algoritmo LSTD es que la función valor se debe aproximar mediante una arquitectura lineal; entre las técnicas que cumplen esta propiedad se encuentran, por ejemplo, las redes formadas por funciones de base radial con base fija o la regresión polinomial multidimensional. En cambio, muchas otras técnicas son incompatibles con el algoritmo LSTD. Entre ellas se encuentran las redes neuronales artificiales (RNAs).

La excelente capacidad de las RNAs para modelar funciones las han convertido en uno de los métodos de regresión más populares. Se han aplicado en multitud de campos, incluyendo el aprendizaje por refuerzo (Bertsekas y Tsitsiklis, 1996). El procedimiento más habitual para aproximar la función valor de un MDP con una RNA consiste en actualizar los pesos de la red mediante la regla TD. Cabe mencionar que la regla TD mostrada en la Ecuación (5.8) se trata de una simplificación para el caso de aproximadores lineales en los parámetros. En el caso general, TD puede combinarse con el algoritmo *backpropagation* fácilmente. A pesar de los beneficios potenciales que pueden aportar las RNAs, su uso práctico junto con TD presenta varias dificultades.

Las RNAs son aproximadores globales, cualidad que le aporta una alta capacidad de generalización y que resulta beneficiosa para aproximar funciones en espacios de alta dimensionalidad. Sin embargo, en la literatura del RL, diversos autores recomiendan el uso de aproximadores locales (Anderson, 1993; Thrun y Schwartz, 1993; Smart y Kaelbling, 2000). La principal motivación es evitar un fenómeno conocido como *interferencia* (Farrell y Berger, 1995). Este fenómeno ocurre cuando la actualización de la función valor para un determinado estado también modifica dicha función para otros estados, posiblemente en la dirección equivocada. El fenómeno de interferencia es inherente a la capacidad de generalización y, por tanto, también ocurre en otros algoritmos de aprendizaje supervisado. En el caso del aprendizaje TD sus efectos pueden ser mucho más peligrosos por dos motivos (Barreto y Anderson, 2008):

- La combinación de interferencia y *bootstrapping* tiende a ser inestable. En el aprendizaje supervisado, la función objetivo suele ser fija. En cambio, en TD, la función que se quiere aproximar es altamente no estacionaria, ya que se va actualizando con cada nueva interacción agente-entorno. De hecho las demostraciones de convergencia de TD se basan en la propiedad de que el operador de Bellman es una contracción, la cual deja de ser cierta cuando se usan aproximadores globales (Jaakkola et al., 1994).
- La naturaleza *online* de TD hace que la distribución de los datos dependa de la política seguida por el agente. En ocasiones, el agente puede permanecer en una determinada región del espacio que no es representativa de todo el dominio. En esta situación, un aproximador global posiblemente dedique muchos de sus recursos para representar esa región, "olvidando" el resto de regiones (Weaver et al., 1998).

Aunque parece claro que los métodos TD no son los más adecuados para explotar

los beneficios de las RNAs, la situación del algoritmo LSTD es diferente. La función valor se construye a partir de un conjunto de datos observados previamente, por lo que no se puede considerar aprendizaje puramente *online* ni tampoco se tiene una función objetivo no estacionaria. La limitación de LSTD a la hora de utilizar RNAs viene dada por la exigencia de que la arquitectura del aproximador tiene que ser lineal en los parámetros.

De acuerdo con la teoría de las máquinas de aprendizaje extremo, tras realizar una asignación aleatoria de los pesos y sesgos de la capa oculta, la red neuronal de tipo SFLN es equivalente a un sistema lineal. Para calcular la salida de una red ELM se realiza un mapeado del vector de entradas a un espacio de características definido por los nodos ocultos de la red y, posteriormente, se combinan linealmente las características. Esta propiedad de las redes ELM resulta especialmente interesante, ya que transforma la red neuronal en un modelo lineal en los parámetros y, por tanto, susceptible de ser combinado con LSTD.

Dado una red SLFN compuesta de M nodos en la capa oculta con la que se pretende aproximar una función valor en el algoritmo LSTD, el espacio de características se define como:

$$\boldsymbol{\phi}(s) = \begin{pmatrix} f(\mathbf{w}_1 \cdot s + b_1) \\ f(\mathbf{w}_2 \cdot s + b_2) \\ \vdots \\ f(\mathbf{w}_M \cdot s + b_M) \end{pmatrix} \tag{5.13}$$

donde $\mathbf{w}_l$ hace referencia a los pesos del l-ésimo nodo oculto, b_l al sesgo del mismo nodo y $f(x)$ es la función de activación de la capa oculta. Después de inicializar los pesos y sesgos aleatoriamente, los parámetros $\mathbf{h}$ de la capa de salida se calculan aplicando:

$$\mathbf{h} = \mathbf{A}^{-1}\mathbf{b}. \tag{5.14}$$

donde la matriz $\mathbf{A}$ y el vector $\mathbf{b}$ se construyen a partir de las trayectorias observadas de acuerdo con la Ecuación (5.12). Igual que en el caso del algoritmo ELM, la matriz $\mathbf{A}$ se invierte de forma robusta mediante la pseudo inversa de Moore-Penrose. El pseudo código que implementa esta versión de LSTD basada en ELM se muestra en el Algoritmo 5.1.

El algoritmo propuesto, denotado como LSTD-ELM, combina la capacidad de aproximación de las redes neuronales al mismo tiempo que evita los problemas derivados del aprendizaje *online* y del *bootstrapping*. Comparado con el algoritmo TD, LSTD-ELM permite el uso de un aproximador global de manera estable y eficiente. Respecto al algoritmo LSTD estándar combinado con aproximadores locales, se espera que LSTD-ELM pueda escalarse a espacios con un número elevado de dimensiones de forma más sostenible.

El pseudo código mostrado en el Algoritmo 5.1 se puede modificar fácilmente para adaptarlo al caso de usar un comité de redes ELM en lugar de una sola red. Si, por ejemplo, se quiere emplear un comité formado por B miembros, los pesos y sesgos de la

Algoritmo 5.1 Aprendizaje LSTD basado en ELM.

Require: Política π que se quiere evaluar, factor de descuento γ

1: Inicializar aleatoriamente los pesos $\mathbf{w}$ y los sesgos $\boldsymbol{b}$ correspondientes a la capa oculta de la red SLFN
2: Definir el espacio de características como $\boldsymbol{\phi}(s) = [f(\mathbf{w}_1 \cdot s + b_1), \ldots, f(\mathbf{w}_M \cdot s + b_M)]^\top$, donde $f(x)$ es la función de activación y M el número de nodos en la capa oculta
3: Fijar $\mathbf{A} = 0$; $\mathbf{b} = 0$; $k = 0$
4: **repeat**
5: Seleccionar un estado inicial s_k
6: **while** $s_k \neq s_{\text{END}}$ **do**
7: Aplicar la política π al sistema, observar la recompensa r_k y el estado siguiente s_{k+1}
8: $\mathbf{A} = \mathbf{A} + \boldsymbol{\phi}(s_k)(\gamma\boldsymbol{\phi}(s_{k+1}) - \boldsymbol{\phi}(s_k))^\top$
9: $\mathbf{b} = \mathbf{b} + \boldsymbol{\phi}(s_k)r_k$
10: $k = k + 1$
11: **end while**
12: **until** alcanzar el número deseado de episodios
13: $\mathbf{h} = \mathbf{A}^{-1}\mathbf{b}$
14: **return** $V^\pi(s) \approx \boldsymbol{\phi}(s)^\top\mathbf{h}$

capa oculta se deben inicializar B veces, una por cada red ELM del comité. También es necesario construir y almacenar B veces la matriz $\mathbf{A}$ y el vector $\mathbf{b}$. Finalmente, el proceso para calcular $V^\pi(s)$ no consiste simplemente en un producto escalar, sino que haría falta multiplicar $\boldsymbol{\phi}(s)$ por los parámetros $\mathbf{h}$ de cada red ELM y después combinar el resultado usando la mediana.

5.6. Experimentos

En esta sección se describen los experimentos llevados a cabo para evaluar empíricamente las propiedades del algoritmo LSTD-ELM. Se utilizaron dos dominios: el problema *Hop-world* (Boyan, 1999) y el péndulo invertido (Lagoudakis y Parr, 2003). En concreto, se ha propuesto una versión generalizada de *Hop-world* en la que puede seleccionarse el número de variables de estado de forma arbitraria. Este dominio, a pesar de ser un ejemplo ilustrativo que no está relacionado con ninguna aplicación real, resulta especialmente adecuado para evaluar las propiedades de LSTD-ELM conforme aumenta el número de dimensiones. Además de la dimensionalidad, también puede elegirse el tamaño de las trayectorias, lo que permite realizar un gran número de experimentos con un coste computacional razonable. Por otra parte, el problema del péndulo invertido proporciona un entorno más realista en el que analizar el comportamiento del algoritmo propuesto.

La calidad de la función valor aproximada con LSTD-ELM se ha medido en términos del error absoluto medio (MAE, *mean absolute error*), calculado como:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\widehat{y}_i - y_i| \tag{5.15}$$

donde $\widehat{y}_i$ es el valor obtenido con el algoritmo e y_i es el valor real. De forma similar a (Boyan, 2002), se utilizó un método de Monte-Carlo para estimar la "verdadera" función valor. Los experimentos también se llevaron a cabo empleando el algoritmo LSTD clásico con una red RBF de base fija con funciones gaussianas, denotado como LSTD-RBF. Este algoritmo fue utilizado como método de referencia para comparar los resultados. En cada uno de los experimentos se realizó un barrido del número de características con el fin de evaluar la sensibilidad de ambos algoritmos respecto a dicho parámetro. El término "características" hace referencia al número de nodos ocultos en el caso de LSTD-ELM y al número de funciones base en LSTD-RBF. Como función de activación en LSTD-ELM se seleccionó la función sigmoide. La teoría de las máquinas de aprendizaje extremo demuestra que cualquier función limitada e infinitamente diferenciable puede ser utilizada como función de activación (Huang y Babri, 1998; Huang et al., 2006); sin embargo, la función sigmoidal es la más extendida y ha sido probada en múltiples aplicaciones prácticas (Yeu et al., 2006; Baboo y Sasikala, 2010; Wu et al., 2013). El factor de descuento γ es el parámetro que controla el horizonte de optimización, o lo que es lo mismo, la relevancia de las recompensas futuras en el estado actual. A pesar de que el valor de γ juega un papel crucial en el aprendizaje de políticas óptimas, en el caso de las funciones valor, el hecho de variar γ únicamente produce una variación en la forma de la función valor. Dado que el objetivo de los experimentos es comprobar la capacidad de aproximación de LSTD-ELM, γ puede ser elegido arbitrariamente dentro del rango $[0, 1)$. En concreto se ha fijado $\gamma = 0.85$ para todos los experimentos.

Tras una comparación entre los algoritmos LSTD-ELM y LSTD-RBF, se realizó un proceso similar para evaluar las ventajas de usar un comité de redes ELM en lugar de una única red. El algoritmo resultante, denotado como LSTD-cELM, se aplicó tanto en el dominio *Hop-world* como en el péndulo invertido.

En las siguientes subsecciones se detallan los dominios utilizados (*Hop-world* generalizado y péndulo invertido), la implementación del algoritmo LSTD-RBF y el procedimiento para estimar la "verdadera" función valor mediante simulación de Monte-Carlo.

5.6.1. *Hop-world* generalizado

Hop-world es un problema ilustrativo propuesto inicialmente en (Boyan, 1999) y más tarde utilizado en otros trabajos de aprendizaje por refuerzo (Xu et al., 2002; Geramifard et al., 2006a; Sutton et al., 2009; Chen et al., 2013). Se trata de una cadena de Markov (o MDP con una política fija) cuyo espacio de estados es discreto y

unidimensional. Sin embargo, para evaluar el algoritmo LSTD-ELM se ha propuesto una generalización del problema *Hop-world* donde el espacio de estados es continuo y la dimensionalidad se puede elegir libremente.

En su forma original, el problema *Hop-world* contiene m estados discretos más un estado terminal, $S = \{0, 1, 2, \ldots, m\}$. Todas las trayectorias comienzan en el estado 0 y terminan en el estado absorbente o terminal m. En la Figura 5.6 se muestra un diagrama del problema. Se puede observar que en cada estado no-terminal se pueden realizar dos acciones, $A = \{a_0, a_1\}$. Ambas acciones acercan el estado actual hacia el final de la trayectoria, pero difieren en la cantidad avanzada. La transición entre cualquier par de estados genera una recompensa igual a -1, excepto la transición entre el estado $m - 1$ y m, cuya recompensa es $-2/3$. La política del agente es fija y consiste en seleccionar ambas acciones con la misma probabilidad.

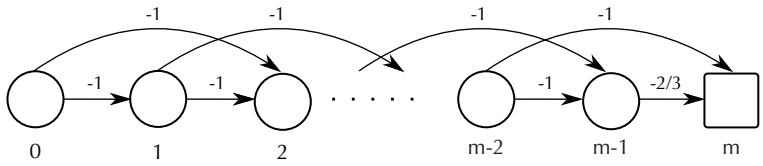


Figura 5.6
Representación esquemática del problema *Hop-world* original. Todas las trayectorias comienzan en el estado 0 y acaban en el estado terminal m. En cada estado no terminal es posible realizar dos acciones. La política es fija y escoge cada acción con probabilidad 0.5.

Se han propuesto dos modificaciones sobre el problema *Hop-world* original con el fin de hacerlo más general. Primero, se ha generalizado a un espacio de estados d-dimensional. Dado que el número de estados discretos por cada dimensión es m, un espacio de d dimensiones contendrá $|S| = m^d$ estados, donde $|\cdot|$ denota la cardinalidad. Por lo tanto, el número de estados aumenta exponencialmente con la dimensionalidad. En la Figura 5.7a se muestra el MPD resultante cuando $d = 2$. En este caso, cada estado está definido por dos variables discretas que se corresponden con los ejes x e y en el plano. El espacio de acciones también aumenta con el número de dimensiones: dos nuevas acciones posibles se añaden al conjunto con cada nueva dimensión, $|A| = 2d$. De forma similar al caso original, el estado inicial se sitúa en uno de los extremos del espacio de estados y el estado terminal en el extremo opuesto. La función de recompensa sigue siendo constante e igual a -1 para todas las transiciones excepto aquellas que alcanzan directamente el estado terminal. La política de selección de acciones también se ha mantenido igual que en el problema original: se escogen todas las acciones aleatoriamente con la misma probabilidad.

Además de generalizar el espacio de estados a d dimensiones, el problema discreto se modificó para transformarlo en continuo. En la versión continua, cada variable de

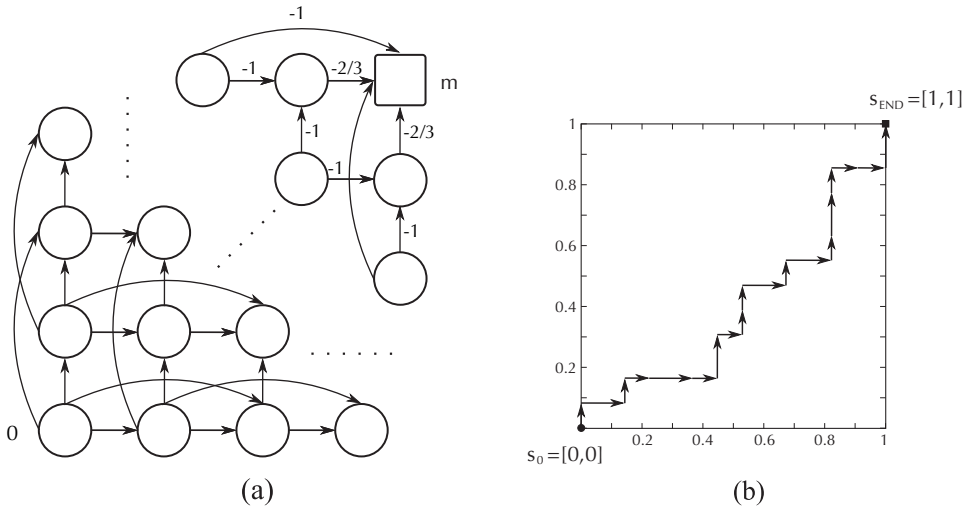


Figura 5.7
(a) Representación esquemática del problema *Hop-world* modificado con un espacio de estados discreto y d-dimensional (en la figura $d = 2$). (b) Ejemplo de una trayectoria típica del problema *Hop-world* generalizado, cuyo espacio de estados es continuo y d-dimensional (en la figura $d = 2$ y $step = 0.083$).

estado puede tomar valores dentro del rango $[0, 1]$, por lo que el espacio de estados deja de ser discreto. Por cada dimensión sigue habiendo dos acciones posibles que llevan el estado actual hacia el estado terminal. En este caso, en lugar de avanzar hasta el estado siguiente o dos estados siguientes, cada acción avanza en una cantidad $step$ o $2 \cdot step$, siendo $step$ un parámetro definido por el usuario. Adicionalmente, cada acción seleccionada fue perturbada con un ruido gaussiano i.i.d. de amplitud igual a $0.2 \cdot step$ con el objetivo de añadir estocasticidad al sistema. Como en los casos anteriores, el estado inicial y el terminal se sitúan en extremos opuestos del espacio de estados. Por ejemplo, si $d = 2$, el estado inicial estaría definido por $s_0 = [0, 0]$ y el estado final por $s_{\text{END}} = [1, 1]$. En la Figura 5.7b se muestra una trajectoria típica correspondiente a este caso.

Durante los experimentos, la dimensionalidad del problema *Hop-world* generalizado se varió desde 1 hasta 6. El parámetro $step$ fue adaptado de acuerdo a la dimensionalidad para asegurar que las trayectorias finalizaran tras un número razonable de acciones. En caso contrario, el tiempo necesario para completar una trayectoria aumentaría de forma exponencial respecto al número de dimensiones. La Tabla 5.1 muestra el parámetro $step$ seleccionado para cada configuración. La cantidad de observaciones (estados y recompensas) empleadas para estimar la función valor se controla mediante el número de trayectorias. Con el objetivo de obtener aproximaciones con una exactitud similar en todos los casos, el número de trayectorias también se incre-

mentó conforme d aumenta. El número de trayectorias correspondiente a las diferentes configuraciones del problema se muestra en la Tabla 5.1.

Tabla 5.1
Parámetros de las diferentes configuraciones del problema *Hop-world* usadas en los experimentos, incluyendo el número de trayectorias que emplearon los métodos LSTD (ambos, LSTD-ELM y LSTD-RBF) y Monte-Carlo. La columna "número de estados de test" indica el número de puntos discretos utilizados para calcular el MAE.

Configuración *Hop-world*		Número de trayectorias		Número de estados de test
Dimensionalidad	*Step*	LSTD	Monte-Carlo	
1	0.033	2000	$4.8 \cdot 10^5$	30
2	0.125	3000	$1.024 \cdot 10^6$	8^2
3	0.143	4000	$5.488 \cdot 10^6$	7^3
4	0.167	5000	$2.0736 \cdot 10^7$	6^4
5	0.25	6000	$5 \cdot 10^7$	5^5
6	0.333	7000	$6.5104 \cdot 10^7$	4^6

5.6.2. Péndulo invertido

El problema del péndulo invertido se ha utilizado ampliamente para evaluar algoritmos tanto en el ámbito del control automático (Wang et al., 1996; Ruan et al., 2007; Rong et al., 2011) como del aprendizaje por refuerzo (Gullapalli, 1995; Lagoudakis y Parr, 2003; Martín H. et al., 2011). En general, el objetivo del problema consiste en mantener en equilibrio una vara sólida colocada sobre un carro móvil. La Figura 5.8 muestra una representación esquemática del problema. El carrito se puede mover libremente en una pista unidimensional mientras que la vara se puede mover únicamente en el plano vértical del carro y la pista. Existen diversas versiones del problema en función de la longitud de la pista, el número de variables de estado, la posición inicial del péndulo, etc. La versión utilizada aquí coincide con la descrita por Lagoudakis y Parr (2003). Se asume que tanto los parámetros del sistema como la ecuación que gobierna su movimiento son desconocidos para el agente y que la única información que conoce del entorno es aquella contenida en las transiciones muestreadas. El espacio de acciones es discreto, siendo posible aplicar tres acciones sobre el carro: una fuerza hacia la izquierda (-50 N), hacia la derecha ($+50$ N) o nula (0 N). A cada una de las acciones se le añade ruido uniformemente distribuido en el rango $[-10, 10]$ N.

La dinámica del péndulo invertido es gobernada por la siguiente ecuación no lineal:

$$\ddot{\theta} = \frac{g \cdot \sin(\theta) - \alpha \cdot m \cdot l \cdot (\dot{\theta})^2 \cdot \sin(2\theta)/2 - \alpha \cdot \cos(\theta) \cdot a}{4 \cdot l/3 - \alpha \cdot m \cdot l \cdot \cos^2(\theta)} \tag{5.16}$$

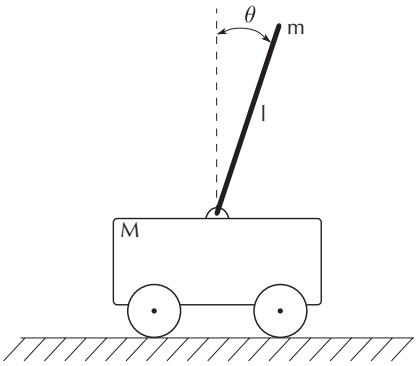


Figura 5.8
Representación esquemática del péndulo invertido.

donde g es la constante de gravedad ($g = 9.8$ m/s^2), m es la masa del péndulo ($m = 2$ Kg), M es la masa del carro ($M = 8$ Kg), l es la longitud del péndulo ($l = 0.5$ m), a es la acción de control y $\alpha = 1/(m + M)$. El espacio de estados del problema consiste en el ángulo vertical del péndulo θ y su velocidad angular $\dot{\theta}$, la cual está limitada al rango $[-5, 5]$ rad/s. Cuando se alcanza un ángulo superior (en valor absoluto) que $\pi/2$ se considera que el péndulo ha perdido el equilibrio y el episodio finaliza.

Los episodios comienzan con un estado seleccionado aleatoriamente entre el conjunto definido por los rangos $\theta = [-\pi/2, \pi/2]$ y $\dot{\theta} = [-5, 5]$. El agente recibe una recompensa igual a 1 siempre que el péndulo esté en equilibrio, es decir, $|\theta| < \pi/2$. En otro caso, la recompensa recibida es 0. Aunque habitualmente la finalidad es calcular una política capaz de balancear el péndulo, en este caso se pretende únicamente estimar la función valor de una política fijada previamente. Para ello, simplemente se utilizó una política que selecciona aleatoriamente entre las tres acciones posibles. El sistema fue simulado mediante la función *ode45* de MATLAB con un paso de simulación de 0.1 s.

5.6.3. LSTD basado en funciones de base radial gaussiana

El uso de redes RBF de base fija como aproximador de funciones es muy común en el campo del RL (Sutton y Barto, 1998; Busoniu et al., 2010a). Normalmente las funciones base suelen ser de tipo gaussiana, las cuales están definidas por dos parámetros: el centro y el ancho (o desviación estándar) (ver Sección 3.2.3). Cuando este tipo de aproximador se utiliza en problemas de aprendizaje supervisado, tanto el centro como la desviación estándar de las funciones se puede seleccionar teniendo en cuenta la distribución de los datos de entrada (Haykin, 2008). De esta forma es posible concentrar los recursos del aproximador en las zonas más críticas. Desafortu-

nadamente, en los problemas de predicción de la función valor los datos de entrada no se conocen de antemano. La solución comúnmente adoptada consiste en distribuir los centros uniformemente a lo largo del espacio de estados. La tarea de distribuir de forma equidistante un número arbitrario de p puntos en un espacio d-dimensional es un problema no trivial que debe resolverse de forma aproximada. Para ello se empleó el algoritmo de agrupamiento k-medias. El número de agrupamientos se fijó igual a p, y como datos de entrada se generó una malla de q^d puntos distribuidos en el espacio de estados. Nótese que cuando el número de puntos es una raíz d-ésima exacta, resulta sencillo distribuirlos uniformemente en el espacio d-dimensional. Si q se selecciona tal que $q^d > p$, los prototipos de los agrupamientos resultantes de aplicar el algoritmo k-medias estarán aproximadamente equiespaciados en todo el espacio de estados.

Respecto al segundo parámetro de las gaussianas, el procedimiento habitual consiste en usar alguna regla heurística para fijar la misma desviación estándar en todas las funciones base. Por ejemplo, en Haykin (2008), las desviaciones son calculadas como:

$$\sigma = \frac{d_{\text{máx}}}{\sqrt{2M}}, \tag{5.17}$$

donde M es el número de centros y $d_{\text{máx}}$ es la distancia máxima entre cualquier par de centros. Esta fórmula asegura que las gaussianas no son ni demasiado estrechas ni demasiado anchas. Si son demasiado estrechas hará falta un número muy elevado de funciones para cubrir todo el espacio, mientras que si son demasiado anchas el aproximador dejará de ser de carácter local, por lo tanto se debe llegar a un compromiso para evitar ambos extremos. Se ha demostrado que la Ecuación (5.17) proporciona una solución cercana a la óptima cuando los datos están uniformemente distribuidos en el espacio de entrada (Benoudjit et al., 2002; Benoudjit y Verleysen, 2003). Otra heurística definida con el mismo propósito es la proporcionada por Alpaydin (2004). Consiste en, después de seleccionar los centros, encontrar la máxima distancia entre ellos y fijar σ igual a la mitad de esta distancia, o un tercio si se desea ser más conservador. En los experimentos realizados en este capítulo se testearon RBFs con estas tres desviaciones estándar, denotadas como $\sigma_{\text{set}} = \{\sigma_{\text{Hay}}, \sigma_{\text{Alp1}}, \sigma_{\text{Alp2}}\}$. Además, también se probaron varios múltiplos de estas tres heurísticas, concretamente $0.5{\cdot}\sigma_{\text{set}}$, $2{\cdot}\sigma_{\text{set}}$ y $4{\cdot}\sigma_{\text{set}}$, lo que hace un total de 12 valores.

5.6.4. Simulación de Monte-Carlo

El error obtenido por LSTD-ELM y LSTD-RBF en la aproximación de la función valor se ha medido utilizando el índice MAE. Para calcular el MAE es necesario disponer de la "verdadera" función valor, la cual se ha estimado mediante simulación de Monte-Carlo. Dada una política π, el valor de un estado s, $V^\pi(s)$, se ha definido previamente como el valor esperado del retorno obtenido cuando el proceso comienza en dicho estado y el agente actúa siguiendo π (ver Sección 2.5). A partir de esta definición resulta evidente que $V^\pi(s)$ se puede calcular almacenando los retornos de diversas realizaciones del proceso y, posteriormente, calculando el promedio de ellos.

Este método de hallar la función valor se puede considerar un ejemplo de los conocidos como métodos de Monte-Carlo (Sutton y Barto, 1998). Desafortunadamente, los retornos son de carácter estocástico y su varianza puede ser elevada, lo que en la práctica significa que es necesario promediar un elevado número de retornos para obtener estimaciones fiables. Por otra parte, si la estimación de $V^{\pi}(s)$ se realiza mientras se interactúa con el sistema, quizás no sea posible forzar un estado inicial arbitrario, en cuyo caso, no es viable aplicar una técnica basada en Monte-Carlo (Szepesvári, 2010). Debido a estos motivos el uso práctico de los métodos de Monte-Carlo es escaso. Sin embargo, para los experimentos realizados en este capítulo proporcionan una forma sencilla de calcular la función valor en un subconjunto discreto de estados.

El número exacto de retornos que se requiere para obtener la estimación depende del problema específico que se esté tratando y se puede determinar empíricamente. Por ejemplo, en la Figura 5.9 se ha representado la estimación de $V^{\pi}(s)$ conforme aumenta el número de trayectorias (cada trayectoria genera un retorno). La figura corresponde al problema *Hop-world* generalizado con $d = 1$ y los estados $s = \{0.13, 0.48, 0.83\}$. Además del valor estimado, en las gráficas se muestra el intervalo de confianza al 95 %. Después de 16000 trayectorias, las estimaciones permanecen prácticamente estables y pueden considerarse suficientemente exactas.

Este mismo procedimiento se llevó a cabo con el resto de problemas para fijar el número de trayectorias. También fue necesario definir el conjunto de estados discretos de test donde se estimó la "verdadera" función valor. El número de estados en dicho conjunto tiene que ser suficientemente grande como para ser representativo de la función valor completa y, al mismo tiempo, suficientemente pequeño como para ser computacionalmente manejable. Los estados se escogieron para que estuvieran uniformemente distribuidos en el espacio de estados. Ambos parámetros, número de trajectorias y número de estados de test, se muestran en la Tabla 5.1 para las seis configuraciones del problema *Hop-world*. En el péndulo invertido, se emplearon $2.56 \cdot 10^6$ trayectorias para calcular la función valor mediante simulación de Monte Carlo en un total de 64 estados (8 por dimensión) distribuidos en el espacio definido por $\theta = [-\pi/2, \pi/2]$ y $\dot{\theta} = [-5, 5]$.

5.7. Resultados

5.7.1. LSTD basado en ELM

En la primera parte de los resultados se ha comparado el algoritmo propuesto, LSTD-ELM, con el algoritmo estándar, LSTD-RBF. Se emplearon seis configuraciones del problema *Hop-world* generalizado donde se varió, entre otros parámetros, la dimensionalidad del espacio de estados. El objetivo fue estudiar la capacidad de ambos métodos para aproximar funciones valor cuando el número de dimensiones aumenta. En cada configuración se hizo un barrido del número de características (funciones base en LSTD-RBF y nodos de la capa oculta en LSTD-ELM) para determinar la

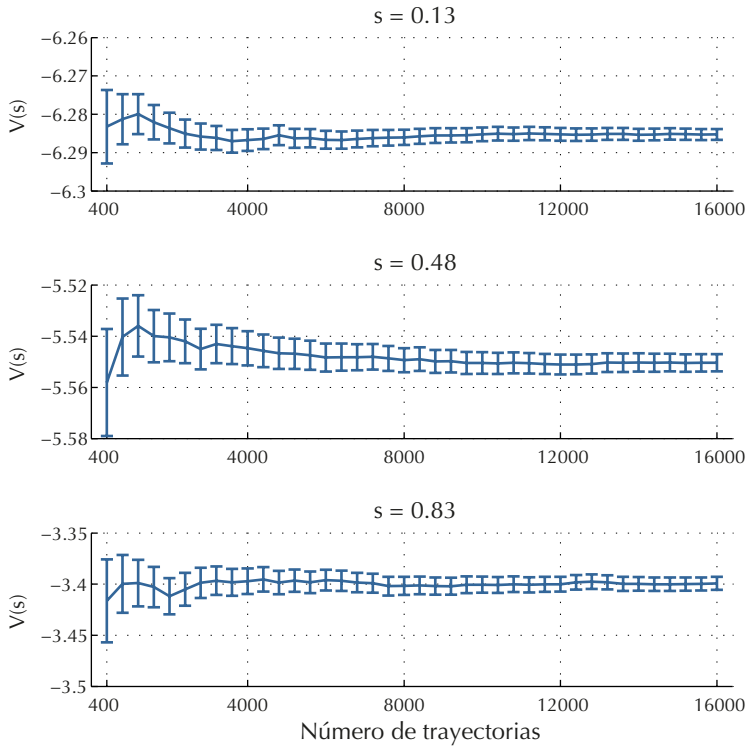


Figura 5.9
Estimaciones de la función valor usando simulación de Monte-Carlo para tres estados del problema *Hop-world* generalizado con $d = 1$. La figura muestra como evolucionan el valor medio y los intervalos de confianza (95 %) conforme aumenta el número de trayectorias usado en la estimación.

robustez de ambos algoritmos frente a este parámetro. El mismo procedimiento se llevó a cabo con el problema del péndulo invertido.

La Figura 5.10 muestra un resumen de los resultados obtenidos para el problema *Hop-world*, donde cada una de las subfiguras corresponde a una configuración diferente. Los resultados obtenidos en el péndulo invertido se muestran en la Figura 5.11. El eje vertical hace referencia al índice MAE y el horizontal al número de características. Cuando se aplica el algoritmo LSTD-ELM, los resultados obtenidos pueden variar al repetir el experimento debido al proceso aleatorio usado para fijar los parámetros de la capa oculta. Las Figuras 5.10 y 5.11 muestran el caso más desfavorable (MAE máximo) tras realizar cada experimento 50 veces de forma independiente. Respecto al algoritmo LSTD-RBF, se probaron 12 valores de σ (desviación estándar de la función gaussiana) y en las figuras únicamente se muestran los tres valores que proporciona-

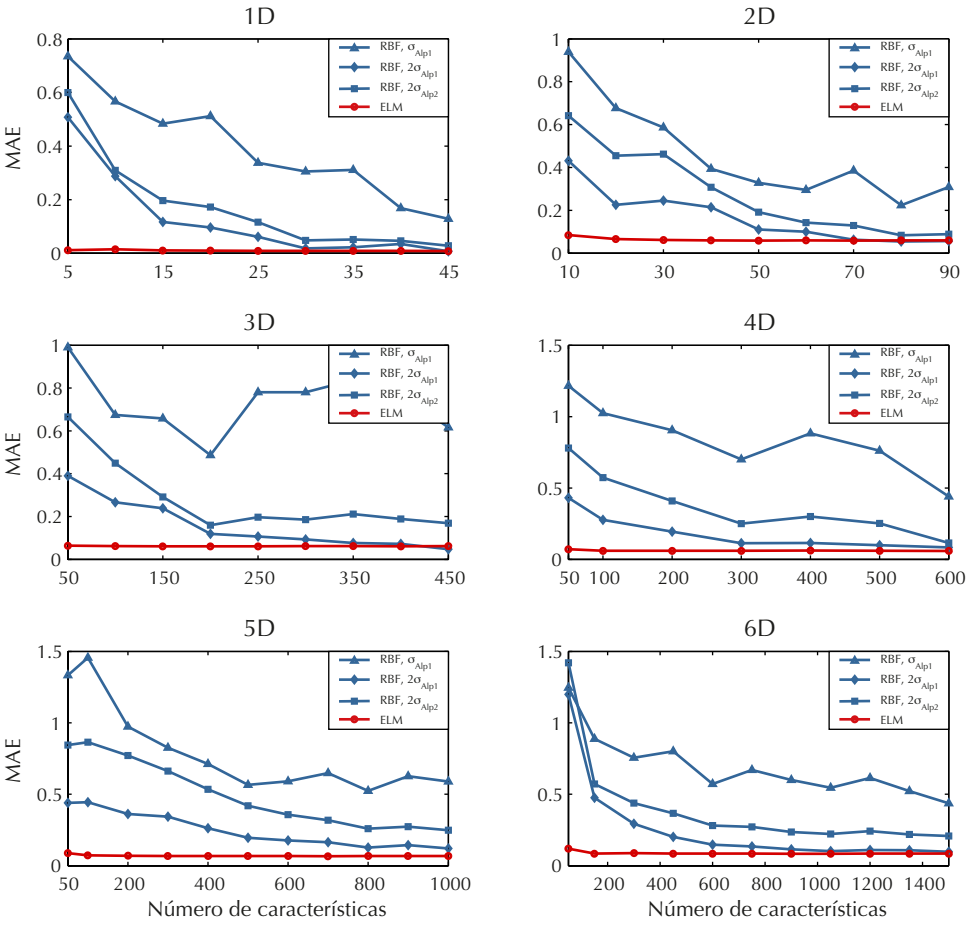


Figura 5.10
Comparación del índice MAE frente al número de características obtenido por los algoritmos LSTD-RBF y LSTD-ELM en el problema *Hop-world* generalizado. Cada una de las subfiguras se corresponde con una configuración de *Hop-world*, donde la dimensionalidad varía desde 1 hasta 6.

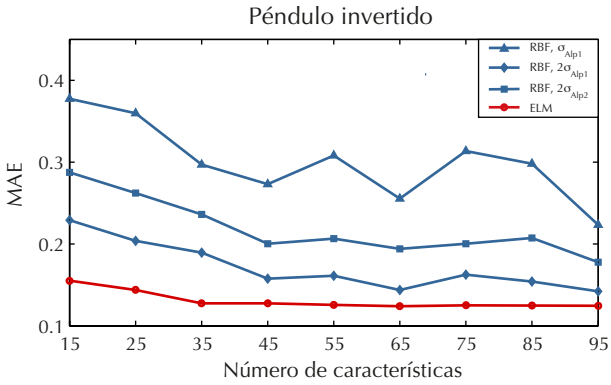


Figura 5.11
Comparación del índice MAE frente al número de características obtenido por los algoritmos LSTD-RBF y LSTD-ELM en el problema del péndulo invertido.

ron mejores resultados. En general, el valor de σ óptimo fue mayor de lo esperado, concretamente el doble de lo indicado por las heurísticas. El motivo de este comportamiento posiblemente esté relacionado con el hecho de que las funciones valor de los experimentos, a pesar de ser no lineales, varían de forma suave. Sin embargo, cuando la función que se quiere aproximar contiene cambios abruptos es preferible usar valores de σ más pequeños.

En las Figuras 5.10 y 5.11 las líneas azules corresponden al algoritmo LSTD-RBF con los tres valores de σ óptimos. En general, se observa que existe una dependencia entre el número de características y el MAE obtenido: conforme aumenta la complejidad de las redes RBF se reduce el error. Esta reducción es más pronunciada al principio de las curvas de error y tiende a suavizarse para un número elevado de características. Al comparar los resultados de los algoritmos LSTD-RBF y LSTD-ELM (en rojo), se puede observar que en algunos casos (*Hop-world* con 1, 2 y 3 dimensiones) la exactitud de las redes RBF es igual o incluso ligeramente superior a la de las redes ELM. La evolución del MAE correspondiente al algoritmo LSTD-ELM sugiere que este algoritmo es capaz de representar las funciones valor de forma más compacta, es decir, empleando menos características. Se observa que el MAE para este caso permanece prácticamente estable, obteniendo un error mínimo incluso con pocas características. Por ejemplo, en el problema *Hop-world* con $d = 5$, LSTD-ELM con 100 características obtuvo un error menor que LSTD-RBF con 1000. En el problema del péndulo invertido las diferencias entre los diferentes métodos son más pequeñas (nótese que la escala va desde 0.1 hasta 0.45), pero aún así, la mejora de LSTD-ELM es evidente.

Los experimentos confirman las ventajas de los aproximadores globales frente a los locales. Como se esperaba, el uso de redes SLFN mejora las propiedades de escalabili-

dad de LSTD. En determinados casos, el algoritmo LSTD-RBF con un elevado número de características puede alcanzar una exactitud mayor que el algoritmo LSTD-ELM. Sin embargo, incluso en esos casos, no existe una ventaja clara de LSTD-RBF: la mejora es pequeña y el algoritmo requiere al ajuste de dos parámetros más (centros y desviaciones estándar de las gaussianas) que LSTD-ELM.

5.7.2. LSTD basado en comités ELM

En este segundo apartado de la sección de resultados se analizan empíricamente los beneficios de aproximar la función valor mediante un comité de redes ELM. Para ello se compararon los resultados obtenidos por LSTD-ELM y LSTD-cELM siguiendo un procedimiento similar al del apartado anterior. Los experimentos se llevaron a cabo en las seis configuraciones de *Hop-world* y en el problema del péndulo invertido. En cada uno de los problemas se calculó el índice MAE obtenido por ambos algoritmos frente al número de características.

Los resultados obtenidos se muestran en las Figuras 5.12 y 5.13. A diferencia del caso anterior, el MAE correspondiente al algoritmo LSTD-ELM (en rojo) es el caso más favorable (MAE mínimo) obtenido entre las 50 repeticiones de cada experimento. LSTD-cELM también se aplicó 50 veces en cada problema y el MAE (en azul) se calculó como el valor medio de las 50 repeticiones. Por tanto, se está comparando el valor esperado obtenido por un comité con el mejor valor posible obtenido por una red ELM. El número de miembros del comité se fijó igual a 10 para todos los casos. Este valor, que se obtuvo empíricamente, proporciona un compromiso entre la exactitud de la aproximación y la carga computacional del método: al emplear más miembros aún es posible reducir el MAE pero, a partir de 10, la mejora es pequeña para el incremento computacional que supone.

Como se puede observar, el error obtenido por ambos métodos decrece rápidamente al principio, cuando el número de características es aún pequeño, hasta alcanzar un valor en el que se mantiene aproximadamente estable. Durante la primera etapa, en algunos casos, el uso del comité produce peores resultados que la red ELM. En cambio, cuando se utilizan suficientes características, LSTD-cELM mejora la calidad de la aproximación de forma estable y consistente. Aunque la mejora es obvia en todos los casos, la reducción de MAE obtenida puede variar de unos problemas a otros. Por ejemplo, al calcular el porcentaje de mejora[2] en *Hop-world* con 1 y 2 dimensiones, se obtuvo un 17.74 % y 21.50 %, respectivamente. Mientras que para Hop-word con 4 dimensiones fue de 6.55 % y en el péndulo invertido de 7.14 %. Así pues, los resultados sugieren que combinar las salidas de varias redes ELM puede dar lugar a un aproximador que mejora la capacidad de cada una de las redes individualmente.

[2]Para calcular el porcentaje de mejora de MAE en cada problema se utilizaron únicamente los valores del índice MAE donde LSTD-cELM supone una reducción respecto a LSTD-ELM. Por ejemplo, en el problema *Hop-world* con $d = 4$, la mejora se produce a partir de 200 características en adelante.

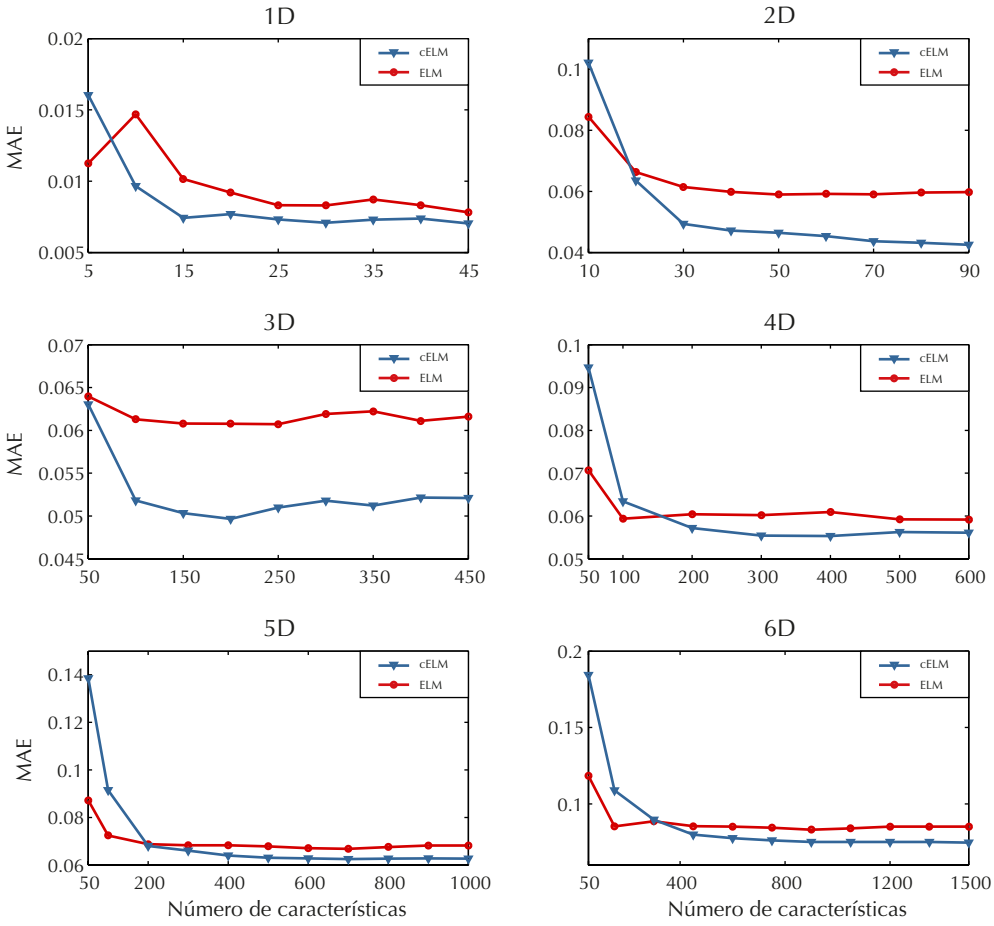


Figura 5.12
Comparación del índice MAE frente al número de características obtenido por los algoritmos LSTD-ELM y LSTD-cELM en el problema *Hop-world* generalizado. Cada una de las subfiguras se corresponde con una configuración de *Hop-world*, donde la dimensionalidad varía desde 1 hasta 6.

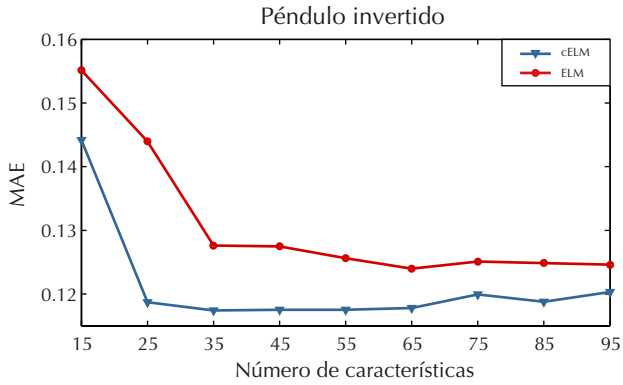


Figura 5.13
Comparación del índice MAE frente al número de características obtenido por los algoritmos LSTD-ELM y LSTD-cELM en el problema del péndulo invertido.

El principal inconveniente de los comités es que implican un incremento de la carga computacional: el número de operaciones necesarios para calcular un modelo de este tipo aumenta linealmente con el número de miembros. La complejidad computacional del algoritmo LSTD-ELM es $\mathcal{O}(Ln^2 + n^3)$, siendo L el número de observaciones y n el número de características; mientras que para el algoritmo LSTD-cELM con c comités la complejidad se multiplica por c. A pesar de este incremento, las redes ELM resultan especialmente adecuadas para combinarlas mediante comités, ya que destacan por su baja carga computacional comparado con otros tipos de redes neuronales. Adicionalmente, aparece un nuevo parámetro que es necesario ajustar: el número de miembros. Por tanto, conviene analizar cada problema concreto para determinar si la mejora aportada por el uso de comités merece la pena o no.

5.8. Conclusiones

En este capítulo se ha propuesto el algoritmo LSTD-ELM para resolver problemas de predicción de la función valor. LSTD-ELM se basa en la teoría de las máquinas de aprendizaje extremo para integrar el algoritmo LSTD con redes neuronales de tipo SLFN. Las principales ventajas del algoritmo propuesto son:

- Permite aproximar las funciones valor empleando redes neuronales, un método de aproximación caracterizado por ser de naturaleza global y, por tanto, adecuado para trabajar en espacios de dimensionalidad elevada. Las aproximaciones obtenidas por LSTD-ELM son más compactas que las que proporcionan los algoritmos LSTD estándar basados en aproximadores locales. Una ventaja adi-

cional es que cuanto más compacto es el aproximador, tiene menos parámetros y se requiere una cantidad de datos menor para ajustarlos.

- Elimina los problemas de convergencia asociados al efecto conocido como "interferencia". Otros algoritmos de RL basados en redes neuronales se pueden volver fácilmente inestables debido, en parte, al aprendizaje *online*. En el algoritmo LSTD-ELM se procesan todos los datos al mismo tiempo, por lo que el algoritmo resulta más estable.

El algoritmo propuesto ha sido evaluado empíricamente en dos dominios: una versión generalizada del problema *Hop-world* y en un péndulo invertido. El dominio *Hop-world* generalizado es un problema ilustrativo que se ha desarrollado para evaluar el funcionamiento de LSTD-ELM en problemas de diferentes dimensionalidades, ya que permite seleccionar el número de variables de estado de forma arbitraria. Por otra parte, el péndulo invertido es un problema de test ampliamente utilizado en el campo del RL. Como método de referencia se ha empleado el algoritmo estándar LSTD-RBF. Los resultados obtenidos sugieren que el algoritmo LSTD-ELM es capaz de obtener aproximaciones de una calidad similar, o superior, con un número de características menor.

También se ha propuesto una variación de LSTD-ELM basada en el uso de varias redes ELM combinadas mediante un comité. El algoritmo resultante, denotado como LSTD-cELM, se ha evaluado en los dominios Hop-word y el péndulo invertido. Los experimentos indican que el hecho de emplear un comité mejora la exactitud de las aproximaciones. Como contrapartida, LSTD-cELM supone un incremento de la carga computacional y añade un nuevo parámetro que debe ser ajustado por el usuario.

En general, los resultados han demostrado el potencial de las redes ELM en el aprendizaje LSTD y abren el camino a investigar su uso en el problema general de RL, donde el objetivo es encontrar políticas óptimas en lugar de predecir la función valor de una política fija. Una posible vía para adaptar el algoritmo LSTD-ELM a dicho caso consiste en modificarlo para aproximar funciones Q y, posteriormente, integrarlo en una arquitectura de iteración de políticas. Otra línea de investigación futura es la introducción de un término de regularización. Cuando el número de características es suficientemente grande comparado con el número de trayectorias observadas, los métodos LSTD tienden a sobreajustarse. La regularización es una técnica comúnmente usada en regresión lineal para evitar el sobreajuste. Sin embargo, incluir un término de regularización en LSTD resulta más complicado que en el caso de la regresión lineal. Aún así existen algunos algoritmos LSTD regularizados que podrían ser usados junto con redes ELM (Kolter y Ng, 2009; Hoffman et al., 2011).

Capítulo 6

Optimización del tratamiento de la anemia en pacientes en hemodiálisis mediante aprendizaje por refuerzo

6.1. Introducción

La anemia es una enfermedad que se caracteriza por una concentración baja de hemoglobina (Hb) en sangre. Puede deberse a diversas causas, entra ellas, la enfermedad renal crónica (ERC). La ERC consiste en la pérdida progresiva e irreversible de las funciones renales. La evolución de la enfermedad se divide en 5 fases que dependen de la capacidad de ultrafiltración de los riñones, siendo conocida la última fase como enfermedad renal terminal (ERT) (Daugirdas, 2011). Se calcula que la ERC afecta a más de 25 millones de personas solo en Estados Unidos, aproximadamente el 13 % de la población adulta. La Figura 6.1 muestra la evolución del número de pacientes afectados por la fase terminal de ERC durante las últimas décadas. Además, también muestra los pacientes segmentados de acuerdo al tipo de tratamiento que han recibido (USRDS, 2010). Como se puede observar, el impacto de la enfermedad, tanto a nivel social como económico, no ha dejado de crecer durante los últimos años y las previsiones indican una tendencia similar en el futuro. En el año 2006, el tratamiento de la ERT supuso al *Medicare*[1] un gasto superior a 23000 millones de dólares, cifra que ha seguido aumentado en los años posteriores (USRDS, 2010).

[1] *Medicare* es el sistema de seguridad social estadounidense creado por el gobierno para garantizar la atención sanitaria a personas mayores de 65 años. También proporciona cobertura a personas más jóvenes con discapacidad o problemas graves de salud como cáncer y ERC.

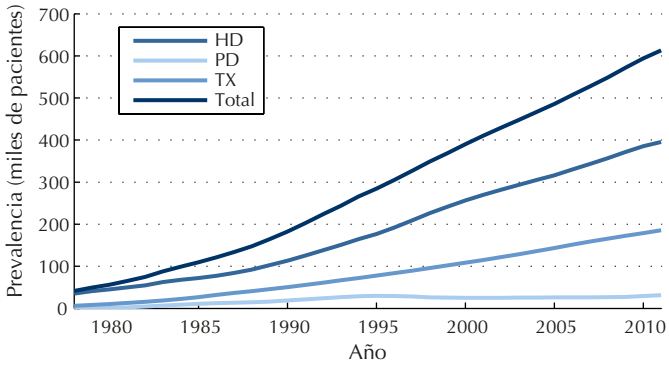


Figura 6.1
HD: Hemodiálisis; PD: diálisis peritoneal; TX: trasplante. Prevalencia de la ERT por modalidad de tratamiento en Estados Unidos desde 1978 hasta 2011.

Una vez alcanzada la fase terminal, los riñones han perdido prácticamente toda la capacidad de eliminar desechos y el paciente requiere terapia de sustitución renal (hemodiálisis, diálisis peritoneal o trasplante renal). Entre otras funciones, los riñones se encargan de regular la producción de glóbulos rojos (o eritrocitos), un tipo de células sanguíneas ricas en Hb. Debido a ello, la anemia afecta, aproximadamente, a un 90 % de pacientes que requieren diálisis (O'Mara, 2008).

El tratamiento estándar de la anemia renal consiste en la administración de agentes estimulantes de la eritropoyesis (AEEs). Este tipo de medicamentos están asociados con diversos efectos secundarios, además de suponer un importante coste para el sistema sanitario (cerca del 10 % del coste total de la ERT). Aunque hace ya casi 30 años que apareció el primer AEE, a día de hoy todavía no se conocen con exactitud los mecanismos de interacción entre el cuerpo humano y los AEEs. Se sabe que los pacientes responden al tratamiento con AEEs de forma muy heterogénea, es decir, la misma dosis de medicamento en diferentes pacientes, o incluso en el mismo paciente pero en diferentes instantes temporales, puede producir efectos dispares. Esta característica se conoce en el ámbito farmacológico como variabilidad inter- e intraindividual. La respuesta individual de cada paciente puede variar en función de numerosos factores, incluyendo sus características físicas (peso, altura, índice de masa corporal, edad, raza, sexo, etc.), gravedad de la enfermedad, comorbilidades, medicaciones concurrentes, etc. (Koch et al., 1974; Ifudu et al., 1996).

La relación entre la condición del paciente y la respuesta al tratamiento con AEEs es compleja; en la práctica resulta difícil calcular la dosis adecuada para cada paciente. Existen varias iniciativas internacionales que, con el objetivo de facilitar la tarea de los nefrólogos, publican periódicamente guías de práctica clínica para el tratamiento de la ERT como, por ejemplo, la *KDOQI Clinical Practice Guidelines and Clinical*

Practice Recommendations for Anemia in Chronic Kidney Disease (KDOQI, 2006) o la *Revised european best practice guidelines for the management of anaemia in patients with chronic renal failure* (Locatelli et al., 2004). La guías clínicas son documentos basados en el análisis sistemático de las últimas investigaciones y que sintetizan los resultados más importantes con el fin de acercarlos a la práctica clínica. A pesar de estos esfuerzos, la administración de AEEs continúa siendo un proceso complejo que requiere continuos ajustes de dosis.

Los resultados de varios estudios sugieren la existencia de un fenómeno conocido como *Hb cycling* que se observa en un elevado porcentaje de los pacientes tratados con AEEs (Macdougall et al., 2005; Collins et al., 2005). *Hb cycling* se ha definido como el movimiento repetitivo y cíclico del nivel de Hb durante el tratamiento con AEEs. La causas exactas de este fenómeno todavía no se conocen con exactitud, pero se han sugerido algunos factores como posibles causas. Fishbane y Berns (2007) destacaron dos aspectos del tratamiento de la anemia como motivos principales del *Hb cycling*. Por una parte, muchas clínicas de diálisis ajustan las dosis de AEEs de acuerdo a protocolos que implementan las indicaciones proporcionadas en las guías clínicas. Sin embargo dichos protocolos suelen ser demasiado rígidos y no tienen en cuenta la alta variabilidad en la respuesta de los pacientes al tratamiento. Por otra parte, la guías suelen recomendar rangos objetivo de Hb excesivamente estrechos, lo cual hace que sea necesario cambiar la dosis frecuentemente. El efecto que produce un cambio de dosis de AEE en el nivel de Hb no alcanza un estado estacionario hasta aproximadamente 70-120 días, tiempo correspondiente al ciclo de vida de los eritrocitos. Cuando la dosis de AEE se cambia numerosas veces en poco tiempo, resulta difícil tener en cuenta los efectos a largo plazo de cada dosis, por lo que a menudo son ignorados (Kalicki y Uehlinger, 2008; Daugirdas, 2011).

La incorporación de las tecnologías de la información y la comunicación en los centros sanitarios ha dado lugar a grandes bases de datos que contienen información detallada sobre los pacientes, incluyendo los tratamientos que reciben y su evolución clínica. Dicha información puede resultar útil para evitar errores clínicos, mejorar tratamientos y reducir efectos secundarios y costes (Patel et al., 2009). El objetivo de este capítulo es estudiar la viabilidad de utilizar las técnicas de RL para optimizar el tratamiento de la anemia renal con AEEs. Hay dos características que hacen al RL especialmente atractivo para afrontar este problema respecto a otros enfoques:

1. Los métodos de RL son capaces de encontrar políticas óptimas para la administración de AEEs a partir únicamente de la experiencia (datos) adquirida previamente, sin la necesidad de ningún modelo matemático del proceso.

2. En lugar de centrarse en un objetivo a corto plazo, el RL incorpora un horizonte temporal en el proceso de optimización, lo cual le permite tener en cuenta los efectos del tratamiento a largo plazo.

El RL puede considerarse como un método basado en datos para resolver problemas secuenciales de toma de decisión formulados mediante MDPs. Calcular las dosis

óptimas de un fármaco para tratar una enfermedad crónica (o de larga duración) es precisamente un problema de toma de decisiones donde el objetivo es encontrar la mejor secuencia de dosis. El marco teórico de los MDPs resulta adecuado para formular este tipo de problemas debido a que captura la naturaleza estocástica del proceso y la incertidumbre asociada al efecto que produce cada dosis (Hauskrecht y Fraser, 2000; Alagoz et al., 2010; Bennett y Hauser, 2013). El procedimiento clásico para resolver MDPs consiste en aplicar las técnicas de programación dinámica. Sin embargo, estas técnicas presentan dos importantes limitaciones: (i) solo son aplicables a problemas con un número finito y reducido de estados, y (ii) requieren un modelo completo del MDP, incluyendo la matriz de probabilidades de transición (ver Sección 2.6). El RL proporciona soluciones para ambas limitaciones y es capaz, en principio, de proporcionar una política de administración de AEEs con un alto grado de personalización incluyendo los efectos del tratamiento a largo plazo. En concreto, se ha escogido el algoritmo *fitted Q iteration* debido a su habilidad para hacer un uso de la experiencia más eficiente que otras técnicas (ver Sección 3.5).

El resto del capítulo se ha organizado como sigue. En la Sección 6.2 se analiza la bibliografía relacionada, destacando las diferencias entre los trabajos previos y la aproximación planteada en este capítulo. La Sección 6.3 proporciona información más detallada sobre la anemia renal, como afecta al nivel de Hb y su tratamiento. Los experimentos realizados para evaluar la metodología propuesta están basados en un modelo computacional que se describe en la Sección 6.4. La Sección 6.5 muestra la formulación del problema mediante procesos de decisión de Markov. Posteriormente se presentan los experimentos llevados a cabo para demostrar la validez del método en la Sección 6.6. Los resultados obtenidos y su discusión se han incluido en la Sección 6.7. Finalmente, la Sección 6.8 concluye el capítulo e introduce algunas posibles líneas de investigación futura.

6.2. Antecedentes

La idea de usar un método basado en datos para optimizar la administración de AEEs no es nueva. Durante la última década varios autores han propuesto métodos basados en redes neuronales artificiales para individualizar las dosis de AEE (Jacobs et al., 2001; Martín-Guerrero et al., 2003b; Gabutti et al., 2006). En general, los métodos propuestos se basan en utilizar el nivel de Hb actual e histórico, las dosis de AEE y otras variables que describen la situación clínica del paciente para predecir el nivel de Hb futuro. El objetivo de desarrollar un predictor de Hb es poder selecionar la dosis de AEE más adecuada para obtener el nivel de Hb deseado. Este planteamiento es correcto únicamente cuando el objetivo es optimizar el nivel de Hb a corto plazo. Por el contrario, el tratamiento con AEE busca la estabilización de la Hb a largo plazo. También se han planteado métodos similares basados en otras técnicas de aprendizaje automático, como lógica borrosa (Gaweda et al., 2008b, 2003), máquinas de vectores soporte (Martín-Guerrero et al., 2003a) o redes bayesianas (Bellazzi, 1993).

El control predictivo, más conocido por sus siglas en inglés MPC (*model predictive control*), es un método de control automático que se caracteriza por incluir un horizonte temporal en el proceso de optimización. Gaweda et al. (2008a) estudió la aplicación del MPC al problema del tratamiento de la anemia, concluyendo que puede mejorar los tratamientos actuales. La principal limitación de esta técnica es que requiere un modelo preciso del sistema, lo cual resulta especialmente complejo en problemas médicos, donde el sistema a modelar es la interacción de fármacos con el cuerpo humano. Incluso si fuera posible disponer de dicho modelo, se ha demostrado que los algoritmos de RL pueden producir resultados competitivos con MPC (Ernst et al., 2009).

El RL en el contexto de la anemia renal ha sido previamente estudiado por Gaweda et al. (2005) y Martín-Guerrero et al. (2009). Ambos estudios coinciden en el potencial del RL como alternativa a los protocolos de tratamiento utilizados actualmente. El algoritmo empleado en esos dos trabajos fue *Q-learning* (Watkins y Dayan, 1992). Este algoritmo destaca por ser muy eficiente computacionalmente, siendo posible aplicarlo en tiempo real debido al bajo número de operaciones que requiere por iteración, característica que lo ha hecho muy popular en ciertos ámbitos como la robótica. Sin embargo el número de datos que necesita para aprender es muy elevado comparado con otros algoritmos más modernos (Wiering y van Otterlo, 2012). *Fitted Q iteration* (FQI) (Ernst et al., 2005a) es un algoritmo relativamente nuevo que reduce drásticamente la cantidad de datos necesaria para obtener políticas óptimas. Actualmente hay un creciente interés dentro del campo de la medicina en aplicar FQI para el tratamiento de diferentes enfermedades como, por ejemplo, el VIH/SIDA (Ernst et al., 2006), desórdenes psiquiátricos (Murphy et al., 2006), epilepsia (Guez et al., 2008; Pineau et al., 2009), esquizofrenia (Shortreed et al., 2010; Lizotte et al., 2012) o adicción al tabaco (Chakraborty et al., 2008).

6.3. Anemia renal

La anemia renal aparece cuando los riñones pierden la capacidad de producir suficiente eritropoyetina para mantener una concentración plasmática de Hb adecuada. Se trata de una de las manifestaciones más visibles de la ERC. En algunos pacientes comienza a aparecer en la fase 3 de la enfermedad y se agrava con el deterioro progresivo de la función renal, siendo en la fase terminal (ERT) donde afecta a un mayor porcentaje de pacientes.

El impacto fundamental de la anemia es la reducción de la capacidad sanguínea para transportar oxígeno. La falta de oxígeno en los órganos activa diversos mecanismos compensatorios y una serie de cambios tanto a nivel fisiológico como metabólico (Nowrousian, 2002). Estos cambios producen en los pacientes un número elevado de síntomas así como limitaciones físicas y mentales y, en general, una reducción de la calidad de vida. Las consecuencias más graves de la anemia son aquellas relacionadas con el sistema cardiovascular. Existe una relación directa entre la anemia y

la hipertrofria ventricular izquierda (HVI). En pacientes anémicos el ritmo cardíaco aumenta para intentar paliar la falta de oxígeno, un sobreesfuerzo que, a menudo, desemboca en HVI. Otras anormalidades cardiacas asociadas con la anemia son la fibrosis y la pérdida capilar (Foley et al., 2000). La presencia de estas anormalidades suponen un incremento en la mortalidad de los pacientes además de provocar otras complicaciones (Xue et al., 2002).

6.3.1. Producción de hemoglobina

La hemoglobina (Hb) es una proteína que forma parte de los glóbulos rojos (o eritrocitos) y cuya principal función es transportar oxígeno y dióxido de carbono entre los órganos respiratorios y el resto de tejidos del cuerpo. En situaciones donde el suministro de oxígeno en los tejidos es insuficiente (hipoxia tisular), los riñones reaccionan secretando eritropoyetina, una hormona que estimula el proceso de formación de eritrocitos, proceso conocido como eritropoyesis. La eritropoyesis se produce principalmente en la médula ósea y está regulada por una serie de etapas que comienzan a partir de una célula madre hematopoyética pluripotente. Esta da lugar a una unidad formadora de células denominada CFU-GEMM que es compartida por otros tipos de células (glóbulos blancos, plaquetas, etc.). El proceso continúa a través de diversas divisiones y diferenciaciones celulares produciendo varios tipos de células tal y como se describe en la Figura 6.2: unidad formadora de brotes eritroides (BFU-E), unidad formadora de células eritroides (CFU-E), proeritroblastos, eritroblastos (basófilos, policromatófilos y ortocromatófilos), retículo inmaduro, retículo maduro y, finalmente, eritrocitos.

La eritropoyetina comienza a aparecer en las BFU-E maduras y más intensamente en las CFU-E y los proeritroblastos. El efecto de la hormona en estas células es doble: evita la muerte celular programada o apoptosis, permitiendo que las células completen su maduración hasta convertirse en eritrocitos y, además, incrementa la población debido a que estimula la proliferación de las mismas (Fried, 2009).

Aparte del número de eritrocitos, existen otros factores que influyen en la concentración plasmática de Hb. Los precursores eritroides necesitan hierro para sintetizar correctamente la Hb. Cuando el nivel de hierro almacenado en el organismo es insuficiente el proceso de síntesis de Hb es defectuoso, produciendo eritrocitos bajos en Hb y con una capacidad de transportar oxígeno limitada. Incluso si la cantidad de hierro almacenado es óptima, existe la posibilidad de que el hierro no esté disponible durante la eritropoyesis. El hígado produce una hormona llamada hepcidina que regula el metabolismo del hierro. La hepcidina tiene la capacidad de bloquear la absorción de hierro en el duodeno e interrumpir su transporte desde las células hasta el plasma sanguíneo. Ante determinadas situaciones, como por ejemplo cuando aparece inflamación, la producción de hepcidina se incrementa como mecanismo de defensa ante organismos patógenos que requieren hierro para reproducirse. Puesto que normalmente los pacientes con ERT también sufren otras enfermedades, por

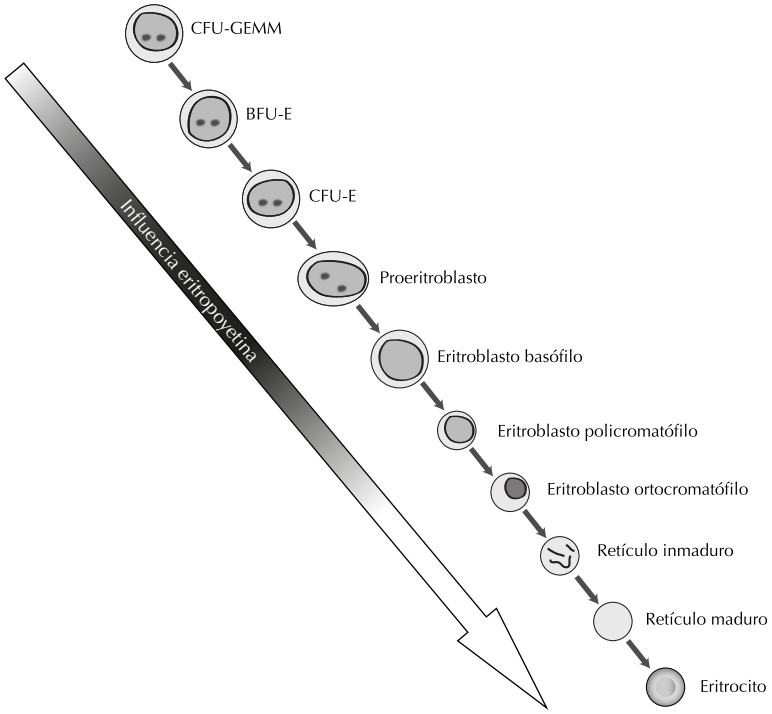


Figura 6.2
Linaje celular de los eritrocitos e influencia de la eritropoyetina (la influencia es mayor en las zonas oscuras).

ejemplo diabetes o hipertensión, por nombrar dos de las más habituales, es común que tengan niveles de inflamación más altos de lo habitual. En consecuencia también producen niveles elevados de hepcidina, lo que provoca un bloqueo del hierro e impide el correcto desarrollo de la eritropoyesis.

Un tercer factor que también puede ser determinante en el desarrollo de la anemia renal es el estado nutricional del paciente. Los pacientes con ERT son proclives a la anorexia y suelen seguir dietas restrictivas, por lo que a menudo presentan deficiencias nutricionales y vitamínicas. El proceso de diálisis también puede causar la pérdida de determinados nutrientes. La vitamina B-12 y el ácido fólico son los dos nutrientes que mayor correlación han mostrado con la aparición de anemia renal hasta el momento (Elliott et al., 2009). Sin embargo, debido a la elevada complejidad del proceso, se ha sugerido la implicación de muchos otros nutrientes (Nissenson y Fine, 2007).

Otras causas menos comunes que afectan a la producción de Hb son el hiperparatiroidismo, la presencia de exceso de aluminio en sangre y los niveles reducidos en la concentración plasmática de levocarnitina entre otras (Nissenson y Fine, 2007).

6.3.2. Tratamiento

Antes del descubrimiento de la eritropoyetina recombinante humana (rHuEPO), muchos de los pacientes con ERT requerían habitualmente transfusiones de sangre para sobrevivir. La rHuEPO es una versión sintética de la eritropoyetina producida mediante técnicas de ADN recombinado y que actúa de manera similar a la hormona natural. El primer fármaco basado en rHuEPO fue la epoetina alfa, a la que más tarde siguieron otros como la epoetina beta, darbepoetina alfa o más recientemente, aprobado en Estados Unidos a principios de 2008, CERA (*continuous erythropoietin receptor activator*). Todos estos fármacos se agrupan bajo el nombre de agentes estimulantes de la eritropoyesis (AEEs). Las primeras pruebas clínicas con rHuEPO se iniciaron en 1985 y en pocos años se convirtió en el tratamiento estándar, lo que supuso una revolución para el tratamiento de la anemia renal.

A pesar de haber sido ampliamente utilizados en las últimas décadas, el uso adecuado de los AEEs dista de ser trivial, ya que requiere del conocimiento de su farmacología, sus efectos clínicos y efectos secundarios, aspectos prácticos sobre su administración, los costes asociados con la terapia y un conocimiento de las prácticas clínicas recomendadas relacionadas con su uso (Nissenson y Fine, 2007).

Los beneficios clínicos del tratamiento con AEEs en pacientes en hemodiálisis han sido documentados en numerosos estudios (Adamson y Eschbach, 1990, 1998; Collins et al., 2001). El incremento del nivel de Hb alcanzando durante el tratamiento con AEEs está correlacionado con una mejora en los índices de calidad de vida (menor fatiga, depresión, deficiencias cognitivas, intolerancia al ejercicio, etc.), menor riesgo de mortalidad, reducción en el número y duración de las hospitalizaciones y un decremento de las disfunciones cardiacas. Por otra parte la terapia basada en AEEs también está asociada con diversos efectos secundarios. Los más significativos son la trombosis del acceso vascular, convulsiones e hipertensión. Por tanto, durante la terapia con AEEs, el paciente debe ser controlado periódicamente para minimizar los riesgos asociados con sus efectos secundarios (Elliott, 2008).

La cantidad de hierro ingerida por los pacientes en hemodiálisis a través de su dieta suele ser insuficiente para cubrir las necesidades de la eritropoyesis, especialmente después de la administración de AEEs. Debido a ello, la terapia con AEEs suele complementarse con la administración de hierro (Eschbach, 2005). Muchos centros de diálisis también suministran rutinariamente ácido fólico para compensar las pérdidas provocadas por el proceso de diálisis (Nissenson y Fine, 2007).

6.4. Modelo de la eritropoyesis durante el tratamiento con darbepoetina alfa

Los experimentos llevados a cabo en este capítulo están basados en un modelo computacional que describe el proceso de eritropoyesis durante el tratamiento con

darbepoetina alfa. Actualmente, este fármaco de tipo AEE es el más utilizado para tratar la anemia renal tanto en la Unión Europea como en Estados Unidos. Antes de aplicar cualquier sustancia, medicamento o técnica en pacientes reales, es necesario llevar a cabo un ensayo clínico para evaluar su seguridad y eficacia. Sin embargo, debido al alto coste que conlleva este tipo de pruebas y el compromiso que supone para la salud de los pacientes implicados, se debe demostrar previamente que el tratamiento que se pretende evaluar puede ser beneficioso para el paciente. Una posible forma de hacerlo es mediante el uso de modelos computacionales.

El modelo computacional utilizado en este trabajo ha sido desarrollado por el departamento *Health Advanced Modelling and Business* de la empresa Fresenius Medical Care en colaboración con el grupo de investigación *Intelligent Data Analysis Laboratory* (IDAL), grupo al que pertenece al autor de la presente tesis. La descripción detallada del modelo se ha incluido en el Apéndice A, por lo que en esta sección únicamente se presentan sus características principales.

Durante la última década se han desarrollado diversos modelos farmacodinámicos que describen la respuesta hematológica a diferentes tipos de AEEs (Uehlinger et al., 1992; Krzyzanski et al., 1999, 2005; Pérez-Ruixo et al., 2005; Krzyzanski y Pérez-Ruixo, 2007; Woo y Jusko, 2007). El modelo introducido aquí se centra en pacientes con ERT y tratados mediante la administración intravenosa de darbepoetina alfa. Las poblaciones de células hematopoyéticas (neutrófilos, eritrocitos, plaquetas, etc.) son ejemplos típicos de sistemas biológicos que se rigen por el ciclo de vida celular. Tal y como se describió en la Sección 6.3.1, los eritrocitos se forman en la médula ósea, donde pasan por una serie de diferenciaciones antes de comenzar a circular en la sangre como reticulitos durante 1-2 días y, posteriormente, como eritrocitos maduros. El tiempo de vida media de los eritrocitos es de 120-140 días en condiciones normales, mientras que en pacientes con ERT se reduce hasta, aproximadamente, 70-90 días (Uehlinger et al., 1992).

La evolución de la concentración de Hb tras administrar darbepoetina alfa se puede describir mediante un modelo multi-compartimental (Jacquez, 1985). Los diferentes tipos de células implicados en la formación de eritrocitos se han agrupado en clases de poblaciones en función de cómo interaccionan con la eritropoyetina (EPO), dando lugar cada clase a un compartimento. El número de células en todos los compartimentos depende de la concentración plasmática total de EPO, la cual consiste en la suma de la hormona producida de forma natural, Ep, y la administrada exógenamente (darbepoetina alfa), E:

$$E_{tot} = E + Ep. \tag{6.1}$$

Se asume que la cantidad de EPO endógena producida en el cuerpo se mantiene constante.

Cuando se administra darbepetina alfa por vía intravenosa, la cantidad total de fármaco se inyecta en un periodo de tiempo pequeño, por lo que es posible asumir que la concentración plasmática de EPO exógena en el momento de la inyección, t_0, es exactamente igual a la cantidad de EPO inyectada. El resultado es un incremento

repentino de la concentración de EPO seguido por un decaimiento exponencial. Este proceso se puede describir mediante una ecuación diferencial de primer orden con una tasa de eliminación constante:

$$E'(t) = -\frac{24}{25}\log(2)E(t), \quad E(t_0) = \frac{D_0}{V_d}, \tag{6.2}$$

donde $E(t)$ es la concentración de EPO exógena en el instante t, D_0 la dosis de fármaco en t_0, y V_d es el volumen de distribución en estado estacionario, el cual se ha fijado como $V_d = 52.4$ (Allon et al., 2002).

El nivel de Hb en el instante t es directamente proporcional a la concentración de eritrocitos $R(t)$:

$$Hb(t) = \text{MCH} \cdot R(t), \tag{6.3}$$

donde MCH denota la concentración de Hb corpuscular media cuyo valor depende del género del paciente (Goldman y Schafer, 2011):

$$\text{MCH} = \begin{cases} 2.7 \text{ g} \cdot \left(10^{11} \text{ células}\right)^{-1} & \text{para hombres.} \\ 2.4 \text{ g} \cdot \left(10^{11} \text{ células}\right)^{-1} & \text{para mujeres.} \end{cases}$$

La Figura 6.3 muestra un diagrama de bloques del modelo compartimental. Se han definido tres clases de células o compartimentos: clase P, formada por células progenitoras y precursoras (CFU-GEMM, BFU-E y CFU-E); clase M, que incluye los eritroblastos (basófilos, policromatófilos y ortocromatófilos) y retículos inmaduros; y, finalmente, clase R, a la que pertenecen los retículos maduros y los eritrocitos. La clase P depende de la eritropoyetina, tanto endógena como exógena. En la clase M es el hierro el elemento que regula la producción de Hb. Finalmente, la concentración plasmática de Hb viene determinada por el número de células en la clase R y el parámetro MCH.

El sistema de ecuaciones diferenciales que describe el número de células en cada compartimento se ha incluido en el Apéndice A. Además de la EPO endógena (Ep) y la concentración de Hb corpuscular (MCH), el modelo tiene otros dos parámetros, Cp y Cr, que dependen de las características individuales de los pacientes y que sirven para modular la respuesta de cada paciente a la darbepoetina alfa. El valor de MCH depende únicamente del sexo del paciente, mientras que los parámetros restantes se calculan con un ajuste por mínimos cuadrados a partir de los datos de laboratorio recogidos en los últimos 6 meses. La solución del sistema de ecuaciones diferenciales se ha obtenido mediante la instrucción *dde*23 de MATLAB 7.14 (R2012a) (Shampine y Thompson, 2001). Esta instrucción implementa un método numérico basado en Runge-Kutta y permite resolver ecuaciones diferenciales con retardo constante.

El modelo computacional, como cualquier otro modelo, es una simplificación del problema real. Hay dos supuestos fundamentales en los que se basa el modelo: por una parte, se asume que el paciente tiene un nivel de inflamación estable, y por otra, que la disponibilidad de hierro para el proceso de eritropoyesis es constante.

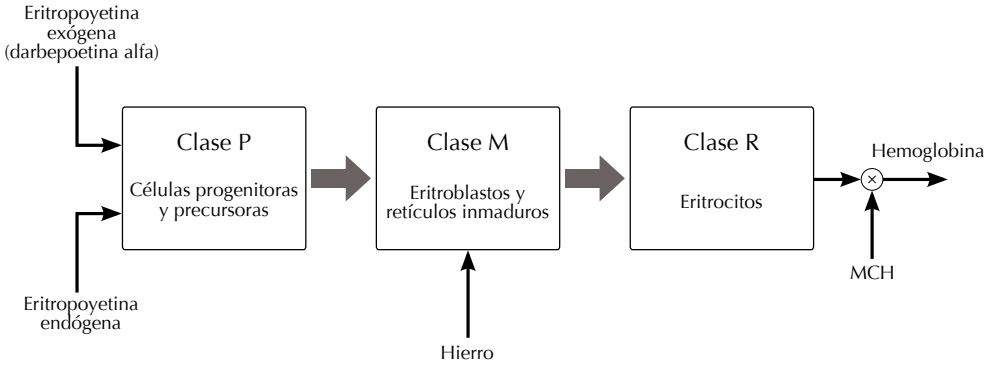


Figura 6.3
Diagrama de bloques del modelo compartimental que describe el nivel de hemoglobina durante el tratamiento con darbepoetina alfa.

Ambos supuestos están relacionados entre sí ya que la inflamación puede afectar a la disponibilidad de hierro (ver Sección 6.3.1). En general, estos supuestos no se cumplen para todos los pacientes durante el periodo completo del tratamiento. Aún así, el modelo es capaz de reproducir la variabilidad en la respuesta de los pacientes al tratamiento y los efectos a largo plazo que produce cada dosis de darbepoetina alfa en el nivel de Hb.

En la Figura 6.4 se compara el nivel de Hb calculado por el modelo (línea continua azul) con el nivel medido (asteriscos rojos) mediante análisis de laboratorio en cuatro pacientes diferentes en un periodo aproximado de 26 meses. En los cuatro casos tanto el nivel de inflamación como la disponibilidad de hierro permanecen aproximadamente constantes. Se puede observar que el modelo puede simular con una precisión razonable el nivel de Hb.

6.5. Formulación del problema mediante MDPs

6.5.1. Tratamiento de la anemia renal: un problema secuencial de toma de decisiones

Los síntomas de la anemia y la respuesta al tratamiento a menudo varían en función de las características del paciente: peso, raza, género, grado de ERT, comorbilidades, medicaciones concurrentes, etc. Durante el periodo de tratamiento es necesario monitorizar su estado clínico por lo que todos los pacientes están sujetos a revisiones mensuales. Una revisión típica consiste en un análisis clínico donde se miden diversas variables que sirven para evaluar el estado del paciente, junto con la prescripción de fármacos que debe recibir hasta la próxima revisión en el siguiente mes. Por lo tan-

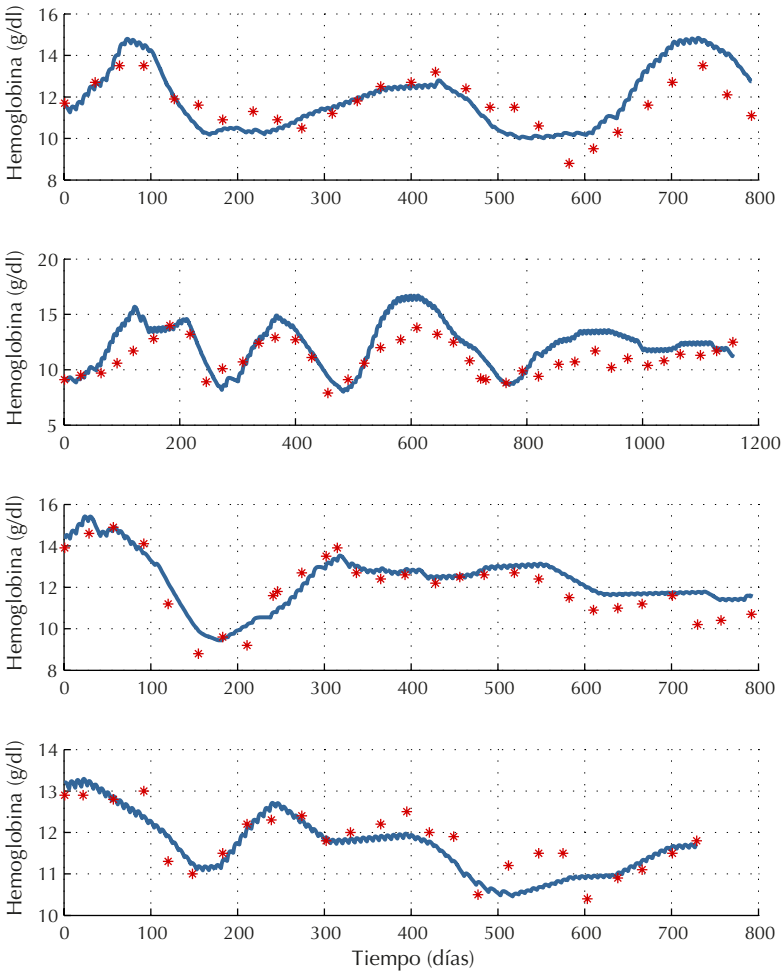


Figura 6.4
Nivel de hemoglobina obtenido mediante el modelo computacional (línea continua azul) y medido mediante análisis de laboratorio (asteriscos rojos) correspondiente a cuatro pacientes durante, aproximadamente, 26 meses de tratamiento. En los cuatro casos se puede considerar que los supuestos en los que se basa el modelo se cumplen.

to, a lo largo de la terapia, cada paciente genera una secuencia de observaciones y tratamientos mensuales de la forma (Lizotte et al., 2012):

$$o_1^j, t_1^j, o_2^j, t_2^j, \cdots, o_T^j, a_T^j, o_{T+1}^j \tag{6.4}$$

donde o_k^j es el estado del paciente j-ésimo en el mes k, t_k^j es la dosis t de darbepoetina alfa para el mismo paciente y mes, y T es la duración del tratamiento completo.

A partir del conjunto de secuencias generado por los pacientes se puede distinguir fácilmente un problema secuencial de toma de decisiones, donde las acciones (o decisiones) se corresponden con la dosis de fármaco prescrito a cada paciente en los diferentes meses (o etapas). Antes de aplicar cualquier método basado en RL es necesario definir una función $s(o_1, a_1, \cdots, o_k, a_k)$ que asigne un estado de acuerdo a la historia y condición clínica del paciente, así como una función escalar de recompensa $\rho(s_k, a_k, s_{k+1})$ que evalúe la transición desde el estado s_k hasta s_{k+1} después de realizar la acción a_k. El resultado de aplicar ambas funciones a (6.4) es un conjunto de secuencias de estados, acciones y recompensas:

$$s_1^j, a_1^j, r_1^j, s_2^j, a_2^j, r_2^j, \cdots, s_T^j, a_T^j, r_T^j \tag{6.5}$$

Dichas secuencias se pueden reescribir como transiciones del tipo (s, a, r, s'). Una vez obtenidas la transiciones, es posible aplicar el algoritmo FQI para calcular una política que seleccione dosis óptimas (de acuerdo con los datos disponibles) de darbepoetina alfa en función de la situación clínica del paciente.

En los siguientes subapartados se detalla la definición de los estados, acciones y función de recompensa para el caso del tratamiento de la anemia renal.

6.5.2. Espacio de estados y acciones

El marco de los MDPs asume que se cumple la propiedad de Markov tanto en la dinámica de transición de estados como en la distribución de la función de recompensa. A efectos prácticos, la condición de Markov implica que, dado un estado s_k y acción a_k, el estado siguiente s_{k+1} y la recompensa r_k son condicionalmente independientes de todos los estados, acciones y recompensas anteriores. Un método obvio para lograr que los estados cumplan la propiedad de Markov es incluir en el estado actual s_k toda la historia previa $(s_0, a_0, s_1, \ldots, a_{k-1})$. En la práctica, este planteamiento no es viable por dos razones:

1. Cada vez que el número de etapas k aumenta, el estado debería incluir dos nuevas variables. Debido a la conocida como "maldición de la dimensionalidad" (ver Sección 3.2), la cantidad de datos necesarios para aprender políticas útiles aumenta rápidamente con la dimensionalidad del espacio de estados, por lo que un MDP con cientos de dimensiones sería intratable.

2. El espacio requerido para almacenar los datos crecería indefinidamente conforme k aumenta.

Idealmente, el espacio de estados debe contener suficiente información para proporcionar un "resumen" de la historia previa y que se cumpla la propiedad de Markov. Al mismo tiempo el número de variables de estado debe ser tan reducido como sea posible para permitir el aprendizaje de políticas a partir de una cantidad de datos limitada. La definición de un espacio de estados adecuado puede ser determinante para aplicar con éxito los algoritmos de RL y, a menudo, suele requerir de un conocimiento profundo del problema abordado.

En el caso de administración de darbepoetina alfa, el espacio de estados del MDP se ha definido como:

$$s = [Hb_k, \Delta Hb_k, DA_k, DA_{k-1}, DA_{k-2}, Pat_{grupo}]$$

donde Hb_k mide el grado de anemia en el mes actual; ΔHb_k, definido como la diferencia entre el nivel de Hb en el mes actual y el anterior, es un indicador de la tendencia de cambio de Hb; y DA_k es la dosis de darbepoetina alfa en el mes k. Debido a que cada administración de darbepoetina afecta al nivel de Hb durante aproximadamente 90 días (el tiempo de vida media de los eritrocitos), la Hb actual del paciente también está influenciada por las dosis administradas en los dos meses previos, razón por la cual DA_{k-1} y DA_{k-2} se han incluido como variables de estado. Dado que el objetivo es que la política personalice el tratamiento, el espacio de estados también debe contener información acerca de las características individuales de cada paciente. En el modelo computacional esta información viene dada por los parámetros MCH, Ep, Cp y Cr. Dichos parámetros podrían introducirse directamente como variables de estado, sin embargo, se ha utilizado el conocimiento previo sobre el problema para reducir la dimensionalidad del espacio de estados. La variable MCH únicamente puede tomar dos valores dependiendo del sexo del paciente. Se decidió calcular una política separada para hombres y mujeres, lo que permite prescindir de la variable MCH. Las variables restantes (Ep, Cp y Cr) fueron analizadas mediante una técnica de agrupamiento o *clustering*, concretamente se utilizó el algoritmo *k-means* (Alpaydin, 2004). El objetivo de este análisis fue encontrar grupos de pacientes con características similares. La idea de esta aproximación es incluir en el espacio de estados únicamente el grupo al que pertenece el paciente en lugar de todas las variables que definen sus características. Es decir, la salida del algoritmo *k-means*, Pat_{grupo}, se puede considerar como una variable que proporciona información sobre las características de los pacientes de forma compacta. Este método asume que una determinada dosis producirá efectos similares entre los pacientes del mismo grupo.

Respecto al espacio de acciones, se ha optado por utilizar un conjunto de acciones discretas. Existen algoritmos de RL que permiten el uso de espacios de acciones continuos (Hafner y Riedmiller, 2011; Lazaric et al., 2007), sin embargo, típicamente tienen que resolver un problema de optimización no convexo en cada iteración, lo cual solo se puede hacer de forma aproximada e implica un coste computacional considerable. La solución más habitual consiste en discretizar el espacio de acciones en un conjunto pequeño de valores tal y como se ha hecho en este problema. El conjunto de acciones

A se definió como:

$$A = \{0, 0.25, 0.50, 0.70, 1\} \ \mu\text{g/kg por semana.}$$

Estos valores corresponden con las posibles dosis que la política podrá administrar a los pacientes. Es importante notar que la dosis están normalizadas por el peso del paciente, por lo que serán diferentes para cada paciente dependiendo de su peso corporal. Los valores seleccionados coinciden aproximadamente con los indicados en la mayoría de protocolos estándar de ajuste de dosis, donde se recomienda una dosis inicial de 0.45 μg/kg e incrementos o decrementos del 25 %.

6.5.3. Función de recompensa

La función de recompensa define el objetivo de la política aprendida por el agente. El principal objetivo del tratamiento con AEEs es mantener el nivel de Hb en un rango saludable para el paciente. De acuerdo con la *European Best Practice Guidelines for the Treatment of Anemia in Chronic Kidney Disease* (Locatelli et al., 2004), la concentración de Hb debe ser superior a 11 g/dl en todos los pacientes y no mayor que 12 g/dl en pacientes con ciertas comorbilidades comunes como, por ejemplo, enfermedades cardiovasculares, diabetes, etc. En vista de estas recomendaciones, el rango objetivo utilizado en este trabajo fue [11, 12] g/dl. Otro factor a tener en cuenta a la hora de administrar AEEs es evitar cambios demasiado bruscos de Hb. Concretamente, cambios (tanto incrementos como decrementos) cercanos a 2 g/dl en un tiempo igual o inferior a cuatro semanas aumentan el riesgo de sufrir episodios cardiovasculares (Locatelli et al., 2004).

Para satisfacer ambos requisitos, la recompensa se diseñó como una función por partes que depende del valor actual de hemoglobina, Hb_k. Cuando Hb_k esté cercano al rango objetivo, la política debe intentar alcanzar dicho rango en el mes siguiente. Sin embargo, si la diferencia entre Hb_k y el objetivo es demasiado grande, el cambio necesario para lograr un nivel de Hb adecuado en un solo mes será demasiado abrupto y debe ser evitado.

La recompensa que recibe el agente cuando $10.5 < Hb_k < 12.5$ se ha definido como sigue:

$$\rho_{Hb}(s_k, a_k) = 1 - \tanh^2\left(\left|\frac{Hb_{k+1} - 11.5}{g}\right| \cdot w\right) \tag{6.6}$$
$$w = \tanh^{-1}\left(\frac{\sqrt{0.95}}{0.01}\right)$$

donde $g = 0.5$ es un parámetro que controla la pendiente de la función. Esta función depende de Hb_{k+1} y tiene una forma acampanada cuyo máximo coincide con $Hb_{k+1} = 11.5$ g/dl. Es decir, la recompensa es máxima cuando el nivel de Hb alcanzado está en el centro del rango objetivo. Conforme el valor de Hb_{k+1} difiere de 11.5 g/dl, la

recompensa decrece suavemente hasta cero, indicando que valores próximos al objetivo son preferibles frente a otros más lejanos. En la Figura. 6.5a se muestra la forma de ρ_{Hb} en función del nivel de Hb.

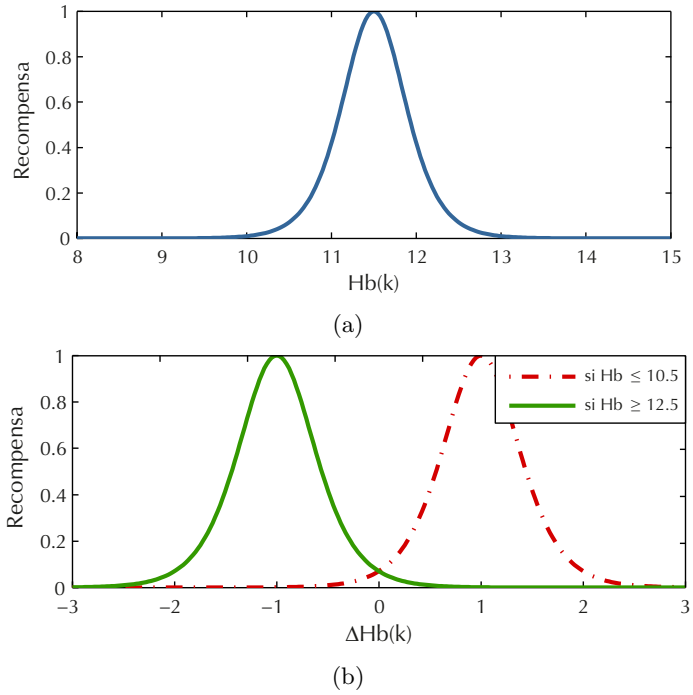


Figura 6.5
Representación de las diferentes partes que forman la función de recompensa. (a) se aplica cuando $10.5 < Hb_k < 12.5$; el agente obtiene la máxima recompensa cuando el nivel de Hb está en el centro del rango objetivo. (b) se aplica cuando $Hb_k \geq 12.5$ (línea continua) o $Hb_k \leq 10.5$ (línea punteada); la recompensa asignada es máxima si el nivel de Hb se reduce o aumenta, respectivamente, en 1 g/dl en un periodo de un mes.

Cuando $Hb_k \geq 12.5$, se ha fijado como objetivo decrementar el nivel de Hb en 1 g/dl:

$$\rho_{-\Delta Hb}(s_k, a_k) = 1 - \tanh^2\left(\left|\frac{\Delta Hb_{k+1} + 1}{g}\right| \cdot w\right) \tag{6.7}$$

donde g y w son iguales que en la Ecuación (6.6) y ΔHb_{k+1} es la diferencia de Hb entre el mes actual y el mes siguiente, es decir, $Hb_{k+1} - Hb_k$. Aunque la forma de la función es la misma que la definida en la Ecuación (6.6), aquí la recompensa no depende del valor de Hb en el mes siguiente, sino de ΔHb_{k+1}. Es decir, la recompensa máxima corresponde con una variación de Hb en lugar de un determinado valor,

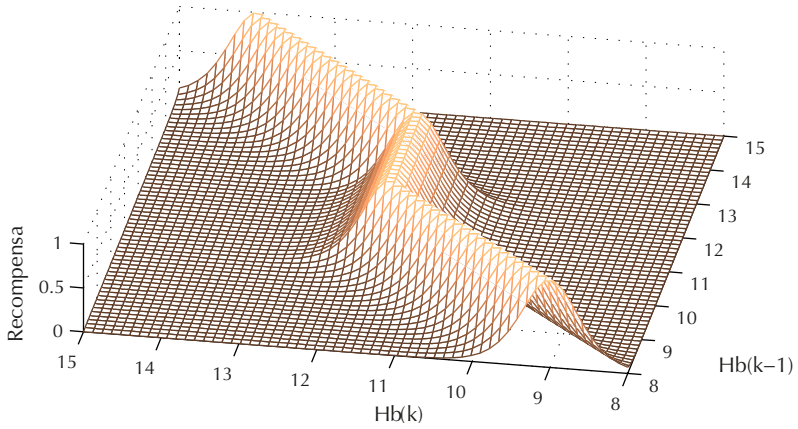


Figura 6.6
Representación de la función de recompensa completa para diferentes valores de Hb_k y Hb_{k+1}.

concretamente -1 g/dl por mes. Variaciones cercanas a -1 g/dl reciben recompensas cercanas a la máxima y van decreciendo conforme se distancian de ese valor.

De forma similar, si $Hb_k \leq 10.5$, al agente debe alcanzar un incremento de 1 g/dl por mes. En este caso la función de recompensa correspondiente es:

$$\rho_{+\Delta Hb}(s_k, a_k) = 1 - \tanh^2\left(\left|\frac{\Delta Hb_{k+1} - 1}{g}\right| \cdot w\right) \tag{6.8}$$

En la Figura 6.5b se muestran las recompensas aplicadas cuando $Hb_k \leq 10.5$ y $Hb_k \geq 12.5$.

Finalmente, la función de recompensa completa se define como:

$$\rho(s_k, a_k) = \begin{cases} \rho_{Hb} & \text{si } 10.5 < Hb_k < 12.5 \\ \rho_{-\Delta Hb} & \text{si } Hb_k \geq 12.5 \\ \rho_{+\Delta Hb} & \text{si } Hb_k \leq 10.5 \end{cases} \tag{6.9}$$

En la Figura 6.6 se ha representado $\rho(s_k, a_k)$ para diferentes valores de Hb_k y Hb_{k+1}.

6.6. Experimentos

El objetivo de los experimentos fue evaluar el método propuesto y comparar la calidad de la política aprendida mediante RL con un protocolo de ajuste de dosis estándar. Tal y como se muestra en el diagrama de bloques de la Figura 6.7, los

experimentos se llevaron a cabo en tres etapas. En la primera etapa se utilizó el modelo computacional para generar la experiencia (datos) requerida por el algoritmo de RL. Concretamente se simularon los datos correspondientes a una población de 5000 pacientes en hemodiálisis y tratados con darbepoetina alfa durante 30 meses. En la segunda etapa los datos se procesaron para transformarlos en un conjunto de estados, acciones y recompensas. Posteriormente se aplicó el algoritmo FQI con el fin de obtener una política de administración de darbepoetina alfa. Finalmente, en la tercera etapa se simuló una nueva población de 60 pacientes, a los cuales se les aplicó el tratamiento indicado por la política aprendida y el tratamiento recomendado en el protocolo. A continuación se describen los detalles de cada una de las etapas.

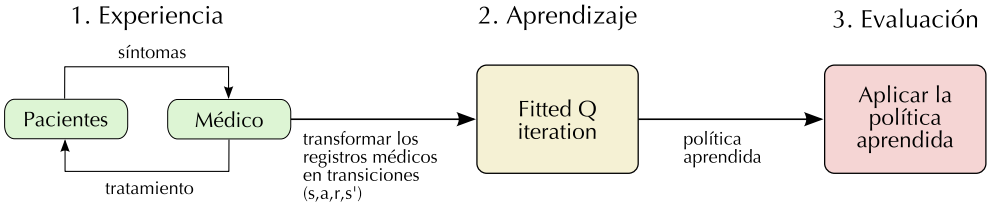


Figura 6.7
Diagrama de bloques de las tres etapas llevadas a cabo en los experimentos. En la primera etapa se han simulado los datos clínicos que se generan durante el tratamiento de los pacientes con anemia renal. Después de procesar dichos datos, en la segunda etapa, se ha aplicado el algoritmo FQI. Finalmente, se han evaluado los resultados obtenidos en un nuevo conjunto de pacientes simulados.

6.6.1. Experiencia

El modelo computacional se empleó para generar un conjunto de datos que simula los datos médicos almacenados durante la práctica clínica habitual. La respuesta individual de cada paciente se describe en el modelo mediante cuatro parámetros (MCH, Ep, Cr y Cp) que se ajustan a partir de la historia clínica del paciente. Se utilizaron datos de 128 pacientes sometidos a hemodiálisis y que reciben tratamiento con darbepoetina alfa. Los datos se recogieron entre enero de 2008 y diciembre de 2010 en tres centros de hemodiálisis localizados en Italia. En la Tabla 6.1 se muestra un resumen de las características principales de los pacientes.

Dado que el método propuesto se aplicó por separado a hombres y mujeres (ver Sección 6.5.2), los datos fueron divididos en dos grupos de acuerdo al género de los pacientes. Debido a la gran similitud de los resultados obtenidos para ambos grupos, el resto de este capítulo se centra únicamente en el caso de los hombres. Tras la división de los datos, se disponía de los parámetros ajustados de 69 pacientes. Para encontrar

Tabla 6.1
Características iniciales de la población empleada para ajustar los parámetros del modelo computacional.

Característica	(n = 128)
Edad (años)	58.53 ± 15.15
Sexo (% hombres)	53.91
Altura (m)	1.62 ± 0.09
Peso (kg)	67.97 ± 12.61
Tratamiento	
Darbepoetin alfa (μ/kg/semana)	0.29 ± 0.17
Hierro IV (mg/semana)	34.74 ± 32.08
Análisis clínico	
Hemoglobina (g/dl)	11.80 ± 1.16
Ferritina (ng/ml)	430.90 ± 178.96
Albúmina sérica (g/dl)	4.09 ± 0.29
Fosfato (mg/dl)	4.52 ± 1.03
Leucocitos (células/mm^3)	6584.1 ± 1260.9

Los datos están expresados en medias ± SD.

una política mediante RL con una capacidad de generalización adecuada, una población de 69 pacientes resultó ser insuficiente. De modo que se utilizó dicha población para generar, artificialmente, nuevos pacientes de acuerdo al siguiente procedimiento:

1. Dado un conjunto de pacientes P representados por los parámetros Ep, Cr y Cp, seleccionar arbitrariamente un paciente $pat_0 \in P$, tal que $pat_0 = [Ep_0, Cr_0, Cp_0]$.
2. Buscar los j pacientes más cercanos a pat_0 usando como criterio la distancia euclídea.
3. Escoger aleatoriamente un paciente $pat_f = [Ep_f, Cr_f, Cp_f]$ entre los j pacientes más cercanos.
4. Seleccionar un nuevo paciente pat_n en el espacio definido entre pat_0 y pat_f de forma aleatoria (ver Figura 6.8). Por ejemplo, el parámetro Ep_n estará dentro del rango $[Ep_0, Ep_f]$.
5. Añadir pat_n al conjunto P y volver al paso 1 hasta generar el número de pacientes deseado.

Este procedimiento permite generar artificialmente los parámetros de nuevos pacientes a partir de los parámetros reales. Puede considerarse un proceso similar a una interpolación pero añadiendo cierto grado de aleatoriedad en los datos generados. El número de pacientes j permite controlar la aleatoriedad introducida, la cual aumenta conforme j aumenta.

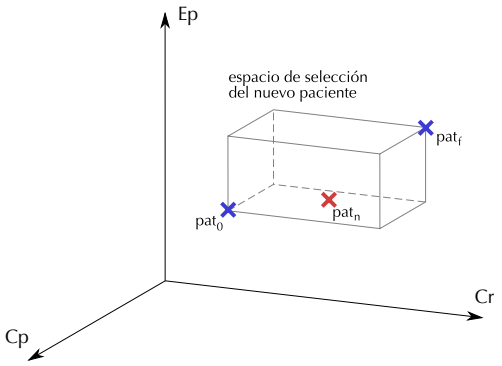


Figura 6.8
Procedimiento seguido para generar artificialmente los parámetros Ep, Cr y Cp a partir de los datos reales. Los parámetros del paciente artificial pat_n se escogen en el espacio definido entre el par de pacientes pat_0 y pat_f.

En total se generaron 4931 pacientes artificiales hasta llegar a tener una población de 5000 pacientes. En la Figura 6.9 se muestran los parámetros generados artificialmente usando $j = 3$ y los parámetros correspondientes a los 69 pacientes reales. Una vez generados los parámetros, se simuló la evolución clínica de los pacientes durante 30 meses de tratamiento. Teniendo en cuenta la ERT es una enfermedad crónica, los pacientes suelen estar bajo tratamiento durante largos periodos de tiempo. Sin embargo 30 meses fue suficiente para obtener una cantidad adecuada de datos.

Durante los 30 meses de tratamiento simulado, en lugar de seguir un protocolo determinado, las dosis administradas a los pacientes se seleccionaron aleatoriamente entre el conjunto discreto $A = \{0, 0.25, 0.50, 0.70, 1\}$ μg/kg/semana. Aunque este método introdujo una alta variabilidad respecto a las dosis, si todos los pacientes se hubiesen simulado siguiendo el mismo protocolo, la política aprendida estaría fuertemente sesgada por dicho protocolo. En el caso real los médicos prescriben tratamientos basándose en protocolos y en su propia experiencia, por lo que es habitual encontrar pacientes con condiciones similares y tratamientos diferentes. Por otra parte, en general, la dosis prescritas por los médicos están más cercanas de las óptimas que las dosis aleatorias usadas en este experimento, lo cual facilita el proceso de aprendizaje.

Las simulaciones generaron un total de $5000 \times 30 = 150000$ observaciones y tratamientos. Debido a la aleatoriedad del tratamiento, en algunas de las observaciones los niveles de Hb eran superiores a 20 g/dl. Estos datos fueron eliminados porque representan situaciones poco realistas. Tras esta restricción el conjunto de datos estaba formado por un total de 138011 observaciones.

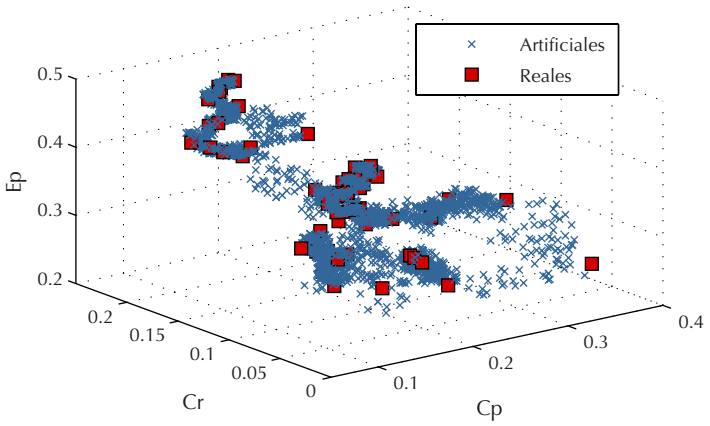


Figura 6.9
Representación de los parámetros correspondientes a 69 pacientes reales (cuadrados rojos) y los generados artificialmente (cruces azules).

6.6.2. Aprendizaje

Los datos generados en la etapa anterior fueron utilizados como entrada del algoritmo de aprendizaje FQI. De acuerdo con la metodología introducida en la Sección 6.5, el conjunto de datos formado por observaciones y tratamientos se debe procesar para convertirlo en un conjunto de estados, acciones y recompensas de la forma (s, a, r, s'). Por otra parte, FQI requiere un método de aprendizaje supervisado para aproximar la función Q. Para estas tareas se emplearon los algoritmos *k-means* y *extremely randomized trees* (Geurts et al., 2006). Ambas técnicas, igual que FQI, dependen de uno o más parámetros que deben ser seleccionados por el usuario. En esta sección se discute la elección de dichos parámetros para el problema del tratamiento de la anemia así como otros aspectos prácticos sobre su implementación.

FQI está basado en un proceso iterativo que comienza con una aproximación arbitraria de la función Q óptima y, conforme aumenta el número de iteraciones, la calidad de la aproximación va mejorando. El número de iteraciones debe ser suficiente para permitir la convergencia hasta la función Q óptima. La velocidad de convergencia es desconocida *a priori* y puede variar dependiendo del problema tratado, por lo que el número de iteraciones se debe ajustar para cada problema concreto. Existen métodos para detener el proceso iterativo de forma automática pero, a su vez, dependen de algún parámetro que debe ser fijado de antemano (Ernst et al., 2005a). La opción más sencilla consiste en representar gráficamente la convergencia del algoritmo y fijar el número de iteraciones a un valor que se considere adecuado. La convergencia se puede medir en términos de la distancia entre dos aproximaciones consecutivas de la

función Q. Dado que tales aproximaciones son funciones continuas, la distancia se suele calcular sobre un número discreto de puntos que sean suficientemente representativos. En este caso se han utilizado todos los pares estado-acción que contiene el conjunto de transiciones para medir la distancia de acuerdo a la expresión:

$$\text{dist}(\widehat{Q}_n, \widehat{Q}_{n-1}) = \frac{\sum_{(s,a)\in S^i\times A}\left(\widehat{Q}_n(s,a) - \widehat{Q}_{n-1}(s,a)\right)^2}{\#(S^i \times A)} \tag{6.10}$$

donde $S^i \times A$ es el conjunto de pares estado-acción y el símbolo # indica número de elementos.

En la Figura 6.10 se ha representado la distancia entre $\widehat{Q}_n$ y $\widehat{Q}_{n-1}$ frente al número de iteraciones. Se puede observar que la curva de convergencia disminuye hasta permanecer prácticamente estable a partir de 35 iteraciones. En vista de este resultado, se fijó un máximo de 40 iteraciones. Otro parámetro que debe ser fijado es el factor de descuento γ, cuyo valor está relacionado con el horizonte temporal de la optimización. En el problema de la anemia renal, γ determina cuánto influye el nivel de Hb en los meses futuros para calcular la dosis de fármaco óptima en el mes actual. El factor de descuento se fijó empíricamente a $\gamma = 0.9$, este valor fue suficiente para estabilizar el nivel de Hb de los pacientes a largo plazo.

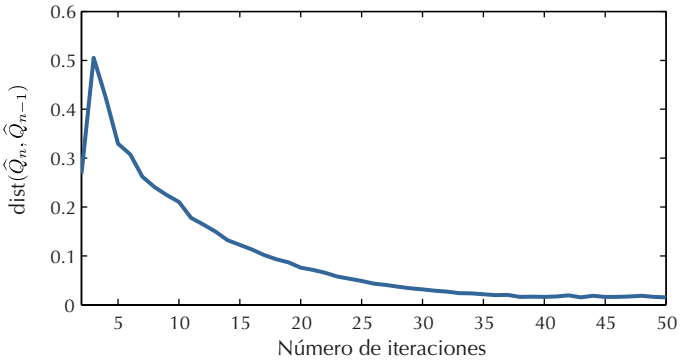


Figura 6.10
Convergencia del algoritmo *fitted Q iteration* medida como la distancia entre $\widehat{Q}_n$ y $\widehat{Q}_{n-1}$ frente al número de iteraciones.

El método de aprendizaje supervisado *extremely randomized trees* forma parte de lo que se conoce como comités (también llamados métodos *ensemble*), es decir, se trata de varios modelos que se combinan para obtener un único modelo más potente. Los modelos que forman el comité son un tipo de árboles de regresión construidos mediante el algoritmo *extra-trees* (Geurts et al., 2006). En total hay tres parámetros que deben ser seleccionados: el número de árboles en el comité, M, el tamaño mínimo

de hoja de los árboles, l_{min}, y un parámetro del algoritmo *extra-trees*, K. La selección de los tres parámetros se hizo siguiendo el procedimiento descrito en (Ernst et al., 2005a). El número de árboles se fijó en 50; K se escogió igual al número de dimensiones del espacio de estados, es decir, $K = 6$; mientras que el tamaño mínimo de hoja se optimizó mediante validación cruzada para cada iteración de FQI entre los posibles valores $l_{min} = \{5, 10, 50, 100\}$. Los resultados de los experimentos realizados en (Ernst et al., 2005a) demuestran que, en general, estos valores proporcionan resultados satisfactorios.

En cuanto al método de agrupamiento, el algoritmo *k-means* comienza con una estimación arbitraria de c centroides (cada uno representa a un agrupamiento o grupo) y utiliza un método iterativo para optimizar la calidad de los grupos. Una posible limitación del algoritmo *k-means* es su sensibilidad respecto a la inicialización de los centroides. Por este motivo se suelen realizar diferentes inicializaciones aleatorias (10 en este caso) y seleccionar aquella que proporciona resultados más adecuados. El número de grupos debe ser seleccionado por el usuario mediante métodos empíricos. El procedimiento seguido fue realizar el análisis de agrupamiento utilizando diferentes valores de c, concretamente desde 3 hasta 10 y para cada valor los agrupamientos resultantes se evaluaron utilizando gráficos de silueta (Rousseeuw, 1987). Estos gráficos son específicos para validar si los grupos obtenidos son adecuados o no; finalmente, se seleccionó $c = 5$.

6.6.3. Evaluación

La política aprendida se evaluó mediante simulaciones en un nuevo conjunto de 60 pacientes cuyos parámetros fueron generados artificialmente. Durante la simulación los pacientes fueron tratados durante un periodo de 30 meses con las dosis de darbepoetina alfa indicadas por la política aprendida. Con el objetivo de tener un tratamiento de referencia, se repitieron las simulaciones con el mismo grupo de pacientes pero siguiendo el tratamiento recomendado por un protocolo estándar. Concretamente se empleó el protocolo descrito por la Agencia Europea de Medicamentos (EMA, 2013) para la administración de Aranesp^{TM} (una de las marcas comerciales que distribuyen la darbepoetina alfa). La indicaciones incluidas en el protocolo se reproducen a continuación:

- La dosis inicial para pacientes en hemodiálisis debe ser 0.45 μg/kg de peso corporal administrada semanalmente.
- Si el incremento de Hb es inadecuado (menor de 1 g/dl en cuatro semanas) se debe incrementar la dosis en aproximadamente un 25 %.
- La dosis no se debe incrementar más de una vez cada 4 semanas.
- Si el incremento de Hb es mayor de 2 g/dl en un periodo de 4 semanas se debe reducir la dosis en un 25 %.

- Si el nivel de Hb es mayor de 12 g/dl, la dosis se debe reducir un 25 %. Después de esta reducción, si el nivel de Hb continua aumentando, el tratamiento con Aranesp^{TM} debe ser interrumpido temporalmente hasta que el nivel comience a disminuir, momento en el cual se debe retomar el tratamiento con una dosis aproximadamente un 25 % menor que la dosis previa.

Todos los pacientes empleados en el proceso de evaluación fueron tratados durante los 4 meses previos con dosis aleatorias. El objetivo de dicho tratamiento fue obtener un conjunto de pacientes con un estado inicial heterogéneo. Por tanto, la recomendación del protocolo sobre la dosis inicial de fármaco no se ha aplicado en ningún caso.

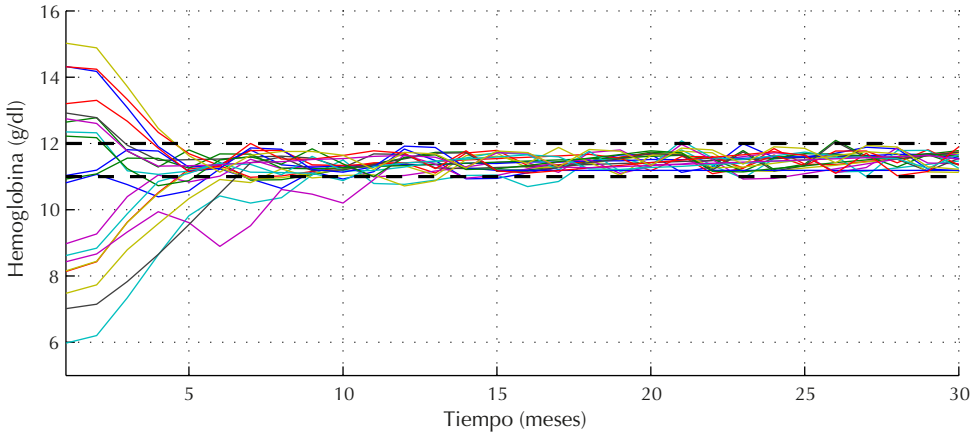
6.7. Resultados y discusión

En esta sección se analizan y comparan los resultados obtenidos por la política basada en RL y la política implementada a partir del protocolo, denotadas por π_{RL} y $\pi_{protocolo}$, respectivamente. Todos los resultados mostrados corresponden al conjunto de 60 pacientes usados para validación y 30 meses de tratamiento simulado.

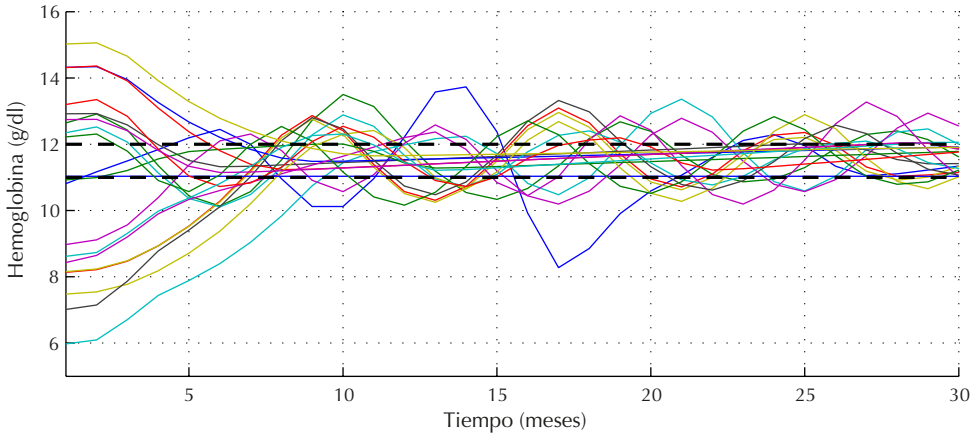
Para un primer análisis cualitativo, en la Figura 6.11 se ha representado la evolución temporal del nivel de Hb obtenido con ambas políticas. Por cuestiones de claridad, solo se han representado 19 pacientes seleccionados aleatoriamente de los 60 utilizados en el proceso de evaluación. La figura también incluye dos líneas horizontales punteadas en 11 y 12 g/dl que indican el rango de Hb objetivo. A pesar de que la figura no muestra información de todos los pacientes, es posible obtener una idea general de los resultados que obtiene cada política. El estado inicial de los pacientes es muy diverso, con niveles de Hb que van desde 6 g/dl hasta casi 15 g/dl. Como era de esperar, durante los primeros meses de tratamiento ambas políticas tienden a llevar el nivel de Hb de los pacientes hacia el rango objetivo. En este sentido π_{RL} es más efectiva, ya que esta tendencia es más pronunciada pero sin llegar a producir cambios de Hb > 2 g/dl. Los primeros signos de *Hb cycling* se empiezan a observar en los pacientes tratados con $\pi_{protocolo}$ a partir del séptimo mes. El nivel de Hb de algunos pacientes comienza a oscilar alcanzando valores de Hb fuera del rango objetivo. También se observa que este fenómeno no se produce en todos los casos, sino que hay determinados pacientes cuyo nivel de Hb permanece estable entre 11 y 12 g/dl. Por otra parte, la figura muestra que π_{RL} es capaz de reducir drásticamente el *Hb cycling*; en algunos pacientes la concentración de Hb oscila ligeramente, pero salvo alguna excepción, no sobrepasa los límites del rango objetivo.

La Figura 6.12 muestra la variación de Hb a lo largo del tratamiento con ambas políticas, pero en este caso en términos de nivel medio (líneas sólidas) y desviación estándar (áreas sombreadas)[2] para el conjunto completo de pacientes. Esta figura

[2]Asumiendo una distribución gaussiana de los niveles Hb, el área sombreada representa el 68.2 % de los datos.



(a)



(b)

Figura 6.11
Evolución temporal del nivel de hemoglobina correspondiente a 19 pacientes seleccionados aleatoriamente del grupo usado para evaluación. En (a) los pacientes han sido tratados de acuerdo a π_{RL}, mientras que en (b) de acuerdo a $\pi_{protocolo}$.

extiende a todos los pacientes las conclusiones obtenidas anteriormente para un subconjunto a partir de la Figura 6.11. Como se puede observar, tras un periodo de corrección de Hb, π_{RL} (en verde) estabiliza los pacientes dentro del rango objetivo. La estrecha desviación estándar indica que este comportamiento es similar en todos los casos. Cuando se compara con $\pi_{protocolo}$ (en rojo), la desviación estándar durante los primeros meses muestra que $\pi_{protocolo}$ necesita más tiempo para corregir el nivel de Hb. A partir del séptimo mes es posible observar las oscilaciones de Hb incluso en el nivel medio. Esto significa que dichas oscilaciones, en general, tienen una duración y fase similares ya que, de lo contrario, se cancelarían. Aunque el nivel medio de Hb correspondiente a $\pi_{protocolo}$ se encuentra dentro del rango objetivo, su elevada desviación estándar muestra que hay pacientes con niveles de Hb inadecuados durante todos los meses del tratamiento. Si se compara la amplitud de la desviación estándar entre los meses 10-15 y 25-30, se observa una ligera reducción, es decir, el fenómeno de *Hb cycling* tiende a reducirse. A pesar de dicha reducción, sus efectos todavía pueden apreciarse claramente después de 30 meses.

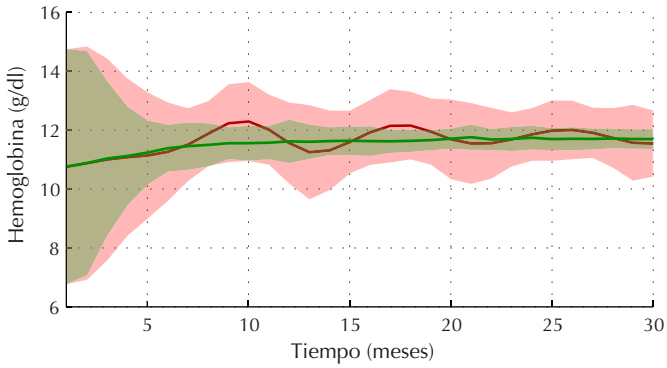


Figura 6.12
Nivel de hemoglobina medio (líneas sólidas) y desviación estándar (áreas sombreadas) a lo largo del tiempo para 60 pacientes tratados con π_{RL} (en verde) y $\pi_{protocolo}$ (en rojo).

La segunda métrica que se ha empleado para comparar las dos políticas es el porcentaje de observaciones, o meses, en los cuales los pacientes alcanzaron el nivel de Hb deseado. Se han definido cinco rangos de Hb que se han agrupado en tres categorías en función de su idoneidad:

- *Adecuada*: incluye los niveles de Hb dentro del rango objetivo, es decir, [11,12] g/dl. Se trata de la categoría más recomendable para los pacientes, permite corregir los síntomas de la anemia a la vez que minimiza los efectos secundarios del tratamiento.
- *Inadecuada*: compuesta por los niveles de Hb dentro de los límites (10,11) y (12,13) g/dl, los dos rangos contiguos al rango objetivo. No es recomendable

que los pacientes permanezcan durante periodos extensos de tiempo en dichos rangos, sin embargo, no supone un peligro inmediato para su supervivencia.

- *Peligrosa*: esta categoría contiene el resto de niveles de Hb, es decir, menores que 10 g/dl y mayores que 13 g/dl. Es la categoría que mayor compromete la salud de los pacientes y debe ser evitada.

En la Tabla 6.2 se muestra el porcentaje de meses en cada una de las categorías. Cuando los pacientes fueron tratados con π_{RL}, en el 82.1 % de los meses de tratamiento su nivel de Hb estuvo dentro de la categoría *adecuada*, lo cual supone una mejora del 27.6 % comparado con el 54.5 % logrado por $\pi_{protocolo}$. En el resto de meses los pacientes están principalmente situados en la categoría *inadecuada*, concretamente 11.02 % para π_{RL} y 34.1 % para $\pi_{protocolo}$. Este porcentaje indica que la política extraída del protocolo también obtiene resultados razonablemente buenos. Por último, hay un pequeño porcentaje de meses con pacientes en la categoría *peligrosa*, sin embargo, las Figuras 6.11 y 6.12 muestran que estos meses corresponden al comienzo del tratamiento. Cabe recordar que, en los primeros meses, hay pacientes con niveles extremos de Hb y que el tratamiento con darbepoetina alfa requiere tiempo para corregir la anemia ya que se deben evitar los cambios abruptos de Hb.

Tabla 6.2
Porcentaje de pacientes en cada una de las tres categorías definidas cuando son tratados con π_{RL} y $\pi_{protocolo}$.

Categoría	Nivel de hemoglobina (g/dl)	Pacientes observados (%)	
		π_{RL}	$\pi_{protocolo}$
Adecuada	[11,12]	82.10	52.30
Inadecuada	$(10, 11)$ y $(12, 13)$	11.02	34.12
Peligrosa	$\leq$10 y $\geq$13	6.88	11.40

La Figura 6.13 muestra, mediante un diagrama de barras, el porcentaje de estados en cada uno de los cinco rangos de Hb previamente definidos. Para cada rango hay dos barras, una correspondiente a $\pi_{protocolo}$ (negro) y otra a π_{RL} (en gris). En general el número de pacientes de $\pi_{protocolo}$ fuera del rango objetivo se reparte de forma homogénea en los niveles altos y bajos de Hb. Respecto a la política de RL se observa una ligera tendencia hacia valores de Hb inferiores al objetivo.

Además de corregir el nivel de Hb, el tratamiento con darbepoetina alfa debía evitar cambios abruptos de Hb. La diferencia entre π_{RL} y $\pi_{protocolo}$ referentes a este aspecto son inapreciables. Durante los experimentos, ambas políticas fueron realmente efectivas evitando cambios de Hb mayores de 2 g/dl por mes, condición que se cumplió en más del 99.5 % de los casos.

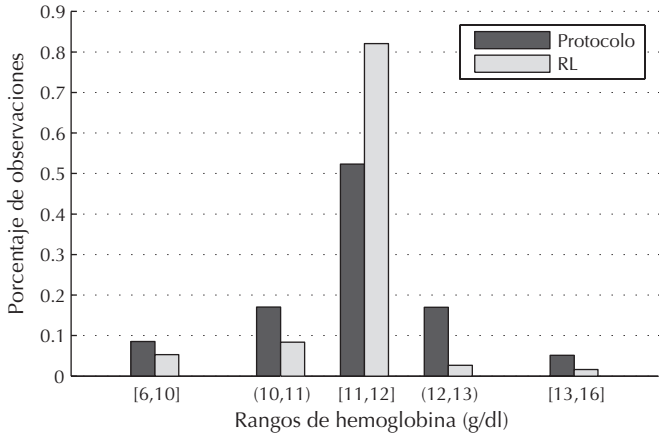


Figura 6.13
Porcentaje de pacientes obtenidos con π_{RL} y $\pi_{protocolo}$ en cada uno de los cinco rangos de hemoglobina definidos.

Debido al alto coste que supone los fármacos de tipo AEE, resulta también interesante analizar si la mejora lograda respecto al nivel Hb supone un incremento, o no, en el coste del tratamiento. Al comparar ambas políticas se obtiene una diferencia significativa ($p< 0.0001$, t-test pareado de cola única) entre la dosis media utilizada por $\pi_{protocolo}$ (0.39 μg/kg/semana) y π_{RL} (0.37 μg/kg/semana). Es decir, π_{RL} no solamente mejora el nivel de Hb sino que, además, reduce en un 5.13 % la cantidad de darbepoetina alfa empleada.

Finalmente, se estudió más detalladamente un caso particular para analizar las diferencias entre ambas políticas. La Tabla 6.3 muestra la evolución detallada de un mismo paciente cuando es tratado con ambas políticas durante 15 meses, incluyendo su nivel de Hb y la dosis de fármaco administrado por cada política. En concreto, el paciente estudiado corresponde a un caso donde $\pi_{protocolo}$ produce *Hb cycling*. Los niveles de Hb destacados en negrita indican que se encuentran dentro del rango objetivo. Se puede observar que, en el quinto mes de tratamiento, $\pi_{protocolo}$ comienza a incrementar la dosis de darbepoetina debido a que la Hb está por debajo de 11 g/dl, dicho incremento continúa durante los meses 6 y 7 hasta alcanzar un nivel de Hb adecuado. Sin embargo, después de alcanzar el rango objetivo, el nivel de Hb sigue creciendo a pesar de que la dosis ha disminuido (meses 9 y 10). Este fenómeno se debe al retardo que existe entre la administración del fármaco y los efectos que produce en la Hb del paciente. Por otra parte, en las dosis administradas por π_{RL}, se puede apreciar que dicho retardo se tiene en cuenta. Por ejemplo, en los meses 7 y 8 se mantiene la misma dosis a pesar de que el nivel de Hb está por debajo del objetivo.

En los siguientes meses la Hb sigue aumentando ligeramente hasta estabilizarse dentro del nivel deseado.

Tabla 6.3
Comparación del tratamiento y la evolución clínica de un paciente tratado con π_{RL} y $\pi_{protocolo}$. Los niveles de hemoglobina dentro del rango objetivo se han destacado en negrita.

Tiempo de tratamiento (meses)	π_{RL}		$\pi_{protocolo}$	
	Hb (g/dl)	Dosis de fármaco (μg/kg/semana)	Hb (g/dl)	Dosis de fármaco (μg/kg/semana)
1	12.65	0.50	12.65	0.75
2	12.78	0.75	12.91	0.56
3	**11.77**	0.75	12.43	0.42
4	**11.28**	0.50	**11.36**	0.42
5	**11.80**	0.50	10.44	0.52
6	**11.43**	0.50	10.13	0.65
7	10.89	0.50	10.57	0.82
8	10.91	0.50	**11.52**	0.82
9	**11.06**	0.50	12.72	0.61
10	**11.22**	0.50	13.50	0.46
11	**11.39**	0.50	13.14	0.34
12	**11.56**	0.50	12.11	0.34
13	**11.73**	0.25	**11.24**	0.34
14	**11.74**	0.50	10.90	0.43
15	**11.19**	0.25	**11.00**	0.54
Dosis total:		7.5		7.71

6.7.1. De pacientes simulados a pacientes reales

Los experimentos realizados a lo largo de este capítulo se han basado en un modelo computacional que simula la respuesta de los pacientes al tratamiento con darbepoetina alfa. Aunque la metodología propuesta es válida para aplicarla en pacientes reales, ciertos aspectos deben ser modificados. La principales diferencias entre el caso simulado y real se discuten a continuación:

- La inclusión de los parámetros del modelo (MCH, Ep, Cp, Cr) en la definición del espacio de estados es, probablemente, la diferencia más importante a la hora de aplicar el método descrito en pacientes reales. Estos parámetros están relacionados con las características individuales de los pacientes y se utilizan para encontrar grupos de pacientes que respondan de forma similar al tratamiento. En la práctica los parámetros no se pueden medir directamente. Aunque se podrían ajustar tal y como se hace en el modelo computacional, una opción más eficaz

es utilizar variables que se miden habitualmente en las revisiones mensuales y que aportan la misma información. Por ejemplo, el nivel de proteína C-reactiva, la albúmina sérica y la concentración de leucocitos se pueden emplear como indicadores del nivel de inflamación. Por tanto, el análisis de *clustering* se realizaría utilizando estas (junto con otras) variables en lugar de los parámetros del modelo.

- Algunos elementos del MDP, como la función de recompensa o el factor de descuento, se deben elegir cuidadosamente para que la política obtenida proporcione resultados adecuados. Cuando se usa un modelo es posible ajustar dichos elementos empleando procedimientos de prueba y error. En el caso de trabajar con pacientes reales, este tipo de procedimientos no son viables. La experiencia ganada con pacientes simulados es indudablemente valiosa y útil, pero aún así, en el caso real se requiere el asesoramiento de médicos expertos para diseñar el MDP.

- Un tercer aspecto que merece la pena mencionar son los supuestos en los que se basa el modelo: nivel de inflamación estable y disponibilidad de hierro para la eritropoyesis constante. Como ya se discutió anteriormente, estos supuestos no tienen por qué cumplirse cuando se trata con pacientes reales. El único requisito para que la política pueda manejar casos de pacientes donde se violen estos supuestos es que los datos utilizados para obtener la política contengan suficientes ejemplos de tales pacientes. Así pues, las limitaciones del modelo no suponen ningún problema cuando los datos empleados en el aprendizaje provienen de pacientes reales.

6.8. Conclusiones

En este capítulo se ha propuesto un método basado en RL para optimizar el tratamiento de la anemia en pacientes en hemodiálisis. Dicho tratamiento se ha formulado como un problema secuencial de toma de decisiones basado en el marco de los procesos de decisión de Markov (MDPs) para, posteriormente, aplicar un algoritmo de RL que ha permitido obtener una política óptima a partir de datos clínicos. Al contrario que otras técnicas para resolver MDPs, los algoritmos de RL no requieren un modelo de la dinámica del sistema, una característica que los hace especialmente útiles en problemas complejos como el abordado aquí.

El método propuesto se evaluó mediante un modelo computacional que describe el efecto del tratamiento en el nivel de Hb. Los resultados obtenidos se compararon con un protocolo estándar de ajuste de dosis. La política aprendida con RL incrementó un 27.6 % la proporción de pacientes con un nivel de Hb adecuado al mismo tiempo que redujo la cantidad de fármaco empleado en un 5.13 %. Dicha reducción supone un ahorro de aproximadamente 3.4 millones de euros anuales en una región con una

población similar a la de la Comunidad Valenciana[3].

Los resultados de la simulación sugieren que la política aprendida es capaz de tener en cuenta los efectos a largo plazo del tratamiento y la alta variabilidad en la respuesta de los pacientes. Estas dos características están ausentes en muchos de los protocolos empleados actualmente en los centros de diálisis y han sido señaladas como las principales causas del *Hb cycling*. En consecuencia, el método propuesto fue más eficaz que el protocolo estándar en mantener niveles de Hb dentro del rango objetivo y prevenir oscilaciones. Por otra parte, la cantidad de fármaco empleado por la política basada en RL fue menor, lo cual indica que administra la darbepoetina alfa de una forma más eficiente.

A pesar de que el modelo utilizado en los experimentos de este capítulo no es representativo de toda la casuística que pueda presentarse en la clínica, sí que reproduce los principales mecanismos del *Hb cycling*. Así pues los experimentos son válidos para demostrar el beneficio potencial del RL en el tratamiento de la anemia.

Los resultados obtenidos en este capítulo usando pacientes simulados han motivado el desarrollo de un sistema de soporte a la decisión clínica basado en RL. Actualmente (marzo, 2014), dicho sistema está siendo sometido a una evaluación clínica en cinco centros de hemodiálisis situados en tres países europeos.

[3]Este cálculo se ha realizado teniendo en cuenta que la prevalencia de la ERC (etapas 3-5) se estima en un 4.9 % de la población total, de los cuales el 12 % sufre anemia renal (McFarlane et al., 2011). El coste medio mensual del tratamiento con darbepoetina alfa en España es de 183.97 euros por paciente (Sanz-Granda, 2009). El número de habitantes de la Comunidad Valenciana es de aproximadamente 5.13 millones.

Capítulo 7

Conclusiones y proyección futura

7.1. Conclusiones

El aprendizaje por refuerzo puede considerarse como un campo del aprendizaje automático enfocado a la resolución de problemas de decisión secuenciales. Debido al carácter general de este tipo de problemas, el aprendizaje por refuerzo tiene aplicaciones en campos tan diversos como control automático, medicina, teoría de juegos, economía, investigación operativa, etc. Los problemas de decisión secuenciales también pueden ser abordados mediante otras técnicas, sin embargo, una de las características diferenciadoras del aprendizaje por refuerzo es que la solución planteada puede estar basada completamente en datos. Esta característica resulta indispensable en los problemas donde es difícil obtener un modelo que describa de forma precisa la evolución del sistema ante las diferentes acciones. Por otra parte, cada vez es más habitual almacenar de forma sistemática datos relativos a todo tipo de procesos, por lo que el número de aplicaciones potenciales del aprendizaje por refuerzo se ha incrementado de forma notable durante los últimos años.

A pesar de la utilidad del aprendizaje por refuerzo en diversos campos, los algoritmos actuales presentan limitaciones que dificultan su aplicación en problemas con un cierto grado de complejidad. Dos limitaciones que aparecen frecuentemente son la escalabilidad a espacios de estados con un número elevado de dimensiones y la gran cantidad de datos requerida para obtener políticas óptimas. La motivación de los algoritmos desarrollados a lo largo de esta tesis es resolver dichas limitaciones. A continuación se resumen las principales conclusiones que se desprenden de la investigación realizada en cada uno de los capítulos.

En el Capítulo 2 se han introducido los **fundamentos del aprendizaje por**

refuerzo. La primera parte del capítulo describe formalmente los diferentes elementos que componen el problema de RL mediante el marco matemático de los procesos de decisión de Markov. Posteriormente se han introducido los conceptos y algoritmos básicos de la programación dinámica para, finalmente, describir los algoritmos clásicos de RL. Además de los fundamentos teóricos del RL, el capítulo muestra la estrecha relación que existe entre los algoritmos de programación dinámica y RL. Estos últimos pueden considerarse como una extensión de los primeros para eliminar la necesidad de disponer del modelo del MDP y hacerlos computacionalmente más eficientes. Es decir, los algoritmos de RL permiten aplicar los principios de funcionamiento de la programación dinámica en un mayor número de problemas. A pesar de ello los algoritmos descritos en el Capítulo 2 asumen que las funciones valor y las políticas se pueden almacenar de forma exacta, por lo que únicamente son válidos en problemas cuyo espacio de estados pueda ser discretizado en un conjunto relativamente pequeño de estados.

El Capítulo 3 pone de manifiesto la necesidad de combinar los algoritmos clásicos de **RL con aproximación de funciones**. Cuando el número de estados que contiene un problema es muy elevado o incluso infinito (cuando las variables de estado son continuas), no resulta viable almacenar las funciones valor y las políticas como tablas con una entrada por cada estado, ni tampoco emplear algoritmos que requieran recorrer todos los estados repetidas veces. La solución para dichos casos consiste en emplear técnicas de aproximación de funciones. Los métodos resultantes no realizan cálculos exactos, por lo que el proceso de aprendizaje converge hacia políticas aproximadamente óptimas. En principio cualquier método de aprendizaje supervisado puede emplearse como aproximador. Sin embargo, en la práctica es probable que surjan problemas de estabilidad al combinar el proceso iterativo de los algoritmos de RL con técnicas de aproximación; de hecho, muchas de las garantías teóricas de convergencia de los algoritmos exactos dejan de ser válidas cuando se combinan con aproximadores.

Desde el punto de vista teórico los aproximadores con una arquitectura lineal en los parámetros (como por ejemplo las redes RBF de base fija) son más adecuados ya que aseguran la estabilidad de ciertos algoritmos de RL. Una desventaja de este tipo de aproximadores es que suelen ser de carácter local, por lo que no son escalables a espacios de alta dimensionalidad. Por este motivo muchas de la aplicaciones reales de RL emplean aproximadores no lineales. A pesar de ello, conviene tener en cuenta que para aplicar los algoritmos de RL con aproximadores no lineales habitualmente se requiere un largo periodo de experimentación hasta encontrar una configuración adecuada tanto del algoritmo como del aproximador.

La última parte del Capítulo 3 está dedicada a una clase de algoritmos caracterizados por hacer un uso eficiente de los datos, conocidos como algoritmos de tipo *batch*. Se ha utilizado el problema del coche en la montaña para realizar un estudio empírico de los algoritmos *batch* más representativos: *experience replay* (ER), *least-squares policy iteration* (LSPI) y *fitted q-iteration* (FQI). Los experimentos realizados han permitido analizar las ventajas e inconvenientes de cada algoritmo. Los resultados

sugieren que el algoritmo FQI combinado con árboles de decisión ofrece un comportamiento más robusto ante factores como la distribución no uniforme de los datos en comparación con ER y LSPI combinados con redes RBF de base fija.

En el Capítulo 4 se ha propuesto el **algoritmo IVAO** cuyo objetivo es mejorar la eficiencia respecto al uso de los datos en problemas de aprendizaje *online*. En muchas aplicaciones reales la velocidad de convergencia de los algoritmos de RL puede resultar excesivamente lenta o, lo que es lo mismo, se requiere una cantidad muy elevada de interacciones agente-entorno para obtener políticas óptimas. La mayoría de algoritmos enfocados en el uso eficiente de los datos están diseñados para funcionar *offline*: durante el proceso de aprendizaje no se permite la interacción entre el agente y el entorno, sino que los datos son almacenados en una etapa previa. El hecho de almacenar los datos permite que sean reutilizados varias veces para actualizar la estimación de la política, al contrario que ocurre en otros algoritmos donde los datos adquiridos son desechados después de realizar una única actualización. El algoritmo IVAO también se basa en el principio de almacenar y reutilizar los datos para extraer más información de cada interacción agente-entorno. Además emplea funciones valor ajustadas para estimar las políticas lo cual permite que el algoritmo sea combinado con aproximadores no paramétricos. El proceso de almacenar y reutilizar los datos varias veces implica una mayor carga computacional por iteración en comparación con otros algoritmos de aprendizaje *online*.

Las propiedades del algoritmo IVAO se han evaluado mediante un estudio experimental empleando tres entornos con diferente grado de complejidad. Los resultados obtenidos indican un incremento de la velocidad de convergencia respecto al algoritmo *Q-learning* con trazas de elegibilidad. Los experimentos también incluyen una sección en la que se analizan los efectos que se producen al variar los parámetros de configuración del algoritmo IVAO.

En el Capítulo 5 se ha propuesto el **algoritmo LSTD basado en máquinas de aprendizaje extremo** (LSTD-ELM). El objetivo de este algoritmo es mejorar la escalabilidad respecto al número de dimensiones del espacio de estados. LSTD-ELM se centra en el problema de estimar la función valor de una política dada, por lo que no resuelve el problema general de encontrar una política óptima a partir de la cual el agente pueda seleccionar acciones. Sin embargo, estimar la función valor de una política es una parte fundamental de cualquier algoritmo de RL basado en iteración de políticas, por lo que cualquier mejora en esta etapa tendrá un impacto relevante en dicha clase de algoritmos.

El método propuesto combina las propiedades del algoritmo LSTD estándar con la capacidad de aproximación de las máquinas de aprendizaje extremo. LSTD es un algoritmo basado en diferencias temporales que converge hacia la misma solución que el algoritmo TD, pero a diferencia de este último, LSTD emplea métodos analíticos para estimar la solución en lugar de hacerlo de forma incremental. Debido a ello el algoritmo LSTD es más robusto y converge a mayor velocidad. Una de las restricciones que impone LSTD es que la función valor tiene que ser representada mediante un apro-

ximador lineal en los parámetros. Los aproximadores lineales suelen ser de carácter local, por lo que, debido al efecto conocido como la "maldición de la dimensionalidad", la complejidad del aproximador aumenta exponencialmente a medida que crece la dimensionalidad del espacio de estados. Este efecto provoca que los aproximadores locales se vuelvan intratables cuando el número de dimensiones es elevado.

Por otra parte, el algoritmo ELM realiza una simplificación del proceso de entrenamiento de las redes neuronales de una sola capa, de forma que este proceso queda reducido, prácticamente, al ajuste de una regresión lineal. Esta característica es empleada en el método propuesto para aprovechar las características de las redes neuronales como aproximadores globales junto con la robustez y velocidad de convergencia del algoritmo LSTD. Comparado con el algoritmo LSTD estándar, LSTD-ELM aporta una mejora en la capacidad de aproximación.

Las propiedades del algoritmo LSTD-ELM se han evaluado experimentalmente en dos entornos. Uno de los entornos es una generalización propuesta en esta tesis del problema *Hop-world*. En la versión generalizada, la dimensionalidad del espacio de estados es un parámetro definido por el usuario, de forma que es posible evaluar fácilmente el comportamiento de un algoritmo determinado frente al número de dimensiones. Los resultados obtenidos muestran la capacidad de LSTD-ELM para obtener estimaciones de la función valor con una exactitud similar al algoritmo LSTD estándar pero de forma más compacta. Esta característica resulta especialmente relevante en problemas de elevada dimensionalidad. Además del algoritmo LSTD-ELM básico, en el Capítulo 5 también se ha estudiado una versión basada en comités de redes ELM. Esta versión alcanza una mayor exactitud en la aproximaciones a costa de incrementar la carga computacional del método.

El Capítulo 6 se centra en la **optimización del tratamiento de la anemia mediante aprendizaje por fuerzo**. Un elevado porcentaje de los pacientes en hemodiálisis sufren anemia renal crónica. El tratamiento estándar de la anemia renal consiste en la administración de darbepoetina alfa (u otros fármacos estimulantes de la eritropoyesis). Determinar la dosis adecuada de esta clase de fármacos es una tarea de vital importancia para mantener a los pacientes en un estado saludable debido a su estrecho rango terapéutico. Dicha tarea resulta compleja principalmente por dos motivos: (i) la respuesta de los pacientes es muy heterogénea y, (ii) el efecto de cada administración de darbepoetina alfa dura aproximadamente tres meses. El procedimiento habitual para determinar la dosis correcta consiste en monitorizar mensualmente el estado de los pacientes mediante análisis de laboratorio y ajustar la cantidad de derbepoetina alfa en función del estado de cada paciente con ayuda de protocolos de administración. Este procedimiento, que se puede identificar fácilmente con un problema de decisión secuencial, es adecuado para ser abordado mediante aprendizaje por refuerzo por dos razones. Por una parte, resulta complicado obtener un modelo que describa la evolución de todos los posibles pacientes cuando son tratados con darbepoetina alfa y, por otra parte, se dispone de una gran cantidad de datos recogida en clínicas de toda Europa durante un periodo largo de tiempo.

Los experimentos realizados a lo largo del capítulo para evaluar la metodología propuesta forman parte de una fase previa a la evaluación clínica, por lo que están basados en un modelo matemático en lugar de pacientes reales. Los resultados obtenidos muestran que la política aprendida a partir de los datos incrementó un 27.6 % el número de pacientes en los que el tratamiento lograba su objetivo comparado con un protocolo estándar de ajuste de dosis. Además, redujo la cantidad empleada de darbepoetina alfa en un 5.13 %, una reducción que es importante tanto desde el punto de vista económico, debido al alto coste de la darbepoetina, como desde el punto de vista médico, debido a los efectos secundarios asociados con dicho fármaco. Cabe destacar que los resultados obtenidos en este capítulo han dado lugar al desarrollo de una prueba piloto de evaluación clínica que se está llevando a cabo actualmente con pacientes reales.

En términos generales, las contribuciones realizadas en esta tesis tienen como objetivo hacer que los métodos de aprendizaje por refuerzo sean más prácticos y efectivos en problemas complejos. A pesar de ello resulta conveniente tener en cuenta otras limitaciones que no han sido tratadas en esta tesis y pueden ser críticas en algunos problemas reales. En primer lugar, en la mayor parte de la tesis se asume que el espacio de acciones puede ser discretizado en un conjunto relativamente pequeño de acciones. Esta asunción es habitual debido a que simplifica en gran medida los algoritmos de aprendizaje por refuerzo; sin embargo, igual que ocurre con el espacio de estados, en ocasiones una solución aproximadamente óptima requiere el uso de acciones continuas. El ámbito de la robótica es un ejemplo claro donde comúnmente se requiere emplear acciones continuas. Incluso en problemas relativamente sencillos en los que teóricamente es viable encontrar una solución adecuada con acciones discretas, en la práctica es posible que dichas acciones no sean deseables debido al estrés mecánico que supone para el sistema. En segundo lugar, se ha asumido que la información que recibe el agente por parte del entorno es suficiente para determinar su estado de forma exacta y fiable. En ciertos problemas reales tal vez no sea posible determinar exactamente el estado del entorno debido, por ejemplo, a que la información aportada por un conjunto de sensores no sea suficiente para distinguir todos los estados o porque la señal contenga ruido. En esta situación no se cumple la propiedad de Markov por lo que dejan de ser válidos los algoritmos que asumen dicha propiedad. Las técnicas para abordar este tipo de problemas suelen pertenecer al ámbito de los procesos de decisión de Markov parcialmente observables, una generalización de los MDP.

7.2. Proyección futura

El trabajo realizado en la presente tesis puede ser continuado de diversas maneras. A continuación se describen algunas cuestiones abiertas sobre los métodos propuestos y se sugieren posibles direcciones de investigación futura que pueden ayudar a resolverlas:

- En el Capítulo 3 se han comparado los tres algoritmos de tipo *batch* más representativos: ER, LSPI y FQI. Dado un problema concreto, los resultados obtenidos por cada uno de los algoritmos depende de un número elevado de factores o parámetros que, debido a las restricciones impuestas por cada algoritmo, no se pueden configurar del mismo modo en los tres casos. Por tanto resulta complicado realizar una comparación en igualdad de condiciones. La solución por la que se ha optado ha sido emplear la configuración típica para cada uno de los algoritmos. Aunque esta solución es razonable, los resultados obtenidos están sesgados por la configuración seleccionada. Sería interesante realizar un estudio más completo que contemple un mayor número de configuraciones, así como una mayor variedad de entornos.

- Continuando con el Capítulo 3, los experimentos realizados demuestran que minimizar el error cuadrático medio de la función valor estimada no garantiza una política de control de mayor calidad. A pesar de que este hecho está ampliamente aceptado por la comunidad científica (Bertsekas y Tsitsiklis, 1996; Sutton y Barto, 1998; Szepesvári, 2010), apenas existen resultados de algoritmos basados en otros criterios.

- El algoritmo IVAO, propuesto en el Capítulo 4, se ha evaluado empíricamente empleando como aproximador de funciones una red RBF de base fija. Se ha escogido este aproximador con el objetivo de que las condiciones fueran lo más parecidas posibles a las del algoritmo usado como referencia (*Q-learning*). Sin embargo, una de las ventajas que ofrece el algoritmo IVAO es la posibilidad de usar aproximadores no paramétricos que, en principio, ofrecen una mayor flexibilidad debido a su capacidad para adaptarse a la complejidad de la función valor. Así pues sería conveniente realizar una evaluación más realista en la que se emplee un aproximador no paramétrico. Adicionalmente también sería interesante comprobar la eficacia del algoritmo IVAO en las condiciones teóricas que aseguran su convergencia, es decir, cuando el mapeado realizado por el aproximador es de tipo no expansivo.

- Actualmente el algoritmo IVAO almacena los datos relativos a las últimas N transiciones realizadas por el agente. Otra opción más sofisticada que conviene explorar consiste en almacenar únicamente aquellas transiciones que aporten información relevante para la estimación de la función Q. De esta forma se podría reducir el número de transiciones almacenadas y, por tanto, la carga computacional del algoritmo. El campo del aprendizaje activo se encarga precisamente de seleccionar muestras relevantes, de hecho, ya existen algunos algoritmos que combinan aprendizaje por refuerzo y aprendizaje activo (Ernst et al., 2005a; Rachelson et al., 2011).

- El Capítulo 5 ha descrito el algoritmo LSTD-ELM y su variante LSTD-cELM. Ambos algoritmos tienen como objetivo resolver el problema parcial de predecir la función valor de una política fija. Para resolver el problema general y permitir que un agente seleccione acciones óptimas, LSTD-ELM debe incluirse en otro

algoritmo basado en iteración de políticas como, por ejemplo, LSPI. Por tanto el siguiente paso lógico consistiría en incluir LSTD-ELM en dicho algoritmo y evaluar su funcionamiento.

- Un problema que puede aparecer bajo determinadas circunstancias en la mayoría de métodos de aprendizaje automático, entre ellos el aprendizaje por refuerzo, es el sobreajuste. Para evitarlo, habitualmente se emplean técnicas de regularización. Aunque existen diversos trabajos que combinan las máquinas de aprendizaje extremo con técnicas de regularización, en el contexto del aprendizaje por refuerzo este proceso resulta más complicado debido a que no se dispone de una señal deseada. Aún así, existen algunos trabajos que utilizan regularización junto con el algoritmo LSTD (Kolter y Ng, 2009; Hoffman et al., 2011) y que podrían extenderse al caso del algoritmo LSTD-ELM.

- La metodología propuesta en el Capítulo 6 para optimizar el tratamiento de la anemia renal ha sido evaluada mediante simulaciones. Para completar el proceso de validación es necesario probarlo con pacientes reales. Esta validación se está llevando a cabo actualmente en un estudio piloto en una serie de clínicas de hemodiálisis de tres países europeos. Finalmente, si los resultados obtenidos son positivos, el sistema basado en aprendizaje por refuerzo será implantado de manera mayoritaria en las clínicas para su uso habitual.

- La función de recompensa en el problema de la anemia renal ha sido diseñada teniendo en cuenta únicamente objetivos relativos a la salud del paciente. A pesar de ello los resultados obtenidos demuestran que un tratamiento óptimo respecto a dichos objetivos también conlleva una reducción en la cantidad de darbepoetina alfa empleada y, por tanto, del coste total del tratamiento. Una opción que puede conseguir una reducción mayor del coste consistiría en incluir en la función de recompensa un término inversamente proporcional a la cantidad de darbepoetina alfa empleada. Actualmente, ante un hipotético paciente sobre el que dos posibles tratamientos produzcan un efecto muy similar, el agente no tiene preferencia sobre ninguno de ellos cuando, realmente, sería más adecuado aquel tratamiento que emplee menor cantidad de fármaco.

- La optimización del tratamiento de la anemia renal solo es una de las diversas aplicaciones prácticas que tienen las técnicas de RL. Otra aplicación en la que actualmente se está trabajando consiste en la optimización del sistema de climatización de la Facultad de Educación perteneciente a la Universidad de León. El sistema de control actual enciende las calderas 3 horas antes del horario de apertura del edificio, mientras que la calefacción comienza a funcionar 2 horas antes con una temperatura de consigna de 23°C. Tanto el horario de funcionamiento como la temperatura de consigna son fijas y no tienen en cuenta factores como la temperatura externa, la carga térmica que puede almacenar el edificio, la eficiencia de las calderas, etc. El objetivo es desarrollar un sistema de control basado en RL que tenga en cuenta estas características para minimizar el con-

sumo de gas al mismo tiempo que mantiene la temperatura de confort durante el horario de uso del edificio.

7.3. Publicaciones científicas

Durante el desarrollo de esta tesis se han obtenido diversos resultados de interés científico. Parte de estos resultados han sido publicados mediante artículos en revistas científicas y participaciones en congresos. En las siguientes secciones se enumerarán dichas publicaciones.

7.3.1. Publicaciones en revistas internacionales indexadas

- Pablo Escandell-Montero, José M. Martínez-Martínez, José D. Martín-Guerrero, Emilio Soria-Olivas y Juan Gómez-Sanchis. Least-squares temporal difference learning based on extreme learning machine. *Neurocomputing*, 141:37−45, 2014.

- Pablo Escandell-Montero, Milena Chermisi, José M. Martínez-Martínez, Juan Gómez-Sanchis, Carlo Barbieri, Emilio Soria-Olivas, Flavio Mari, Joan Vila-Francés, Andrea Stopper, Emanuele Gatti y José D. Martín-Guerrero. Optimization of anemia treatment in hemodialysis patients via reinforcement learning. *Artificial Intelligence in Medicine*. Aceptado con revisiones, 2014.

7.3.2. Publicaciones en conferencias

- Pablo Escandell-Montero, José M. Martínez-Martínez, José D. Martín-Guerrero, Emilio Soria-Olivas, Joan Vila-Francés y Rafael Magdalena-Benedito. Adaptive treatment of anemia on hemodialysis patients: a reinforcement learning approach. *IEEE Symposium on Computational Intelligence and Data Mining*, páginas 44-49, París, Francia, 2011.

- Pablo Escandell-Montero, José M. Martínez-Martínez, José D. Martín-Guerrero, Emilio Soria-Olivas y Juan Gómez-Sanchis. Regularized committe of extreme learning machine for regression problems. En *20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2012*, páginas 251-256, Brujas, Bélgica, 2012.

- Pablo Escandell-Montero, José M. Martínez-Martínez, José D. Martín-Guerrero, Emilio Soria-Olivas y Juan Gómez-Sanchis. Least-squares temporal difference learning based on extreme learning machine. En *21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013*, páginas 233-238, Brujas, Bélgica, 2013.

- Pablo Escandell-Montero, José M. Martínez-Martínez, Emilio Soria-Olivas, Joan Vila-Francés y José D. Martín-Guerrero. Ensembles of extreme learning machine networks for value prediction. En *22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning ESANN 2014*, páginas 129-134, Brujas, Bélgica, 2014.

Apéndice A

Modelo de la eritropoyesis

Este apéndice describe las ecuaciones del modelo de la eritropoyesis introducido en la Sección 6.4. Se trata de un modelo compartimental en el que cada compartimento agrupa los diferentes tipos de células implicados en la formación de eritrocitos en función de su interacción con la EPO (ver Figura 6.3). Se han considerado tres clases de células o compartimentos:

- Compartimento P: formado por las células progenitoras y precursoras (CFU-GEMM, BFU-E y CFU-E).
- Compartimento M: contiene los eritroblastos (basófilos, policromatófilos y ortocromatófilos) y retículos inmaduros. Ambos tipos de células dependen del hierro para diferenciarse. Como se asume que todos los pacientes tienen suficiente hierro disponible, el compartimento M no modifica el número de células, sino que únicamente introduce un retardo entre las células que entran y salen.
- Compartimento R: a este compartimento pertenecen los retículos maduros y los eritrocitos.

Las ecuaciones que gobiernan el número de células en cada compartimento son ecuaciones de balance (Krzyzanski et al., 1999). En cada compartimento entra un flujo de células nuevas y sale un flujo de células diferenciadas o apóptoticas. Se asume que las células de un mismo compartimento permanecen vivas durante aproximadamente el mismo periodo de tiempo, denotado por $\mathbf{T}_P$ para las células del compartimento P y $\mathbf{T}_R$ para las células del compartimento R (Krzyzanski et al., 1999). Esta asunción determina a que se eliminan las células, ya que en un momento dado t el número de células que mueren tiene que ser igual al que nacen en el mismo instante pero retardado por su periodo de vida. El flujo de células entrantes al compartimento P depende de la respuesta de los progenitores al efecto estimulante del fármaco. De acuerdo con (Krzyzanski et al., 1999), esta respuesta puede ser descrita con la función

Hill:

$$H(E_{\text{tot}}) = \frac{E_{\text{tot}}}{E_{50} + E_{\text{tot}}}$$

donde E_{tot} (definido en la Ecuación 6.1) representa la concentración plasmática total de EPO, y $E_{50} = 100/V_d$. La sensibilidad E_{50} es la mitad de la concentración máxima efectiva, un parámetro usado habitualmente en farmacología para medir la potencia de una sustancia.

El flujo de salida del compartimento P se corresponde con el flujo de entrada del compartimento R retardado por $\mathbf{T}_M$, ya que se ha considerado en el compartimento M no se elimina ninguna célula.

En la última etapa de su vida útil, los eritrocitos se vuelven senescentes y son eliminadas del flujo sanguíneo. La tasa de eliminación de eritrocitos es un promedio ponderado de la tasa de entrada en el compartimento R. Dicho promedio consisten en una combinación convexa cuyos coeficientes son proporcionado por una distribución gausiana con media igual al tiempo medio de vida útil de los eritrocitos y varianza 30. La distribución gaussiana indica que el efecto del fármaco es mayor en las células mayores de $\mathbf{T}_R$.

La evolución del número de células en los compartimentos P y R es la siguiente:

$$\begin{cases} P'(t) = C_p \dfrac{E_{\text{tot}}(t)}{E_{50} + E_{\text{tot}}(t)} P(t) \\ \qquad -C_p \dfrac{1}{\mathbf{T}_P} \displaystyle\sum_{T_j=1}^{\mathbf{T}_P} \dfrac{E_{\text{tot}}(t - T_j)}{E_{50} + E_{\text{tot}}(t - T_j)} P(t - T_j) \\ R'(t) = C_r \dfrac{E_{\text{tot}}(t - (\mathbf{T}_P + \mathbf{T}_M))}{E_{50} + E_{\text{tot}}(t - (\mathbf{T}_P + \mathbf{T}_M))} P(t - (\mathbf{T}_P + \mathbf{T}_M)) \\ \qquad -\dfrac{C_r}{S_{P,M,R}} \displaystyle\sum_{T_j=\mathbf{T}_P+\mathbf{T}_M+1}^{\mathbf{T}_P+\mathbf{T}_M+\mathbf{T}_R} g_{T_j} \dfrac{E_{\text{tot}}(t - T_j)}{E_{50} + E_{\text{tot}}(t - T_j)} P(t - T_j) \end{cases}$$

donde

$$g_{T_j} = G(T_j - (\mathbf{T}_P + \mathbf{T}_M))),$$

$G(T)$ es la distribución gaussiana con media $\mathbf{T}_R$ y varianza 30 evaluada en el instante T, y $S_{P,M,R}$ es un factor de normalización definido como

$$\begin{aligned} S_{P,I,R} &= \sum_{T_j=\mathbf{T}_P+\mathbf{T}_M+1}^{\mathbf{T}_P+\mathbf{T}_M+\mathbf{T}_R} G\Big(T_j - (\mathbf{T}_P + \mathbf{T}_M)\Big) \\ &= \sum_{T_j=1}^{\mathbf{T}_R} G(T_j) \end{aligned}$$

Durante los experimentos se han empleado los valores $(\mathbf{T}_P, \mathbf{T}_M, \mathbf{T}_R) = (9, 4, 70)$. En el instante t_0 de la primera administración, $P_0 = 1$ and $R_0 = 1$ (es decir, hay 10^{11} células madurando en el compartimento P). El resto de parámetros, Ep, Cp y Cr se han estimado utilizando los datos históricos de cada paciente correspondientes a 6 meses de tratamiento.

Índice de figuras

Índice de tablas

Índice de algoritmos

Bibliografía

Sander Adam, Lucian Busoniu, and Robert Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(2):201–212, March 2012.

John W. Adamson and Joseph W. Eschbach. Treatment of the anemia of chronic renal failure with recombinant human erythropoietin. *Annual Review of Medicine*, 41(1): 349–360, February 1990.

John W. Adamson and Joseph W. Eschbach. Erythropoietin for end-stage renal disease. *The New England Journal of Medicine*, 339(9):625–627, August 1998.

Oguzhan Alagoz, Heather Hsu, Andrew J. Schaefer, and Mark S. Roberts. Markov decision processes: A tool for sequential decision making under uncertainty. *Medical Decision Making*, 30(4):474–483, January 2010.

James S. Albus. A new approach to manipulator control: the cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control*, 97:220–227, October 1975.

James S. Albus. *Brains, behavior, and robotics*. BYTE Books, Peterborough, N.H., 1981.

Michael Allon, Kenneth Kleinman, Michael Walczyk, Charles Kaupke, Louise Messer-Mann, Kurt Olson, Anne C. Heatherington, and Bradley J. Maroni. Pharmacokinetics and pharmacodynamics of darbepoetin alfa and epoetin in patients undergoing dialysis. *Clinical pharmacology and therapeutics*, 72(5):546–555, November 2002.

Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, October 2004.

Charles W. Anderson. Q-learning with hidden-unit restarting. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 81–88, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

Santhosh S. Baboo and Sankar Sasikala. Multicategory classification using an extreme learning machine for microarray gene expression cancer diagnosis. In *2010 IEEE International Conference on Communication Control and Computing Technologies (ICCCCT)*, pages 748–757, 2010.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37, 1995.

Poornima Balakrishna, Rajesh Ganesan, Lance Sherry, and Benjamin S. Levy. Estimating taxi-out times with a reinforcement learning algorithm. In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 3–12, 2008.

Poornima Balakrishna, Rajesh Ganesan, and Lance Sherry. Accuracy of reinforcement learning algorithms for predicting aircraft taxi-out times: A case-study of tampa bay departures. *Transportation Research Part C: Emerging Technologies*, 18(6):950–962, December 2010.

André M. S. Barreto and Charles W. Anderson. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence*, 172(4–5):454–482, March 2008.

Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, January 2003.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

Andrew G. Barto, Richard S. Sutton, and Christopher Watkins. Learning and sequential decision making. In *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, page 539–602. The MIT Press, 1989.

Riccardo Bellazzi. Drug delivery optimization through bayesian networks: An application to erythropoietin therapy in uremic anemia. *Computers and Biomedical Research*, 26(3):274–293, June 1993.

Richard Bellman. *Dynamic Programming*. Princeton University Press, July 1957.

Casey C. Bennett and Kris Hauser. Artificial intelligence framework for simulating clinical decision-making: a markov decision process approach. *Artificial intelligence in medicine*, 57(1):9–19, January 2013.

Nabil Benoudjit and Michel Verleysen. On the kernel widths in radial-basis function networks. *Neural Processing Letters*, 18(2):139–154, October 2003.

Nabil Benoudjit, Cédric Archambeau, Amaury Lendasse, John A. Lee, and Michel Verleysen. Width optimization of the gaussian kernels in radial basis function networks. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning 2002*, pages 425–432, 2002.

Hamid R. Berenji. Fuzzy reinforcement learning and dynamic programming. In Anca L. Ralescu, editor, *Fuzzy Logic in Artificial Intelligence*, number 847 in Lecture Notes in Computer Science, pages 1–9. Springer Berlin Heidelberg, January 1994.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. I.* Athena Scientific, 3rd edition, November 2005.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II.* Athena Scientific, 3rd edition, January 2007.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1st edition, May 1996.

Brett Bethke, Jonathan P. How, and Asuman Ozdaglar. Approximate dynamic programming using support vector regression. In *47th IEEE Conference on Decision and Control, 2008. CDC 2008*, pages 3811–3816, 2008.

Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, USA, 1st edition, January 1995.

Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 1st edition 2006, corr. 2nd printing edition, October 2007.

Justin A. Boyan. Least-squares temporal difference learning. *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56, 1999.

Justin A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2-3):233–246, November 2002.

Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 369–376. The MIT Press, 1995.

Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001.

Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees.* Chapman and Hall/CRC, 1st edition, January 1984.

Lucian Busoniu. *Reinforcement Learning in Continuous State and Action Spaces.* PhD thesis, Delft University of Technology, Delft, 2008.

Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, 2008a.

Lucian Busoniu, Damien Ernst, Bart De Schutter, and Robert Babuska. Consistency of fuzzy model-based reinforcement learning. In *IEEE International Conference on Fuzzy Systems, 2008. FUZZ-IEEE 2008. (IEEE World Congress on Computational Intelligence)*, pages 518–524, 2008b.

Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators.* CRC Press, 1st edition, April 2010a.

Lucian Busoniu, Bart Schutter, and Robert Babuška. Approximate dynamic programming and reinforcement learning. In Janusz Kacprzyk, Robert Babuška, and Frans C. A. Groen, editors, *Interactive Collaborative Information Systems*, volume 281, pages 3–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010b.

Lucian Busoniu, Damien Ernst, Bart De Schutter, and Robert Babuska. Cross-entropy optimization of control policies with adaptive basis functions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 41(1):196–209, February 2011.

Feilong Cao, Bo Liu, and Dong Sun-Park. Image classification based on effective extreme learning machine. *Neurocomputing*, 102(15):90–97, 2013.

Andrea Castelletti, Simona Galelli, Marco Restelli, and Rodolfo Soncini-Sessa. Tree-based reinforcement learning for optimal water reservoir operation. *Water Resources Research*, 46(9):W09507, September 2010.

Bibhas Chakraborty, Victor Strecher, and Susan A. Murphy. Bias correction and confidence intervals for fitted q-iteration. In Daphne Koller, Dale Shuurmans, Yoshua Bengio, and Leon Bottou, editors, *Neural Information Processing Systems - Workshop on Model Uncertainty and Risk in Reinforcement Learning*, 2008.

Shenglei Chen, Geng Chen, and Ruijun Gu. An efficient l_2-norm regularized least-squares temporal difference learning algorithm. *Knowledge-Based Systems*, 45:94–99, June 2013.

Pawel Cichosz. An analysis of experience replay in temporal difference learning. *Cybernetics and Systems*, 30(5):341–363, 1999.

Allan J. Collins, Suying Li, Wendy Peter, Jim Ebben, Tricia Roberts, Jennie Z. Ma, and Willard Manning. Death, hospitalization, and economic associations among incident hemodialysis patients with hematocrit values of 36 to 39 %. *Journal of the American Society of Nephrology: JASN*, 12(11):2465–2473, November 2001.

Allan J. Collins, Robert M. Brenner, Joshua J. Ofman, Eric M. Chi, Nina Stuccio-White, Mahesh Krishnan, Craig Solid, Norma J. Ofsthun, and J. Michael Lazarus. Epoetin alfa use in patients with ESRD: an analysis of recent US prescribing patterns and hemoglobin outcomes. *American journal of kidney diseases: the official journal of the National Kidney Foundation*, 46(3):481–488, September 2005.

Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In David S. Touretzky, Michael Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1017–1023. The MIT Press, 1996.

Christian Darken and John E. Moody. Towards faster stochastic gradient search. In John E. Moody, Stephen Jose Hanson, and Richard Lippmann, editors, *Advances in Neural Information Processing Systems*, pages 1009–1016. Morgan Kaufmann, 1992.

Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, STOC '08, pages 537–546, New York, NY, USA, 2008. ACM.

Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Virkumar Vazirani. *Algorithms*. McGraw-Hill Higher Education, Boston, 2008.

John T. Daugirdas. *Handbook of Chronic Kidney Disease Management*. Lippincott Williams & Wilkins, 1st edition, May 2011.

Scott Davies. Multidimensional triangulation and interpolation for reinforcement learning. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 1005–1011. The MIT Press, 1997.

Marc P. Deisenroth, Carl E. Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7–9):1508–1524, March 2009.

Marc P. Deisenroth, Dieter Fox, and Carl E. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99(1):1, 2013a.

Marc P. Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1):388–403, jun 2013b.

Eric V. Denardo. Contraction mappings in the theory underlying dynamic programming. *SIAM Review*, 9(2):165–177, April 1967.

Kevin R. Dixon, Richard J. Malak, and Pradeep K. Khosla. Incorporating prior knowledge and previously learned information into reinforcement learning. Technical report, Carnegie Mellon University, 2000.

Stuart Dreyfus. Richard bellman on the birth of dynamic programming. *Operations Research*, 50(1):48–51, February 2002.

Sjoerd Dries and Marco A. Wiering. Neural-fitted TD-Leaf learning for playing othello with structured neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 23(11):1701–1713, November 2012.

James Elliott, Dennis Mishler, and Rajiv Agarwal. Hyporesponsiveness to erythropoietin: Causes and management. *Advances in Chronic Kidney Disease*, 16(2):94–100, March 2009.

Sadie F. Elliott. Erythropoiesis-stimulating agents and other methods to enhance oxygen transport. *British Journal of Pharmacology*, 154(3):529–541, June 2008.

EMA. European medicines agency, summary of product characteristics document, March 2013. http://www.ema.europa.eu/docs/en_GB/document_lib rary/EPAR_-_Product_Information/human/000332/WC500026149.pdf (accessed July 9, 2013).

Damien Ernst. Selecting concise sets of samples for a reinforcement learning agent. In *International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore, 2005.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, March 2005a.

Damien Ernst, Mevludin Glavic, Pierre Geurts, and Louis Wehenkel. Approximate value iteration in the reinforcement learning context. Application to electrical power system control. *International Journal of Emerging Electric Power Systems*, 3(1), January 2005b.

Damien Ernst, Guy-Bart Stan, Jorge Gongalves, and Louis Wehenkel. Clinical data based optimal STI strategies for HIV: a reinforcement learning approach. In *Decision and Control, 2006 45th IEEE Conference on*, pages 667–672, 2006.

Damien Ernst, Mevludin Glavic, Florin Capitanescu, and Louis Wehenkel. Reinforcement learning versus model predictive control: a comparison on a power system problem. *Transanctions on Systems, Man and Cybernetics, Part B*, 39(2):517–529, April 2009.

Joseph W. Eschbach. Iron requirements in erythropoietin therapy. *Best Practice & Research Clinical Haematology*, 18(2):347–361, June 2005.

Amir M. Farahmand, Mohammad Ghavamzadeh, Csaba Szepesvari, and Shie Mannor. Regularized fitted q-iteration for planning in continuous-space markovian decision problems. In *American Control Conference, 2009. ACC '09.*, pages 725–730, 2009.

Jay A. Farrell and Torsten Berger. On the effects of the training sample density in passive learning control. In *American Control Conference, Proceedings of the 1995*, volume 1, pages 872–877, 1995.

Fernando Fernández and Daniel Borrajo. VQQL. applying vector quantization to reinforcement learning. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, number 1856 in Lecture Notes in Computer Science, pages 292–303. Springer Berlin Heidelberg, January 2000.

Steven Fishbane and Jeffrey S. Berns. Evidence and implications of haemoglobin cycling in anaemia management. *Nephrology Dialysis Transplantation*, 22(8):2129–2132, August 2007.

Robert N. Foley, Patrick S. Parfrey, Janet Morgan, Paul E. Barré, Patricia Campbell, Pierre Cartier, Douglas Coyle, Adrian Fine, Paul Handa, Iris Kingma, Cathy Y. Lau, Adeera Levin, David Mendelssohn, Norman Muirhead, Brendan Murphy, Richard K. Plante, Gerald Posen, and George A. Wells. Effect of hemoglobin levels in hemodialysis patients with asymptomatic cardiomyopathy. *Kidney international*, 58(3): 1325–1335, September 2000.

Jordan Frank, Shie Mannor, and Doina Precup. Reinforcement learning in the presence of rare events. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 336–343, New York, NY, USA, 2008. ACM.

Walter Fried. Erythropoietin and erythropoiesis. *Experimental Hematology*, 37(9):1007–1015, September 2009.

Luca Gabutti, Nathalie Lötscher, Josephine Bianda, Claudio Marone, Giorgio Mombelli, and Michel Burnier. Would artificial neural networks implemented in clinical wards help nephrologists in predicting epoetin responsiveness? *BMC Nephrology*, 7:13–26, September 2006.

Adam E. Gaweda, Alfred A. Jacobs, and Michael E. Brier. Fuzzy rule-based approach to automatic drug dosing in renal failure. In *IEEE International Conference on Fuzzy Systems*, volume 2, pages 1206–1209, 2003.

Adam E. Gaweda, Mehmet K. Muezzinoglu, George R. Aronoff, Alfred A. Jacobs, Jacek M. Zurada, and Michael E. Brier. Individualization of pharmacological anemia management using reinforcement learning. *Neural Networks*, 18(5-6):826–834, April 2005.

Adam E. Gaweda, Alfred A. Jacobs, George R. Aronoff, and Michael E. Brier. Model predictive control of erythropoietin administration in the anemia of ESRD. *American Journal of Kidney Diseases: The Official Journal of the National Kidney Foundation*, 51(1):71–79, January 2008a.

Adam E. Gaweda, Alfred A. Jacobs, and Michael E. Brier. Application of fuzzy logic to predicting erythropoietic response in hemodialysis patients. *The International journal of artificial organs*, 31(12):1035–1042, December 2008b.

Matthieu Geist and Olivier Pietquin. An algorithmic survey of parametric value function approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6): 845 – 867, February 2013.

Alborz Geramifard, Michael Bowling, and Richard S. Sutton. Incremental least-squares temporal difference learning. In *In Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*, pages 356–361. AAAI Press, 2006a.

Alborz Geramifard, Michael Bowling, Martin Zinkevich, and Richard S. Sutton. iLSTD: eligibility traces and convergence analysis. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems*, pages 441–448. The MIT Press, 2006b.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

Mohammad Ghavamzadeh, Alessandro Lazaric, Odalric-Ambrym Maillard, and Rémi Munos. LSTD with random projections. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23, pages 721–729. The MIT Press, 2010.

Pierre. Y. Glorennec. Reinforcement learning: An overview. In *Proceedings European Symposium on Intelligent Techniques*, pages 17–35, Aachen, Germany, 2000.

Fred Glover and Manuel Laguna. *Tabu search*. Kluwer, Boston, Massachusetts, 1997.

David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, 1989.

Lee Goldman and Andrew I. Schafer. *Goldman's Cecil Medicine: Expert Consult Premium Edition*. Saunders, 24 edition, August 2011.

Geoffrey Gordon. Stable function approximation in dynamic programming. Technical Report CMU-CS-95-103, Computer Science Department, Pittsburgh, PA, January 1995a.

Geoffrey Gordon. Online fitted reinforcement learning. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1052–1058. The MIT Press, 1995b.

Geoffrey Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, School of Computer Science Carnegie Mellon University, 1999.

Geoffrey J. Gordon. Chattering in SARSA(lambda) - a CMU learning lab internal report. Technical report, Carnegie Mellon University, 1996.

Ivo Grondman, Lucian Busoniu, Gabriel A. D. Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(6):1291 –1307, November 2012.

Arthur Guez, Robert D. Vincent, Massimo Avoli, and Joelle Pineau. Adaptive treatment of epilepsy via batch-mode reinforcement learning. In *Proceedings of the 20th national conference on Innovative applications of artificial intelligence - Volume 3*, pages 1671–1678, Chicago, Illinois, 2008. AAAI Press.

Vijaykumar Gullapalli. Direct associative reinforcement learning methods for dynamic systems control. *Neurocomputing*, 9(3):271–292, December 1995.

Roland Hafner and Martin Riedmiller. Reinforcement learning in feedback control. *Machine Learning*, 84(1):137–169, February 2011.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2009.

Milos Hauskrecht and Hamish Fraser. Planning treatment of ischemic heart disease with partially observable markov decision processes. *Artificial intelligence in medicine*, 18 (3):221–244, March 2000.

Simon Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 3rd edition, November 2008.

Bernhard Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, page 243–250, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

Félix Hernández-del Olmo, Elena Gaudioso, and Antonio Nevado. Autonomous adaptive and active tuning up of the dissolved oxygen setpoint in a wastewater treatment plant using reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(5):768 –774, September 2012.

Félix Hernández-del Olmo, Félix H. Llanes, and Elena Gaudioso. An emergent approach for the control of wastewater treatment plants by means of reinforcement learning techniques. *Expert Systems with Applications*, 39(3):2355–2360, February 2012.

Matthew Hoffman, Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Regularized least squares temporal difference learning with nested l_2 and l_1 penalization. In *European Workshop On Reinforcement Learning, EWRL'11*, pages 102–114, 2011.

Tadashi Horiuchi, Akinori Fujino, Osamu Katai, and Tetsuo Sawaragi. Fuzzy interpolation-based q-learning with continuous states and actions. In *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems, 1996*, volume 1, pages 594–600, 1996.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, February 1991.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, March 1989.

Ronald A. Howard. *Dynamic programming and Markov processes.* The MIT press, Cambridge, Massachusetts, 1960.

Guang-Bin Huang. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Transactions on Neural Networks*, 14(2):274–281, August 2003.

Guang-Bin Huang and Haroon A. Babri. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE Transactions on Neural Networks*, 9(1):224–229, November 1998.

Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501, December 2006.

Onyekachi Ifudu, Joseph Feldman, and Eli A. Friedman. The intensity of hemodialysis and the response to erythropoietin in patients with end-stage renal disease. *The New England Journal of Medicine*, 334(7):420–425, February 1996.

Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6): 1185–1201, November 1994.

Alfred Jacobs, Peter Lada, Jacek M. Zurada, Michael E. Brier, and George R. Aronoff. Predictors of hematocrit in hemodialysis patients as determined by artificial neural networks. *Journal of American Nephrology*, 12:387–395, March 2001.

John A. Jacquez. *Compartmental analysis in biology and medicine.* University of Michigan Press, Ann Arbor, 1985.

Sébastien Jodogne, Cyril Briquet, and Justus H. Piater. Approximate policy iteration for closed-loop learning of visual tasks. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, number 4212 in Lecture Notes in Computer Science, pages 210–221. Springer Berlin Heidelberg, January 2006.

Lionel Jouffe. Fuzzy inference system learning by reinforcement methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 28 (3):338–355, January 1998.

Leslie P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, January 1996.

Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2): 99–134, May 1998.

Robert M. Kalicki and Dominik E. Uehlinger. Red cell survival in relation to changes in the hematocrit: more important than you think. *Blood Purification*, 26(4):355–360, November 2008.

Shivaram Kalyanakrishnan. *Learning Methods for Sequential Decision Making with Imperfect Representations*. PhD thesis, University of Texas at Austin, December 2011.

Shivaram Kalyanakrishnan and Peter Stone. Batch reinforcement learning in a complex domain. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, Honolulu, Hawaii, 2007. ACM.

KDOQI. KDOQI clinical practice guidelines and clinical practice recommendations for anemia in chronic kidney disease. *American journal of kidney diseases: the official journal of the National Kidney Foundation*, 47(5 Suppl 3):11–145, May 2006.

Karl M. Koch, Wolf D. Patyna, Stanley Shaldon, and Eckhard Werner. Anemia of the regular hemodialysis patient and its treatment. *Nephron*, 12(5):405–419, September 1974.

J. Zico Kolter and Andrew Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 521–528, New York, NY, USA, 2009. ACM.

Vijay R. Konda. *Actor-critic algorithms*. Phd thesis, Massachusetts Institute of Technology, February 2002.

Vijay R. Konda and John N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, April 2003.

Wojciech Krzyzanski and Juan J. Pérez-Ruixo. An assessment of recombinant human erythropoietin effect on reticulocyte production rate and lifespan distribution in healthy subjects. *Pharmaceutical research*, 24(4):758–772, April 2007.

Wojciech Krzyzanski, Rohini Ramakrishnan, and William J. Jusko. Basic pharmacodynamic models for agents that alter production of natural cells. *Journal of pharmacokinetics and biopharmaceutics*, 27(5):467–489, October 1999.

Wojciech Krzyzanski, William J. Jusko, Mary C. Wacholtz, Neil Minton, and Wing K. Cheung. Pharmacokinetic and pharmacodynamic modeling of recombinant human erythropoietin after multiple subcutaneous doses in healthy subjects. *European journal of pharmaceutical sciences: official journal of the European Federation for Pharmaceutical Sciences*, 26(3-4):295–306, November 2005.

Michail G. Lagoudakis and Ronald Parr. Model-free least-squares policy iteration. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 1547–1554. The MIT Press, 2001.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

Yuan Lan, Yeng Chai Soh, and Guang-Bin Huang. Ensemble of online sequential extreme learning machine. *Neurocomputing*, 72(13–15):3391–3395, August 2009.

Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems*, pages 833–840, 2007.

Frank L. Lewis and Draguna Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine*, 9(3):32–50, November 2009.

Lihong Li, Michael L. Littman, and Christopher R. Mansley. Online exploration in least-squares policy iteration. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 733–739, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, June 1992.

Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, January 1993.

Simeng Liu and Gregor P. Henze. Experimental analysis of simulated reinforcement learning control for active and passive building thermal storage inventory: Part 1. theoretical foundation. *Energy and Buildings*, 38(2):142–147, February 2006a.

Simeng Liu and Gregor P. Henze. Experimental analysis of simulated reinforcement learning control for active and passive building thermal storage inventory: Part 2: Results and analysis. *Energy and Buildings*, 38(2):148–161, February 2006b.

Daniel J. Lizotte, Michael Bowling, and Susan A. Murphy. Linear fitted-q iteration with multiple reward functions. *Journal of Machine Learning Research*, 13:3253–3295, November 2012.

Lennart Ljung and Torsten Söderström. *Theory and practice of recursive identification*. The MIT Press, 1983.

Francesco Locatelli, Pedro Aljama, Peter Bárány, Bernard Canaud, Fernando Carrera, Kai-Uwe Eckardt, Walter H Hörl, Iain C. Macdougal, Alison Macleod, Andrzej Wiecek, and Stewart Cameron. Revised european best practice guidelines for the management of anaemia in patients with chronic renal failure. *Nephrology, Dialysis, Transplantation: Official Publication of the European Dialysis and Transplant Association - European Renal Association*, 19 Suppl 2:1–47, May 2004.

Hui-juan Lu, Chun-lin An, En-hui Zheng, and Yi Lu. Dissimilarity based ensemble of extreme learning machine for gene expression data classification. *Neurocomputing*, 128:22–30, March 2014.

Iain C. Macdougall, Paul D. Wilson, and Alison Roche. Impact of hemoglobin variability in haemodialysis patients receiving erythropoiesis stimulating agents for the management of renal anaemia. *Journal of the American Society of Nephrology*, 16: 899–911, January 2005.

Hamid R. Maei. *Gradient Temporal-Difference Learning Algorithms*. PhD thesis, University of Alberta, 2011.

Hamid R. Maei, Csaba Szepesvári, Shalabh Bhatnagar, and Richard S. Sutton. Toward off-policy learning control with function approximation. In *International Conference on Machine Learning ICML*, pages 719–726, 2010.

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, October 2007.

José D. Martín-Guerrero, Gustavo Camps-valls, Emilio Soria-Olivas, Antonio J. Serrano-López, Juan J. Pérez-Ruixo, and N. Víctor Jiménez-Torres. Dosage individualization of erythropoietin using a profile–dependent support vector regression. *IEEE Transactions on Biomedical Engineering*, 50:1136–1142, August 2003a.

José D. Martín-Guerrero, Emilio Soria-Olivas, Gustavo Camps-Valls, Antonio J. Serrano-López, Juan J. Pérez-Ruixo, and Víctor Jiménez-Torres. Use of neural networks for dosage individualisation of erythropoietin in patients with secondary anemia to chronic renal failure. *Computers in Biology and Medicine*, 33(4):361–373, July 2003b.

José D. Martín-Guerrero, Faustino Gómez, Emilio Soria-Olivas, Jürgen Schmidhuber, Mónica Climente-Martí, and Víctor Jiménez-Torres. A reinforcement learning approach for individualizing erythropoietin dosages in hemodialysis patients. *Expert Systems with Applications*, 36(6):9737–9742, February 2009.

José M. Martínez-Martínez, Pablo Escandell-Montero, Emilio Soria-Olivas, José D. Martín-Guerrero, Rafael Magdalena-Benedito, and Juan Gómez-Sanchis. Regularized extreme learning machine for regression problems. *Neurocomputing*, 74(17): 3716–3721, October 2011.

José A. Martín H., Javier de Lope, and Darío Maravall. Robust high performance reinforcement learning through weighted k-nearest neighbors. *Neurocomputing*, 74 (8):1251–1259, March 2011.

Samy I. McFarlane, Shu-Cheng Chen, Adam T. Whaley-Connell, James R. Sowers, Joseph A. Vassalotti, Moro O. Salifu, Suying Li, Changchun Wang, George Bakris, Peter A. McCullough, Allan J. Collins, and Keith C. Norris. Prevalence and associations of anemia of CKD: kidney early evaluation program (KEEP) and national health and nutrition examination survey (NHANES) 1999-2004. *American Journal of Kidney Diseases*, 51(4):S46–S55, April 2011.

Francisco S. Melo and Manuel Lopes. Fitted natural actor-critic: A new algorithm for continuous state-action MDPs. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, number 5212 in Lecture Notes in Computer Science, pages 66–81. Springer Berlin Heidelberg, January 2008.

Francisco S. Melo, Sean P. Meyn, and M. Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 664–671, New York, NY, USA, 2008. ACM.

Yoan Miche, Antti Sorjamaa, Patrick Bas, Olli Simula, Christian Jutten, and Amaury Lendasse. OP-ELM: optimally pruned extreme learning machine. *IEEE Transactions on Neural Networks*, 21(1):158–162, January 2010.

Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

Marvin Minsky. Steps toward artificial intelligence. In *Computers and Thought*, page 406–450. McGraw-Hill, 1963.

Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, October 1993.

Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(3):199–233, December 1995.

Rémi Munos and Andrew W. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49(2-3):291–323, November 2002.

Susan A. Murphy. A generalization error for q-learning. *Journal of machine learning research*, 6:1073–1097, July 2005.

Susan A. Murphy, David W. Oslin, A. John Rush, and Ji Zhu. Methodological challenges in constructing effective treatment sequences for chronic psychiatric disorders. *Neuropsychopharmacology*, 32(2):257–262, November 2006.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, page 278–287. Morgan Kaufmann, 1999.

Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics, ISER 2004*, pages 330–443, 2004.

Allen R. Nissenson and Richard N. Fine. *Handbook of Dialysis Therapy*. Saunders, 4th edition, September 2007.

Mohammad R. Nowrousian. Impact of anemia on organ functions. In Prof Dr M. R. Nowrousian, editor, *Recombinant Human Erythropoietin (rhEPO) in Clinical Oncology*, pages 147–172. Springer Vienna, January 2002.

Neeta B. O'Mara. Anemia in patients with chronic kidney disease. *Diabetes Spectrum*, 21(1):12–19, January 2008.

Dirk Ormoneit and Saunak Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2-3):161–178, March 2002.

Martjin V. Otterlo. *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for Adaptive Sequential Decision Making under Uncertainty*. IOS Press, February 2009.

Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, November 2005.

Vimla L. Patel, Edward H. Shortliffe, Mario Stefanelli, Peter Szolovits, Michael R. Berthold, Riccardo Bellazzi, and Ameen Abu-Hanna. The coming of age of artificial intelligence in medicine. *Artificial intelligence in medicine*, 46(1):5–17, May 2009.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks: the official journal of the International Neural Network Society*, 21(4):682–697, May 2008a.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7–9):1180–1190, March 2008b.

Markus Peters, Wolfgang Ketter, Maytal Saar-Tsechansky, and John Collins. A reinforcement learning approach to autonomous decision-making in smart electricity markets. *Machine Learning*, 92(1):5–39, July 2013.

Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27(1):335–380, November 2006.

Joelle Pineau, Marc G. Bellemare, A. John Rush, Adrian Ghizaru, and Susan A. Murphy. Constructing evidence-based treatment strategies using methods from computer science. *Drug and alcohol dependence*, 88(Suppl 2):S52–S60, May 2007.

Joelle Pineau, Arthur Guez, Robert Vincent, Gabriella Panuccio, and Massimo Avoli. Treating epilepsy via adaptive neurostimulation: a reinforcement learning approach. *International Journal of Neural Systems*, 19(4):227–240, August 2009.

Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley, 2nd edition, September 2011.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1st edition, March 2005.

Juan J. Pérez-Ruixo, Hui C. Kimko, Andrew T. Chow, Vladimir Piotrovsky, Wojciech Krzyzanski, and William J. Jusko. Population cell life span models for effects of drugs following indirect mechanisms of action. *Journal of pharmacokinetics and pharmacodynamics*, 32(5-6):767–793, December 2005.

Emmanuel Rachelson, François Schnitzler, Louis Wehenkel, and Damien Ernst. Optimal sample selection for batch-mode reinforcement learning. In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART 2011)*, Rome, Italy, 2011.

Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, page 463–471, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

Radhakrishna Rao and Sujit K. Mitra. *Generalized Inverse of Matrices and Its Applications*. John Wiley & Sons Inc, 1st edition, January 1972.

Carl E. Rasmussen and Christopher K. Williams. *Gaussian processes for machine learning*. The MIT Press, Cambridge, Massachusetts, 2006.

Bohdana Ratitch. *On characteristics of Markov decision processes and reinforcement learning in large domains*. PhD thesis, McGill University, Montreal, 2005.

Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. *In 16th European Conference on Machine Learning*, pages 317–328, 2005.

Hai-Jun Rong, Yew-Soon Ong, Ah-Hwee Tan, and Zexuan Zhu. A fast pruned-extreme learning machine for classification problem. *Neurocomputing*, 72(1-3):359–366, December 2008.

Hai-Jun Rong, Sundaram Suresh, and Guang-She Zhao. Stable indirect adaptive neural controller for a class of nonlinear system. *Neurocomputing*, 74(16):2582–2590, September 2011.

Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, November 1987.

Xiaogang Ruan, Mingxiao Ding, Daoxiong Gong, and Junfei Qiao. On-line adaptive control for inverted pendulum balancing based on feedback-error-learning. *Neurocomputing*, 70(4–6):770–776, January 2007.

Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer, 2004.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

Gavin A. Rummery and Mahesan Niranjan. On-line Q-learning using connectionist systems. Technical report, Cambridge University Engineering Department Technical Report CUED/F-INFENG/TR 166., 1994.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1st edition, January 1995.

Angel Sanz-Granda. Análisis probabilístico de minimización de costes de darbepoetin alfa frente a epoetina alfa en el tratamiento de la anemia secundaria a insuficiencia renal crónica. valoración en la práctica clínica española. *Farmacia Hospitalaria*, 33 (4):208–216, July 2009.

Ralf Schoknecht and Artur Merke. Convergent combinations of reinforcement learning with linear function approximation. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Scholkopf, editors, *Advances in Neural Information Processing Systems*, pages 1579–1586. The MIT Press, 2003.

Bernhard Scholkopf, Christopher J. C. Burges, and Alexander J. Smola. *Advances in kernel methods support vector learning*. The MIT Press, Cambridge, Massachusetts, 1999.

Lawrence F. Shampine and Skip Thompson. Solving DDEs in Matlab. *Applied Numerical Mathematics*, 37(4):441–458, June 2001.

Daniel Shapiro, Pat Langley, and Ross Shachter. Using background knowledge to speed reinforcement learning in physical agents. In *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, page 254–261, New York, NY, USA, 2001. ACM.

John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, New York, 2004.

Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, March 1950.

Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In *Proceedings of the 6th International Conference on Abstraction, Reformulation and Approximation*, SARA'05, page 194–205, Berlin, Heidelberg, 2005. Springer-Verlag.

Susan M. Shortreed, Eric Laber, Daniel J. Lizotte, T. Scott Stroup, Joelle Pineau, and Susan A. Murphy. Informing sequential clinical decision-making through reinforcement learning: an empirical study. *Machine Learning*, 84:109–136, December 2010.

Olivier Sigaud and Olivier Buffet. *Markov decision processes in artificial intelligence: MDPs, beyond MDPs and applications*. ISTE ; Hoboken, NJ : Wiley, London, 2010.

David Silver, Richard S. Sutton, and Martin Müller. Reinforcement learning of local shape in the game of go. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, page 1053–1058, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 361–368. The MIT Press, 1995.

Satinder P. Singh, Richard S. Sutton, and Leslie P. Kaelbling. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, December 1996.

Satinder P. Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, March 2000.

Steven S. Skiena. *The Algorithm Design Manual*. Springer-Verlag New York, Inc., New York, NY, USA, 1998.

Burrhus F. Skinner. *The behavior of organisms: an experimental analysis*. Appleton-Century, New York, NY, 1938.

Jean-Jacques E. Slotine and Weiping Li. *Applied nonlinear control*. Prentice Hall, Englewood Cliffs, N.J., 1991.

William D. Smart and Leslie P. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 903–910, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

Emilio Soria-Olivas, Juan Gómez-Sanchis, José D. Martín, Joan Vila-Francés, Marcelino Martínez, José R. Magdalena, and Antonio J. Serrano. BELM: bayesian extreme learning machine. *IEEE Transactions on Neural Networks*, 22(3):505–509, March 2011.

Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, MA, 1984.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, July 1988.

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *In Proceedings of the Seventh International Conference on Machine Learning*, page 216–224. Morgan Kaufmann, 1990.

Richard S. Sutton. Introduction: The challenge of reinforcement learning. *Machine Learning*, 8(3-4):225–227, September 1992.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. The MIT Press, 1995.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, March 1998.

Richard S. Sutton and Steven D. Whitehead. Online learning with random representations. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 314–321. Morgan Kaufmann, 1993.

Richard S. Sutton, Andrew G. Barto, and Ronald J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems*, 12(2):19–22, February 1992.

Richard S. Sutton, Csaba Szepesvári, and Hamid R. Maei. A convergent O(n) temporal-difference algorithm for off-policy learning with linear function approximation. In Daphne Koller, Dale Shuurmans, Yoshua Bengio, and Leon Bottou, editors, *Advances in Neural Information Processing Systems*, volume 20, pages 1609–1616. The MIT Press, 2008.

Richard S. Sutton, Hamid R. Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000, Montreal, Quebec, Canada, 2009. ACM.

Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, January 2010.

Csaba Szepesvári and William D. Smart. Interpolation-based q-learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 100–109, New York, NY, USA, 2004. ACM.

Torsten Söderström and Petre Stoica. *Instrumental variable methods for system identification*, volume 161. Springer-Verlag Berlin, 1983.

Shinichi Tamura and Masahiko Tateishi. Capabilities of a four-layered feedforward neural network: four layers versus three. *IEEE Transactions on Neural Networks*, 8 (2):251–255, December 1997.

Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995.

Philip S. Thomas, William C. Dabney, Stephen Giguere, and Sridhar Mahadevan. Projected natural actor-critic. In Christopher J.C. Burges, Leon Bottou, Max Welling, Zoubin Ghahramani, and Killan Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, pages 2337–2345. The MIT Press, 2013.

Edward L. Thorndike. *The elements of psychology.* A. G. Seiler, New York, NY, 1905.

Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. *In Proceedings of The Fourth Connectionist Models Summer School*, 1993.

Hui-Xin Tian and Zhi-zhong Mao. An ensemble ELM based on modified AdaBoost.RT algorithm for predicting the temperature of molten steel in ladle furnace. *IEEE Transactions on Automation Science and Engineering*, 7(1):73–80, February 2010.

Hui-Xin Tian and Bo Meng. A new modeling method based on bagging ELM for day-ahead electricity price prediction. In *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pages 1076–1079, 2010.

Stephan Timmer and Martin Riedmiller. Fitted q iteration with CMACs. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, pages 1–8, 2007.

John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, January 1997.

Dominik E. Uehlinger, Frank A. Gotch, and Lewis B. Sheiner. A pharmacodynamic model of erythropoietin therapy for uremic anemia. *Clinical pharmacology and therapeutics*, 51(1):76–89, January 1992.

University of Michigan. CTM: Digital control Tutorial. http://www.engin.umich.edu/group/ctm/digital/digital.html, 1996. Accesed 10 September 2012.

USRDS. USRDS 2010 annual data report: Atlas of chronic kidney disease and end-stage renal disease in the united states. Technical report, National Institutes of Health, National Institute of Diabetes and Digestive and Kidney Diseases, Bethesda, MD, 2010.

Draguna Vrabie, Octavian Pastravanu, Murad Abu-Khalaf, and Frank L. Lewis. Adaptive optimal control for continuous-time linear systems based on policy iteration. *Automatica*, 45(2):477–484, February 2009.

Anthony Waldock and Brian Carse. Fuzzy q-learning with an adaptive representation. In *IEEE International Conference on Fuzzy Systems, 2008. FUZZ-IEEE 2008. (IEEE World Congress on Computational Intelligence)*, pages 720–725, 2008.

Dianhui Wang and Monther Alhamdoosh. Evolutionary extreme learning machine ensembles with size control. *Neurocomputing*, 102:98–110, February 2013.

Hua O. Wang, Kazuo Tanaka, and Michael F. Griffin. An approach to fuzzy control of nonlinear systems: stability and design issues. *Fuzzy Systems, IEEE Transactions on*, 4(1):14 –23, February 1996.

Christopher Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, England, 1989.

Christopher Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, May 1992.

Scott Weaver, Leemon Baird, and Marios Polycarpou. Preventing unlearning during online training of feedforward networks. In *Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings*, pages 359–364, 1998.

Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

Marco A. Wiering and Martijn van Otterlo, editors. *Reinforcement Learning: State-of-the-Art*. Springer, 1st edition, February 2012.

Sukyung Woo and William J. Jusko. Interspecies comparisons of pharmacokinetics and pharmacodynamics of recombinant human erythropoietin. *Drug metabolism and disposition: the biological fate of chemicals*, 35(9):1672–1678, September 2007.

Si Wu, Youyi Wang, and Shijie Cheng. Extreme learning machine based wind speed estimation and sensorless control for wind turbine power generation system. *Neurocomputing*, 102:163–175, February 2013.

Xin Xu, Hangen He, and Dewen Hu. Efficient reinforcement learning using recursive least-squares methods. *Journal of Artificial Intelligence Research (JAIR)*, 16:259–292, December 2002.

Xin Xu, Dewen Hu, and Xicheng Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, March 2007.

Jay L. Xue, Wendy Peter, James P. Ebben, Susan E. Everson, and Allan J. Collins. Anemia treatment in the pre-ESRD period and associated mortality in elderly patients. *American journal of kidney diseases: the official journal of the National Kidney Foundation*, 40(6):1153–1161, December 2002.

Chee-Wee T. Yeu, Meng-Hiot Lim, Guang-Bin Huang, Amit Agarwal, and Yew-Soon Ong. A new machine learning paradigm for terrain reconstruction. *IEEE Geoscience and Remote Sensing Letters*, 3(3):382–386, October 2006.

Jun-hai Zhai, Hong-yu Xu, and Xi-zhao Wang. Dynamic ensemble extreme learning machine based on sample entropy. *Soft Computing*, 16(9):1493–1502, September 2012.

Weiwei Zong and Guang-Bin Huang. Face recognition based on extreme learning machine. *Neurocomputing*, 74(16):2541–2551, September 2011.

Aprendizaje por refuerzo en espacios continuos: algoritmos y aplicación al tratamiento de la anemia renal

Pablo Escandell Montero, abril de 2014