

DEPARTAMENTO DE INTELIGENCIA ARTIFICIAL  
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

# MODELADO DE COMPORTAMIENTO DE CONDUCTORES CON TÉCNICAS DE INTELIGENCIA COMPUTACIONAL

## TESIS DOCTORAL

**Alberto Díaz Álvarez**  
Máster en Ciencias y Tecnologías de la Computación

### DIRECCIÓN

**Dr. Francisco Serradilla García**  
Doctor Ingeniero en Informática  
**Dr. Felipe Jiménez Alonso**  
Doctor Ingeniero Industrial



hoja según tribunal



Alberto Díaz Álvarez

*Modelado de comportamiento de conductores con técnicas de Inteligencia Computacional*

Tesis doctoral, 5 de mayo de 2018

Dirección: Dr. Francisco Serradilla García, Dr. Felipe Jiménez Alonso

**Instituto Universitario de Investigación del Automóvil**

Universidad Politécnica de Madrid

Campus Sur UPM, Carretera de Valencia (A-3), km7  
28031 Madrid

This work is licensed under a Creative Commons “Attribution-NonCommercial-ShareAlike 3.0 Unported” license.





*Dedicatorias. O en blanco, como me dé.*



## *Resumen*

El tráfico por carretera es un sistema complejo cuyo estudio requiere la inmersión en el sistema físico o bien su simulación. Si el objetivo es abarcar el mayor número posible de casos abstrayéndose de los límites de la realidad física, la opción es el uso de micro-simuladores. En general, este tipo de simuladores están basados en arquitecturas de [Sistemas Multiagente \(MASs, Multiagent Systems\)](#) donde cada elemento es modelado individualmente. El incremento de los recursos computacionales nos permite aumentar la complejidad de las simulaciones, pero los agentes que modelan a los conductores se suelen basar en funciones simplificadas de su comportamiento real. Esto provoca que dichos modelos se adapten poco al comportamiento de un conductor en concreto.

Por otro lado, estamos viviendo una época donde los avances de la [Inteligencia Computacional \(CI, Computational Intelligence\)](#) en predicción, clasificación y modelización están siendo abrumadores. Esto hace pensar que la aplicación de estas técnicas en “agentes conductores” pueda hacer que se comporten “mejor” (en el sentido “más humano”), permitiendo un mejor análisis del tráfico y sus componentes.

Esta tesis explora cómo la [inteligencia computacional](#), aplicada a la modelización de conductores, puede mejorar las simulaciones de tráfico, en el sentido de hacer más reales sus resultados. Concretamente, se centra en técnicas basadas en [Redes Neuronales Artificiales \(ANNs, Artificial Neural Networks\)](#) y [Sistemas De Control Borroso \(FCSs, Fuzzy Control Systems\)](#) para estudiar cómo se comportan las diferentes técnicas al reproducir comportamientos humanos en micro-simuladores a partir de datos reales.

El estudio se centra en dos características existentes en este tipo de simulaciones: el comportamiento longitudinal y el control lateral (cambio de carril). Se concluye que las técnicas basadas en [redes neuronales artificiales](#) son claramente superiores a la hora de modelar el comportamiento en estas tareas, y dentro de ellas, las [Redes Convolucionales \(CNNs, Convolutional Neural Networks\)](#) son idóneas frente a los [Perceptrones Multicapa \(MLPs, Multilayer Perceptrons\)](#) interpretando el entorno del vehículo.

Las implicaciones de estos resultados son variadas. Aún con las limitaciones que nos imponen los micro-simuladores, es posible modelar estos dos comportamientos a partir de datos reales con precisión suficiente como para decir que los agentes se comportan de manera más humana. Además, estos comportamientos son dependientes de los perfiles de conductor que se utilizaron como referencia para obtener los datos, por lo que es posible su uso no sólo para modelar comportamiento más humano, sino para evaluar determinados perfiles dentro de escenarios.



## *Abstract*

Traffic might be conceived as a complex system, the study of which requires immersion in the real system or its simulation. If the goal is to obtain greater behavior trustworthiness or to evaluate a specific element in this environment, the better choice is the use of micro-simulators, commonly based in [Multiagent Systems](#) architectures, in which each element is modelled separately. The increase in computational resources allows us to enhance the complexity of the simulations over time, but modelled driver agents are typically based on functions, more or less complex, that have nothing to do with a true driver's behaviour.

This causes that those models does not adapt to a particular driver. On the other hand, we are living in a thriving era with regard to [Computational Intelligence](#) progress, such as prediction, classification and modelization. This makes us wonder whether the application of this techniques in driver agents could yield better results ("better" as "more human"), allowing us to obtain a better traffic and driver agents analysis.

The present thesis explores how real drivers modelled [computational intelligence](#) could improve traffic simulation making their results more close to reality as possible. Specifically, it focuses on [Artificial Neural Network](#) based techniques and [Fuzzy Control Systems](#). It will study how different techniques mimic human behaviour inside micro-simulators, based on real driver's data.

The study focuses into two existent characteristics inside this type of simulations: the longitudinal and lane-change behaviours. The study concludes that [artificial neural network](#) based techniques are far superior in modelling human behaviour. In lane-change the best among them are [Convolutional Neural Networks](#), which are better in perceiving the surrounding environment patterns than [Multilayer Perceptrons](#).

The implications of these findings are varied. Micro-simulators have some limitations, but it is possible to accurately modellate those behaviours based on real data in order to agents act more humanly. In addition, those behaviours are driver profile dependents, consequently it is possible to use them not only for modelling more human behaviour, but also to evaluate those profiles inside certain situations.



# *Índice general*

<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	2
1.2. Objetivos . . . . .	4
1.3. Estructura de la tesis . . . . .	6
<b>I Estado de la cuestión</b>	<b>7</b>
<b>2. Inteligencia Computacional</b>	<b>9</b>
2.1. De inteligencia artificial a inteligencia computacional . . . . .	9
2.2. El rol del aprendizaje en la inteligencia computacional	13
2.3. Redes Neuronales Artificiales . . . . .	18
2.4. Lógica Borrosa . . . . .	30
2.5. El paradigma de los agentes inteligentes . . . . .	37
<b>3. Simulación de tráfico</b>	<b>43</b>
3.1. Clasificación de simuladores de tráfico . . . . .	44
3.2. Modelos de micro-simulación . . . . .	48
3.3. Software de simulación . . . . .	51
3.4. Entorno seleccionado: SUMO . . . . .	53
<b>4. Modelos de comportamiento</b>	<b>57</b>
4.1. Comportamientos modelados . . . . .	59
4.2. Modelado de conductores clásico . . . . .	60
4.3. Modelado basado en inteligencia computacional . . . . .	64

<b>II Modelos de comportamiento</b>	<b>71</b>
<b>5. Metodología</b>	<b>73</b>
5.1. Perspectiva general del modelo de conducción . . . . .	73
5.2. Extracción de datos de conducción . . . . .	75
5.3. Preparación de los datos . . . . .	79
5.4. Entrenamiento de modelos . . . . .	80
<b>6. Comportamiento longitudinal</b>	<b>83</b>
6.1. Modelo basado en sistemas de control borroso . . . . .	85
6.2. Modelo basado en perceptrones multicapa . . . . .	87
6.3. Comparación entre modelos . . . . .	89
<b>7. Comportamiento lateral</b>	<b>91</b>
7.1. Descripción de los datasets . . . . .	92
7.2. Modelo basado en perceptrones multicapa . . . . .	97
7.3. Modelo basado en redes convolucionales . . . . .	97
7.4. Comparación entre modelos . . . . .	100
<b>8. Modelos específicos de conductores</b>	<b>101</b>
8.1. Modelo longitudinal . . . . .	101
8.2. Control lateral . . . . .	102
8.3. Personalización en modelos de conducción específicos	103
<b>9. Implementación en simulador</b>	<b>105</b>
9.1. Comportamiento en el modelo longitudinal . . . . .	107
9.2. Comportamiento en el modelo de cambio de carril . .	108
<b>III Conclusiones</b>	<b>109</b>
<b>10. Conclusiones</b>	<b>111</b>
10.1. Hipótesis planteadas . . . . .	111
10.2. Sobre los datos, los sensores y su utilidad . . . . .	112
10.3. Sobre las técnicas de inteligencia computacional . . .	113
10.4. Sobre el entorno de simulación . . . . .	114

<b>11. Resultados y líneas de investigación futuras</b>	<b>115</b>
11.1. Propuestas de líneas de investigación futuras . . . . .	115
11.2. Diseminación de resultados . . . . .	116
<b>IV Apéndices</b>	<b>119</b>
<b>A. Ajuste de sistemas de control borroso basado en descenso del gradiente</b>	<b>121</b>
A.1. Representación de un FCS como grafo computacional .	122
A.2. Ejemplo de ajuste de un sistema de control borroso . .	126
<b>B. Visión general de ROS</b>	<b>129</b>
<b>Referencias</b>	<b>131</b>



# Índice de figuras

2.1.	Experimento mental de la <i>Habitación China</i> , por John Searle . . . . .	10
2.2.	Sinergias entre técnicas del Soft Computing . . . . .	11
2.3.	Diferentes objetivos perseguidos por la inteligencia artificial . . . . .	12
2.4.	Separación clásica de conjuntos de datos en problemas de aprendizaje automático . . . . .	14
2.6.	Capacidad de los modelos en función de la cantidad de datos . . . . .	17
2.5.	Ciclo de aplicación de soluciones basadas en inteligencia computacional . . . . .	17
2.7.	Sección de neocórTEX humano . . . . .	18
2.8.	Modelo de neurona artificial de McCulloch y Pitts . . . . .	18
2.9.	Funciones de activación: sigmoide y tangente hiperbólica	19
2.10.	Funciones de activación: ReLU y Leaky-ReLU . . . . .	20
2.11.	Diferencias entre redes de tipo <i>feed-forward</i> y <i>red neuronal recurrente</i> . . . . .	21
2.12.	Estructura general de una red convolucional . . . . .	25
2.13.	Descripción de la capa de convolución . . . . .	26
2.14.	Diferentes operaciones de muestreo . . . . .	27
2.15.	Ejemplo de la operación de dropout para tres epochs sucesivos . . . . .	29
2.16.	Ejemplo de grafo computacional . . . . .	29
2.17.	Derivadas parciales sobre un grafo computacional . . . . .	30
2.18.	Gráfica de funciones de pertenencia triangular y trapezoidal . . . . .	32
2.19.	$t$ -normas de mínimo y producto algebraico. . . . .	32
2.20.	$t$ -conormas de máximo y suma probabilística . . . . .	33

2.21. Esquema de un sistema de control borroso . . . . .	35
2.22. Matriz de representación de reglas borrosas . . . . .	35
2.23. Esquema general de un agente . . . . .	39
2.24. Esquema de agente basado en objetivos . . . . .	39
2.25. Esquema de agente reactivo . . . . .	40
2.26. Esquema de agente basado en utilidad . . . . .	40
3.1. Aspectos medibles del tráfico . . . . .	44
3.2. Clasificación de simuladores según granularidad . . .	45
3.3. Ejemplo de simulador basado en autómata celular . .	46
3.4. Ejemplo de modelo lineal en un espacio continuo . . .	47
3.5. Ejemplo de efecto de ondas de choque en simulación de tipo Nagel-Scherckenberg . . . . .	47
3.6. Simulación de comportamiento en intersección basada en un sistema multiagente . . . . .	49
3.7. Captura de pantalla del simulador MovSim . . . . .	50
3.8. Características obligatorias y deseables del simulador a elegir . . . . .	51
3.9. Captura de pantalla del simulador SUMO . . . . .	53
3.10. Ejemplo de forma de envío de mensajes a través de TraCI	54
3.11. Arquitectura de la plataforma TraaS . . . . .	55
4.1. Los tres niveles jerárquicos de la conducción . . . . .	57
4.2. Diferentes regímenes de aceleración para modelos lon- gitudinales . . . . .	59
4.3. Operaciones involucradas en el proceso de cambio de carril . . . . .	60
4.4. Evolución de los tres tipos generales de modelo de <i>car-following</i> . . . . .	60
4.5. Estructura del modelo de comportamiento propuesto por [Toledo et al., 2007] . . . . .	64
4.6. Principales áreas de aplicación de la CI en los ITS . . .	65
4.7. Ejemplo de aproximación <i>neuro-fuzzy</i> al control de se- ñales de tráfico . . . . .	68
5.1. Esquema general del modelo de conductor planteado .	74
5.2. Vehículo instrumentado utilizado en los ensayos . . . .	75

5.3.	LiDAR modelo VLP-16 de empresa Velodyne . . . . .	76
5.4.	Dispositivo CAN BUS de LACIWEL AB . . . . .	77
5.5.	Cámara Microsoft Kinect . . . . .	77
5.6.	Mapa de los dos recorridos planteados para la captura de datos de conducción . . . . .	78
6.1.	Perfiles de aceleración a ajustar por los modelos. Conjuntos de entrenamiento y de test . . . . .	83
6.2.	Evolución del error durante el entrenamiento en las arquitecturas de FCS para el modelo longitudinal . . .	86
6.3.	Particiones borrosas después de la operación de ajuste en el modelo longitudinal basado en FCS . . . . .	86
6.4.	Comparación del perfil de aceleración real y el inferido por los modelos entrenados . . . . .	87
6.5.	Evolución del error durante el entrenamiento en las arquitecturas de MLP para el modelo longitudinal . .	88
6.6.	Comparación del perfil de aceleración real y el inferido por los modelos entrenados . . . . .	88
6.7.	Comparación entre los dos tipos de modelo longitudinal	89
7.1.	Mapa de profundidad de ejemplo . . . . .	93
7.2.	Ejemplo de la técnica de <i>mirroring</i> . . . . .	95
7.3.	Ejemplo de la técnica de <i>shaking</i> . . . . .	95
7.4.	Momento $t_i$ en el conjunto de datos . . . . .	96
7.5.	Comparación de las precisiones alcanzadas por los modelos de tipo MLP . . . . .	97
7.6.	Esquema de modificación de CNNs para incluir características no espaciales . . . . .	98
7.7.	<i>Padding</i> artificial en los extremos del mapa de profundidad . . . . .	99
7.8.	Gráfica de la tasa de aprendizaje adaptativo por <i>epoch</i> usada para entrenar CNNs . . . . .	99
7.9.	Comparación de las precisiones alcanzadas por los modelos de tipo red convolucional . . . . .	99
7.10.	Matriz de confusión del conjunto de test para el modelo de cambio de carril seleccionado . . . . .	100

8.1.	Comparativa de la evolución del RMSE en test para los sujetos de la arquitectura seleccionada para el modelo longitudinal . . . . .	101
8.2.	RMSE en test entre los sujetos de la arquitectura seleccionada para el modelo de cambio de carril . . . . .	102
8.3.	Diferencia entre los perfiles de aceleración para los diferentes sujetos . . . . .	103
9.1.	Círculo de prueba en el entorno virtual . . . . .	105
9.2.	Medias y desviaciones típicas de los indicadores del modelo longitudinal . . . . .	107
A.1.	Ejemplo de operación en el bloque de fuzzificación . .	122
A.2.	Grafo computacional de la función de pertenencia para la línea ascendente . . . . .	123
A.3.	Grafo computacional de la función de pertenencia para el trapecio . . . . .	123
A.4.	Ejemplo de operación de fuzzificación como grafo computacional . . . . .	124
A.5.	Producto cartesiano de variables borrosas de entrada .	125
A.6.	Particiones borrosas del controlador de ejemplo . . . .	126
A.7.	Superficie de la función que modela el sistema de control borroso de ejemplo . . . . .	126
A.8.	Evolución del sistema de control borroso de acuerdo al conjunto de datos extraido del . . . . .	127

# *Índice de tablas*

3.1. Tabla comparativa de los simuladores seleccionados . . . . .	53
5.1. Resumen de los indicadores obtenidos en los recorridos del experimento . . . . .	81
6.1. Descripción de los conjuntos de datos . . . . .	84
6.2. Resumen de las arquitecturas sistema de control borroso (FCS, fuzzy control system) para el modelo longitudinal . . . . .	85
6.3. Resumen de las arquitecturas de perceptrones multicapa para el modelo longitudinal . . . . .	87
7.1. Descripción de los conjuntos de datos . . . . .	97
7.2. Resumen de las arquitecturas MLP para el modelo de cambio de carril . . . . .	97
7.3. Resumen de las arquitecturas CNN para el modelo de cambio de carril . . . . .	98
8.1. Resumen de los valores de para los modelos específicos de comportamiento longitudinal . . . . .	102
8.2. Precisión alcanzada para los modelos específicos de cambio de carril . . . . .	102
8.3. Comparación de los errores de aceleración en los diferentes modelos longitudinales . . . . .	104
8.4. Comparación de la precisión para los diferentes modelos de cambio de carril . . . . .	104
9.1. Indicadores reales frente a indicadores capturados en simulación . . . . .	106



# Glosario

*AORTA (Approximately Orchestrated Routing and Transportation Analyzer).* Entorno de microsimulación multiagente de tráfico desarrollado en el departamento de Ciencias de la Computación de la Universidad de Austin (Texas). Url: <http://www.aorta-traffic.org/>.

[52](#)

*autoencoder* Técnica de aprendizaje para reducir el tamaño de un vector de entrada transformándolo en una representación de ésta en una menor dimensión. En *ML* se utilizan tanto para reducción de dimensión de entradas, recuperación de datos a partir de vectores de características o reducción de ruido en señales entre otros. [14](#), [22](#)

*car-following* Modelo de control de conductor longitudinal que basa su comportamiento en el del vehículo del mismo carril que le precede. [46](#), [47](#), [53](#), [59–63](#), [65–67](#), [76](#)

*deep belief network* En castellano, “redes profundas de creencia”, se tratan de modelos probabilísticos para aprender representaciones jerárquicas de datos de manera no supervisada. Están compuestas de múltiples capas de *RNNs* conectadas entre si dos a dos. [14](#), [17](#)

*framework* Conjunto estandarizado de herramientas, conceptos, normas y metodología para solucionar problemas de características similares de una manera organizada. [127](#), [128](#)

*free-flow* Modelo de control de conductor longitudinal que genera su comportamiento de acuerdo al estado de la vía cuando no hay vehículo que le preceda. [62](#), [65](#)

*GPL (General Public License)*. Licencia de software que garantiza las libertades del software. Asegura que cualquier versión, extensión o software derivado de éste permanecerá siendo software libre. Su última versión es la 3.0. Url: <https://www.gnu.org/licenses/gpl-3.0.html>. [52–54](#)

*GPU (Graphics Processing Unit)*. Componente similar a una CPU especializado en el cálculo de operaciones sobre matrices para el

procesamiento gráfico en videojuegos, también usada para el trabajo con grandes cantidades de datos que requieren ese tipo de operaciones (e.g. [aprendizaje automático](#)). [17](#), [78](#)

*Hard Computing* Forma de denominar a la computación clásica en contraposición al término *Soft Computing*. [13](#)

*HBEFA* *The Handbook Emission Factors for Road Transport* es una guía que propone unas tablas de emisiones específicas en g km<sup>-1</sup> para todas las categorías actuales de vehículos de carretera (concretamente automóviles de pasajeros, vehículos ligeros, vehículos pesados, autobuses y motocicletas).. [114](#)

*Inteligencia Artificial Distribuida* Rama de la [inteligencia artificial \(AI, artificial intelligence\)](#) donde se estudian las técnicas de aplicación, coordinación y distribución de conocimiento en un entorno multiagente. [41](#)

*MatSIM* (*Multi-Agent Transport Simulation*). Software de microsimulación multiagente desarrollado en la ETH Zürich. Url: <http://matsim.org>. [52](#)

*MitSIM* (*Microscopic Traffic SIMulator*). Software de microsimulación desarrollado por el laboratorio de sistemas inteligentes de transporte del MIT. Url: <https://its.mit.edu/software/mitsimlab>. [52](#), [63](#)

*MovSim* (*Multi-model Open-source Vehicular-traffic Simulator*). Software de microsimulación que implementa tanto modelo multiagente y como modelo basado en CAs. Url: <http://www.movsim.org/>. [XVIII](#), [50](#), [52](#)

*NMEA* En el dominio del geoposicionamiento, es un formato estándar de intercambio de datos soportado por todos los fabricantes de [GPS](#). [74](#)

*OSI* Estándar de comunicaciones en red desarrollado por la ISO en 1980 para servir como modelo de referencia para la implementación de protocolos de red basados en arquitectura en capas. [75](#)

*Python* Lenguaje de programación OSS. Url: <https://www.python.org/>. [XXV](#), [52](#), [54](#), [128](#)

*RMSE* *Root Mean Squared Error*, una métrica para el cálculo del error entre dos conjuntos de medidas sobre los mismos elementos y definida como

$$\sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}}$$

Siendo  $y_i$  e  $\hat{y}_i$  cada una de las  $n$  medidas tomadas sobre el mismo conjunto . [XX](#), [XXI](#), [23](#), [82](#), [83](#), [86](#), [99–101](#)

*sistema experto* Sistema que emula a un humano en la toma de decisiones de un dominio en el que es experto. [11](#)

*Soft Computing* Paradigma de computación por el que se hace uso de técnicas de resolución de problemas que manejan información incompleta, inexacta o con ruido. [XVII](#), [XXIV](#), [13](#)

*SOM Self-Organizing Maps* o Mapas Autoorganizados, son un tipo de [ANN](#) que aprenden una representación topológica de los datos de entrada de una forma no supervisada. [14](#), [22](#)

*SUMO (Simulation of Urban MObility)*. Entorno de micro y mesosimulación multiagente desarrollado por el instituto de sistemas de transporte del DLR (Centro Aeroespacial Alemán). Url: <http://www.dlr.de>. [XIII](#), [XVIII](#), [XXV](#), [52–55](#), [71](#), [103](#), [112](#), [114](#)

*TCP/IP* Protocolo de comunicación desarrollado en 1970 para permitir la comunicación entre dispositivos a través de una red de comunicaciones. [127](#)

*TORCS (The Open Racing Car Simulator)*. Software de simulación creado en un principio como videojuego de carreras y que ha evolucionado hacia plataforma de simulación de técnicas de inteligencia artificial aplicadas a la conducción. Url: <http://torcs.sourceforge.net/>. [43](#), [113](#)

*TraaS (TraCI as a Service)*. APIs basado en SOAPs para interactuar con el simulador [SUMO](#) cuando está funcionando en modo servidor. Url: <http://traas.sourceforge.net/cms/>. [XVIII](#), [55](#)

*TraCI (Traffic Control Interface)*. Término que sirve tanto para denominar al protocolo de comunicación ofrecido por [SUMO](#) en modo servidor para la interacción remota con la simulación como para denominar a la librería desarrollada para abstraer dicho protocolo cuando se trabaja desde [Python](#). Url: <http://www.sumo.dlr.de/wiki/TraCI>. [54](#), [55](#)



# *Abreviaturas*

*AASIM* Autonomous Agent SIMulation Package. 66

*ACL* Agent Communications Language. 41

*ADAS* Advanced Driver Assistance System. 57

*AI* artificial intelligence. XIII, XVII, XXIV, 1, 9–13, 37, 38, 41, 63

*ANN* artificial neural network. IX, XI, XIII, XXV, 5, 9–11, 16–18, 21, 23, 29, 38, 64–67, 82, 125

*API* Application Programming Interface. 52, 127, 128

*BDI* Belief-Desire-Intention. 41

*BSD* Berkeley Software Distribution. 127

*CA* cellular automaton. XVIII, 44, 46–49

*CAN* Controller Area Network. XIX, 73, 75

*CI* computational intelligence. IX, XI, XIII, XIV, XVII, XVIII, 1, 2, 4–6, 9, 11–15, 17, 19, 21, 23, 25, 27–29, 31, 33, 35, 37–39, 41, 49, 59, 63–65, 67, 71, 90, 109, 111, 114, 119

*CNN* convolutional neural network. IX, XI, XIV, XVII, XIX, XXI, 22, 24–28, 79, 89, 94–97, 111, 112

*COA* centroid of area. 36

*DLC* Discretionary Lane Change. 61–63, 66, 67

*DVO* Driver-Vehicle Object. 59

*DVU* Driver-Vehicle Unit. 43, 49, 51, 52, 59, 81

*FCS* fuzzy control system. IX, XI, XIV, XV, XVIII–XXI, 32, 34–36, 50, 66, 67, 79, 81–84, 87, 111, 119–125

*FL* fuzzy logic. XIII, 9, 11, 30–35, 38, 64–67

*FLOWSIM* Fuzzy LOgic motorWay SIMulation. 66

- GA* genetic algorithm. 11
- GHR* Gazis-Herman-Rothery. 60, 66
- GM* Generalized Model. 60
- GPS* Global Positioning System. XIX, XXIV, 73–75, 110
- HMM* Hidden Markov Model. 67
- IDM* Intelligent Driver Model. 54
- IMU* Intertial Measurement Unit. 67
- INSIA* Instituto Universitario de Investigación del Automóvil. 74
- ITS* Intelligent Transport Systems. XVIII, 1, 2, 57, 63–65, 112
- KQML* Knowledge Query and Manipulation Language. 41
- LDA* Latent Dirichlet Allocation. 67
- LiDAR* Light Detection and Ranging. XVIII, 63, 73, 74, 78, 91, 92, 94, 103, 110, 111
- LOLIMOT* LOcal LInear MOdel Tree. 67
- LRN* local response normalization. 27
- LSTM* Long-Short Term Memory. 22
- MAS* multiagent system. IX, XI, XVIII, 2, 4, 41, 45, 47–51, 57
- ML* machine learning. XVII, XXIII, XXIV, 11, 15, 64
- MLC* Mandatory Lane Change. 61–63, 66, 67
- MLP* multilayer perceptron. IX, XI, XIV, XIX, XXI, 14, 22, 23, 25–28, 65, 79, 81, 82, 85, 87, 89, 95, 98, 111, 112
- NLP* Natural Language Processing. 12
- OSRF* Open Source Robotics Foundation. 127
- OSS* Open Source Software. 51, 52
- ReLU* Rectified Linear Unit. XVII, 17, 20, 21, 86, 90
- RNN* recurrent neural network. XVII, XXIII, 14, 21, 22, 65, 114
- ROS* Robot Operating System. 74, 75, 127, 128
- RS* recommender system. 11, 16

*SVM* Support Vector Machine. 67

*V<sub>2</sub>I* Vehicle-to-Infraestructure. 50

*V<sub>2</sub>V* Vehicle-to-Vehicle. 50



# **1** *Introducción*

La **inteligencia artificial** como área de conocimiento ha experimentado un creciente interés en los últimos años. Esto no siempre ha sido así; desde su nacimiento ha ido alternando épocas de mucha actividad con otras de apenas avance debido a las altas expectativas que genera cada nuevo avance en el área. Sin embargo, en la actualidad es muy difícil encontrar un campo que no se beneficie directamente de sus técnicas.

Una de las razones es su carácter multidisciplinar ya que, aunque pertenece al campo de la computación, es transversal a muchos otros campos de naturaleza muy diferente como, por ejemplo, la biología, la neurología o la psicología.

Dentro del área de la **inteligencia artificial** es común diferenciar dos tipos de aproximaciones para representar el conocimiento: el enfoque **clásico**, que postula que se puede reducir a un conjunto de símbolos con operadores para su manipulación, y el enfoque de la **inteligencia computacional**, que defiende que el conocimiento se alcanza a través del aprendizaje, y que basa sus esfuerzos en la simulación de elementos de bajo nivel esperando que el conocimiento “emerga” de su interacción.

El límite entre ambos conjuntos no está perfectamente definido, más aún si tenemos en cuenta las diferentes terminologías existentes, las sinergias entre distintas técnicas dentro del área y los diferentes puntos de vista sobre éstas por parte de los autores. Sin embargo, una de las principales diferencias de ambos paradigmas es el punto de vista a la hora de solucionar problemas, siendo la aproximación **top-down** la usada en problemas de **inteligencia artificial** clásica y la **bottom-up** la típica usada en la **inteligencia computacional**. Revisaremos las diferencias entre conceptos de diferentes autores en el capítulo **Inteligencia Computacional**.

Uno de los campos de aplicación es el de los **Sistemas Inteligentes de Transporte (ITS, Intelligent Transport Systems)**. Éstos se definen como un conjunto de aplicaciones orientadas a gestionar el transporte en todos sus aspectos (e.g. conducción eficiente, gestión del tráfico o señalización en redes de carreteras) para hacerlos más eficientes y seguros. El interés es tal que en el año 2010 se publicó la directiva 2010/40/UE (ver [Directive, 2010]) en la que se estableció el marco

de implantación de los [ITS](#) para toda la Unión Europea, quedando éstos definidos como:

“[Los [ITS](#) …] son aplicaciones avanzadas que, sin incluir la inteligencia como tal, proporcionan servicios innovadores en relación con los diferentes modos de transporte y la gestión del tráfico y permiten a los distintos usuarios estar mejor informados y hacer un uso más seguro, más coordinado y más inteligente de las redes de transporte.”

La *conducción* es una actividad muy compleja que involucra la ejecución de muchas tareas cognitivas pertenecientes a diversos niveles de abstracción. El concepto del *tráfico* puede verse como un sistema complejo que emerge de las interacciones de agentes muy diversos, incluyendo a aquellos que realizan la tareas de conducir. El comportamiento durante la tarea de conducción es un objeto interesante de estudio: la evaluación de los conductores para conocer su manera de actuar en determinados escenarios nos permite, por ejemplo, detectar qué factores pueden afectar más o menos sobre determinados indicadores. Sin embargo, la evaluación en algunos casos puede no ser posible debido a limitaciones como el tiempo, el dinero o la peligrosidad del escenario, entre otros.

Los simuladores de tráfico son una solución para muchas de estas limitaciones, pero suelen basar su funcionamiento en modelos de conductor que responden a funciones más o menos simplificadas, alejadas de la realidad, además con pocas o ninguna opción de personalización. Esto provoca que dichos modelos se adapten poco al comportamiento de un conductor en concreto.

Esta tesis pretende explotar la generación de modelos de conductor para simuladores que respondan al comportamiento de conductores reales usando, para ello, técnicas pertenecientes al campo de la [inteligencia computacional](#). Estos perfiles extraídos se aplicarán a un entorno de simulación basado en [sistemas multiagente](#). Así, una vez configurado el entorno, se podrán estudiar aspectos generales como la evolución del tráfico con determinados perfiles o particulares como el estilo de conducción o el impacto de los sistemas de asistencia.

### 1.1 Motivación

Los conceptos introducidos al comienzo del capítulo obedecen a una necesidad de la sociedad en la que vivimos, y que afecta tanto a nuestra generación como a las venideras: la eficiencia en el transporte. Dado que es imprescindible saber que existe un problema para resolverlo, nada mejor que puntualizar algunos hechos de sobra conocidos:

- En el año 2016, el número de vehículos a nivel mundial superó los 1,350 millones, con una tendencia creciente [[OICA, 2015](#)]. Reducir en un pequeño porcentaje el consumo evita la emisión de toneladas

de gases considerados nocivos para el medio ambiente y el ser humano.

- Aunque existen diferentes puntos de vista acerca de cuándo se agotarán las reservas de petróleo, los combustibles fósiles son recursos **finitos**. Lo más probable es que no se lleguen a agotar debido a la ley de la oferta y la demanda, pero hay que recordar que el petróleo se usa como base para la producción de otros muchos tipos de productos, como por ejemplo la vaselina, el asfalto o los plásticos.
- A pesar de que existen estudios que asocian el calentamiento global a causas no humanas [Jaworowski, 2004], existe consenso en la comunidad científica de que la emisión de gases, por parte de la acción humana, está correlacionada con el aumento de la temperatura del planeta [Oreskes, 2018]. De ser así, es necesaria una reducción drástica en su emisión para mitigar y revertir sus efectos.
- La conducción eficiente afecta directamente a factores relacionados con el número de accidentes de tráfico. Un factor de sobra conocido es el de la velocidad, que influye no sólo en el número sino en la gravedad de los accidentes ([Imprialou et al., 2016]). Otros indicadores son las aceleraciones, deceleraciones y maniobras de cambio de dirección, cuyas frecuencias son directamente proporcionales a la agresividad, falta de seguridad y accidentes, e inversamente proporcionales a la eficiencia ([Dingus et al., 2006, Lerner et al., 2010]).

Éstos son sólo algunos hechos que ponen de manifiesto la necesidad de conseguir que la conducción sea una actividad más eficiente y segura. Por ello, la **conducción eficiente** o *eco-driving* se define como sigue:

"[Eco-driving is . . .] the practice of driving in such a way as to minimize fuel consumption and the emission of carbon dioxide."  
—*Oxford Dictionary*.

Si es posible discriminar entre conductores eficientes y no eficientes, se pueden identificar los hábitos recurrentes en estos últimos y adecuar la formación para eliminar dichos hábitos. Más aún teniendo en cuenta la relación existente entre la peligrosidad y algunas conductas agresivas. Un ejemplo donde la identificación de perfiles no eficientes pueden tener impacto claramente económico y social es el de las empresas cuya actividad se basa en el transporte de mercancías o de personas.

Sin embargo, identificar la conducta de un conductor no es sencillo, dado que su comportamiento se ve condicionado por muchos factores como el estado de la ruta, el del tráfico o el estado físico o anímico. Además, la ambigüedad de las situaciones dificulta todavía más la identificación. Por ejemplo, un conductor puede ser clasificado en un momento como agresivo o no eficiente únicamente porque su

comportamiento ha sido condicionado por las malas reacciones de otros conductores.

El análisis de todos los posibles casos es una tarea prácticamente imposible. Por ello, las simulaciones pueden dar una estimación de los posibles resultados de un estudio en el mundo real. Las simulaciones con *sistemas multiagente* representan a los conductores como agentes independientes, permitiendo la evaluación del comportamiento tanto individual como general del sistema en base a sus individuos a través de iteraciones discretas de tiempo.

Si el comportamiento de dichos agentes es extraído a partir de los datos reales de conductores, su comportamiento dentro de la simulación podría ser considerado como fuente de datos aproximada del comportamiento en situaciones de tráfico del mundo real. De esta forma, se dispondría de un marco de trabajo para la comparación de diferentes conductores sin necesidad de exponerlos a todos y cada uno de los posibles eventos posibles. También sería factible evaluar sistemas de asistencia evitando los problemas de no comparabilidad de condiciones del entorno entre pruebas.

Demostrar que la evaluación de un modelo del conductor en entornos simulados es equivalente a la evaluación de conductores en entornos reales implica que se pueden comparar dos conductores usando un criterio objetivo, es decir, sin depender del estado del resto de factores a la hora de realizar la prueba de campo. Dicho de otro modo, implicaría que es posible comparar la eficiencia de dos conductores independientemente del estado del tráfico e incluso, sobre rutas diferentes.

## 1.2 *Objetivos*

El objetivo general de esta tesis es el de simular el comportamiento humano de conductores en entornos de micro-simulación, de tal manera que (i) se aumente elrealismo de los modelos existentes de conductor en estos entornos, y (ii) éstos puedan ser usados tanto como fuentes de datos así como marco de referencia para la comparación de características entre conductores.

Para ello, se tratarán de demostrar las hipótesis H<sub>1</sub> y H<sub>2</sub>, quedando dicha demostración dentro de los límites impuestos por los supuestos y restricciones indicados más adelante.

**Hipótesis 1 (H<sub>1</sub>):** *La aplicación de técnicas pertenecientes al campo de la inteligencia computacional sobre datos de conductores reales para la obtención de un modelo de conducción permitirá incorporar características humanas no reproducibles por los modelos lineales existentes, mejorando la calidad de dichas simulaciones.*

**Hipótesis 2 (H<sub>2</sub>):** *La aplicación de técnicas pertenecientes al campo de la inteligencia computacional con datos extraídos de un entorno de micro-*

*simulación de espacio continuo y tiempo discreto basado en sistemas multi-agente permitirá modelar, de manera fiel a la realidad, el comportamiento de conductores reales.*

Se consideran además los siguientes objetivos específicos:

- Estudiar y aplicar técnicas de la **inteligencia computacional** sobre el área de la conducción.
- Realizar un estudio naturalista de conducción<sup>1</sup> sobre conductores reales para:
  1. Generar modelos personalizados de conductor a partir de los datos de conducción obtenidos.
  2. Aplicar modelos de conductores a entornos de simulación multiagente.
  3. Validar los modelos de conductor contra conductores reales.
- Estudiar la efectividad de sistemas de asistencia encaminados a mejorar la eficiencia y analizar el comportamiento de conductor.

#### 1.2.1 Supuestos

- La circulación se supone por la derecha de la vía en el sentido de la circulación, siendo los carriles de lento a rápido de derecha a izquierda respectivamente.
- La captura de datos se realizará durante el día, con buena visibilidad y sin lluvia para minimizar el impacto de la variabilidad del entorno en el comportamiento.
- El tipo de vehículo sobre el que modelar el comportamiento será de tipo *turismo*, al ser (i) la principal tipología de vehículo usada en las carreteras y (ii) la tipología del vehículo instrumentado del que dispone nuestro grupo.
- El conductor a modelar pertenecerá al grupo de conductores varón de 30 a 34 años, edades con las que se corresponden los sujetos que aceptaron formar parte del experimento.

#### 1.2.2 Restricciones

- Los modelos generados se corresponderán a entornos urbanos.
- Reduciremos el comportamiento del conductor a los de circulación en línea y cambio de carril<sup>2</sup>.
- La resolución máxima del modelo creado será de 10 Hz.
- En el caso de los modelos que hacen uso de **redes neuronales artificiales**, las razones del comportamiento inferido no podrán ser explicadas.

<sup>1</sup> Un estudio naturalista de conducción basa su funcionamiento en la captura masiva de datos de conducción, normalmente involucrando una gran cantidad de sensores, para analizar el comportamiento del conductor, las características del vehículo, la vía, etcétera. La cantidad de sensores y la frecuencia hacen que la tarea de analizar y extraer conclusiones sea una tarea prácticamente imposible para un humano, por lo que es necesario el uso de técnicas de análisis de datos que suelen recaer en los campos de la estadística y del aprendizaje automático.

<sup>2</sup> Son conocidos en la literatura como modelos de **aceleración** y modelos de **cambio de carril**. Entraremos en detalle sobre ambos conceptos en el capítulo **Modelos de comportamiento**.

### 1.3 *Estructura de la tesis*

La tesis se divide en tres partes. La primera se corresponde con el estado de la cuestión, compuesta por los capítulos [Inteligencia Computacional](#), [Simulación de tráfico](#) y [Modelos de comportamiento](#), explicando en qué punto se encuentra la literatura de los temas en los que se apoya la presente tesis.

En la segunda parte se desarrolla el tema del modelado de conductores: En [Metodología](#) se describe la adquisición y tratamiento de datos, junto con algunas de las decisiones tomadas durante el proceso; después, en [Comportamiento longitudinal](#) y [Comportamiento lateral](#) se desarrollan los procesos de obtención de los modelos longitudinal y de cambio de carril respectivamente. En [Modelos específicos de conductores](#) se extiende el proceso entrenamiento a conductores en particular y en [Implementación en simulador](#), se introducen los comportamientos de los modelos de conductor en un simulador.

Por último, se expondrán las [Conclusiones](#) junto con una serie de posibles [Resultados y líneas de investigación futuras](#) consideradas interesantes tras la realización de la tesis.

## PARTE I

### ESTADO DE LA CUESTIÓN



## 2 *Inteligencia Computacional*

Todo elemento dentro de un entorno se ve influenciado por multitud de variables que determinan en mayor o menor grado su comportamiento. En la mayoría de los casos es muy complicado determinar el grado de efecto de estas variables, identificar las relaciones que existen entre las mismas o incluso determinar cuáles son.

No existe una definición globalmente aceptada de **inteligencia computacional**. Dependiendo del autor, se la considera desde un sinónimo de la **inteligencia artificial** hasta un campo completamente diferenciado.

En esta tesis hablaremos desde el punto de vista mayoritario, es decir, como rama de la **inteligencia artificial** donde se agrupan todas aquellas técnicas que tratan de aprender soluciones a problemas a través del análisis de información presente en conjuntos de datos, en el entorno o ambos [Rutkowski, 2008, Siddique and Adeli, 2013].

El resto del capítulo ofrecerá una visión histórica para explicar el por qué de esta definición, justificará la importancia que tiene el aprendizaje dentro de este área, describirá las **redes neuronales artificiales** y la **lógica borrosa** necesarias para sentar las bases del posterior desarrollo de la tesis y, por último, dará nociones de qué son los agentes y por qué este punto de vista es útil para nuestro cometido.

### 2.1 *De inteligencia artificial a inteligencia computacional*

Es difícil datar el comienzo del interés del ser humano por emular la inteligencia humana. Los silogismos en la antigua Grecia para modelar el conocimiento como reglas, los autómatas mecánicos de los filósofos modernos (siglos XVII al XIX) donde los cuerpos vivos son como un reloj, o la electricidad (siglos XIX y XX) capaz de animar constructos, son sólo ejemplos de cómo cada nuevo avance en la ciencia ha ido acompañado de un intento de simular cuerpo y mente humanos.

Podemos aventurarnos a decir que a principios del siglo XX se comienza a gestar el área de la **AI** con los trabajos relacionados con los principios del **conexionismo**<sup>1</sup>. Estas ideas saltaron al mundo de la computación hacia mediados del siglo XX, cuando Warren S. McCu-

<sup>1</sup> El enfoque del **conexionismo** postula que tanto la *mente* como el *conocimiento* son comportamientos complejos que emergen de redes formadas por unidades sencillas (i.e. neuronas) interconectadas. Se puede considerar a Santiago Ramón y Cajal como principal precursor de esta idea por sus trabajos acerca de la estructura de las neuronas y sus conexiones (e.g. [Ramón y Cajal, 1888] y [Ramón y Cajal, 1904]).

<sup>2</sup> Muchos autores prefieren nombrar este hito, junto con el trabajo "The organization of behavior" [Hebb, 1949] de Donald O. Hebb como el punto de partida del área debido a su connotación computacional

<sup>3</sup> El **Test de Turing** es una metodología para probar si una máquina es capaz de exhibir comportamiento inteligente similar al del ser humano. En ella, dos humanos ( $H_1$  y  $H_2$ ) y una máquina ( $M$ ) están separados entre sí pero pudiendo intercambiarse mensajes de texto.  $H_2$  envía preguntas a  $H_1$  y  $M$  y éstos le responden. Si  $H_2$  no es capaz de identificar qué participante es la máquina, se puede concluir que ésta es inteligente.

<sup>4</sup> El concepto de "pensar" es un tema controvertido incluso en el propio ser humano: ¿es inherentemente biológico? ¿surge de la mente? Tanto si sí como si no, ¿de qué forma lo hace? Por ello existen detractores de la validez del Test de Turing. Un ejemplo es el experimento de la habitación china (Figura 2.1), donde se demuestra la invalidez argumentando que la máquina ha aprendido a realizar acciones sin *entender* lo que hace y por qué lo hace. Sin embargo, ¿qué garantías tenemos de que el humano sí es capaz? Si los ordenadores operan sobre símbolos sin comprender el verdadero contenido de éstos, ¿hasta qué punto los humanos lo hacen de forma diferente?.

<sup>5</sup> El **AI Winter** no sólo se produjo por el efecto gurú del libro *Perceptrons*, aunque éste fue la gota que colmó el vaso. A la emoción inicial por los avances le siguieron muchos años de promesas incumplidas, investigación sin resultados significativos, limitaciones de hardware y el aumento de la complejidad del software (los comienzos de la crisis del software [Dijkstra, 1972]). Todo ello provocó un desinterés y una disminución de la financiación que se retroalimentaron la una a la otra.

Figura 2.1: La *Habitación China* de John Searle es un experimento mental por el que se trata de demostrar la invalidez del Test de Turing. Partiendo de un Test de Turing donde la máquina ha aprendido a hablar chino. Reemplazamos la máquina por un humano sin idea de chino pero con un manual de correspondencias de ideogramas. Cuando una persona le manda mensajes en chino, esta otra responde usando el manual, por lo que podemos afirmar que ni la persona ni la máquina saben chino, aunque ambas hayan pasado el Test de Turing.

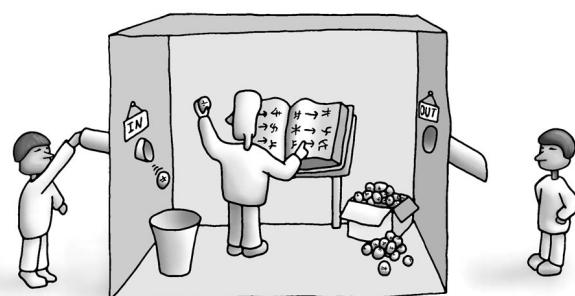
lloch y Walter Pitts publicaron su trabajo "A logical calculus of the ideas immanent in nervous activity" [McCulloch and Pitts, 1943]<sup>2</sup>, donde se describe el primer modelo artificial de una neurona.

El trabajo suscitó tanta expectación que se comenzó a especular sobre la posibilidad de emular (una vez más) la inteligencia humana en máquinas. Uno de los resultados fue la publicación de un artículo por parte de Alan Turing que comenzaba con la frase "Can machines think?" [Turing, 1950], introduciendo el famoso Test de Turing<sup>3</sup> por el que el autor pretendía establecer una metodología para determinar si una máquina podía ser considerada inteligente y por tanto podría llegar a pensar<sup>4</sup>.

Pocos años después de la publicación del artículo, en el año 1956, se celebró la **Conferencia Dartmouth** [McCarthy et al., 1956]. En ésta, el tema de la conferencia fue la pregunta del artículo de Turing, y fue John McCarthy en este preciso momento cuando acuñó el término que todos conocemos como **AI**.

A partir de este momento, la investigación en el área recibió mucha atención por parte de investigadores y gobiernos. Después de todo era un área nueva, muy prometedora y con mucho trabajo por delante. Tras su nacimiento el campo comenzó a dar resultados, pero la expectación y las promesas no dejaban ver que los resultados se obtenían en problemas relativamente simples, muy formales y en general estériles, donde en realidad no era necesaria demasiada información para generar un conocimiento del entorno en el que los modelos se movían.

Dado que los estudios en el área estaban dominados por aquellos relacionados con las ideas del conexionismo, la publicación del libro "Perceptrons" [Minski and Papert, 1969] de Marvin Minsky y Seymour Papert en 1969 supuso un notable varapalo para las investigaciones. En él se expusieron las limitaciones de los modelos de **redes neuronales artificiales (ANNs, artificial neural networks)** desarrollados hasta la fecha, y el impacto fue de tal envergadura que la investigación en el área se abandonó casi por completo. Concretamente el conexionismo prácticamente desapareció de la literatura científica durante dos décadas. Es lo que se conoce como el primer *AI Winter*<sup>5</sup>.



El interés por el campo volvió de nuevo a principios de los 80 con la aparición en escena de los primeros **sistemas expertos**, considerados como el primer caso de éxito en la **inteligencia artificial** ([Russell et al., 2003]).

A finales de la década, sin embargo, empezaron a resurgir de nuevo los enfoques conexionistas, debido en gran parte a la aparición de nuevas técnicas de entrenamiento en perceptrones multicapa y por el concepto de activación no lineal en neuronas [Rumelhart et al., 1985, Cybenko, 1989]. En este momento, los **sistemas expertos** empezaron a perder interés frente al nuevo avance del conexionismo<sup>6</sup>. Esta época se suele identificar como el segundo *AI Winter*, ya que tanto la investigación como las inversiones en el área de **sistemas expertos** disminuyeron. Sin embargo, el efecto no fue ni mucho menos equiparable al del primero.

Junto con el resurgir del conexionismo, otras técnicas alineadas como la **lógica borrosa** o los **Algoritmos Genéticos** (GAs, Genetic Algorithms) también ganaban popularidad, y entre ellas retroalimentaban los éxitos gracias a sus sinergias. Esto provocó una explosión de terminologías para diferenciar las investigaciones en curso de la propia AI clásica. Por un lado se evitaba el conflicto, nombrando las áreas de trabajo con un término más acorde con el comportamiento o técnica utilizada. Por otro, se separaba de las connotaciones negativas que fue cosechando la **AI** con el paso de los años (i.e. promesas, pero no resultados).

Lo verdaderamente interesante es ver la evolución de la literatura durante estos años. En el nacimiento del campo, se buscan literalmente máquinas que piensen como humanos, o al menos seres racionales, con mente. Con el paso de los años, el área va tendiendo hacia la búsqueda de conductas y comportamientos inteligentes cada vez más específicos. Este hecho se hace más patente en este momento, donde cada investigación se nombra de cualquier forma menos con el término **Inteligencia Artificial** (e.g Aprendizaje Automático (ML, Machine Learning), Sistemas De Recomendación (RSs, Recommender Systems), o Procesamiento De Lenguaje Natural (NLP, Natural

<sup>6</sup> Esto, evidentemente no sentó bien a los autores prolíficos en **sistemas expertos**. Mientras que el enfoque en estos sistemas es el clásico en la computación, donde los problemas (en este caso el conocimiento experto) son resueltos mediante operaciones sobre un lenguaje de símbolos, el enfoque del conexionismo postula que la *mente*, el comportamiento inteligente, emerge de modelos a más bajo nivel. Por ello, algunas voces se alzaron contra lo que se consideraba el *enfoque incorrecto* de la **inteligencia artificial**. Después de todo, los modelos desarrollados en los métodos clásicos son fáciles de interpretar mientras que los del enfoque conexionista no son del todo deducibles, más aún si estos problemas son de naturaleza estocástica.

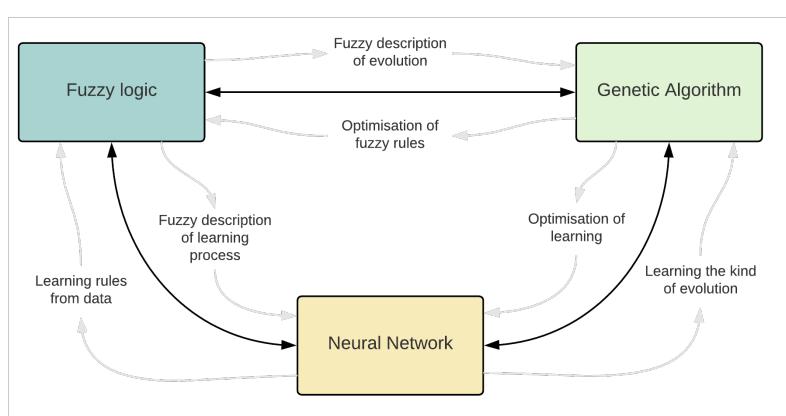


Figura 2.2: Las **redes neuronales artificiales**, junto con otras técnicas como la **lógica borrosa** o los **algoritmos genéticos** se complementan perfectamente, y esta una de las razones de la vuelta a la investigación en el área de la **inteligencia artificial**.

Figura 2.3: Diferentes objetivos perseguidos por la [inteligencia artificial](#). Las filas diferencian entre pensamiento o comportamiento mientras que las columnas separan entre inteligencia humana o el ideal de la inteligencia (racionalidad). Fuente: *Artificial Intelligence: A Modern Approach* (3<sup>rd</sup> Ed.), [Russell et al., 2003].

	Humanly	Rationally
Thinking	<i>The exciting new effort to make computers think... machines with minds, in the full and literal sense.</i>	<i>The study of the computations that make it possible to perceive, reason and act.</i>
Acting	<i>The study of how to make computers do things at which, at the moment, people are better.</i>	<i>Intelligent behaviour in artifacts.</i>

Language Processing)). Es evidente que la [inteligencia artificial](#) se puede observar desde diferentes puntos de vista, todos perfectamente válidos. En [Russell et al., 2003], tras un análisis de las definiciones existentes en la literatura por parte de diferentes autores, se hace énfasis en este hecho mostrando los diferentes puntos de vista a la hora de hablar de lo que es la [inteligencia artificial](#). El resumen se puede observar en la figura 2.3.

Volviendo a la terminología, muchas de las técnicas se fueron agrupando dentro de diferentes áreas. Una de ellas es la conocida como [Inteligencia Computacional](#). Dado que persigue el mismo objetivo a largo plazo que la [inteligencia artificial](#) parece lógico mantenerla como un subconjunto y no como un nuevo campo del conocimiento humano. Sin embargo, algunos autores abogan por que es un nuevo campo diferenciado de la [inteligencia artificial](#).

Podemos definir la [inteligencia computacional](#) como la “*rama de la inteligencia artificial que aporta soluciones a tareas específicas de forma inteligente a partir del aprendizaje mediante el uso de datos experimentales*” [Rutkowski, 2008]. A diferencia de la aproximación clásica de la [inteligencia artificial](#), se buscan aproximaciones a las soluciones y no las soluciones exactas. Esto es debido a que muchos problemas son de naturaleza compleja, ya sea debido a la relación entre sus múltiples variables, la falta de información o la imposibilidad de traducirlos a lenguaje binario o la explosión combinatoria.

Se puede establecer el año 1994 como en el que la [inteligencia computacional](#) nace formalmente como área, coincidiendo con el cambio de nombre del *IEEE Neural Networks Council* a *IEEE Computational Intelligence Society*<sup>7</sup>. Poco antes, en 1993, Bob Marks presentaba las que él consideraba diferencias fundamentales entre la [inteligencia artificial](#) clásica y la [inteligencia computacional](#), resumiéndolas en la siguiente frase:

“Neural networks, genetic algorithms, fuzzy systems, evolutionary programming, and artificial life are the building blocks of [computational intelligence](#).”

<sup>7</sup> <http://cis.ieee.org/>

Durante estos años también iba ganando popularidad el concepto del **Soft Computing** en contraposición con el **Hard Computing**<sup>8</sup>. El **Soft Computing** engloba las técnicas que buscan resolver problemas con información incompleta o con ruido. Debido a que el conjunto de técnicas definidas como constituyentes del **Soft Computing** son las mismas que se usan en la **inteligencia computacional** algunos autores consideran ambos términos equivalentes. Nosotros consideramos que el **Soft Computing** es un punto de vista de la computación a diferencia de la **inteligencia computacional**, la cual es un área de específica dentro de la **AI** que hace uso de métodos incluidos en el concepto **Soft Computing**.

Hoy en día la **inteligencia computacional** es un área con muchas aplicaciones prácticas en una variedad muy distinta de campos de la ciencia y con muchos temas de investigación por explorar. Por ello, esta tesis pretende la exploración de una parte concreta de este área en el tema del modelado de comportamiento humano en el problema de la conducción.

<sup>8</sup> **Hard Computing** y **Soft Computing** son la forma de diferenciar dos puntos de vista a la hora de resolver problemas computacionalmente. El **Hard Computing** basa sus técnicas en aquellas basadas en modelos analíticos definidos de forma precisa y que en ocasiones requieren mucho tiempo de cómputo. Están basados en lógica binaria, análisis numérico, algoritmos y respuestas exactas. El **Soft Computing** por otro lado es tolerante a la imprecisión y al ruido y tiene de a llegar a soluciones aproximadas de manera más rápida. Se basa en modelos aproximados, emergencia de algoritmos y modelos estocásticos.

## 2.2 *El rol del aprendizaje en la inteligencia computacional*

El cambio más notorio entre los dos puntos de vista de la **AI** tradicional y la de la **inteligencia computacional (CI, computational intelligence)** es el concepto de entrenamiento, es decir, pasar de “*desarrollar un programa para resolver un problema*” a “*entrenar un modelo para que aprenda la solución*”. Éste es el concepto de **aprendizaje**, y al proceso de ajuste del modelo a la solución buscada se le denomina **entrenamiento**.

Las técnicas de aprendizaje se clasifican dependiendo de la forma en la que entrena los modelos. Podemos identificar tres clases principales de técnicas de entrenamiento las cuales se describen a continuación:

- **Aprendizaje supervisado.** Suponiendo que disponemos de un modelo denotado por  $M_V(V, I) = O$  donde  $V$  es el conjunto de variables de determinan el comportamiento de  $M_V$  y donde  $I$  es un conjunto de valores (características) de entrada para las que el modelo obtiene un conjunto  $O$  de valores de salida. Entonces, la forma de entrenar al modelo (el *algoritmo de entrenamiento*), se encargará de, a partir de un conjunto de la forma  $D = (I_i, O_i) | \forall i \in \mathbb{N}$ , donde cada  $O_i$  es la salida esperada del modelo a la entrada  $I_i$ , modificar los valores de las variables del conjunto  $V$  para ajustar lo más posible  $O_i$  a  $O$  dado  $I_i$ .
- **Aprendizaje no supervisado.** Es el proceso por el cual un modelo aprende a partir de datos en bruto sus relaciones y extrae patrones, sin necesidad de saber qué son esos datos ni recibir supervisión (a diferencia del aprendizaje supervisado donde los datos incluyen

un valor de entrada y su salida correspondiente). En general, los algoritmos pertenecientes a esta categoría se dedican al problema del *clustering*, es decir, identificar grupos de elementos cercanos en el espacio basándose en la suposición de que el comportamiento de dos elementos es más parecido cuanto más cerca están el uno del otro. Algunos ejemplos de técnicas que basan su entrenamiento en un esquema no supervisado son los **SOM**, los **autoencoders** o las **deep belief network**.

- **Aprendizaje por refuerzo.** En este paradigma, el algoritmo ajusta el modelo de acuerdo a políticas de recompensa o penalización en función de lo bien o lo mal que el modelo está desempeñando la tarea de acuerdo a una métrica determinada.

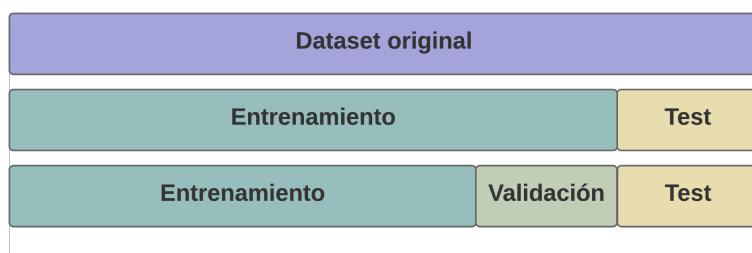
Algunos autores hacen uso de técnicas híbridas para suplir deficiencias u optimizar/acerlar el aprendizaje. Un claro ejemplo lo podemos ver en [Hinton, 2006], donde los autores hacen uso de **autoencoders** como técnica no supervisada para la inicialización de los pesos de una red neuronal, y posteriormente realizan un entrenamiento supervisado para su optimización.

Esta tesis se dedica al modelado de comportamiento entrenando a partir de datos reales de conducción, por lo que el discurso se centrará únicamente en el esquema de aprendizaje supervisado. En él, los algoritmos de entrenamiento dependen del modelo a usar (no es lo mismo un algoritmo de entrenamiento para una **Red Neuronal Recurrente (RNN, Recurrent Neural Network)** que para un **perceptrón multicapa**), y suelen ser usados principalmente para la solución a problemas de **clasificación** (i.e. determinar si un elemento dada sus características pertenece o no a un determinado conjunto) y de **regresión**, esto es, ajustar las salidas de un modelo para ajustarse lo máximo posible al valor real del sistema modelado.

### 2.2.1 Epochs, conjuntos de entrenamiento, test y validación

La terminología general que se usa en la **CI** no es demasiado compleja (con la salvedad de los nombres de técnicas y algoritmos). La Figura 2.4 muestra un esquema de esta terminología.

Figura 2.4: Los diferentes tipos de conjuntos de datos existentes. El conjunto de datos original está separado en dos conjuntos, entrenamiento y test. Del conjunto de entrenamiento se separa una porción para comprobar cómo evoluciona el entrenamiento y así determinar qué arquitectura y meta-parámetros son más útiles.



Un **dataset** es todo el conjunto de datos que disponemos para nuestro

experimento. Cuando el dataset es el resultado de una observación directa, también se le denomina *ground truth*. La calidad del conjunto de datos es esencial, ya que de éste depende la efectividad de las soluciones que lo han usado. Sin entrar demasiado en detalle, es recomendable que la distribución de datos del dataset sea aproximadamente la misma que la distribución del sistema en el mundo real<sup>9</sup>.

Para problemas que requieren soluciones del área del [aprendizaje automático](#), este conjunto se suele partir en varios subconjuntos<sup>10</sup>, cada uno para un cometido específico. Éstos comparten la característica de que, para ser lo más útiles posibles, han de mantener en la medida de lo posible la misma distribución que la del dataset del que provienen.

El primero de ellos es el **conjunto de entrenamiento** o *training set*. Éste es usado para entrenar los modelos y suele ser el mayor de los subconjuntos.

El conjunto de validación o **validation set** tiene como objetivo validar el modelo **durante** el entrenamiento ya que, dependiendo del modelo que se está explorando, lo más común es realizar una serie de modificaciones en metávariables para ajustarlo. Al usar un conjunto distinto que el de entrenamiento, ajustamos el modelo para unos datos que no ha visto durante el proceso de entrenamiento.

Sin embargo, al realizar estos ajustes sí estamos incurriendo en un sesgo hacia el conjunto de validación. Esto es, el modelo se está entrenando de acuerdo a los datos que se le presentan en el conjunto de entrenamiento, y las metávariables a lo que le dicta el conjunto de validación. En este punto surge el **conjunto de test** o *test set*<sup>11</sup>. Su objetivo es el de, precisamente, evaluar todo el proceso de entrenamiento **al final** de éste para comprobar que el modelo entrenado es lo suficientemente bueno para realizar la tarea para la que ha sido entrenado.

El último concepto es el de **epoch**. Se suele referir a una iteración sobre todo el conjunto de ejemplos del conjunto de entrenamiento. Sin embargo, en la actualidad estos conjuntos pueden llegar a ocupar demasiado por lo que, dependiendo del contexto, un *epoch* se puede referir a una iteración sobre una porción del conjunto total de entrenamiento.

En otros contextos la definición de *epoch* se mantiene y a cada una de las porciones se las denomina *batch* o *mini-batch*.

### 2.2.2 Problemas del aprendizaje en la [inteligencia computacional](#)

El entrenamiento de modelos en la [CI](#) adolece de dos principales problemas que, además, no tienen una solución general<sup>12</sup> ya que dependen tanto de la estructura de los datos sobre la que vamos a trabajar como de los hiperparámetros<sup>13</sup> del modelo. Estos son la

<sup>9</sup> Desgraciadamente, éste es un dato a priori desconocido en muchos problemas. Por eso la máxima de “cuantos más datos, mejor” suele funcionar ya que, cuantos más datos diferentes contiene el dataset, más se aproxima su función de distribución a la del sistema a modelar.

<sup>10</sup> Evidentemente disjuntos.

<sup>11</sup> En la literatura existe cierta confusión entre estos dos términos. Históricamente el conjunto de datos se solía dividir entre los conjuntos de entrenamiento y test. Posteriormente se vio la utilidad de separar el conjunto de test en dos diferentes para diferenciar la evaluación del modelo durante y después del proceso de entrenamiento.

Muchos autores prefieren la definición aquí expuesta, pero otros invierten los significados de validación y test indicando que el test se usa para el ajuste del modelo **durante** el proceso de entrenamiento mientras que la validación se realiza **al final** del proceso.

<sup>12</sup> Es el llamado *non free-lunch theorem* [Wolpert and Macready, 1997].

<sup>13</sup> En general, hablaremos de *parámetros* cuando nos referimos a valores que son inherentes al modelo (e.g. en una [ANN](#), los valores de los pesos) y de *hiperparámetros* cuando nos referimos a aquellos parámetros que modifican los elementos que modifican los parámetros (e.g. en una [ANN](#), el factor de aprendizaje).

<sup>14</sup> En la literatura también se habla de *high variance* o *over-fitting* y de *high bias* o *under-fitting* para referirse a la sobreespecialización y subespecialización respectivamente.

sobreespecialización y la subespecialización <sup>14</sup>.

El objetivo de un entrenamiento es conseguir modelos lo suficientemente buenos para que aprendan a generalizar sobre datos no conocidos, pero sin fallar demasiado. Cuando un modelo sufre de **sobreespecialización**, es porque aunque ha aprendido los ejemplos, falla a la hora de generalizar (podemos decir que ha aprendido los ejemplos *de memoria*). El caso contrario, la **subespecialización**, pasa cuando el modelo no es lo suficientemente complejo como para aprender suficientemente el problema y por lo tanto generaliza demasiado. Hablaremos de casos concretos y soluciones más adelante.

### 2.2.3 Deep learning

La tónica general en la ciencia es que cada pocos años algún término se escape del mundo de la investigación y se convierta en la palabra de moda que inunde la prensa especializada y no especializada. Palabras con fuerza como Big Data, Cloud Computing, Web Services o SaaS que lo más normal es que sean conceptos ya existentes como proceso de datos masivos, computación distribuida o ejecución remota.

Sí es cierto que en ocasiones estas palabras captan sutilidades o información que las hace más adecuadas que los antiguos conceptos y que incluso pueden llegar a abrir en el futuro nuevas ramas dentro del área al que pertenecen. Un ejemplo podrían ser los **Sistemas De Recomendación (RSs, Recommender Systems)**, un caso particular de *sistemas de filtrado de información* cuyo nombre estuvo de moda durante unos años y que en la actualidad conserva su entidad como una subrama de la rama principal.

El *deep learning* es una de las palabras con las que últimamente se inunda la prensa, pero lo cierto es que desde la aparición del término, parece que los avances dentro de éste no tienen límites. En esta tesis se considera al *deep learning* a la nueva generación de enfoques en la que confluyen varios factores que han hecho posible una mejora sustancial en el entrenamiento y la operación de modelos con técnicas que, por otro lado, ya existían previamente. Estos factores son los siguientes:

- **Disponibilidad de datos.** En la última década, la cantidad de datos que generamos como especie ha crecido en muchos órdenes de magnitud. El abaratamiento de los costes de producción de dispositivos y de sensores o la acumulación temporal de los datos históricos son sólo dos factores que nos permiten en la actualidad el acceso a una cantidad ingente de datos con la que trabajar, algo impensable en la década anterior.
- **Capacidad computacional.** Aunque obvio, es un factor también crucial. Una mayor capacidad computacional es directamente pro-

porcional a una mayor velocidad en el ciclo de experimentación de modelos (ver Figura 2.5). El verdadero impacto de esta década ha sido el del uso de las **GPU** de las tarjetas gráficas como plataforma donde distribuir el cómputo.

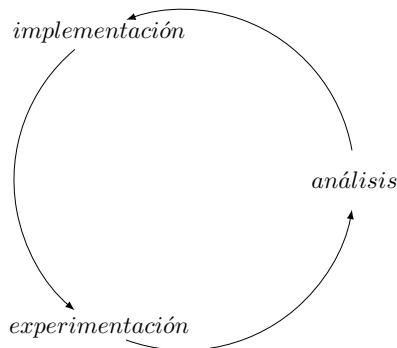


Figura 2.5: Ciclo de aplicación de soluciones basadas en **computational intelligence**. Los sistemas inteligentes se conciben, se implementan y se prueban. En la fase de experimentación se determina cómo de bien o de mal lo está haciendo y cuáles son las decisiones a tomar para el nuevo ciclo. Cuanto más rápido se puede realizar este ciclo, más soluciones se pueden probar, por ello el impacto de la mayor capacidad computacional y la optimización de las técnicas de entrenamiento es tan importante.

- **Algoritmos más eficientes.** Nuevos elementos como las funciones de activación **ReLU**, la representación de las redes como grafos computacionales para facilitar su distribución o innovaciones en los algoritmos de entrenamiento son algunas de las mejoras en este aspecto que redundan, como no, en la optimización de la capacidad computacional de las máquinas y por tanto sobre el ciclo de experimentación.

El adjetivo *deep*, en el contexto de las redes (e.g. **redes neuronales artificiales**, **deep belief network**, ...), donde se origina este término, se refiere a una red con más capas de lo habitual<sup>15</sup> (normalmente más de dos o tres). Este tipo de capas, históricamente han sido más difíciles de entrenar, debido entre otras cosas a:

- Las técnicas de entrenamiento, donde los parámetros tendían a disolverse (*vanishing gradient*) según se aumentaba el número de capas entre la entrada y la salida.
- La disponibilidad de conjuntos de datos, los cuales al no ser muy grandes, las redes grandes tendían a sobre-especializarse.

Los factores antes mencionados han hecho posible la operación sobre redes más grandes y profundas de órdenes de magnitud muy superiores. El resultado en la actualidad es que disponemos de modelos que mejoran en mucho a los modelos anteriores, y no parece haber cota superior en su capacidad de aprendizaje<sup>16</sup>. La Figura 2.6 introduce, con una ilustración un tanto informal, las capacidades del **deep learning** frente a las de los modelos entrenados antes de esta época.

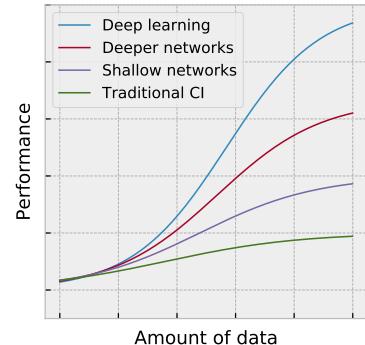
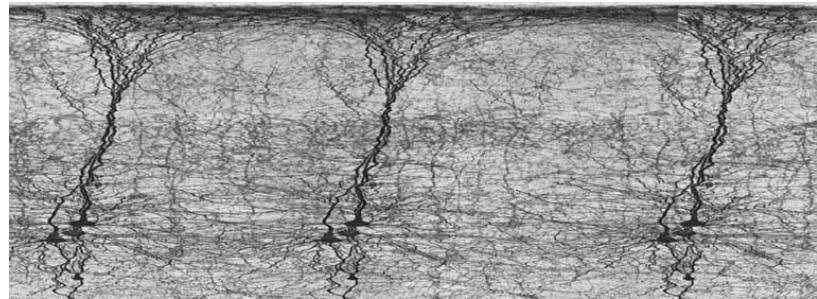


Figura 2.6: La enorme cantidad de datos junto con la capacidad computacional y la mejora de las técnicas de entrenamiento hacen posible que en la actualidad, con las técnicas asociadas al contexto del deep learning, los modelos entrenados sean más eficientes. Imagen adaptada de la charla *How scale is enabling deep learning* de Andrew Y. Ng, accesible <https://youtu.be/LcfLo7YP804>.

<sup>15</sup> De aquí surge el nombre de **shallow network** o red superficial, en contraposición a **deep network** o red profunda.

<sup>16</sup> Dentro de un contexto de aplicación específica, creemos que aún estamos muy lejos de crear vida inteligente.

Figura 2.7: Sección de neocórtex humano, región asociada a las capacidades cognitivas y que supone alrededor de un 76% del volumen total del cerebro humano. Está distribuido en 6 capas y miles de columnas que las atraviesan, cada una con alrededor de 10,000 neuronas y un diámetro de 0,5mm. Como dato anecdótico, se estima que sólo en el neocórtex humano existen alrededor de 20,000 millones de neuronas, cada una de las cuales conectada a entre 100 y 100,000 neuronas vecinas ([Pakkenberg and Gundersen, 1997]). Esto supone entre  $2 \cdot 10^{12}$  y  $2 \cdot 10^{15}$  conexiones. Fuente: Blue Brain Project EPFL, <http://bluebrain.epfl.ch/>.



## 2.3 Redes Neuronales Artificiales

Son herramientas que tratan de replicar las funciones cerebrales de un ser vivo de una manera muy fundamental, esto es, desde sus componentes más básicos, las neuronas. Para ello se basan en estudios de neurobiología y de ciencia cognitiva moderna del ser humano<sup>17</sup>.

Una **Red Neuronal Artificial (ANN, Artificial Neural Network)** es independiente del problema a solucionar. Se la puede considerar como una caja negra que aprende las relaciones que subyacen en los datos de un problema para abstraer el modelo a partir de éstos. Estas características de aprendizaje y abstracción son los factores determinantes por los que son usadas en prácticamente todas las áreas de la ciencia y de la ingeniería ([Du and Swamy, 2006]).

El primer trabajo en la disciplina se le atribuye a los investigadores McCulloch-Pitts por su modelo de neurona artificial ilustrado en la figura 2.8 ([McCulloch and Pitts, 1943]). Existen diferentes tipologías y formas de operar con redes, pero todas funcionan de la misma manera: unidades (e.g. neuronas) conectadas entre sí mediante enlaces por los que fluye la información de manera unidireccional, donde algunas de dichas unidades sirven de entrada al sistema (i.e. entradas o sensores), otras sirven de salida del sistema (i.e. salidas y actuadores) y otras como elementos internos (i.e. ocultas), y donde los pesos de sus conexiones se ajustan mediante un proceso denominado *entrenamiento*, imitando los principios de la teoría hebbiana [Hebb, 1949].

Este primer modelo de neurona proponía una función escalón para determinar si la neurona se activaba o no, como analogía del funcionamiento de la neurona biológica. Sin embargo, este modelo es muy limitado. La verdadera potencia de las redes surge tanto del uso de funciones de activación no lineales como de la formación de estructuras más complejas de neuronas.

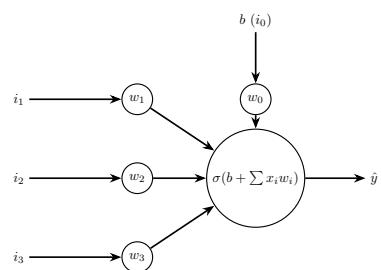
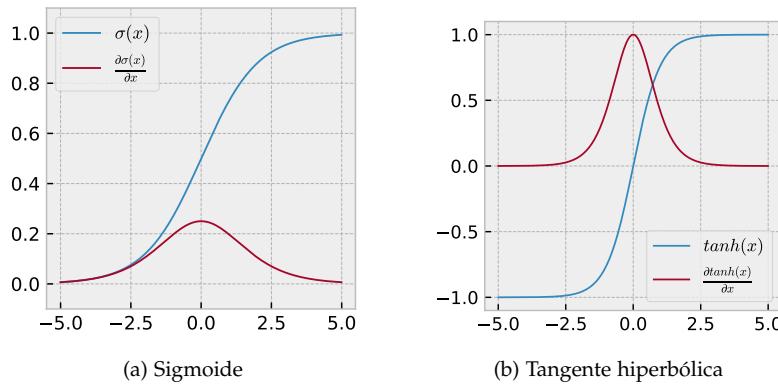


Figura 2.8: Variación de la representación del modelo de neurona artificial propuesto por McCulloch y Pitts. En éste, cada una de las entradas  $x_i$  es incrementada o inhibida aplicando el producto con su peso asociado  $w_i$ . La activación vendrá determinada por la aplicación de una función (denominada “de activación”) a la suma de los valores. Esta variación en concreto incluye una entrada  $x_0$  y un peso  $w_0$  como bias de la neurona para la variación dinámica del umbral de activación.

### 2.3.1 Funciones de activación

El valor de salida de una neurona queda determinado por la aplicación de una función sobre la entrada neta a ésta. Esta función dictamina el grado de activación de la neurona y para un correcto



funcionamiento de la red en términos generales debe ser no lineal<sup>18</sup>.

Las funciones de activación clásicas usadas en redes neuronales han sido la función sigmoidal (eq. 2.1) y la tangente hiperbólica (eq. 2.2). Por un lado, son funciones no lineales que mantienen normalizadas las activations de las neuronas, son derivables a lo largo de  $\mathbb{R}$ , siendo su derivada además fácilmente computable. En la Figura 2.9 se muestra una representación gráfica de éstas funciones junto con su derivada.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x)). \quad (2.1)$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad \frac{d\tanh(x)}{dx} = 1 - \tanh^2(x) \quad (2.2)$$

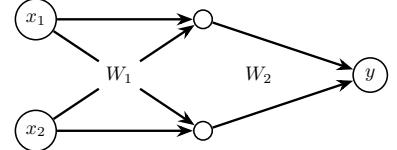
En general, la tangente hiperbólica es superior a la sigmoidal en todos sus aspectos. Computacionalmente es menos costoso el cálculo de una función sigmoidal, pero con la potencia de cómputo actual esta diferencia se puede considerar despreciable. Además de tener una forma similar a la sigmoidal, la tangente hiperbólica permite enviar señales de activación negativas. Además, tiende a centrar los datos de una capa a otra en lugar de mantener un sesgo hacia el 0,5 (recordemos que mientras que la sigmoidal está definida en el intervalo (0, 1), la tangente hiperbólica se encuentra definida entre los intervalos (-1, 1)). Por tanto, en un caso general, la tangente hiperbólica suele ser preferible a la sigmoidal.

Aún así, en algunas situaciones sí podría tener sentido el uso de funciones sigmoides en lugar de tangentes hiperbólicas. Por ejemplo, en un problema de clasificación, mantener en la capa de salida funciones de activación sigmoides permite ajustar la salida en el intervalo (0, 1), sin necesidad de realizar una posterior normalización.

Sin embargo, el principal problema de estas funciones es cuando los valores netos de las entradas son muy grandes. En ese caso, las funciones se acercan a los extremos, aproximándose sus gradientes a 0, y por tanto frenando el aprendizaje. Este error es denominado frecuentemente como *vanishing gradient* en la literatura. Es una de las

Figura 2.9: Las funciones de activación sigmoidal y tangente hiperbólica. Ambas se usan como funciones de activación no lineales gracias a que sus derivadas son continuas a lo largo de todo el dominio y además son fácilmente computables.

<sup>18</sup> Supongamos una red neuronal con una estructura como la siguiente:



Supongamos, además, la función de activación de las neuronas es lineal (e.g.  $f(x) = x$ ). La salida se puede expresar como:

$$\begin{aligned} \mathbf{y}^1 &= f(W_1 * \mathbf{x} + \mathbf{b}_1) \\ \mathbf{y}^2 &= f(W_2 * \mathbf{y}^1 + \mathbf{b}_2) \end{aligned}$$

Por tanto:

$$\begin{aligned} \mathbf{y} &= f(W_2 * f(W_1 * \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) \\ &= (W_2 * W_1) * \mathbf{x} + (W_2 * \mathbf{b}_1 + \mathbf{b}_2) \\ &= \mathbf{W}' * \mathbf{x} + \mathbf{b}' \end{aligned}$$

Siendo  $\mathbf{y}^l$  el vector columna de salida de la capa  $l$ ,  $\mathbf{b}_l$  el vector de los bias de la capa  $l$  y  $\mathbf{x}$  el vector columna de las entradas.

Es decir, usando funciones lineales da igual el número de capas que tengamos, ya que la composición de dos funciones lineales es siempre una función lineal y la arquitectura se reducirá a una única capa de activación lineal. Por ello no tiene demasiado sentido el uso de funciones lineales en las capas ocultas de una red.

razones por las que en la fase de inicialización de una red neuronal se tiende a valores en torno al 0 en los pesos. Unos valores altos hacen que las entradas netas sean muy grandes ralentizando el aprendizaje.

Uno de los avances dentro del Deep learning ha sido el uso de un tipo de función de activación denominada **Rectified Linear Unit (ReLU)** (eq. 2.3). Por un lado, evita el problema del *vanishing gradient* dado que su derivada es constante en el intervalo  $(0, \infty)$ . Por otro, su cálculo es tremadamente simple comparado con el resto de funciones (no deja de ser un máximo).

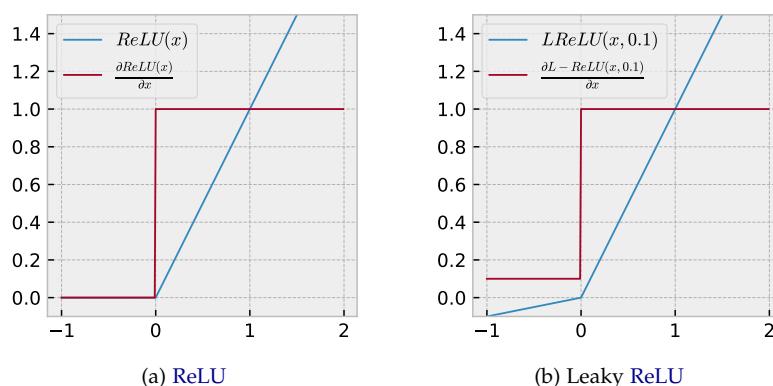
$$ReLU(x) = \max(0, x) \quad \frac{\partial ReLU(x)}{\partial(x)} \approx \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \quad (2.3)$$

$$LReLU_\epsilon(x) = \max(0, x) \quad \frac{\partial LReLU_\epsilon(x)}{\partial(x)} \approx \begin{cases} \epsilon & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \quad (2.4)$$

Este tipo de neurona tiene una serie de características que merece la pena comentar:

- No existe derivada en 0. El aprendizaje, basado en el descenso del gradiente, se encuentra con una indeterminación en 0. Sin embargo, es fácilmente subsanable incluyendo el valor 0 o 1 en la derivada en el punto 0. Aunque no es matemáticamente correcto, computacionalmente se está reemplazando la derivada por una función muy aproximada a ésta, en la que el algoritmo de descenso del gradiente se comporta de forma muy similar. Por ejemplo, la representación en la Figura 2.10, la derivada es 1 en el punto  $x = 0$ .
- Sin cota superior. Las activaciones muy fuertes representan relaciones entre elementos muy próximos entre sí, por lo que no tener una cota superior en el peso puede ser una ventaja a la hora de acelerar un entrenamiento, o una desventaja porque puede llegar a anular el impacto del resto de entradas. En general, esta desventaja aparece cuando el factor de aprendizaje es notoriamente alto.

Figura 2.10: La función de activación **Rectified Linear Unit (ReLU)** evita el problema del estancamiento cuando la entrada neta de la neurona es muy alta. La función de activación **Leaky ReLU** (en este ejemplo, con  $\epsilon = 0,1$ ) es una de las posibles soluciones cuando se permite que la entrada neta a la red sea menor que 0, ya que en el caso de la función ReLU la derivada es 0 y por tanto el gradiente deja de indicar hacia dónde ha de descender el error.



- Derivada 0 para  $x < 0$ . En el momento que el gradiente de una neurona se hace 0, ésta “muere”, por lo que la red estará compuesta sólo por activaciones positivas. Esta anulación puede ser una ventaja si el modelo se ajusta a las neuronas necesarias o una desventaja si se anulan demasiadas.

Por último, la función de activación *Leaky ReLU* (eq. 2.4) es una evolución de la *ReLU* para el uso en problemas donde es ventajoso que una neurona no llegue a tener nunca un gradiente de 0. Van acompañadas de un parámetro  $\epsilon \approx 0$  que determina la pendiente para todos aquellos valores menores de 0. Un ejemplo de esta neurona se ilustra en la figura 2.10. Su uso no está muy extendido, pero existen casos en los que está justificado.

### 2.3.2 Estructura y clasificación de redes neuronales

Las **redes neuronales artificiales** reciben ese nombre debido a que son sistemas formados por multitud de neuronas artificiales simples. Dependiendo de topología y configuración, éstas serán más adecuadas para unos u otros problemas (es decir, unas serán más adecuadas para regresión, otras para clasificación, otras para *clustering*, etcétera).

Típicamente, estas redes se componen de neuronas conectadas, por lo que se puede pensar en ellas como un grafo ponderado donde los nodos se corresponden a las neuronas, las aristas a las conexiones entre entradas y salidas y los pesos de las aristas a los pesos de las conexiones de entrada. Existen diferentes topologías o arquitecturas dependiendo de qué forma toma el grafo que modela las neuronas y sus conexiones. Estos dos tipos son los siguientes:

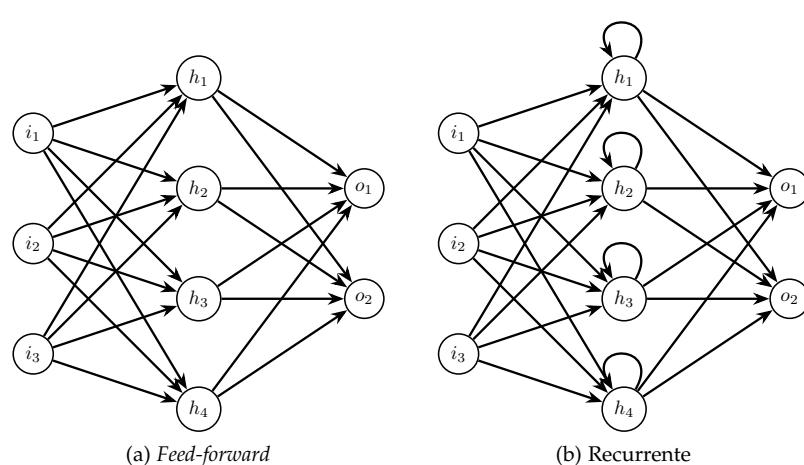


Figura 2.11: Diferencias entre los grafos que representan las ANNs de tipos (a) *feed-forward* y (b) *red neuronal recurrente*. Las **redes neuronales recurrentes** presentan ciclos entre sus nodos que permiten la retroalimentación interna entre las neuronas. Suelen ser más útiles a la hora de modelar eventos en el tiempo, aunque su entrenamiento es más complejo.

- Redes *feed-forward* (o prealimentadas). Su representación como grafo no presenta ningún ciclo (por tanto ninguna retroalimentación entre neuronas, como se ilustra en la figura 2.11). Es la topología más usada en aplicaciones prácticas debido a su sencillez y su efectividad. En éstas, el flujo de información sigue un camino

<sup>19</sup> Más adelante, en la sección [Perceptrones multicapa](#), explicamos esta nomenclatura.

desde un conjunto de neuronas denominadas *entradas* o *input* hasta otro conjunto de neuronas denominado *salidas* o *output*. No es requisito que las neuronas se agrupen en capas, aunque sí suele ser la disposición más común. A las redes de más de dos/tres capas ocultas <sup>19</sup> se las suele denominar *profundas* o *deep*. Representantes clásicos de esta categoría pueden ser el [perceptrón multicapa](#) [Rumelhart et al., 1985], los [autoencoders](#) [Hinton, 2006] o los [SOM](#) [Kohonen, 1998].

- **Red Neuronal Recurrente.** Tienen al menos un ciclo dentro de su representación, de manera que el flujo de información de salida de una neurona puede llegar a afectar a su propio estado. Aunque representan de manera más fiel las bases biológicas del cerebro, durante muchos años han sido más complejas a la hora de operar y sobre todo, entrenar, debido a estas relaciones. Sin embargo, durante la última década han aparecido nuevas técnicas relacionadas con el *deep learning* que facilitan su operación. Algunos casos particulares de este tipo de arquitectura son las Redes de Hopfield [Hopfield, 1982] o las redes [Long-Short Term Memory \(LSTM\)](#) [Hochreiter and Schmidhuber, 1997].

Esta tesis se centra en redes pertenecientes al primer grupo, concretamente en las arquitecturas [perceptrón multicapa](#) y [red convolucional](#), con las que se cerrará la presente sección. Ambos tipos de redes han probado su efectividad en diferentes dominios, aunque las segundas están demostrando su efectividad con las nuevas técnicas surgidas a partir del *deep learning*.

### 2.3.3 Perceptrones multicapa

En un [Perceptrón Multicapa \(MLP, Multilayer Perceptron\)](#), las neuronas se encuentran agrupadas en capas de tal manera que todas las salidas de las neuronas de una capa se conectan a todas las entradas de cada una de las neuronas de la capa siguiente. A la primera y última capa de la red se las denomina respectivamente capa de entrada y de salida, mientras que las capas intermedias son denominadas capas ocultas.

La salida se calcula como sigue: supongamos que tenemos dos capas,  $l - 1$  y  $l$ , compuestas por  $n^{l-1}$  y  $n^l$  neuronas respectivamente cada una con su función de activación  $f$  y, como conexiones (pesos) entre ambas capas, tendremos una matriz  $W^l$  de dimensión  $(l - 1, l)$ . Cada una de las neuronas deberá tener una entrada de *bias*, por lo que tendremos también un vector columna  $\mathbf{b}^l$  que los representará. La salida  $\mathbf{s}^l$  de esa capa será un vector columna que se obtendrá a partir de la ecuación 2.5:

$$\mathbf{s}^l = f(W^l \mathbf{s}^{l-1} + \mathbf{b}^l) \quad (2.5)$$

Como se puede observar, la salida  $s^l$  de la capa  $l$  depende directamente de la salida de la capa  $l - 1$ . El proceso de inferencia es introducir los valores en la primera capa, recoger los valores de salida, convertirlos en la nueva entrada de la capa siguiente y repetir hasta que se llega a la última capa, es decir, la de salida.

El algoritmo base de aprendizaje en un **perceptrón multicapa** se denomina *back-propagation* [Rumelhart et al., 1985] y se basa en la regla delta para el perceptrón simple [Widrow and Hoff, 1960]. Usa una técnica denominada *descenso del gradiente* donde lo que se intenta es minimizar el valor de una función  $f(x)$  que representa el error (en realidad el *coste*<sup>20</sup>) de la red calculando cómo aumenta o disminuye ésta con pequeñas variaciones de  $x$ .

El algoritmo trata de aplicar un error desde la salida de la red sobre todos los pesos de la misma en función de cuánto han colaborado en dicho error bajo la suposición de que, cuanto mayor es una entrada, más ha contribuido a éste. El problema es cómo se calcula este error. En la última capa es sencillo (conocemos las salidas real y esperada), pero la genialidad del algoritmo es que el error en las interiores se determina a partir de la retropropagación del error de las sucesivas capas, apoyándose en el descenso del gradiente.

Supongamos que nos encontramos en la capa  $l$  para la cual conocemos el error de salida que denotaremos como el vector columna  $\delta s^l$ , y donde todas las neuronas usan la función de activación  $f$ . Entonces, de acuerdo al gradiente, el error neto que produce nuestra salida se corresponde con:

$$\delta e = \delta s^l \circ f'(W^l \cdot s^{l-1} + b^l) \quad (2.8)$$

Nótese que  $\circ$  denota al producto de Hadamard (elemento a elemento). También es interesante fijarse en que  $W^l \cdot s^{l-1} + b^l$  se corresponde al cálculo de la entrada neta de la capa  $l$  antes de aplicarle la función de activación  $f$ .

Una vez conocido este error podemos pasar a calcular cómo varían los parámetros de la capa (eq. 2.9b y 2.9c) y el error de salida (eq. 2.9a) de la capa anterior de la siguiente manera:

$$\delta s^{l-1} = W^l \cdot \delta e \quad (2.9a)$$

$$\delta W^l = \delta e \cdot \delta s^{l-1} \quad (2.9b)$$

$$\delta b^l = \delta e \quad (2.9c)$$

El valor  $\delta s^{l-1}$  será el valor de entrada para la capa anterior, mientras que los valores  $\delta W^l$  y  $\delta b^l$  serán los gradientes calculados. La forma

<sup>20</sup> Existen dos términos asociados al error en **redes neuronales artificiales**: **error** y **coste**, que en la literatura se denominan *loss* y *cost* respectivamente. El primero se refiere al error existente entre la salida de la **red neuronal artificial** y la salida esperada de un ejemplo en concreto, mientras que el segundo se refiere al error sobre todo el conjunto de entrenamiento. Es de esperar que durante el proceso de entrenamiento este error baje.

Existen diferentes funciones para calcular el error y el coste de un modelo en concreto. En el caso del coste, lo más común es usar la media entre todo el conjunto de entrenamiento como:

$$C(M) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}_i, y_i) \quad (2.6)$$

Donde  $M$  es el modelo actual,  $m$  el número total de ejemplos y  $L$  la función de error entre las salidas  $\hat{y}$  e  $y$  de cada ejemplo  $i$ . En algunos casos se usa la media ponderada para dar más importancia a determinados casos, pero no es lo común.

Sin embargo, en el caso del error, existen varias funciones dependiendo de cuál sea la tarea. Por ejemplo, para tareas de regresión, lo común es usar el error cuadrático medio o su raíz cuadrada (**RMSE**). Para tareas de clasificación, una función de error muy útil que ha remplazado a la función *hit* (precisión, o relación aciertos/fracasos) es la entropía cruzada, que tiene la siguiente forma:

$$L(\hat{y}_i, y_i) = (1 - y) \log(1 - \hat{y}) - y \log \hat{y} \quad (2.7)$$

La razón es que en esta última se capturan sutilezas como lo cercano que ha estado un acierto. Por ejemplo, si nuestro objetivo es una salida de clasificación  $(1, 1, 0)$  y tenemos dos modelos, uno que dice  $(0, 9, 0, 9, 0, 6) \rightarrow (1, 1, 1)$  y otro que dice  $(0, 6, 0, 6, 0, 6) \rightarrow (1, 1, 1)$ , según la función *hit*, ambos modelos funcionan igual mientras que según la entropía cruzada el primero funciona mejor que el segundo.

Curiosamente, la entropía cruzada también funciona bien en problemas de regresión, aunque es más sencillo interpretar los resultados de un **RMSE** y por tanto su uso no está extendido.

más común de aplicar el error es la que se muestra en las ecuaciones 2.10a y 2.10b:

$$W^l = W^l + \alpha \delta W^l \quad (2.10a)$$

$$\mathbf{b}^l = \mathbf{b}^l + \alpha \delta \mathbf{b}^l \quad (2.10b)$$

Esta es la forma original del algoritmo de *back-propagation*. Al valor  $\alpha$  que aparece en las ecuaciones se le denomina factor de aprendizaje (en inglés *learning-rate*), y como se puede apreciar, su valor determina lo rápido que cambian los pesos. Suele tomar valores entre 0,1 y 0,01, pero dependiendo de la evolución del entrenamiento y de la variación del algoritmo, éste puede llegar a tomar valores mucho más bajos. Algunas de las variaciones sobre el algoritmo inicial son las siguientes:

- **Momento de inercia** [Qian, 1999]. Al algoritmo se le añade un factor por el cual los movimientos en el mismo sentido durante sucesivos epochs se acumulan. De esta manera, el riesgo de caer en mínimos locales disminuye al ser capaz de “saltar” pequeños baches.
- **Adagrad** [Duchi et al., 2011], **RMSProp** [Tieleman and Hinton, 2012] y **Adadelta** [Zeiler, 2012]. Los tres se basan en el mismo principio. Al factor de aprendizaje de la red se le añade la existencia de otro factor individual por cada peso de tal manera que se actualiza de forma diferente en función de la evolución de su gradiente individual.
- **Adam** [Kingma and Ba, 2014]. Este algoritmo combina los funcionamientos del momento junto con los del gradiente adaptativo del algoritmo **RMSProp**. Este algoritmo es el usado de base en los últimos años debido a su buen desempeño.

#### 2.3.4 Redes Convolucionales

Tal y como su nombre indica, las **Redes Convolucionales** (CNNs, *Convolutional Neural Networks*) son redes que usan convoluciones para su funcionamiento. Aunque también está estructurada en capas, su funcionamiento es diferente. Para comenzar, su estructura se divide en dos regiones bien diferenciadas, una que se dedica a la extracción de características de la entrada (la denominaremos *extracción de patrones*) y otra que se dedica a la clasificación o regresión de la entrada a partir de las características extraídas (que denominaremos *región de inferencia*). En la Figura 2.12 podemos ver una ilustración del esquema general de una **red convolucional** donde se identifican ambas regiones.

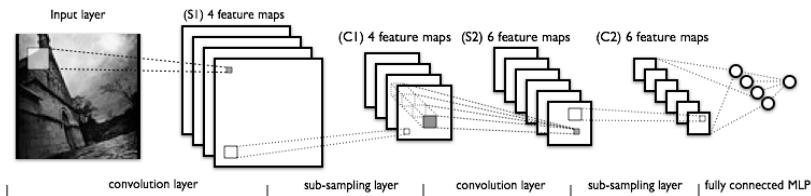


Figura 2.12: Estructura general de una **red convolucional**. El primer conjunto, el de extracción de patrones, se compone de operaciones de convolución y muestreo (*subsampling* o *pooling*). El segundo conjunto, el de inferencia, se compone de capas de neuronas conectadas una a una (la misma estructura que un **MLP**). Fuente: <http://deeplearning.net>.

La región de inferencia es, en esencia, un **perceptrón multicapa**. A ésta le llega un conjunto de entradas deducidas en la región anterior y realiza la operación de regresión o de clasificación que le corresponda. El funcionamiento se explica en el apartado anterior dedicado a este tipo de redes. La región de extracción de patrones es más interesante y merece más contenido.

**Convoluciones** Una convolución es una operación matemática que funciona como filtro sobre una estructura espacial (en este contexto, normalmente en forma de matriz o de cubo) para identificar patrones y/o para transformar estructuras identificadas. Para simplificar la explicación en este apartado, supondremos que la entrada se trata de una matriz bidimensional (e.g. una imagen de un sólo canal de color), y que los filtros son también bidimensionales, pero es común tener espacios de entrada de más dimensiones<sup>21</sup>. La ecuación 2.3.4 describe el resultado de aplicar una convolución de un filtro sobre una matriz.

$$\begin{pmatrix} 8 & 7 & 4 & 7 \\ 0 & 1 & 1 & 3 \\ 3 & 4 & 7 & 1 \\ 5 & 2 & 5 & 0 \end{pmatrix} \circledast \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 9 & 8 & 7 \\ 4 & 8 & 2 \\ 5 & 9 & 7 \end{pmatrix}$$

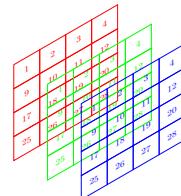
El símbolo  $\circledast$  denota la operación de convolución. Esta operación hace recorrer el filtro por todas las posiciones hasta que recubre todo el espacio inicial<sup>22</sup>. Para cada posición, se realizará el sumatorio del producto elemento a elemento, y el resultado se asignará en dicha posición. Esto implica que el tamaño de la matriz resultante es inversamente proporcional al tamaño del filtro. Concretamente, si  $(M_w, M_h)$ ,  $(F_w, F_h)$  y  $(R_w, R_h)$  son el ancho y el alto para las matrices origen, filtro y resultado, se cumple que:

$$R_w = M_w - F_w + 1 \quad (2.11)$$

$$R_h = M_h - F_h + 1 \quad (2.12)$$

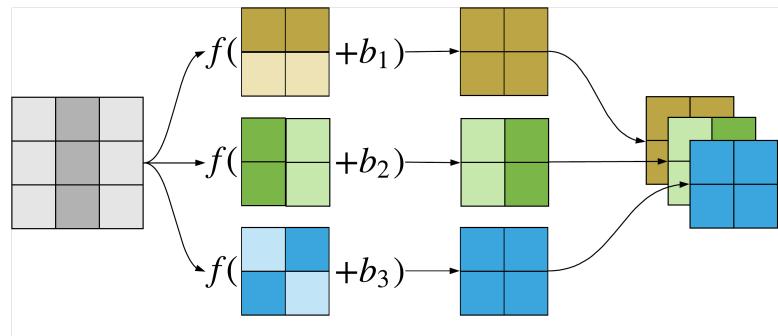
No obstante este funcionamiento de las convoluciones tiene un problema: las matrices tienden a ser cada vez más pequeñas con cada operación de convolución. En el contexto del *deep-learning*, el número de capas es finito y viene determinado por el tamaño del filtro.

<sup>21</sup> De hecho, para analizar imágenes lo normal es trabajar sobre los diferentes canales de color como capas diferentes, por lo que una imagen es en realidad una matriz tridimensional de dimensiones  $(w, h, c)$  donde  $w$  es el ancho,  $h$  es el alto y  $c$  es el número de canales.



<sup>22</sup> La operación básica recorre las posiciones una a una. Existe un parámetro, denominado *stride* que puede modificar este comportamiento. Nosotros nos ceñiremos a un *stride* de tamaño  $1 \times 1$  para simplificar el discurso.

Figura 2.13: Descripción de la capa de convolución. Como entrada se recibe una matriz tridimensional con una capa de profundidad. Los filtros han de tener la misma profundidad, mientras que la matriz resultado tiene como profundidad el número de filtros. A las convoluciones (más un *bias*) se les aplica una función no lineal como en los [perceptrón multicapa](#). Por lo tanto, los parámetros que el algoritmo de aprendizaje debe ajustar serán los valores del filtro junto con los *bias*.



<sup>23</sup> En general, el *padding* genera la cantidad de celdas con o alrededor de la imagen para que a la hora de aplicarlo el resultado sea igual. Es una solución válida, pero dependiendo del problema puede interesar usar diferentes formas de *padding*. Por ejemplo, en nuestro problema, el cálculo de los valores de *padding* horizontal es diferente. En la parte dedicada al desarrollo de la tesis se explicará el cálculo y el por qué.

La solución a este problema es la aplicación de una técnica denominada *padding* que aumenta la matriz de origen para que la matriz resultante tras la operación de convolución quede del mismo tamaño <sup>23</sup>. El cálculo del tamaño del *padding* para que la matriz resultado sea del mismo tamaño deberá ser:

$$P_w = \frac{F_w - 1}{2} \quad (2.13)$$

$$P_h = \frac{F_h - 1}{2} \quad (2.14)$$

Por tanto es necesario que los filtros sean de dimensión impar.

*Capas de convolución* Las capas más importante en una [red convolucional](#) son las capas de convolución. Éstas se componen de un número variable de filtros que representan las características que queremos extraer de la matriz de entrada. La entrada a esta capa de convolución será la matriz a la que aplicar los filtros, y la salida será una no linearización sobre el resultado de la convolución, modificándose en el proceso la dimensión de la matriz resultado. En la figura 2.13 se ilustra esto.

El proceso de aprendizaje en una [redes convolucionales](#), sin embargo, no es trivial. Sigue apoyándose en el cálculo de los gradientes en función de los parámetros, que en este caso son los valores del filtro junto con un *bias* por cada filtro. Las ecuaciones de este cálculo en una capa bidimensional (para cada uno de los filtros) son las siguientes:

$$\delta W_f = \sum_{w=0}^{F_h} \sum_{h=0}^{F_w} S^{l*} \times \delta C_{w,h} \quad (2.15a)$$

$$\delta b_f = \sum_{w=0}^{F_h} \sum_{h=0}^{F_w} \delta C_{w,h} \quad (2.15b)$$

En estas ecuaciones,  $\delta C_{w,h}$  se refiere al gradiente del coste respecto a la salida de la capa de convolución correspondiente al filtro  $f$  en la

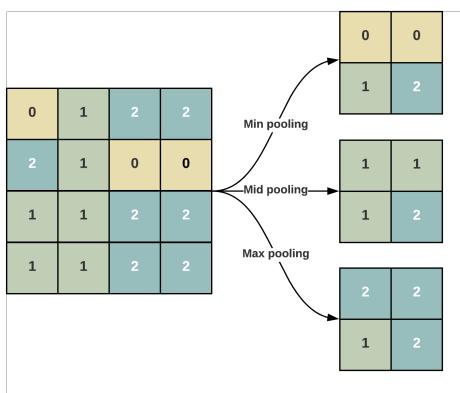


Figura 2.14: Ejemplo de diferentes operaciones de muestreo sobre una matriz de valores. El filtro de tamaño  $2 \times 2$  recorre la matriz en saltos (*strides*) de 2 celdas horizontales y 2 verticales extrayendo de cada región filtrada el menor valor, el valor medio o el mayor valor, dependiendo de si la operación es *min-pooling*, *mid-pooling* o *max-pooling*. La más común en CNN suele ser la operación *max-pooling*.

celda  $(w, h)$ . Así mismo,  $S^{l*}$  se corresponde con la región (slice) de la entrada que se usó para calcular el correspondiente  $\delta C_{w,h}$ .

Estas ecuaciones son más complejas en función de cómo varía el número de filtros y de dimensiones de los mismos, pero al final usa el mismo principio de propagación que un [perceptrón multicapa](#).

*Pooling y normalización* Junto con las capas de convolución, normalmente se usan otros dos tipos de capa diferentes: las capas de *pooling* y las de *normalización*.

El *pooling* es una operación de reducción del espacio de entrada. El objetivo es reducir las dimensiones de las entradas para las siguientes capas. Esto mejora el coste computacional y ayuda contra la sobre-especialización, ya que hay menos parámetros sobre los que trabajar para la misma entrada.

Su funcionamiento es similar al de las convoluciones (después de todo se tratan también de filtros), con la diferencia de que se mueven en ventanas en principio no solapables, generando un único valor para cada ventana y reduciendo por tanto el tamaño de la entrada<sup>24</sup>. Existen tres tipos fundamentales que son el máximo, el mínimo y la media, y su diferencia estriba en el cálculo del valor de salida a partir de los valores de la entrada (Figura 2.14).

La normalización es otra operación, ésta a nivel de capa, que reafirma las diferencias entre los valores existentes en la matriz, similar a un aumento de contraste en una imagen. Permite destacar diferencias y en general los resultados demuestran que acelera el proceso de aprendizaje de una [red convolucional](#) drásticamente cuando se incluye como capa oculta.

En los últimos años, los tipos más comunes de normalización son el [local response normalization \(LRN\)](#) [Robinson et al., 2007] y el [batch normalization](#) [Ioffe and Szegedy, 2015].

<sup>24</sup> Es curioso que por un lado se inventen mecanismos para mantener los tamaños de entradas-salidas entre convoluciones y luego no se use el comportamiento de las convoluciones con un *stride* del tamaño del filtro para realizar esa reducción. De hecho existen algunos estudios que afirman que el uso del *pooling* conlleva un aumento computacional sobre el uso de convoluciones para reducir el tamaño sin implicar una pérdida de rendimiento a la hora de predecir. Un ejemplo de esto lo tenemos en [Howard et al., 2017] donde afirman:

"We find that max-pooling can simply be replaced by a convolutional layer with increased stride without loss in accuracy on several image recognition benchmarks"

### 2.3.5 Solucionando los problemas de entrenamiento

Anteriormente hablábamos de los problemas a los que se enfrentan los procesos de entrenamiento en las técnicas de [inteligencia computacional](#), la **sub-especialización** o *underfitting* y la **sobre-especialización** u *overfitting*.

En líneas generales, los problemas de sub-especialización pueden darse por tres razones diferentes:

- La red no es lo suficientemente grande. Puede ocurrir que el conjunto de datos requiera de más propiedades o parámetros. La solución es incrementar el número de capas o de neuronas por capa a fin de que el modelo acabe aprendiendo al menos los datos del conjunto de entrenamiento.
- El modelo no ha sido entrenado lo suficiente. Este caso es más raro, pero puede ser que el modelo sea lo suficientemente complejo para requerir más ciclos de entrenamiento. También puede ser que algunos hiperparámetros como el factor de entrenamiento o las funciones de activación predisponen al modelo a aprender más lentamente.
- La topología del modelo no se adecúa a los datos. Puede ocurrir que el modelo que estamos tratando de aprender aprenda mucho mejor en una topología que en otras. Por ejemplo, para aprender a clasificar imágenes, las [redes convolucionales](#) funcionan mejor, entre otra serie de razones, porque mantienen una coherencia espacial entre los píxeles de la imagen desde el principio, algo que con un [perceptrón multicapa \(MLP, multilayer perceptron\)](#) no ocurre. Quizá representar los parámetros de entrada de una forma diferente o probar otra topología puede hacer que aprenda el problema de forma diferente.

El caso de la sobreespecialización es quizás algo más complejo. Cuando un modelo está sobreentrenado falla al generalizar, aunque los datos del conjunto de entrenamiento estén perfectamente aprendidos. Suele ser causado por dos razones principales:

- No hay suficientes datos. El modelo no generaliza, no porque no sea capaz, sino porque todavía le queda por aprender. La solución suele ser aumentar la cantidad de datos que existen en el conjunto de entrenamiento.
- La red está sobredimensionada. Suele ser el caso más común, y es que la red tiene tantos parámetros que al final ha aprendido a predecir uno a uno casi todos los ejemplos del conjunto de entrenamiento. Existen dos posibles soluciones no excluyentes, la simplificación de la red y la aplicación de técnicas de regularización.

En el caso concreto de la regularización, el objetivo de esta técnica es tratar de penalizar o limitar el entrenamiento a través de la inhibición de los parámetros. Existen diferentes técnicas para la regularización de parámetros, como las  $L_1$  y  $L_2$  (explicadas en detalle en [Ng, 2004]). En esta tesis se ha escogido una técnica de regularización denominada *dropout* [Srivastava et al., 2014].

En el *dropout*, la idea es que en cada *epoch* de entrenamiento del modelo se desactivan una serie de neuronas de manera aleatoria. Estas neuronas no participan ni en la predicción ni en el posterior reajuste de pesos. Es muy fácil de implementar, computacionalmente muy eficiente y mejora sustancialmente el proceso de aprendizaje en redes que sufren de una alta especialización. La Figura 2.15 describe un ejemplo de funcionamiento en tres epochs de un perceptrón multicapa con una tasa de dropout del 0,5.

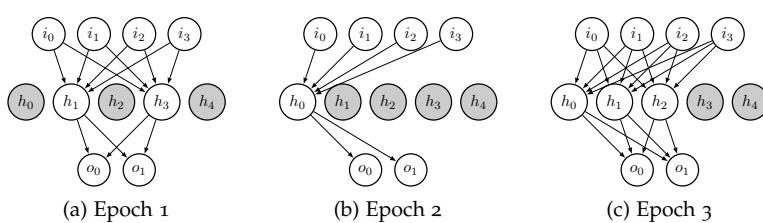


Figura 2.15: Ejemplo del funcionamiento del dropout en tres epochs sucesivos. Suponiendo una tasa de dropout de 0,5, en cada epoch cada neurona tiene una probabilidad del 50 % de ser desactivada, y por tanto el error no se propagará hacia sus pesos de entrada.

### 2.3.6 Grafos computacionales

En la actualidad el concepto de grafo computacional se relaciona directamente con las *redes neuronales artificiales*, y por ello se introduce el concepto en este apartado. Sin embargo, no se trata de un concepto exclusivo de esta técnica. De hecho, ni siquiera es un concepto perteneciente a la *inteligencia computacional*. Es simplemente una forma para representar las operaciones como un grafo, de tal manera que es fácil ver cuál es el flujo de los datos y las operaciones que se realizan sobre ellos.

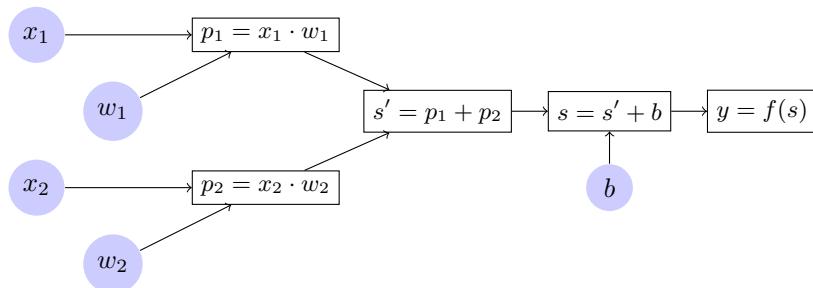
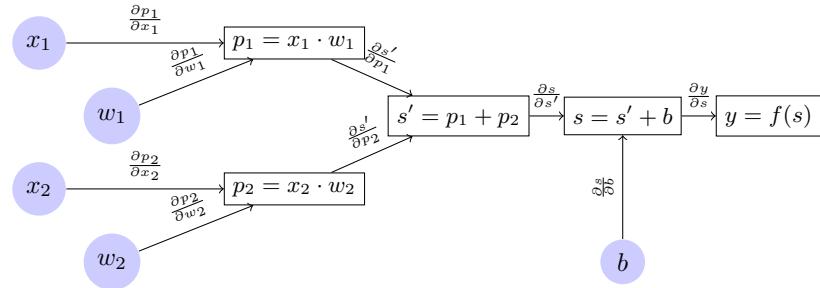


Figura 2.16: Ejemplo de grafo computacional para un perceptrón simple de dos neuronas de entrada y una de salida.

Formalmente, un **grafo computacional** es un grafo dirigido donde los vértices representan operaciones sobre datos mientras que las aristas representan el flujo de dichos datos. La figura 2.16 describe un posible grafo computacional para un perceptrón simple.

Una ventaja al representar un modelo como grafo computacional es que ayuda a abstraerse de la formas de las entradas y las salidas, facilitando el trabajo de operaciones en batch. Otra ventaja, todavía mayor, es que, al organizar de entrada a salida (en la figura 2.16 de izquierda a derecha) las operaciones que se necesitan para obtener una salida a partir de una entrada, en los casos donde el objetivo es optimizar la salida, permiten fácilmente representar el gradiente al organizarlo del modo contrario (es decir, de las salidas a la entrada o, en el caso de la figura 2.17 de derecha a izquierda).

Figura 2.17: Es fácil representar las derivadas parciales sobre un grafo computacional y, a partir de ahí, obtener el efecto de las variaciones de una variable sobre el resto.



Como regla general, para conocer el gradiente de una variable  $a$  con respecto a otra variable  $b$ , se aplicaría la *regla de la cadena multivariable*, que es equivalente a sumar todos los posibles caminos que van de  $a$  a  $b$  del grafo, multiplicando las derivadas parciales de cada arista.

## 2.4 Lógica Borrosa

<sup>25</sup> La lógica nace en el siglo IV a.C. dentro de la física Aristotélica, que permaneció inalterada hasta la revolución científica (alrededor del siglo XVI d.C.), momento en que se separó y permaneció como disciplina paralela perteneciente más al campo de la filosofía que de la física y la matemática. Empezó a relacionarse de nuevo con la matemática a principios del siglo XIX y a principios del siglo XX la lógica y la teoría de conjuntos pasaron a convertirse en partes indispensables la una de la otra. Por ello suelen ir de la mano cada vez que se habla de la una y de la otra. La evolución de la teoría de conjuntos y su unión con la lógica es una época bastante convulsa dentro de la historia de la matemática.

<sup>26</sup> Las lógicas multivaluadas son aquellas donde las premisas pueden tomar más de dos valores.

La lógica matemática <sup>25</sup> (y por extensión la teoría de conjuntos) tiene como misión servir de fundamento del razonamiento matemático. Se basa en la definición precisa y con rigor de un razonamiento, evitando cualquier tipo de ambigüedad y de contradicción. Es por ello que la lógica tradicional no suele servir como fundamento de razonamientos del mundo real.

Los conceptos que se manejan en el mundo real suelen ser vagos y estar llenos de imprecisiones. Además tienden a ser nombrados cualitativamente, no cuantitativamente, y cuando existe una correspondencia, ésta suele estar marcada por la subjetividad de los términos. La **lógica borrosa** se puede considerar como perteneciente al conjunto de lógicas multivaluadas <sup>26</sup>, donde se trabaja con este tipo de conceptos.

La idea tras este punto de vista es que raramente el ser humano piensa en términos absolutos o completamente definidos. En su lugar, su forma de razonar conlleva una serie de abstracciones que diluyen el carácter de las observaciones, las cuales a su vez pueden ser también imprecisas. Ya que en la lógica clásica es imposible expresar la complejidad que implica la incertidumbre de este proceso de razonamiento, en la **lógica borrosa** los conceptos de verdad o mentira

se relajan, existiendo una transición entre estados no abrupta, sino suave y progresiva.

A lo largo de la sección se describirá el concepto de conjunto borroso, cómo se usa como mecanismo de razonamiento y su utilidad dentro del área de sistemas de control.

#### 2.4.1 Ampliando la teoría de conjuntos tradicional

La teoría de conjuntos borrosos nació a mediados del siglo XX de la mano de Lofti A. Zadeh [Lofti A., 1965] como solución para la representación de procesos de inferencia conociendo a priori los grados de verdad de las premisas <sup>27</sup>. A diferencia de los conjuntos tradicionales (*crisp* en la terminología de la lógica borrosa), los conjuntos borrosos expresan el grado de pertenencia de un elemento a la categoría representada por el conjunto.

*Variables lingüísticas* El primer concepto importante a entender en la teoría de conjuntos borrosos es el de **variable lingüística**, las cuales sirven de base para toda la teoría porque sus posibles valores son, precisamente, conjuntos borrosos. Se define como:

“[...] variable whose values are words or sentences in a natural or artificial language [...]” [Zadeh, 1975]

El uso de variables lingüísticas permite representar fenómenos del mundo real como variables cualitativas. Por ejemplo, la variable *velocidad*, puede tomar los valores *atrás rápido*, *atrás lento*, *parado*, *adelante lento* y *adelante rápido*. Es cierto que la variable velocidad a su vez está definida sobre  $\mathbb{R}$ , y que los conjuntos tienen una definición sobre este dominio, pero la imprecisión que nos dan los términos (conjuntos borrosos) se hereda a lo largo de todo el proceso de deducción, de manera similar a como lo hacen los humanos.

*Conjuntos borrosos* Una posible definición de conjunto borroso <sup>28</sup> podría ser la siguiente:

Dada  $X$  una colección de elementos, se define al **conjunto borroso**  $F$  como un conjunto ordenado de pares de la forma:

$$F = \{(x, \mu_F(x)) | x \in X\} \quad (2.16)$$

Siendo  $\mu_F(x) \in [0, 1] \forall x \in X$ .

A la función  $\mu_F(x)$  se la denomina **función de pertenencia**, y caracteriza unívocamente a un conjunto borroso del dominio de  $X$  <sup>29</sup>.

La teoría de conjuntos clásica también define los conjuntos de acuerdo a una función, en este caso denominada *función característica*. Las ecuaciones 2.17a y 2.17b muestran las diferencias entre los valores de

<sup>27</sup> La lógica borrosa es uno de esos casos curiosos en la ciencia donde la aplicación nace antes que la propia teoría.

<sup>28</sup> Según Zadeh, la teoría de conjuntos borrosos debería servir de ejemplo de lo que él denominó *principio de extensión* [Zadeh, 1975], esto es, el proceso por el cual generalizar cualquier teoría definida en un dominio discreto hacia su versión continua.

<sup>29</sup> Un ejemplo clásico que pone de relieve la utilidad es el de una variación de la paradoja del montón (o paradoja sotiles, donde se aplica el método inductivo para demostrar que un montón de arena es y no es un montón de arena) por parte de [Klir and Yuan-Yu, 1997]. En el artículo se argumenta que si tenemos un montón de arena y vamos quitando los granos uno por uno, llegará un momento que no tendremos dicho montón. Está claro que un grano, dos, tres no son un montón de arena, pero un montón menos uno, dos o tres granos tampoco deja de serlo. Desde el punto de vista de la lógica clásica, es imposible determinar el límite de granos donde un montón de arena deja de serlo. Este ejemplo representa una de las muchas situaciones donde es inevitable la incertidumbre.

$f$  siendo ésta una función característica de la teoría de conjuntos clásica o una función de pertenencia de la teoría de conjuntos borrosos respectivamente.

$$f(x) = \begin{cases} 0 & \text{si } x \notin F \\ 1 & \text{si } x \in F \end{cases} \quad (2.17a)$$

$$f(x) = \mu_F(x) \quad (2.17b)$$

<sup>30</sup> Puede porque también se puede definir un conjunto *crisp* desde el punto de vista de la lógica borrosa.

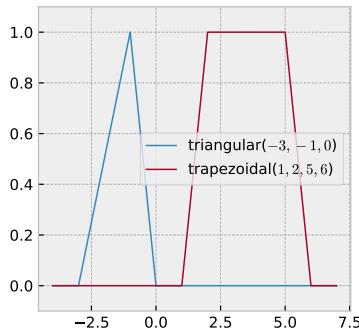


Figura 2.18: Las funciones de pertenencia triangular y trapezoidal son las dos funciones más usadas a la hora de definir conjuntos borrosos, tanto manualmente como en técnicas de ajuste. La razón es su sencillez, ya que captan la esencia de la imprecisión a la hora de definir un término sobre un dominio.

<sup>31</sup> Las mismas nociones de operaciones en teoría de conjuntos son aplicables a la teoría de conjuntos borrosos.

Mientras que un conjunto tradicional se basa en si un valor pertenece o no al conjunto, un elemento en un conjunto borroso puede <sup>30</sup> tomar todos los valores posibles en el intervalo [0, 1].

*Funciones de pertenencia* Esta función de pertenencia  $\mu_F$  puede tomar cualquier forma siempre que estén definidas en el dominio  $[0, 1] \subset \mathbb{R}$  (independientemente de si son discretas o no). Lo habitual es definirlas como funciones triangulares o trapezoidales.

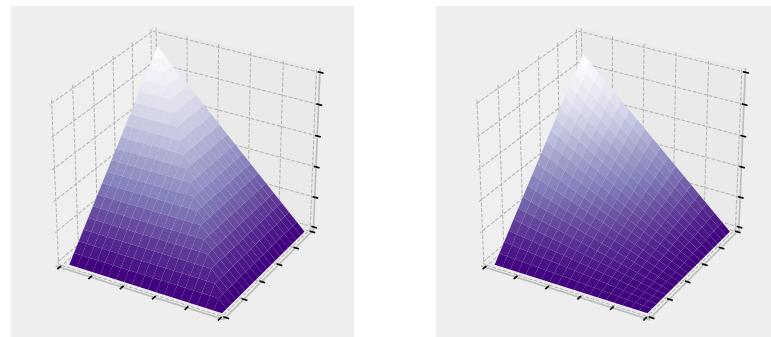
Las funciones de pertenencia caracterizan a los conjuntos borrosos. Su notación habitual es la que se presenta en la ecuación 2.16, donde  $F$  es el conjunto definido,  $\mu$  su función de pertenencia y  $x$  el valor real del dominio sobre el que se define la variable lingüística.

Dos casos particulares de funciones de pertenencia son las funciones cuadradas (equivalente a conjuntos *crisp*) y las funciones *singleton* (El valor de pertenencia lo tiene un único valor en todo el dominio).

*Operaciones entre conjuntos* Una operación entre conjuntos borrosos <sup>31</sup> es aquella que, a partir de uno o más conjuntos borrosos genera uno nuevo definido sobre el mismo universo de discurso. Son 1 *t*-norma, la *t*-conorma y el complemento.

La *t*-norma es la generalización del operador conjunción en lógica clásica. En la Figura 2.19 se describen las dos operaciones más usadas como *t*-norma en sistema de control borroso.

Figura 2.19: Las *t*-normas de mínimo y producto algebraico son dos representantes de un conjunto muy amplio de funciones



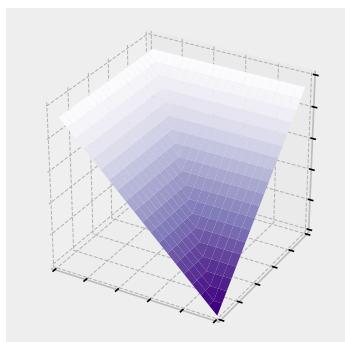
(a) Mínimo

(b) Producto Algebráico

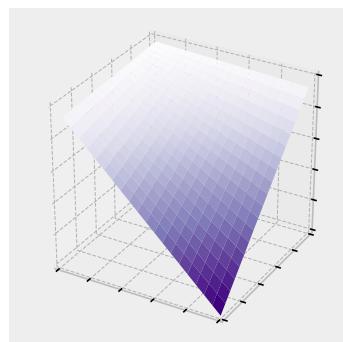
Son un conjunto de aplicaciones de la forma  $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$

tales que cumplen las propiedades *comutativa*, *asociativa*, *monótona* e *identidad*<sup>32</sup>.

La *t*-conorma (o *s*-norma), por otro lado, generaliza el concepto de unión. Una *t*-conorma  $S$  se define a partir de una *t*-norma  $T$  como  $S = 1 - T(1 - x, 1 - y)$ , es decir, la función complementaria a la *t*-norma. De esta forma se consigue que las Leyes de De Morgan se cumplan de una forma generalista. Dos ejemplos de *t*-conormas se ilustran en la Figura 2.20.



(a) Máximo



(b) Suma probabilística

Las propiedades que ha de cumplir una *t*-conorma son las mismas que las de la *t*-norma<sup>33</sup>.

El complemento de un conjunto vendrá calculado a partir de la función de pertenencia de dicho conjunto. Esto es similar a la teoría de conjuntos clásica donde el complemento de un conjunto contiene todos aquellos elementos que no estaban originariamente en dicho conjunto.

Formalmente, si  $F$  es un conjunto definido por la función de pertenencia  $\mu_F$ , la función de pertenencia que definirá al conjunto complementario  $F^C$  será  $\mu_{F^C} = \mu_F(x) \circ f$ , siendo  $f$  una aplicación de la forma  $f : [0, 1] \rightarrow [0, 1]$  tal que cumple las propiedades de **equivalencia** con teoría de conjuntos clásica, **estrictamente decreciente** e **involución**<sup>34</sup>.

El operador más común para el complemento es el probabilístico, denominado también *complemento de Zadeh* en [lógica borrosa](#). Existen sin embargo otros complementos, como el complemento de Yager o el complemento de Sugeno, ambos deducidos de una familia de funciones denominadas negaciones fuertes<sup>35</sup>.

#### 2.4.2 Razonamiento

El razonamiento o inferencia es el proceso de obtener consecuencias a partir de hechos (premisas). En [lógica borrosa](#), el proceso de inferencia lógica se amplía a través de relaciones borrosas, trasladando valores de verdad aproximados.

<sup>32</sup> Las propiedades son:

- Comutativa:  $T(x, y) = T(y, x)$ .
- Asociativa:  $T(x, T(y, z)) = T(T(x, y), z)$ .
- Monótona:  $x \leq z, y \leq t \rightarrow T(x, y) \leq T(z, t)$ .
- Identidad:  $\exists 1 | T(x, 1) = x$ .

Figura 2.20: Las *t*-conormas de máximo y suma probabilística son las complementarias a *t*-normas mostradas previamente, y como ocurre en estas, existen tantas *t*-conormas como *t*-normas definidas.

<sup>33</sup> De hecho, son deducibles a partir de la *t*-norma con una salvedad, la identidad, que se define como  $\exists 0 | S(x, 0) = x$ .

<sup>34</sup> Las propiedades son:

- Equivalencia con teoría de conjuntos clásica en el caso de usar conjuntos *crisp*, es decir,  $f(1) = 0 \wedge f(0) = 1$ .
- Estrictamente decreciente:  $\forall x, y \in [0, 1], x > y \implies f(x) < f(y)$ .
- Involución:  $\forall x \in [0, 1], f(f(x)) = x$ .

<sup>35</sup> El **complemento de Yager** es toda una familia de complementos que obedece a la fórmula  $f_\lambda(x) = (1 - x^\lambda)^{1/\lambda}$  para  $\lambda = 0$ . El **complemento de Sugeno** es otra familia cuya función es diferente,  $f_\lambda(x) = \frac{1-x}{1-\lambda x}$ . A partir de ambas se puede definir el complemento de Zadeh variando el valor de  $\lambda$ .

Toda regla borrosa está compuesta por un antecedente y un consecuente. Tanto el uno como el otro están compuestos de sentencias que asignan valores de variables borrosas (conjuntos borrosos) a éstas. La forma más común de representar las reglas borrosas es a través de reglas IF . . . THEN, donde las relaciones entre elementos (i.e. AND, OR y NOT) se corresponden con las operaciones de  $t$ -norma,  $t$ -conorma y complemento.

La implicación lógica  $A \rightarrow B$  es equivalente a  $\neg A \vee B$ , pero en **lógica borrosa** esta equivalencia arroja valores contra-intuitivos. Una definición ampliamente extendida es la del uso de una  $t$ -norma, como es el caso de la **Implicación de Mamdani**, pero existen muchos tipos diferentes<sup>36</sup>.

<sup>36</sup> En los trabajos [Kiszka et al., 1985a] y [Kiszka et al., 1985b] se estudian hasta 72 alternativas al operador  $A \rightarrow B \equiv \neg A \vee B$  de implicación en **lógica borrosa**.

*Modus Ponens Generalizado* Se trata de la extensión del método del *modus ponens* para obtener el valor de verdad de un consecuente a partir de un antecedente ambos borrosos. En el *modus ponens* el esquema de razonamiento es el siguiente (ecuación 3).

1.  $P(x) \rightarrow Q(x)$
2.  $P(x)$
3. Entonces  $Q(x)$

Sin embargo, debido a que en **fuzzy logics** los valores de las reglas de inferencia son conjuntos borrosos, la conclusión de un silogismo no será la conclusión de la regla, sino algo aproximado a ella, como se ilustra en la ecuación 3

1.  $P(x) \rightarrow Q(x)$
2.  $P'(x)$
3. Entonces  $Q'(x)$

La *regla composicional de inferencia* es el método para determinar qué grado le asignamos a un consecuente a partir de las premisas de los antecedentes en un proceso de *modus ponens generalizado*, o dicho de otro modo, cómo obtener la función de pertenencia del conjunto resultado a partir de las funciones de pertenencia de las premisas.

Cada regla de tipo IF  $x$  is  $A$  THEN  $y$  is  $B$  se puede representar como una función  $I$  tal que  $I(\mu_A(x), \mu_B(y))$  (*Regla Composicional de Zadeh*).

#### 2.4.3 Acrlongplspfcs

Los **Sistema De Control Borroso (FCS, Fuzzy Control System)** son el caso de éxito de la **lógica borrosa** que más resultados ha cosechado tanto a nivel académico como a nivel industrial. Se trata sistemas que utilizan el razonamiento borroso para inferir una respuesta a partir

de un conjunto de entradas<sup>37</sup>. Su uso se aplica tanto a sistemas de control manual como a sistemas donde los elementos a controlar son muy difíciles o incluso imposibles de diseñar (e.g. procesos de control con un alto número de variables relacionadas entre sí).

Se puede definir como un componente que asigna salidas de dominios no borrosos (*crisp*) a entradas de dominios también no borrosos (*crisp*) de forma no lineal haciendo uso de **lógica borrosa**. La arquitectura de estos sistemas está compuesta por cuatro bloques: *fuzzificador*, *bloque de reglas*, *razonador* (bloque de inferencia) y *defuzzificador*. En la Figura 2.21 se describe el esquema de un **sistema de control borroso** general.

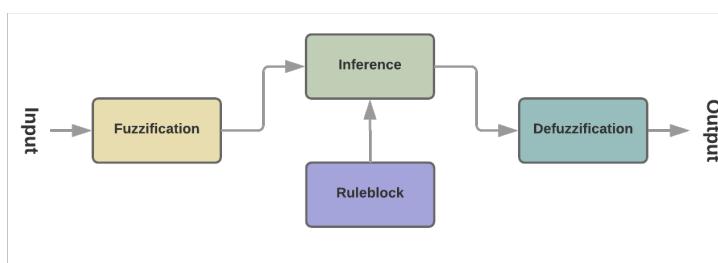


Figura 2.21: Un **sistema de control borroso** es un componente compuesto por cuatro bloques principales: *fuzzificador*, *bloque de reglas*, *razonador* y *defuzzificador*.

**Fuzzificador** Este bloque recibe las entradas *crisp* al sistema (los cuales pertenecerán al dominio sobre el que las variables están definidas) y las transforma a valores de pertenencia de sus respectivos conjuntos borrosos.

**Bloque de reglas** Define el conjunto de reglas que gobiernan el proceso deductivo del **sistema de control borroso**. A estas reglas se les puede aplicar pesos en caso de que se quiera simbolizar la mayor importancia de una regla frente al resto. Se suele representar como un parámetro  $w_i > 0 \in \mathbb{R}$  para la regla  $i$ -ésima, el cual se multiplica al resultado de la regla para ponderar la importancia de la misma.

La forma más común de representación es a través de reglas de la forma **IF ... THEN ...** por su facilidad de comprensión. Sin embargo, otra representación muy usada (sobre todo como representación interna) es la de  $m$  matrices  $n$ -dimensionales, donde  $m$  es el número de conjuntos de salida y  $n$  cada variable lingüística, con los conjuntos borrosos de éstas como índices y con los valores que toman esas intersecciones como los valores de dicha salida<sup>38</sup> (Figura 2.22).

$$\begin{matrix}
 & F_B^1 & F_B^2 & F_B^3 & F_B^4 & F_B^5 \\
 F_A^1 & \left[ \begin{matrix} 1 & 1 & 1 & 1 & 1 \end{matrix} \right] \\
 F_A^2 & \left[ \begin{matrix} 0 & 1 & 0 & 0 & 1 \end{matrix} \right] \\
 F_A^3 & \left[ \begin{matrix} 0 & 0 & 1 & 0 & 1 \end{matrix} \right] \\
 F_A^4 & \left[ \begin{matrix} 0 & 0 & 0 & 1 & 1 \end{matrix} \right]
 \end{matrix}$$

<sup>38</sup> Más adelante veremos un caso en el que esta representación es útil a la hora de inferir un **sistema de control borroso** a partir de un conjunto de entrenamiento.

Figura 2.22: Representación matricial de un conjunto de reglas borrosas para un conjunto borroso de salida  $O_k$ . Siendo  $F_A^i$  y  $F_B^j$  cada uno de los  $i$  y  $j$  conjuntos borrosos de las variables  $A$  y  $B$  respectivamente, los valores de la matriz representan si la conjunción entre ambos conjuntos borrosos existe o no.

*Razonador* Se encarga de realizar todo el procesamiento de inferencia haciendo uso de las entradas fuzzy (las salidas del bloque fuzzificador) y las reglas del bloque de reglas.

A partir de éstos realizará un proceso de deducción (normalmente *modus ponens generalizado*) tras el cual se obtendrán las funciones de pertenencia para cada uno de los conjuntos borroso resultado de las conclusiones del proceso de inferencia.

*Defuzzificador* Realiza el proceso inverso del bloque fuzzificador. Dado que a la salida del razonador sólo contamos con valores de pertenencia al conjunto borroso de salida, es necesario una transformación al dominio de la variable lingüística de salida para convertirlo en un valor interpretable por el sistema a controlar.

$$f(x) = \frac{\int_F x \cdot \mu_F(x) \cdot dx}{\mu_F(x)} \quad (2.18a)$$

$$f(x) = \frac{\sum_F x \cdot \mu_F(x) \cdot dx}{\sum_F \mu_F(x)} \quad (2.18b)$$

Existen diferentes algoritmos para la transformación de un conjunto borroso (expresado a partir de su función de pertenencia  $\mu_F$ ) a un valor *crisp*. Suelen ser comunes en conjuntos de salida continuos el **Centroid Of Area (COA)**, ecuación 2.18a y en conjuntos de tipo singleton su versión *media ponderada* (ecuación 2.18b)<sup>39</sup>.

<sup>39</sup> En el trabajo [Defuzzification: criteria and classification] se realiza una clasificación muy completa de una gran cantidad de algoritmos de defuzzificación.

#### 2.4.4 Tipos de sistemas de control borroso

Los **sistemas de control borroso** se han usado en muchos dominios diferentes. Son tantos los autores que han trabajado sobre ello que los sistemas tienden a variar en su funcionamiento. Existen dos tipos de controladores que son los que dominan la literatura, los cuales de describen brevemente a continuación.

*Controlador borroso tipo Mamdani* Este controlador fue el primer intento de control destinado automatizar un motor de vapor y su caldera utilizando para ello un conjunto de reglas borrosas del tipo *IF ... THEN ...* obtenidas directamente de la experiencia de los operarios. Su estructura es la básica presentada previamente (ver Figura 2.21).

Es conseciente de un sistema de tipo *Mamdani* es siempre un conjunto borroso y por tanto requiere de un proceso de defuzzificación que es costoso. De ahí el sentido el siguiente tipo de controlador.

*Controlador borroso tipo Takagi-Sugeno-Kang* Este tipo de controlador borroso fue propuesto por Takagi, Sugeno y Kang en los tra-

bajos [Takagi and Sugeno, 1993] y [Sugeno and Kang, 1988], aunque se suele abreviar a tipo Takagi-Sugeno o, directamente, Sugeno.

En este tipo de controlador, el bloque de defuzzificación desaparece y los consecuentes de las reglas de inferencia son reemplazados por una función (normalmente un polinomio basado en los parámetros del antecedente) la cual nos devuelve directamente un valor *crisp*.

En función del orden del polinomio de la función de salida, el controlador recibe también el mismo orden. De esta manera, por ejemplo, si tenemos dos valores de entrada  $x$  e  $y$ , cuando  $f(x, y) = c$  se dice que el controlador es de orden 0. Si  $f(x, y) = ax + by + c$ , se dice que es de orden 1, y así sucesivamente<sup>40</sup>.

El consecuente en un controlador de tipo Sugeno es directamente un valor del dominio de la variable de salida, y por tanto no necesita proceso de defuzzificación. Sin embargo, la respuesta pierde significado semántico, a diferencia de un controlador de tipo Mamdani.

<sup>40</sup> Los controladores de tipo Takagi-Sugeno-Kang de orden 0 son equivalentes a controladores de tipo Mamdani cuyos conjuntos borrosos de salida son de tipo singleton.

## 2.5 El paradigma de los agentes inteligentes

En la figura 2.3 se mostraban los cuatro objetivos perseguidos por la **inteligencia artificial**. En uno de ellos en particular se la entiende como el estudio para conseguir que entidades (e.g. sistemas, software, ...) actúen de la manera más inteligente posible.

Este concepto es una metáfora denominada **agente**<sup>41</sup> en el campo de la **inteligencia artificial**, pero que se ha extendido a lo largo de todo el área de la computación, especialmente en las áreas de los sistemas distribuidos o de las simulaciones basadas en sistemas multiagentes.

Es difícil encontrar un consenso en la definición de agente, y más todavía cuando entra en juego el adjetivo *inteligente*. Sin embargo, sí existen una serie de características que coinciden dentro de la literatura que exponemos a continuación:

- Operan siempre en un **entorno**, ya sea físico (e.g. una red de carreteras para un vehículo autónomo) o virtual (e.g. un cliente de correo electrónico para un clasificador de spam).
- Tienen la capacidad de **percibir** el entorno por medio de *sensores* y de **actuar** sobre él por medio de *actuadores*.
- Son **autónomos** en el sentido de que pueden actuar sin intervención externa (e.g. humana u otros agentes) teniendo control sobre su estado interno y su comportamiento. Algunos autores les presuponen una autonomía absoluta mientras que otros hablan de que sólo es necesaria cierta autonomía parcial.
- Tienen **objetivos** a cumplir, actuando para ello sobre el entorno de la manera que les indique su comportamiento.

<sup>41</sup> En [Russell et al., 2003] se define como "... just something that acts" alegando que la palabra *agent* proviene del latín *agere*. Para clarificar esto, *agere* es la forma verbal para *hacer*, pero improme un significado de movimiento/actividad diferente que no tiene mucho que ver con *hacer* como forma verbal para *crear* o *dar forma* (de lo que se ocupa el verbo *facere*). Por ello, el verbo *actuar* es un verbo que se relaciona con *agere* y de ahí la definición.

- Pueden ser **sociales**, es decir, tienen la capacidad de comunicarse con otras entidades (e.g. otros agentes) para llevar a cabo sus objetivos.

<sup>42</sup> El término preferido en esta tesis es precisamente este último, **agentes racionales**, dado que capture la esencia de lo que es un comportamiento inteligente. El problema de este punto de vista es que, como bromean en [Russell et al., 2003], hasta un elemento tan rudimentario como un termostato puede ser considerado un elemento inteligente, ya que realiza siempre la mejor acción para cumplir sus objetivos por muy simples que puedan parecer. Pero, ¿qué hace a un agente inteligente? Según algunos autores, el hecho de que posea unos objetivos y autonomía suficiente para cumplirlos ya denota inteligencia (e.g. en [Russell et al., 2003]). Según otros, es necesario que el comportamiento sea flexible, o lo que es lo mismo, que sea **reactivo** (reacciona ante el entorno que percibe), **proactivo** (iniciativa para tratar de cumplir sus objetivos) y **social** (capaz de interactuar con otros agentes para cumplir sus objetivos) (e.g. [Wooldridge et al., 1995]). Y otros directamente exigen, además, un comportamiento racional a la hora de cumplir los objetivos para calificarlo de inteligente (e.g. [Shoham, 1993]). Como dijimos anteriormente, dónde está el límite a partir del cual comenzar a considerar a un ente inteligente cae dentro de los dominios de la filosofía.

<sup>43</sup> Tanto es así que en algunos trabajos se define el objetivo de la **inteligencia artificial** como la implementación de la función agente, esto es, la función que realiza la correspondencia de una percepción a una acción, para un problema dado.

Nosotros hablaremos de estas entidades desde el punto de vista de la **inteligencia artificial**, y las denominaremos indistintamente *agentes*, *agentes inteligentes* o *agentes racionales*<sup>42</sup>. Por tanto usaremos la siguiente definición:

“Un agente es una entidad física o virtual que realiza una acción de manera total o parcialmente autónoma dada una secuencia de percepciones del entorno en el que se ubica.”

Según ésta, un agente es considerado **agente inteligente** cuando éste realiza la mejor acción posible (según un criterio de medida). En este contexto, *la mejor acción posible* se refiere en términos de objetivos a cumplir y comprensión del entorno en el que se desarrolla la acción, que puede ser o no correcta.

Las nociones de agentes inteligentes y la de **inteligencia computacional** van de la mano. Esto es debido a que su definición funciona a la perfección para las técnicas que la constituyen, esto es, agentes autónomos que perciben el entorno (problema) y actúan de la mejor manera posible sobre él (resuelven) de acuerdo a su conocimiento del medio y su estado interno (en base a algoritmos como **artificial neural network**, **fuzzy logic**, ...). Por ello desde mediados de la década de 1990 el concepto de agente inteligente ha ganado tanta popularidad<sup>43</sup>.

### 2.5.1 Tipos de entorno

La tupla (*entorno, agente*) es esencialmente una metáfora para referirse a la tupla (*problema, solución*) por lo que existen casi tantos entornos diferentes como problemas.

Afortunadamente es posible caracterizar los entornos de acuerdo a un conjunto de propiedades o dimensiones. Este conjunto es usado por la totalidad de la literatura a la hora de caracterizar entornos:

- **Observable.** Un entorno es **totalmente observable** cuando el agente es capaz de captar toda la información relevante para la toma de una decisión y no necesita mantener ningún modelo interno del entorno, **parcialmente observable** cuando la información obtenida es incompleta o tiene ruido y **no observable** cuando el agente no posee sensores.
- **Multiagente o monoagente.** Un entorno es **multiagente** cuando requiere de múltiples agentes interactuando para llegar a una solución mientras que es **monoagente** cuando sólo requiere de uno para ello.

- **Determinista o no determinista.** Si el estado del entorno actual depende totalmente del estado anterior, se dice que el entorno es **determinista**. Si no es así, se considera **no determinista** o **estocástico**<sup>44</sup>.
- **Episódico o secuencial.** Un entorno en el que las acciones se dividen atómicamente donde cada una de ellas conlleva un ciclo de (percepción, decisión, acción) y sin relación una con otra se denomina episódico. Si en lugar de ello la acción del agente puede afectar a las decisiones futuras se dice que el entorno es **no episódico o secuencial**.
- **Estático o dinámico.** Si durante la toma de decisión en entorno no cambia, se dice que el entorno es **estático**. En caso contrario, se dice que es **dinámico**.
- **Discreto o continuo.** Esta dimensión en realidad se divide en cuatro, estado del entorno, tiempo en el entorno, percepciones y acciones. La dimensión es **discreta** cuando ésta se divide en una partición discretizada, y **continua** cuando no. Por ejemplo, en el Juego de la Vida de Conway, si se modela en un sistema multiagente, tanto el estado (i.e. tablero) como el tiempo (i.e. turnos) como las percepciones y acciones están discretizadas. Sin embargo, en un entorno de conducción automática se puede determinar que las cuatro dimensiones son continuas.
- **Conocido o desconocido.** Un entorno es **conocido** cuando es posible determinar cuál va a ser el resultado de una acción. Si por el contrario no es posible, entonces se dice que es **desconocido**<sup>45</sup>.

### 2.5.2 Arquitecturas

Existe una serie de arquitecturas básicas o tipos de agentes que dependen principalmente de cómo perciben el entorno y de qué forma se comportan aunque, dependiendo de los autores, las nomenclaturas, tipologías y esquemas pueden variar. Por ello, hemos decidido ofrecer una abstracción donde poner de manifiesto las partes comunes y no comunes entre arquitecturas.

La Figura 2.23 muestra el esquema de las partes principales de un agente. En general, toda arquitectura de agente inteligente está cortada por el mismo patrón y obedece al siguiente funcionamiento:

1. El agente, a través de sus **sensores**, percibe el entorno que el que se encuentra inmerso.
2. Genera una **interpretación del entorno** tal y como supone el agente que es, es decir, percibe el entorno y, de acuerdo a sus sensaciones, lo entiende de una determinada forma.
3. Esta interpretación del entorno es pasada a un proceso de **inferencia** el cual, en función la implementación para la consecución

<sup>44</sup> En general, los entornos del mundo real tienden a ser tan complejos que es imposible para un agente abarcar todos los aspectos medibles de éste. Por lo tanto, sea o no la naturaleza del entorno determinista, en general se suele suponer éste como no determinista.

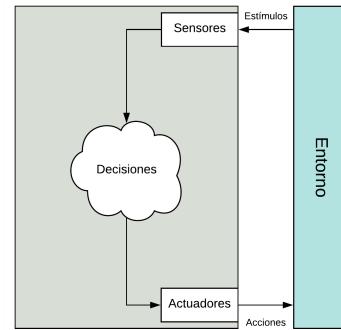


Figura 2.23: Aunque no existe una definición comúnmente aceptada de agente, sí que existe una serie de propiedades que los que los identifican. Es autónomo y realiza acciones sobre un entorno, el cual puede incluir otros agentes.

<sup>45</sup> Que el conocimiento del entorno no sea total es un factor clave que diferencia la racionalidad de la omnisciencia. La omnisciencia significa conocer el resultado de toda acción antes de realizarla y por tanto implica el conocimiento de absolutamente todos los detalles del entorno. La racionalidad existe dentro de un contexto de conocimiento limitado.

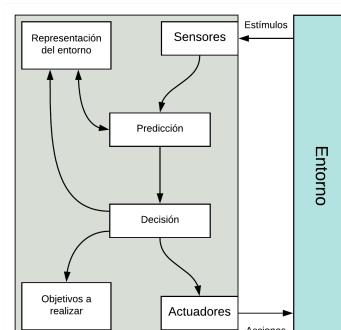


Figura 2.24: Los agentes basados en objetivos basan su comportamiento en un conjunto de objetivos que describen las situaciones deseables, lo que permite que el agente tenga una base de conocimiento que le ayude a elegir entre distintas opciones.

de sus objetivos, generará una serie de acciones a realizar sobre el entorno.

4. Estas acciones serán ejecutadas sobre el entorno a través de una serie de **actuadores**, provocando probablemente una modificación en éste que será percibida de nuevo en momentos sucesivos.

La primera diferencia clave surge en la manera que se ofrece al bloque de inferencia la interpretación del entorno y genera la primera clasificación:

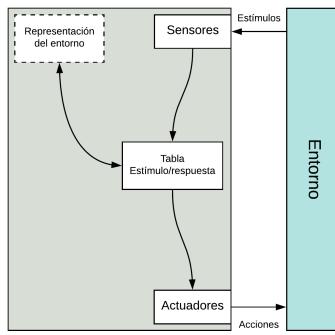


Figura 2.25: Esquema de agente reactivo. Los agentes reactivos se basan en un conjunto de reglas de tipo percepción → acción simples. En algunos casos pueden incluir una pequeña representación del entorno muy simple para dirigir la selección de las reglas.

- **Sin modelo de entorno.** Si el agente ofrece su interpretación del entorno directamente, sin hacer uso de información histórica sobre el entorno que se ha movido. Otras formas de denominar a estos agentes es *agentes reactivos* o *simple-reflex agents* ([Russell et al., 2003]), aunque algunos autores usan dichas expresiones para la forma de inducción de acciones a partir de percepciones. Además, los agentes reactivos sí pueden incorporar un modelo básico del entorno (Figura 2.25) y por ello preferimos la denominación *sin modelo de entorno*.
- **Con modelo de entorno.** El agente genera su interpretación del entorno a partir de las percepciones que llegan desde los sensores y del histórico del entorno que mantiene. Algunos autores los denominan *agentes con estado* o *Model-based*, pero lo hemos denominado de esta manera para diferenciar que el modelo que se mantiene en este punto pertenece únicamente al entorno.

La siguiente clasificación viene motivada por la forma de deducir el conjunto de acciones a ser aplicadas por parte de los sensores. En este sentido podemos identificar tres tipos distintos de agentes:

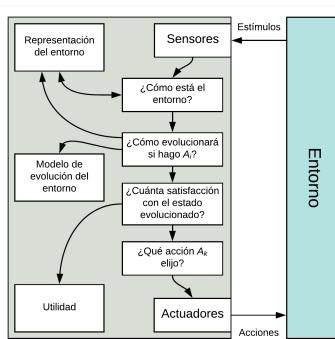


Figura 2.26: Los agentes basados en utilidad son similares a los basados en objetivo, pero añadiéndoles una medida de cómo de buenas son las acciones que podemos ejecutar, eligiendo la que mayor utilidad le ofrece (a menudo se habla de esta utilidad como *felicidad*).

- **Reactivos** (Figura 2.25). Son aquellos donde el uso de un proceso de razonamiento explícito es demasiado costoso para producir una conducta en un tiempo aceptable. Se suelen implementar como correspondencias (percepción → acción) sin ningún razonamiento adicional.
- **Basados en objetivos** (Figura 2.24). Plantean una deducción de forma que determinan cuál sería el estado del entorno tras aplicar varias o todas las acciones que puede realizar. En base a los resultados, selecciona la acción que se corresponde con sus propios objetivos.
- **Basados en utilidad** (Figura 2.26). Plantean una deducción similar a los basados en objetivos con la diferencia de que, mientras los primeros sólo diferencian entre entorno objetivo o no objetivo, éstos asignan un valor (i.e. *utilidad*) a cada uno de los escenarios de entorno posibles para seleccionar el mejor (e.g. el que mayor utilidad tiene).

En la literatura se describen muchos tipos de agente, como por ejemplo los agentes **Belief-Desire-Intention (BDI)**<sup>46</sup> o los agentes lógicos<sup>47</sup>. Sin embargo, éstos pueden definirse en los términos aquí expuestos.

### 2.5.3 Sistemas Multiagente

Son aquellos sistemas compuestos de dos o más agentes que interactúan de alguna manera para llegar a una solución.

Cuando los agentes son inteligentes y el problema cae dentro del dominio de la **inteligencia artificial**, el ámbito de estudio es el de la **Inteligencia Artificial Distribuida**, la rama dedicada a la resolución de problemas mediante procesamiento descentralizado.

Desde el punto de vista de la ingeniería de sistemas, y a pesar del aumento de complejidad, los **Sistemas Multiagente (MASs, Multiagent Systems)**, al ser sistemas inherentemente descentralizados, ofrecen múltiples ventajas frente a los sistemas centralizados tradicionales:

- Los sistemas son más robustos y fiables frente a fallos, ya que los agentes son autónomos e independientes del resto.
- La modificación del sistema se puede realizar sobre la marcha, agente a agente sin necesidad de parar el sistema al completo.
- Su diseño fuerza a desacoplar las dependencias entre agentes.
- Son inherentemente paralelizables y por tanto pueden llegar a ser más eficientes que sus homólogos centralizados. Este punto es quizás el más controvertido, ya que esta ganancia en eficiencia se puede perder rápidamente en función de la cantidad de comunicación existente entre agentes.
- Debido al nivel de complejidad alcanzado en los sistemas existentes en la actualidad, la computación se distribuye a través de múltiples sistemas, normalmente heterogéneos. La tendencia además es a la alza. La definición de los **sistemas multiagente** hace natural su implementación en este tipo de arquitecturas.

Desde el punto de vista de la **inteligencia artificial** podemos añadirles la ventaja de que permiten el estudio de conductas complejas de poblaciones a partir del comportamiento de sus elementos básicos, facilitando el estudio de modelos y de teorías sobre éstos.

LA COMUNICACIÓN ENTRE AGENTES se trata de una característica clave en un **sistema multiagente**, ya que para denominarse de esta manera dos o más agentes deben interactuar/comunicarse entre sí. Esta interacción puede implementarse de diversas maneras<sup>48</sup> y siempre toman una o las dos formas siguientes:

<sup>46</sup> Deciden las acciones (*intentions*) que creen que ayudarán (*beliefs*) a la consecución de sus objetivos (*desires*) a partir de las entradas y su conocimiento del entorno

<sup>47</sup> El entorno se representa con reglas lógicas y se infiere mediante métodos como por ejemplo deducción lógica o prueba de teoremas

<sup>48</sup> Las formas clásicas de comunicación son el paso de mensajes, los sistemas de pizarra y la estigmergia. Para los dos primeros existen dos propuestas para estándar de lenguaje de comunicación, **Knowledge Query and Manipulation Language (KQML)** ([Finin et al., 1994]) y **Agent Communications Language (ACL)** ([Poslad, 2007]). La tercera forma de comunicación suele ser muy dependiente del problema y no se apoya en lenguajes estándares. Se trata de una forma de comunicación basada en la modificación del entorno, como la efectuada por las hormigas en la búsqueda de alimento, donde éstas dejan rastros de feromonas modificando el entorno para modificar el comportamiento del resto de la colonia.

- **Cooperación.** Los agentes intercambian información entre sí para llegar a una solución. Esta solución puede ser fragmentada (i.e. cada agente posee parte de la solución y se comunican para ir avanzando de forma común hacia la solución global) o centralizada uno o varios agentes que usan al resto como componentes para llegar ir avanzando hacia la solución.
- **Competición.** Los agentes compiten dentro de un entorno, generalmente mediante la adquisición de recursos limitados. Un ejemplo de este tipo de sistemas multiagente puede ser aquellos sistemas de vida artificial.

## *3 Simulación de tráfico*

El tráfico es un sistema de comportamiento caótico que hace muy difícil la tarea de extraer de modelos de funcionamiento. Por un lado, la cantidad de variables existentes es muy numerosa y en muchos casos con relaciones no detectables a primera vista. Por otro lado, es un sistema que funciona en el mundo real, es decir, donde las mediciones en unos casos afectan a los resultados y en otros, directamente no se pueden realizar, ya sea por regulaciones vigentes o por imposibilidad física.

Los simuladores de tráfico son herramientas de software que, usando diferentes modelos para representar sus componentes, describen el tráfico como sistema, permitiendo, entre otros:

- Extracción de resultados y conclusiones de escenarios de tráfico determinados.
- Implementación de técnicas determinadas en tráfico simulado para su evaluación sin necesidad de alterar el tráfico real.
- Introducción de modificaciones en puntos determinados (e.g. espaciales o temporales) de un escenario conocido para estudiar la divergencia en la evolución del tráfico.

El objetivo principal de un simulador de tráfico es el de hacer que sus modelos se parezcan lo máximo posible a la realidad. En este capítulo vamos a ver cuál es la realidad actual de este tipo de simuladores, cuáles son sus diferentes tipologías y formas de modelar los diferentes aspectos del tráfico y, posteriormente, qué simulador de los disponibles en el mercado es el idóneo para nuestro trabajo.

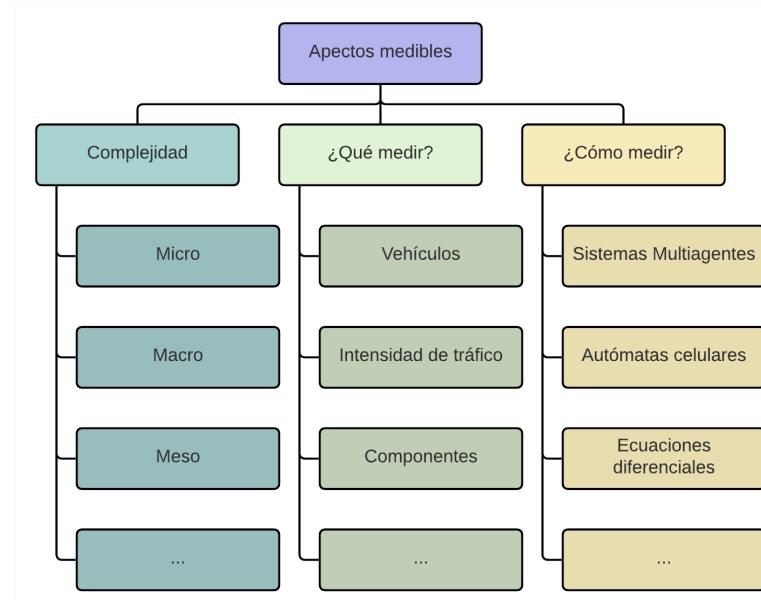
Limitaremos nuestro estudio a los simuladores de **Driver-Vehicle Units (DVUs)**, obviando otros tipos de simulación de tráfico que nada tienen que ver con esta temática, como por ejemplo los orientados a la evaluación de sistemas de señalización inteligentes (e.g. [Jin et al., 2016]), a la estimación de emisiones (e.g. [Quaassdorff et al., 2016]) o los de carreras (e.g. [Wymann et al., 2013]).

A lo largo del capítulo, se utilizarán indistintamente los términos **Driver-Vehicle Unit (DVU)**, conductor y vehículo para referirse al mismo concepto. En caso de no ser así, se indicará de manera explícita.

### 3.1 Clasificación de simuladores de tráfico

Los aspectos simulables y medibles del problema del tráfico son muy diversos (Figura 3.1), dependiendo sobre todo:

Figura 3.1: Los aspectos medibles del problema del tráfico son muy diversos, y dependen del nivel de granularidad (complejidad) al que se quiere llegar, de qué queremos medir y de cómo lo queremos hacer.



- Del nivel de **complejidad** del tráfico (e.g. modelar una vía por la que circula un centenar de coches no es lo mismo que modelar una ciudad por la que circulan cientos de miles).
- De **qué** queremos medir (e.g. evaluar a un conductor en una situación determinada o evaluar la evolución del flujo de tráfico en un cuello de botella causado por un accidente).
- De **cómo** (e.g. un **autómata celular** se modela de forma diferente a un modelo lineal de vías o carriles).

El resto de la sección ofrece una visión de las principales categorías existentes para clasificar a los simuladores de tráfico.

#### 3.1.1 Tipos de simulador en función de la complejidad

La complejidad en una simulación se refiere al nivel de detalle que queremos alcanzar durante la ejecución de la misma y/o en sus resultados. Es evidente que según aumentamos el detalle en la simulación aumenta la cantidad de cálculo. Por ejemplo, si queremos modelar el comportamiento de 10 billones de canicas cayendo por un tubo es considerablemente más eficiente modelarlas como un fluido con

una serie de parámetros que como una colección de elementos individuales, cada uno con sus propiedades (e.g. masa, aceleración, ...) e interaccionando entre sí.

El caso de los simuladores de tráfico es similar. En éstos existe un amplio intervalo de granularidades, desde por ejemplo el flujo de entrada en una autovía hasta el consumo de carburante de un vehículo en ciudad. Lo más común es clasificar los simuladores dentro de dos grandes grupos (Figura 3.2):

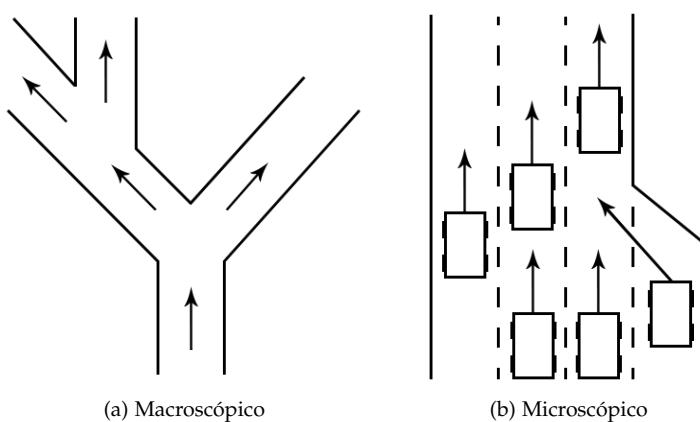


Figura 3.2: Clasificación clásica de simuladores en función de la granularidad (complejidad) de la simulación. En la imagen de la izquierda se muestra un ejemplo clásico de macrosimulador donde el tráfico se modela como un flujo a través de las vías. En la de la derecha, se ilustra un modelo clásico de microsimulación donde cada elemento (en este caso vehículos) circula por un carril de la vía.

- **Microsimulación** o simulación de tipo **micro**. Su objetivo es estudiar, desde un punto de vista de granularidad fina (e.g. vehículos o peatones), las micropropiedades del flujo de tráfico como, por ejemplo, los cambios de carril, las aproximaciones a vehículos de lanteros o los adelantamientos, para evaluar su comportamiento. Sus dos principales ventajas son la posibilidad de estudiar el tráfico como un todo a partir de sus elementos más simples (ofreciendo una representación más fiel de éste) y la posibilidad de estudiar cada elemento por separado. Sin embargo, su principal desventaja es que cada elemento de la simulación requiere de cómputo independiente y por tanto simulaciones con alto contenido de elementos pueden llegar a ser inviables<sup>1</sup>.
- **Macrosimulación** o simulación de tipo **macro**. Este tipo de modelos centran su esfuerzo en estudiar el flujo de tráfico como un todo (generalmente como fluido), explorando sus macropropiedades (e.g. evolución del tráfico, efectos onda, velocidad media o flujo en vías). Su ventaja principal es que a nivel macroscópico permiten estudiar propiedades que a nivel microscópico requerirían una cantidad ingente de recursos. Sin embargo, con este modelo es imposible obtener información precisa de un elemento en particular del tráfico.

Aunque ésta es la categorización típica de modelos, en la literatura aparecen otros tipos de modelo con granularidades que pueden considerarse no pertenecientes a ninguno de estos dos conjuntos. Éste es el caso de los simuladores de tipo **submicro** y de tipo **meso**.

<sup>1</sup> Existen técnicas de computación distribuida que superan ampliamente los límites impuestos por la computación en un único nodo. Un ejemplo relativamente reciente es el simulador de IBM Megaffic. Éste implementa un modelo de granularidad micro donde cada elemento es un agente independiente (i.e. **sistemas multiagente**) usando para ello entornos con cientos de núcleos de proceso que proveen de capacidad suficiente para modelar ciudades enteras como Tokio (ver [Osogami et al., 2012] y [Suzumura and Kanezashi, 2012]).

Los **submicro-modelos** (también denominados como *nano-modelos* en algunos trabajos) especifican granularidades por debajo del nivel de “vehículo” o “peatón”. Por ejemplo, en ([[Minderhoud, 1999](#)]) trabaja a nivel de funcionamiento del control de crucero inteligente de un vehículo en función del entorno del vehículo.

Por otro lado los **meso-simuladores** nacen para amortiguar los problemas inherentes a la complejidad en los micro-modelos y a la falta de resolución en los macro-modelos. Este tipo de simulador mantiene la representación de vehículo como unidad básica de tráfico, pero agrega dinámicas de tráfico que, de otro modo, aumentarían el consumo de recursos sin aportar información relevante sobre la tarea en la que trabajan.<sup>b</sup> Los trabajos en la línea de meso-modelos no se basan en una simulación completa a nivel meso, sino que trabajan desde una perspectiva híbrida, como por ejemplo el de [[Munoz et al., 2001](#)] donde utilizan una aproximación meso para reducir el coste computacional de simulaciones micro a una escala muy grande, o el de [[Casas et al., 2011](#)], un estudio similar, pero específico para [Aimsun](#).

Dado que el objetivo de la tesis es la evaluación de modelos de comportamiento de conductores concretos, nos ceñiremos al uso de simuladores que modelen un nivel de granularidad **micro**.

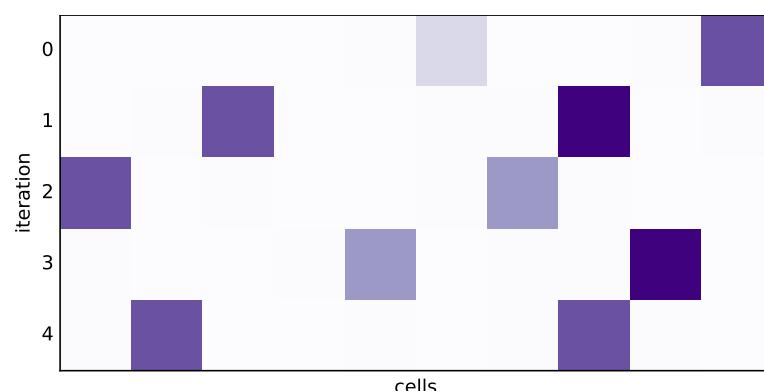
### 3.1.2 Tipos de simulador en función del espacio y el tiempo

Existen otras dos formas de clasificar los simuladores en función de cómo evolucionan en la simulación las dimensiones **espacio** y **tiempo**. Sin embargo, aunque *complejidad, espacio* y *tiempo* son dimensiones diferentes a la hora de clasificar, el tipo de simulador según una de ellas tiende a determinar en gran medida los tipos en las demás.

En el caso de la dimensión **espacio**, la clasificación diferencia las simulaciones que se mueven por un espacio discreto o por uno continuo:

- **Espacio discreto.** Simulación donde el espacio está dividido en celdas que (normalmente) sólo pueden estar ocupadas por un elemento en un momento determinado. Este es el caso, por ejemplo, de los simuladores basados en [autómatas celulares](#) (Figura 3.3).

Figura 3.3: Simulador de tráfico basado en [autómata celular](#). El espacio se divide en celdas que pueden estar vacías u ocupadas por un vehículo a una velocidad (más oscuro implica más lento). Concretamente muestra la evolución a lo largo del tiempo del movimiento de un modelo de [car-following](#) de 2 vehículos donde en eje x representa la posición en la vía y el eje y el momento temporal (iteración) de la vía.



- **Espacio continuo.** Simulación que transcurre en una secuencia infinita de puntos en el espacio. Es el caso por ejemplo de los simuladores basados en modelos lineales (Figura 3.4).

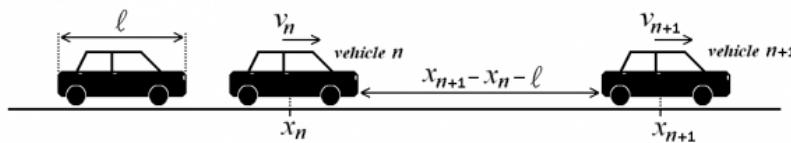


Figura 3.4: Ejemplo de un modelo lineal en un espacio continuo. La posición del vehículo es un valor  $x \in \mathbb{R}$ . Este ejemplo muestra un modelo de *car-following* donde el comportamiento de la aceleración del vehículo es determinado por la distancia al coche siguiente. Fuente: [Tordeux et al., 2011].

En el caso de la dimensión **tiempo**, la división se realiza en los mismos términos que en los del espacio:

- **Tiempo discreto.** También denominada *simulación de eventos discretos*, divide el tiempo en intervalos discretos, generalmente de longitud fija durante toda la simulación. Los simuladores basados en **autómatas celulares** son típicamente discretos, ya que cada posición en el espacio se va calculando para cada intervalo discreto de tiempo (Figuras 3.3 y 3.5).

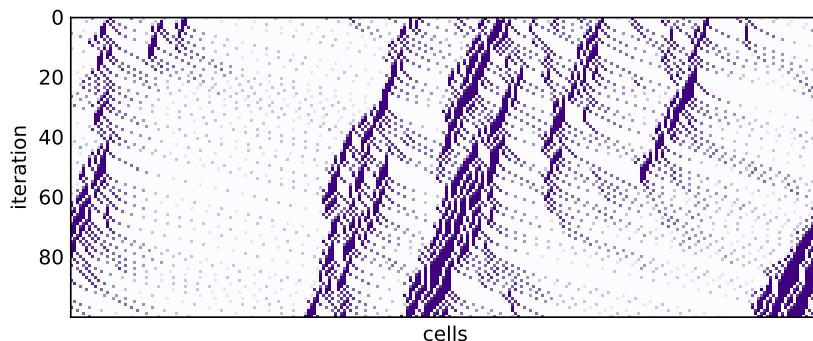


Figura 3.5: Aparición de retenciones en una autopista de 250 celdas usando el modelo Nagel-Scherckenberg. La densidad de ocupación es de 50 coches en la vía, la velocidad máxima es de  $5c/\Delta t$  y la probabilidad de frenada es de  $p = 0,5$ . Se puede observar en la figura cómo se desplazan las "olas" del atasco a lo largo de las 100 iteraciones.

- **Tiempo continuo.** En estos simuladores el tiempo es un factor más para un modelo de ecuaciones diferenciales. La Figura 3.4 ilustra un modelo de *car-following* que puede implementarse en una simulación de tiempo continuo si la aceleración viene determinada por un modelo que entre otros factores incluye el tiempo.

En nuestro caso, queremos conocer la situación exacta del vehículo en lugar de una situación aproximada en un dominio espacial discreto. También realizaremos la recolección de datos en intervalos cuantificables de tiempo, los cuales serán usados para modelar los comportamientos de los conductores y para contrastar los resultados.

Nuestro estudio se ceñirá, por tanto, hacia el trabajo sobre microsimuladores de **espacio continuo** y **eventos discretos**.

### 3.2 Modelos de micro-simulación

Los simuladores que se basan en un modelo de granularidad micro están en su mayoría implementados en dos tipos de paradigma: **autómatas celulares** y **sistemas multiagente**.

Existe un tercer punto de vista a la hora de implementar este tipo de modelos, que es el de los sistemas de partículas. Sin embargo, su ámbito de aplicación es el mismo que el del punto de vista macroscópico, esto es, usar sistemas de partículas para el análisis del tráfico como fluido. Por tanto, el resto de la sección describirá los dos tipos principales sin tener en cuenta éste último.

#### 3.2.1 Micro-simulación basada en *autómatas celulares*

<sup>2</sup> Existen arquitecturas diseñadas para operar de esta manera, esto es, arquitecturas basadas en **autómata celular** (**CA**, *cellular automaton*) (e.g. [Margolus, 1995]). En ellas, cada ciclo de reloj actualiza todas las celdas de memoria del autómata. Éstas arquitecturas se suelen usar para la implementación de modelos físicos superando en varios órdenes de magnitud la capacidad computacional de las arquitecturas tradicionales.

El **modelo Nagel-Scherckenberg** es un autómata celular que basa su funcionamiento en los siguientes aspectos:

- La vía está dividida en celdas de longitud  $7,5m$ . La razón de este valor es que ésta es la distancia media entre los parachoques traseros de dos coches consecutivos en un atasco.
- La celda puede tener dos estados, vacía o con un vehículo a velocidad  $v = \{0, \dots, v_{max}\} \in \mathbb{N}$ . La unidad de medida es  $c/\Delta t$  (celdas por unidad de tiempo).
- $\Delta t$  queda establecido en  $1s$ , considerado el tiempo medio de reacción de un conductor ante una eventualidad. Esto hace, por ejemplo, que una velocidad de  $6c/\Delta t$  sea  $45m/s$  ( $162km/h$ ).
- En cada ciclo y para cada vehículo, se realizan tres acciones de manera consecutiva: (i) acelerar una unidad si no está a la máxima velocidad o frenar si se ve obligado, (ii) freno aleatorio (la velocidad se reduce en una unidad hasta un mínimo de  $v = 1c/\Delta t$  con una probabilidad de  $p = 0,5$ ) y (iii) reposicionamiento.

Un **Autómata Celular (CA, Cellular Automaton)** es una colección ordenada de celdas (*células*) en un espacio  $n$ -dimensional que parcelan el universo de estudio. Cada una de ellas se encuentra en un estado (e.g. contiene un valor numérico), y el estado de toda la malla se actualiza de manera síncrona <sup>2</sup> (i.e., todas a la vez) en intervalos regulares de tiempo denominados *ciclos*. El cambio de estado de cada célula depende de los valores de las células vecinas y del mismo algoritmo de modificación al que responden todas y cada una de las células.

Por su naturaleza, los modelos de de micro-simulación basados en este paradigma se clasifican como simuladores de tiempo y espacio discreto, y son usados por su facilidad de implementación y porque son fácilmente paralelizables.

El modelo clásico de esta aproximación es el propuesto por Nagel-Scherckenberg en su artículo *A cellular automaton model for freeway traffic* [Nagel and Schreckenberg, 1992], un modelo teórico creado para la simulación de tráfico en autopistas. La Figura 3.5 muestra la evolución del tráfico en una autopista a lo largo del tiempo en una implementación basada en este paradigma.

En general los modelos de la literatura suelen ser una variación del de Nagel-Scherckenberg con modificaciones para estudiar aspectos concretos de modelos de tráfico o para dotarle de un mayor realismo. Algunos ejemplos de estas variaciones son la modificación del paso de *aleatorización* (e.g. [Barlovic et al., 1998]), reglas para determinar niveles de molestia a vehículos vecinos ([Wagner et al., 1997]), celdas más pequeñas (e.g. [Krauss et al., 1997]) para comprobar la metaestabilidad del flujo de tráfico, o modelos y reglas para cambio de carril en vías de dos carriles ([Brilon and Wu, 1999, Nagel et al., 1998]).

Los modelos basados en **autómatas celulares** no son suficientemente realistas desde un punto de vista microscópico. Por poner un ejemplo, en una situación típica de un modelo Nagel-Scherckenberg, los

vehículos realizan aleatoriamente aceleraciones y deceleraciones de  $27 \text{ km h}^{-2}$ . Es más, en una situación favorable, cualquier vehículo puede realizar una aceleración de  $0 \text{ km h}^{-1}$  a  $162 \text{ km h}^{-1}$  en tan sólo 6 s segundos. Por tanto, no ofrecen una visión demasiado realista ni fiable en caso de querer realizar estudios muy detallados de tráfico.

### 3.2.2 Micro-simulación basada en *sistemas multiagente*

Por otro lado, en un *sistemas multiagente* cada uno de los agentes tiene su propia entidad dentro del sistema. Esto es, perciben tanto el entorno como al resto de agentes y actúan de acuerdo a lo percibido y a su comportamiento. Basarse no sólo en las magnitudes físicas del resto de vehículos (e.g. distancia, aceleración, ...) sino también en un comportamiento de conducción ofrece un interesante campo de estudio a nivel cognitivo. Se habla más en detalle sobre los *sistemas multiagente* en el capítulo [Inteligencia Computacional](#) y sobre los comportamientos concretos de agentes de interés para esta tesis en el capítulo [4](#). Por ello, este apartado únicamente hará una pequeña introducción a estudios existentes y aplicaciones de simuladores basados en este modelo.

A diferencia de los *autómatas celulares*, los *sistemas multiagente* pueden emplazarse en un entorno virtual que represente un espacio continuo y no discreto. Esto permite modelar con mayor fidelidad magnitudes físicas asociadas a cada agente (e.g. posición y velocidades actuales, dimensiones del vehículo, masa, velocidad máxima permitida, ...). Sin embargo, aun así no es una propiedad inherente de éstos. No existe ninguna limitación en cuanto a la representación del espacio y es perfectamente posible representar un modelo basado en *autómatas celulares* usando para ello *sistemas multiagente*.

Cada uno de los agentes es independiente del resto, y una consecuencia directa es que el comportamiento de cada individuo permite evaluar comportamientos grupales complejos, como el descrito en la Figura [3.6](#). Esta independencia da la posibilidad de tener todos los agentes diferentes entre sí, ofreciendo la ventaja de permitir experimentar con diferentes perfiles de conducción (e.g. un perfil agresivo en un flujo de tráfico dominado por conductores tranquilos).

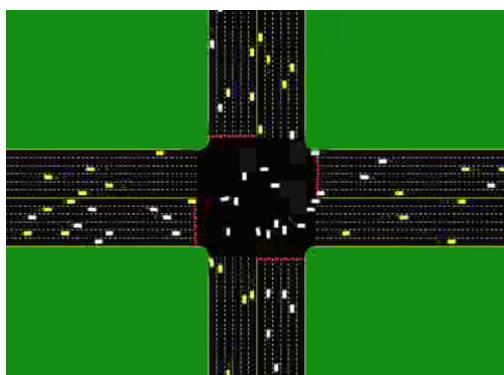


Figura 3.6: Simulación de comportamiento en intersección basada en un *sistema multiagente*. En ésta, cada uno de los vehículos incorpora un controlador para hacerlo autónomo. Modelar este caso de estudio con una arquitectura basada en *sistemas multiagente* permite centrarse en el diseño del agente en concreto (i.e. el controlador de conducción del vehículo) y estudiar el comportamiento emergente surgido de la interacción de todos los agentes. Fuente: Proyecto AIM (<http://www.cs.utexas.edu/~aim/>).

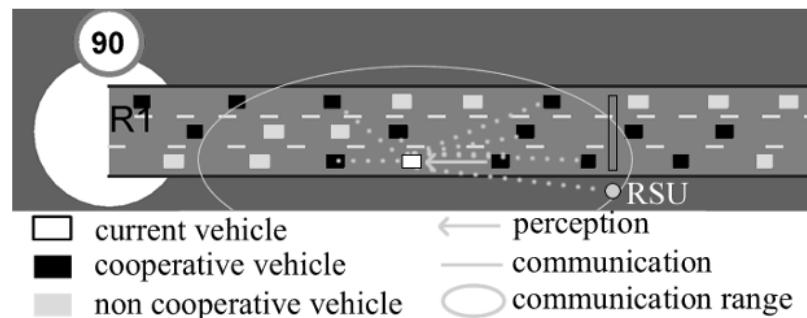
Esto es debido a que en un **sistema multiagente** cada agente es una parte del sistema y las decisiones de cómo se ha de comportar las toma él mismo. Desde el punto de vista de un **autómata celular**, el comportamiento existe en cada celda, sin dar control al contenido o estado de cada celda.

En general los estudios basados en este modelo suelen seguir el patrón 1 **DVU**  $\equiv$  1 agente, dando así una enorme cantidad de posibilidades a experimentar. Por ejemplo en [Das et al., 1999] se hace uso de **sistemas de control borroso** para decidir cómo comportarse en la vía mientras que en [Ehlert and Rothkrantz, 2001] se hace uso de un patrón reactivo. Otros, como [Dia, 2002] o [Balmer et al., 2004] hacen uso de encuestas o censos para establecer las propiedades y calibrar los parámetros de diferentes tipos de agentes.

Los estudios en materia de simuladores de tráfico con sistema multiagente no se limitan a vehículos, sino que se usan también en otras áreas como el control de luces de tráfico o agentes para peatones entre otros. Por ejemplo el estudio presentado en [Clymer, 2002], los agentes del sistema son las señales de tráfico luminosas y no los vehículos, y el objetivo es adaptar la señalización en una red de carreteras para minimizar al máximo el tiempo de espera por parte de los vehículos en las intersecciones gestionadas por las señales. Otro ejemplo es el propuesto en [Galis and Rao, 2000], donde los agentes, en lugar de ser los vehículos son los tramos de las carreteras; en él, los vehículos poseen comportamiento, pero lo reciben del agente que les guía de acuerdo a la zona en la que se encuentran. Esto tiene la ventaja de que el paso de información a vehículos dentro de la misma zona se realiza mucho más rápido en un entorno distribuido.

En los últimos años, otro concepto que está en auge es el de las redes intervehiculares e intravehícuales, **Vehicle-to-Vehicle (V2V)** y **Vehicle-to-Infrastructure (V2I)**. El modelo de **sistemas multiagente** permite la implementación rápida de diferentes políticas y protocolos de comunicación via sensores y actuadores para estudiar estos tipos de redes de comunicación (Figura 3.7).

Figura 3.7: Captura de pantalla del simulador **MovSim**. Este simulador implementa un modelo multiagente donde los vehículos incorporan sistemas de comunicación vehicular. El estudio se centra en el uso de la comunicación entre vehículos para el acoplamiento dinámico de vehículos en sus respectivos carriles. Fuente: [Guériaud et al., 2015].



Estudios como por ejemplo [Shiose et al., 2001] o [Galis and Rao, 2000] hacen uso de un **sistema multiagente** para implementar diferentes

formas de **V2V** con el objetivo de aliviar congestiones de tráfico (en el primer caso) y por el propio estudio de las comunicaciones en sí (en el segundo caso). En el caso de redes **V2I**, un buen ejemplo es [Dresner and Stone, 2004], donde se representan como agentes tanto los vehículos como las intersecciones de la vía. Éstas gestionan un sistema de reservas de tokens que los vehículos solicitan cuando van a entrar en la intersección y devuelven cuando salen, comunicando en todo momento mediante eventos los cambios en dicho sistema. El estudio concluye que una comunicación de este tipo es más eficiente que una intersección clásica basada en señales de tráfico luminosas.

### 3.3 Software de simulación

Para la realización de esta tesis es necesario contar con un paquete de simulación que permita modelar un sistema multiagente en el que poder ejecutar los modelos de comportamiento desarrollados.

Aunque en un principio se ha valorado el desarrollo de una solución propia, la oferta de simuladores en el mercado es muy amplia, cada uno de ellos implementando uno o varios modelos diferentes bajo distintas licencias. Por ello se ha optado por la elección de un paquete de simulación ya desarrollado.

Para elegir el mejor simulador que se adapte a nuestras necesidades se ha realizado un listado de características obligatorias y, de este modo, realizar una primera criba eliminando simuladores no aptos (resumidos en la figura 3.8):

1. **Tipo de simulador.** Para nuestras necesidades es necesario un simulador que implemente **microsimulación**, ya que es el único tipo de granularidad que permite evaluar el comportamiento de un conductor independientemente del resto de la simulación. Además, debido a la forma en la que se recolectan los datos, es necesario que represente un **espacio continuo** y una dimensión de **tiempo discreto** con una resolución de al menos 1 segundo.
2. **Modelo de simulación.** Debe ofrecer un entorno basado en un **sistema multiagente** donde cada **DVU** se comporte como agente individual.
3. **Entorno de simulación.** Debe ofrecer un entorno de **simulación de tráfico general**, permitiendo la creación de escenarios. Quedan excluidos los simuladores de propósito específico o de casos particulares como simuladores de autopistas, congestiones o colisiones.
4. **Extensibilidad.** El simulador debe permitir extender de alguna manera la ejecución de los modelos desarrollados en los agentes (**DVUs**). Aunque se puede considerar que si es simulador **Software Abierto (OSS, Open Source Software)**, se puede modificar su comportamiento para adecuarlo a los modelos desarrollados, es mejor

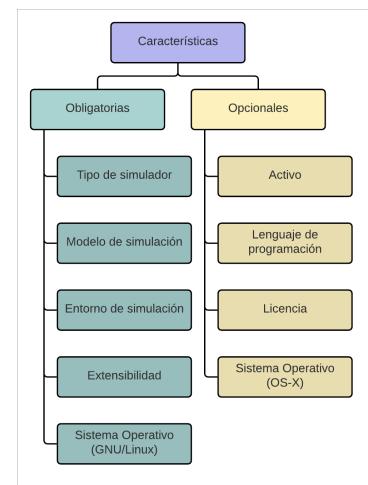


Figura 3.8: Características obligatorias y deseables del simulador donde implementar nuestros modelos personalizados de conductor.

que el propio software ofrezca los mecanismos necesarios para la integración sin necesidad de tocar los fuentes del sistema.

5. **Sistema operativo.** Es imprescindible que el software se ejecute sobre sistemas operativos GNU/Linux por la configuración de los sistemas sobre los que se trabaja.

Posteriormente se ha desarrollado un listado de características deseables. No son determinantes para descartar simuladores pero sí favorecen la elección de unos sobre otros.

1. **Activo.** Es preferible que el sistema esté activamente desarrollado porque eso favorece la aparición de parches y mejoras sobre el software. En caso contrario, se trata de un proyecto con poca actividad por parte de sus autores.
2. **Lenguaje de programación.** Es favorable la implementación de los modelos en código [Python](#).
3. **Licencia.** Es preferible una licencia de tipo [OSS](#) ya que, en caso de error o falta de funcionalidad, es posible acceder a los fuentes para modificarlos.
4. **Sistema operativo.** Es favorable que el sistema se ejecute en entornos tipo OS X.

### *3.3.1 Entornos de simulación a estudiar*

El primer listado de características deja atrás la mayoría de simuladores (una gran cantidad de ellos son o bien de propósito específico, están desarrollados para sistemas operativos Windows o no permiten extender su modelo). Tras la selección, nos quedamos con los siguientes simuladores: [AORTA](#), [MatSIM](#), [MitSIM](#), [MovSim](#) y [SUMO](#).

Dichos entornos están prácticamente igualados en las características presentadas, tal y como se puede observar en la Tabla 3.1. Sin embargo, en materia de extensibilidad, [SUMO](#) es el único que permite el desarrollo de [DVUs](#) de manera externa. El resto requiere la modificación del código fuente del simulador para variar los comportamientos de los conductores. [AORTA](#) además no es un proyecto que se mantenga activo en la actualidad (las últimas modificaciones del repositorio datan de principios del año 2014).

No obstante se ha tratado de modificar los comportamientos de los conductores en los cuatro simuladores para validar este hecho y ha quedado patente que es mucho más eficaz usar [SUMO](#) como simulador para nuestro estudio.

Tabla 3.1: Tabla comparativa donde se contrastan las características de los simuladores seleccionados. El simulador **SUMO** es el que más características cumple de las originalmente planteadas.

	AORTA	MatSIM	MitSIM	MovSim	SUMO
Activo	✗	✓	✗	✓	✓
Lenguajes de programación	Scala	Java	C++	Java	C++/Python
Licencia propietaria	✗	✗	✗	✗	✗
Licencia OSS	✓	✓	✓	✓	✓
Licencia <b>GPL</b>	✓	✓	✗	✓	✓
API	✗	✗	✗	✗	✓
GNU/Linux	✓	✓	✓	✓	✓
OS X	✓	✓	✗	✓	✓
Windows	✓	✓	✗	✓	✓

### 3.4 Entorno seleccionado: **SUMO**

En definitiva, el simulador que más se adapta a nuestras necesidades y el que se usará como simulador base en el desarrollo de esta tesis será **SUMO** [Krajzewicz et al., 2002, Behrisch et al., 2011, Krajzewicz et al., 2012].

**SUMO** es un entorno de microsimulación licenciado bajo la **GPL** versión 3.0 y desarrollado por el instituto de sistemas de transporte del Centro Aeroespacial Alemán. Implementa un modelo discreto en el tiempo y continuo en el espacio.

Además de simulación clásica, incorpora una interfaz gráfica (se puede ver una captura de la vista gráfica en la figura 3.9) donde se puede ver el comportamiento de cada vehículo durante la simulación. Es interesante para obtener de un vistazo información acerca del funcionamiento del modelo en concreto a controlar. Otras de las características que el simulador ofrece son las siguientes:

- Granularidad micro y meso.

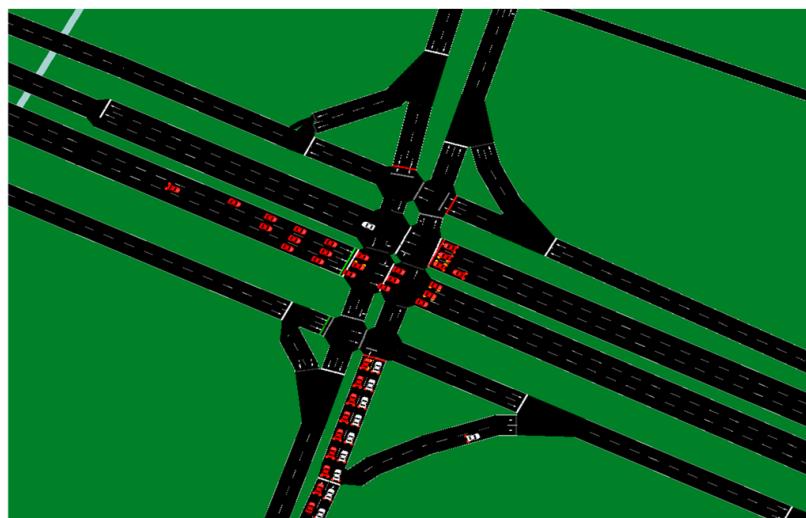


Figura 3.9: Captura de pantalla del simulador **SUMO**. Además de software de simulación propiamente dicho, **SUMO** provee de una interfaz gráfica que permite una visualización general, de zonas y de elementos en concreto a la vez que permite la variación de configuración de la simulación durante el desarrollo de la misma.

- Multimodalidad permitiendo modelar no sólo tráfico de vehículos sino de peatones, bicicletas, trenes e incluso de barcos.
- Simulación con y sin colisiones de vehículos.
- Diferentes tipologías de vehículos y de carreteras, cada una con diferentes carriles y éstas con diferentes subdivisiones de subcarriles (diseño conceptual para permitir modelar comportamientos en vehículos como motocicletas y similares).

**SUMO** usa como modelo por defecto de *car-following* el modelo de Stefan Krauß [Jin et al., 2016], debido a su simplicidad y su velocidad de ejecución y como modelo de cambio de carril el modelo de Gipps [Krajzewicz et al., 2002]. No obstante, se encuentran para seleccionar otros modelos como el **Intelligent Driver Model (IDM)**, el modelo de tres fases de Kerner [Kerner et al., 2008] y el modelo de Wiedemann [Wiedemann, 1974].

Al estar licenciado bajo la licencia **GPL**, su distribución implica a su vez la distribución de su código fuente. Esto permite la modificación de su comportamiento y el desarrollo de nuevos modelos integrados dentro del simulador. Sin embargo nosotros no haremos uso de esta característica, sino que usaremos **SUMO** como aplicación servidor y el módulo **TraCI** como aplicación cliente desde donde gestionar todos los aspectos de la simulación.

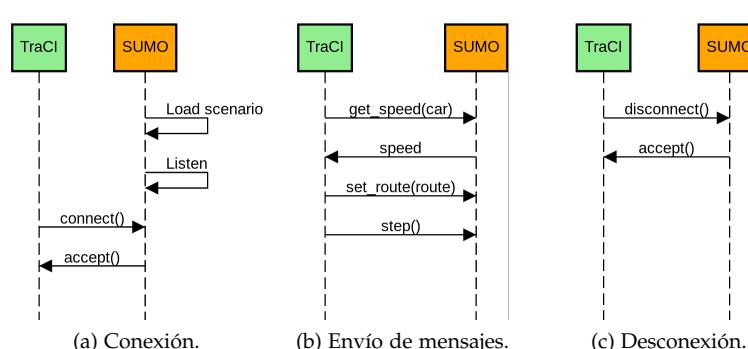
#### 3.4.1 La interfaz **TraCI**

**TraCI** [Wegener et al., 2008] es tanto el nombre del protocolo de comunicación expuesto por **SUMO** en su versión servidor como el nombre de la librería escrita en **Python** para interactuar con el mismo.

Como protocolo, la interacción a través de cliente/servidor comienza especificando a **SUMO** que se desea trabajar de este modo. En ese momento, **SUMO** se inicializa en modo servidor dejando abierto un puerto TCP para la conexión del cliente (figura 3.10 (a)).

Una vez el servidor se encuentra en ese estado, el cliente se conecta enviando una señal de conexión indicando que él se encargará de controlar la simulación. Desde ese momento y hasta que el cliente no

Figura 3.10: **SUMO** ofrece la posibilidad de interactuar con la simulación desde cualquier aplicación a través del uso del protocolo **TraCI**. En la figura podemos ver, de izquierda a derecha, ejemplos de comunicación a través de la interfaz como el *handshake* o inicialización, mensajes de obtención de información y modificación de la misma más una solicitud de avance de paso en la simulación y una señal de finalización de simulación y desconexión.



envíe una señal de desconexión (figura 3.10 (c)), el cliente podrá enviar y recibir todos los mensajes que desee para capturar información y modificar los detalles de la simulación, incluido el mensaje *step*, que es el encargado de avanzar un paso en la simulación (figura 3.10 (b)).

Como librería, **TraCI** es un módulo desarrollado en **Python 2.7**. Aunque es posible trabajar directamente con el protocolo de comunicación a través de sockets, una librería abstrae todos los detalles dando una interfaz de trabajo más clara y sencilla. Por ello, aunque no se usarán en la tesis, existen otras dos implementaciones que merece la pena mencionar:

- **TraCI4J**<sup>3</sup>. El homólogo de la librería de abstracción de **Python** pero para el lenguaje Java. Está desarrollada por un tercero.
- **TraaS**<sup>4</sup>. Una plataforma ofrecida como SaaS que proporciona una interfaz de servicios web bajo protocolo SOAP para abstraer el protocolo en mensajes HTTP (figura 3.11).

<sup>3</sup> <https://github.com/egueli/TraCI4J>.

<sup>4</sup> <http://traas.sourceforge.net/cms/>.

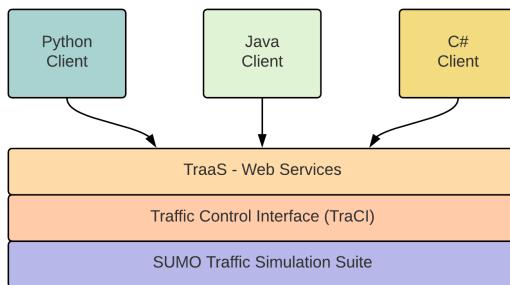


Figura 3.11: Concepto arquitectural de la plataforma **TraaS**. La plataforma se conecta como cliente a **SUMO** y ofrece un API basado en SOAP de mensajes que traduce en mensajes del protocolo **TraCI**, lo que independiza completamente la elección de lenguaje de programación a la vez que abstrae los detalles del protocolo de comunicación.



## 4 Modelos de comportamiento

El objetivo que persigue la simulación de tráfico es hacer cada vez más realistas los modelos generados. Cuando el simulador está basado en **sistemas multiagente**, el realismo aumenta cuanto más se parece el comportamiento de los agentes al de los conductores reales.

Conducir implica la ejecución de múltiples tareas en paralelo, cada una de ellas pertenecientes a un nivel cognitivo. Además, las acciones no están limitadas a la interacción con el vehículo; el conductor ha de tener en cuenta otros factores como pueden ser señales, peatones o **Sistema Avanzado de Ayuda a la Conducción (ADAS, Advanced Driver Assistance System)**.

El modelo de [Michon, 1985] define tres niveles de abstracción para las tareas que requieren procesos cognitivos, entre ellas la tarea de conducir: el nivel de **control**, donde se encontrarían las tareas de más bajo nivel como el mantenimiento de la velocidad o los cambios de marcha, el **táctico** donde se engloban tareas encargadas de mantener la interacción con el entorno como los cambios de carril y el **estratégico**, al que pertenecen tareas de más alto nivel como el razonamiento y la planificación de rutas (ver figura 4.1).

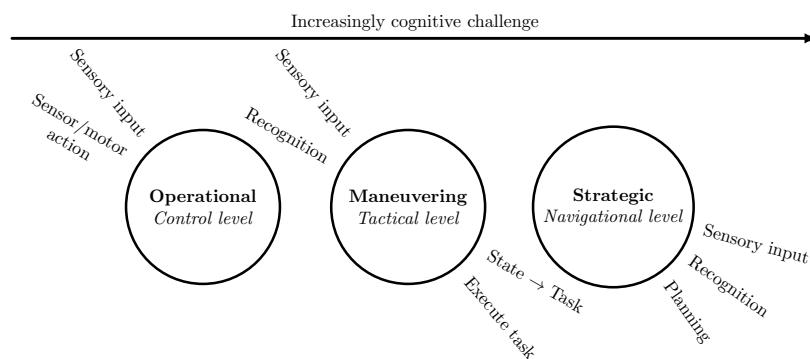


Figura 4.1: Los tres niveles jerárquicos que describen la tarea de conducción según [Michon, 1985]: *estrategia* (i.e. las decisiones generales), la *maniobra* (i.e. decisiones durante la conducción de más corto plazo) y *control* (i.e. automatismos).

A pesar de que se trata de una clasificación subjetiva, es un modelo ampliamente aceptado dentro del área de los **ITS**. Algunos estudios llegan incluso a definir intervalos de tiempo de razonamiento para las tareas de cada nivel, como por ejemplo [Alexiadis et al., 2004], donde se llegan a proponer tiempos que separan unos niveles de otros: alrededor de 30 s para las tareas del nivel de planificación, entre 5 s

a 30 s para las tareas de nivel táctico y por debajo de los 5 s para las tareas de control.

Otro modelo jerárquico de tres niveles muy referenciado en la literatura es el *skill-rule-knowledge* de [Rasmussen, 1986], una generalización del modelo propuesto por [Michon, 1985] al comportamiento y razonamiento humano. Postula que éste se puede basar en **habilidades** (actividades completamente automatizadas de la forma *percepción → ejecución*), **reglas** (situaciones familiares o estereotipadas de la forma *percepción → reconocimiento de la situación → planificación → ejecución*) y **conocimiento** (actividades conscientes que implican resolución de problemas y toma de decisiones de la forma *percepción → reconocimiento de la situación → toma de decisión → planificación de la ejecución*) que suelen ser necesarias en situaciones poco familiares). Algunos trabajos se basan en este modelo en lugar del de [Michon, 1985] como el presentado por [Chaib-draa and Levesque, 1994] donde abarca los tres niveles de abstracción representando en un escenario urbano tres tipos diferentes de situaciones: rutinaria, familiar y no familiar.

El comportamiento de un conductor al volante tiene una relación directa con el nivel de abstracción táctico. Se puede entender como el nivel encargado de planificar acciones a corto plazo para conseguir objetivos a corto plazo. Las tareas de control son automáticas e influyen poco o nada en la toma de decisiones de tareas como *cuánto acelerar en esta situación* o *cuándo cambiar de carril*. Las tareas estratégicas están a un nivel más alto de abstracción (e.g. la ruta a seguir hasta mi destino) y tampoco afectan demasiado al comportamiento en situaciones concretas. No obstante algunos trabajos han demostrado que en ocasiones la planificación de la ruta sí afecta a decisiones normalmente asociadas el nivel táctico como por ejemplo la preferencia de un conductor por uno u otro carril de la vía [Wei et al., 2000, Toledo et al., 2003].

Nuestro interés es el uso de agentes como unidades en simulación. Después de todo trabajamos con sistemas multiagente. Sin embargo, y aunque en los trabajos más modernos exista una cierta predisposición hacia este paradigma, no todos los trabajos se basan en él. El auge de su uso coincide con el renacimiento de la inteligencia artificial, alrededor de los años 1990, y los modelos que se describen en este apartado, sobre todo posteriores a esta fecha, se basan en este tipo de sistemas.

Las tipologías de agentes son muy variadas. Existen desde trabajos que explotan las características de los agentes reactivos (e.g. agentes que continuamente van realizando decisiones de control para mantenerse en la vía [Ehlert and Rothkrantz, 2001]) hasta aquellos que proponen complejos frameworks para definir comportamientos (e.g. cuatro unidades de funcionamiento interconectadas, *percepción, emoción, toma de decisiones y ejecución*, que gestionan el comportamiento inteligente del agente en cada situación [Al-Shihabi and Mourant, 2001]).

La información que manejan estos agentes dentro de los simuladores suele limitarse a tipologías de vehículo (e.g. turismos o vehículos de grandes dimensiones) y magnitudes físicas (tamaño, velocidad máxima). Dependiendo del trabajo, algunos autores añaden más conocimiento a los agentes; por ejemplo, en [Hidas, 2002], el autor incluye en los agentes, denominados **Driver-Vehicle Objects (DVOs)** (otra denominación del concepto de DVU), información adicional sobre el tipo de conductor y el nivel de conocimiento de la red de carreteras.

El resto del capítulo introducirá los modelos de comportamiento más conocidos y hará especial hincapié en el estado más reciente de modelos basados en técnicas de la **inteligencia computacional**.

#### 4.1 Comportamientos modelados

Las tareas que se realizan en el nivel táctico del comportamiento son aquellas orientadas a circular dentro del flujo de tráfico interactuando con éste. En la literatura, estas tareas se centran en dos clases generales de problema diferentes: el **longitudinal** de aceleración y el del **cambio de carril**.

Los modelos de **aceleración** se ocupan de gestionar las alteraciones de la tasa de aceleración (positiva o negativa) en un entorno lineal como lo es un carril de tráfico.

Tras la aparición de los modelos psico-físicos se comprobó que el umbral en las percepciones, y por tanto el comportamiento, podía variar dependiendo de las situaciones. Por ello, el *car-following* no es más que una entre diferentes clases o regímenes de aceleración. Algunos de los regímenes más usados en la literatura son *free-flow*, *car-following*, *approaching*, y *emergency*, aunque algunos autores definen nuevos regímenes, cada uno con sus límites de aplicación (Figura 4.2).

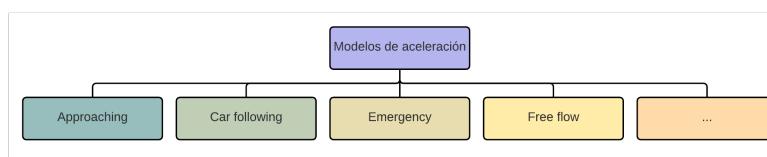


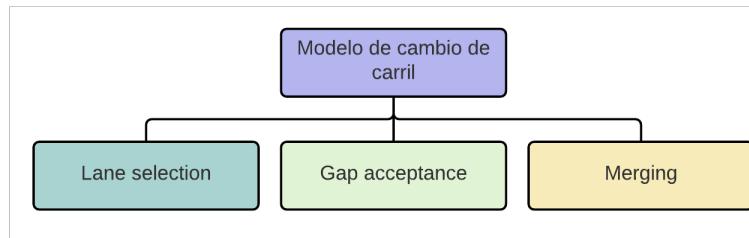
Figura 4.2: Diferentes regímenes de aceleración para modelos longitudinales.

El tráfico real, sin embargo, no está compuesto por un sólo carril, sino por varios. Los modelos de **cambio de carril** tienen como objetivo representar las maniobras de cambio de carril, ya sea porque el agente quiere mejorar su circulación (e.g. quiere realizar un adelantamiento) o porque su ruta lo requiere (e.g. está próxima la rampa de salida que quiere tomar en una autopista).

Se divide tradicionalmente en una operación que involucra dos pasos. La selección de carril (*lane-selection*) al que cambiarse y la ejecución del cambio (*merging*). En la operación de merging se suele involucrar

otra operación denominada *gap-acceptance*, aunque algunos autores la tratan como operación independiente ( (Figura 4.3)). Otros autores pueden llegar a añadir operaciones todavía más especializadas.

Figura 4.3: Operaciones involucradas en el proceso de cambio de carril.



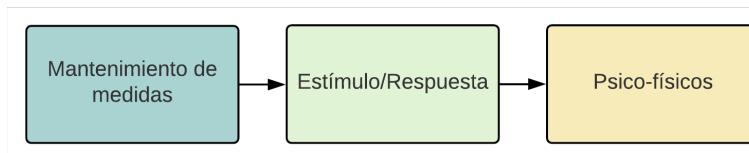
En la literatura los modelos de aceleración han sido mucho más estudiados que los de cambio de carril, entre otras cosas por la dificultad en la captura de los datos y, por tanto, por su escasez.

El estudio del comportamiento en los cambios de carril es muy interesante debido a que tiene efectos opuestos según la intensidad de tráfico de la vía en la que se ejecutan. Por un lado, si ésta es baja, mejora la velocidad media del flujo de la vía. Sin embargo, según aumenta, los cambios de carril comienzan a afectar a éste en formas de ondas de choque ([[Sasoh and Ohara, 2002](#), [Jin, 2006](#)]) e interferir incluso más que los modelos de *car-following* ([[Laval and Daganzo, 2006](#)]).

## 4.2 Modelado de conductores clásico

Los primeros trabajos sobre modelos de conducción datan de comienzo de los años 50 con estudios sobre el concepto denominado ***car-following***, acuñado por [[Reuschel, 1950](#)]. En este comportamiento, la velocidad está condicionada por el vehículo que le precede. En el primer modelo concreto ([[Pipes, 1953](#)]), el comportamiento responde a mantener un espacio variable en función de las velocidades.

Figura 4.4: Evolución de los tres tipos generales de modelo de *car-following*: mantenimiento de medidas, estímulo→respuesta y psico-físicos. Con la llegada de los psico-físicos se vio que *car-following* no era más que uno de tantos regímenes distintos dentro de los modelos de aceleración.



Este modelo se puede considerar de una clase que denominaremos **mantenimiento de medida** (Figura 4.4) dado que su objetivo es mantener constantemente una distancia segura, determinada a partir de la ecuación de la velocidad cuando el tiempo no baje de 1,02 segundos. Otros trabajos trabajan con el mantenimiento de otras medidas como distancia relativa al parachoques delantero o trasero.

Más adelante, a finales de los años 1950 se presentó el modelo **Gazis-Herman-Rothery (GHR)**, también conocido como **Generalized Model (GM)** ([Chandler et al., 1958]). Fue desarrollado en el seno de la *General Motors* y sirvió como base para el desarrollo de muchos modelos posteriores antes de la introducción del modelo de Gipps. Este modelo dio origen a una nueva clase de modelos de conducción, los de tipo **estímulo → respuesta**.

Se caracteriza por el uso del concepto *estímulo → respuesta* (Figura 4.4), donde la respuesta del vehículo (el cambio en la tasa de aceleración) es debida a la activación de un estímulo (la variación en la distancia con el vehículo delantero) tras pasar un tiempo de retardo  $\tau$ . Concretamente calcula el valor de la aceleración  $a$  en un instante  $t$  como:

$$a(t) = cv^m(t) \frac{\Delta v(t - \tau)}{\Delta x^l(t - \tau)} \quad (4.1)$$

Siendo  $t$  es el instante actual,  $a(t)$  la aceleración del vehículo,  $\delta v(t)$  y  $\delta x(t)$  son la velocidad y distancia relativas al siguiente coche respectivamente,  $v$  la velocidad del vehículo y  $c, m, l$  y  $\tau$  constantes, siendo ésta última el tiempo de reacción del conductor.

En realidad los modelos *estímulo → respuesta* son la evolución lógica de los modelos anteriores, donde se pasa de un cálculo de velocidad en función de la distancia (u otra medida) a un sistema de control donde la variable a controlar es la aceleración en función de uno o varios estímulos de entrada, además con un retardo simulando el tiempo de reacción. Algunas modificaciones sobre el algoritmo original son la asimetría en la tasa de cambio de aceleración y deceleración [Gazis et al., 1959] o la inclusión más coches precedentes [Bexelius, 1968].

Los métodos de estas dos clases tienen un problema principal: suponen que el conductor es capaz de percibir incluso el más ínfimo cambio en las variables observadas, cuando la realidad no es así. Por ello, a mediados de los años 1970 apareció una nueva clase de modelos de *car-following*, denominados posteriormente como **psicofísicos** [Wiedemann, 1974], donde se introduce el concepto de *umbral perceptual* como solución a dicha limitación. El *umbral perceptual* de una medida es el límite a partir del cual se percibe un cambio en dicha medida. Mediante su uso, las acciones de los vehículos se limitan únicamente a los cambios **perceptibles** en los coches delanteros.

A finales de la década, en 1978, se cumplió otro hito en el desarrollo de modelos de conducción. [Sparmann, 1978] define el primer modelo de cambio de carril, inspirándose en las clases de modelo psicofísico. El verdadero interés de este modelo es que sentó las bases de dos conceptos que perduran hoy en día. El primero, la diferenciación entre cambio a carriles rápidos y lentos <sup>1</sup>. El segundo, la diferenciación entre la selección de carril o *lane-selection* y la ejecución del cambio o *merging*.

<sup>1</sup> En [Sparmann, 1978] el autor entiende la derecha como el carril lento y la izquierda como el carril rápido, y es como se entiende dentro del contexto de esta tesis. Sin embargo en otros países esta correspondencia es al revés.

La viabilidad en un cambio de carril se determina haciendo uso de modelos denominados *gap-acceptance*, donde los vehículos calculan si caben o no en un determinado hueco y actuan en consecuencia.

En su origen los modelos de *gap-acceptance* se desarrollaron para resolver situaciones en intersecciones e incorporaciones. En la actualidad son los modelos usados tras seleccionar el cambio de carril y antes de ejecutar físicamente el cambio, y dependiendo del modelo, es incluido como operación dentro de uno u otro.

El modelo típico del **gap-acceptance** responde a la ecuación 4.2, donde en un momento  $t$ , el cambio a un carril  $l$  es viable ( $f_{g_l}(t) = 1$ ) o no ( $f_{g_l}(t) = 0$ ) dependiendo de si el espacio en el carril destino  $g_l(t)$  es mayor o menor que un “hueco crítico” (en inglés critical gap)  $g_l^{crit}(t)$ .

$$f_{g_l}(t) = \begin{cases} 0 & \text{si } g_l(t) < g_l^{crit}(t) \\ 1 & \text{si } g_l(t) \geq g_l^{crit}(t) \end{cases} \quad (4.2)$$

Por otro lado, existen autores que definen factores de influencia que modifican el modelo típico. Algunos ejemplos pueden ser la velocidad absoluta del vehículo ([Gipps, 1986, K. et al., 1996]), el tipo de cambio (MLC o DLC, usado en [Ahmed, 1999, Toledo et al., 2007]), la relativa con los vehículos delantero y trasero del carril destino ([Ahmed, 1999]) o incluso el peso de encontrarse o no en una situación de cooperación ([Ahmed, 1999, Hidas, 2002]).

Con los modelos psicofísicos se llegó a la conclusión que no todas las situaciones eran iguales, sino que en función del entorno y el momento los umbrales podían variar, y que el *car-following* no era sino un subtipo más de una clase más amplia que se definió como *modelos de aceleración* (figura 4.2).

Debido a eso y a la irrupción de los modelos de cambio de carril, los posteriores modelos y frameworks desarrollados se componen de dos o más submodelos que responden a diferentes umbrales. Sin embargo, esto provoca que los modelos desarrollados sean más complejos ya que, cuantos más regímenes se tratan de agrupar en un mismo modelo, más aumenta el número de factores a generalizar y ajustar.

El trabajo de Gipps es uno de los primeros modelos que agrupa varios regímenes distintos (concretamente *car-following* y *free-flow*) [Gipps, 1981]. Sin embargo consideramos más interesante su posterior trabajo, [Gipps, 1986] ya que puede considerarse como la primera solución para el cambio de carril.

Introduce el concepto de que los cambios de carril obedecen a diferentes motivaciones. Por un lado, los cambios pueden ser **obligatorios** (denominado en la literatura como *Mandatory Lane Change (MLC)*) cuando los vehículos se ven obligados a abandonar el carril que ocupan. Por otro lado, pueden ser **discretionales** (denominados como *Discretionary Lane Change (DLC)*) cuando el cambio obedece a motivaciones más relacionadas con la mejora del confort o de la situación

actual de conducción.

En su modelo, Gipps propone un modelo para el cambio de carril al aproximarse a un cambio de dirección. Dicho modelo identifica tres distancias que caracterizan el comportamiento del conductor en función de cómo de lejos está dicho punto: (i) **lejos**, en el que no existe condicionamiento en la decisión de cambio de carril, (ii) **medio**, donde el conductor empieza a ignorar los cambios que dan ventaja de velocidad si no hacia carriles distanciados del de salida y (iii) **cerca** donde los vehículos deben estar en el carril de cambio de salida.

Otro concepto que incluye el modelo de Gipps y que exporta a modelos posteriores de cambio de carril es el de ampliar el número de *critical gaps* a más de un hueco: las distancias hasta el vehículo delantero y hasta el vehículo trasero, forzando a que durante el proceso de *gap-acceptance* las condiciones de ambos huecos tengan que ser aceptables.

Posteriormente, en [Weidemann and Reiter, 1992] se desarrolla un framework similar pero teniendo en cuenta los cambios a carriles lentos (para representar, por ejemplo, obstrucciones como accidentes o un vehículo lento) y a carriles rápidos (para situaciones como condiciones de la ruta). Concretamente, se proponen hasta cuatro clases diferentes de modelos de aceleración en función de las posiciones y velocidades relativas entre el vehículo sujeto y el siguiente:

- Modelo **free-flow**: El comportamiento no se ve afectado por el del vehículo delantero.
- Modelo **car-following**: El comportamiento sí se ve influenciado por el vehículo delantero, obligando a disminuir la velocidad deseada en el conductor en cuestión.
- Modelo **approaching**: Situación intermedia entre **free-flow** y **car-following**.
- Modelo **emergency**: Situación es crítica (e.g. colisión inminente) obligando, normalmente, a respuestas extremas.

Además el autor incluye un modelo para influir en su desempeño en función del entorno actual (las características de los vehículos de alrededor) y el entorno potencial (la estimación de las características del entorno en momentos posteriores). Otros autores identifican diferentes situaciones dependiendo del alcance del trabajo como por ejemplo el **car-following** o el **stop-and-go** [Toledo et al., 2003, Liu and Li, 2013]).

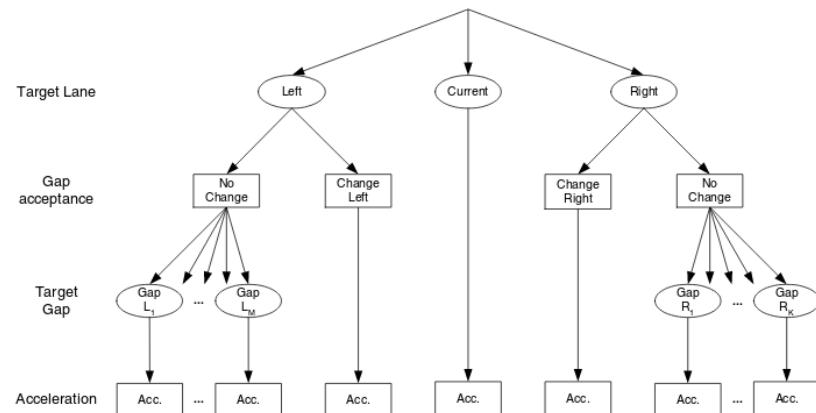
El modelo de Gipps sin embargo, sufre de dos problemas clave, heredados además por muchos trabajos posteriores. El primero es que un cambio de carril no sólo involucra al conductor que lo ejecuta. En [Hidas, 2002], el autor resalta el problema de que, en situaciones de congestión, el cambio ha de ser o bien forzado o bien a través de colaboración; en caso contrario, los vehículos no abandonarán el carril congestionado.

Uno de los primeros trabajos en abordar el comportamiento colaborativo es el de [Fritzsche and Ag, 1994]. En éste, se describe un modelo de microsimulación para analizar cuellos de botella (e.g. un accidente donde se bloquea uno de los carriles). Los autores describen el problema pero no consideran la modificación de los modelos en cambio de carril. [Yang and Koutsopoulos, 1996] sin embargo presentan un entorno de simulación (MitSIM) que introduce, entre otros, un modelo de cambio de carril en el que se habla específicamente de comportamiento colaborativo. Introducen el concepto de función de cortesía (*courtesy yielding function*) la cuál afecta al modelo de *car-following* de un vehículo cuando otro intenta incorporarse al carril. Sin embargo, los detalles de dicho proceso no están especificados en el artículo.

El segundo problema es el del uso de **árboles secuenciales para evaluar los factores**. En el momento que la evaluación encuentra una situación favorable, el resto de factores no son evaluados. Por ejemplo, tal y como está formado el modelo de Gibbs, y algunos basados en éste como [Hidas, 2002], un *MLC* inhibe cualquier posibilidad de realizar un *DLC* independientemente de la utilidad de ambos.

Para evitar esta limitación, algunos autores hacen uso de técnicas como modelos probabilísticos [Toledo et al., 2003, Toledo et al., 2007, Wei et al., 2000]. Por ejemplo, en [Toledo et al., 2007] el modelo que proponen se basa en el concepto de “objetivo a corto plazo” para elaborar un “plan a corto plazo” apoyándose, para ello, en un arbol de decisión (Figura??). Aunque mantiene la clasificación, es probabilístico y existe opción de realizar un *DLC* en lugar de un *MLC* aun en situaciones donde ambas clases se activen. Para ello implementa agentes basados en utilidad donde ésta se calcula teniendo en cuenta cada uno de los nodos en un árbol de decisión

Figura 4.5: Estructura del modelo de comportamiento de los vehículos propuesto por [Toledo et al., 2007]. Fuente: [Toledo et al., 2007].



#### 4.3 Modelado basado en inteligencia computacional

Desde mediados de los años 1990 empezó a crecer el interés por aplicar la *inteligencia artificial* a los *ITS*. Hay dos razones por las

que esto es así: la primera, los éxitos cosechados por la **inteligencia computacional**, los cuales atrajeron a investigadores de multitud de áreas incluida ésta. La segunda, el rápido desarrollo de la tecnología, que posibilita la existencia de conjuntos de datos masivos con la capacidad de explotarlos y aprender de ellos.

Algunos autores ([[Zhang et al., 2011](#)]) se atreven a afirmar incluso que el futuro de los **ITS** son las técnicas de la **inteligencia computacional**, y que los resultados que se puedan cosechar de técnicas basadas en el desarrollo convencional de sistemas es marginal comparado con las que se obtendrán con el nuevo enfoque.

Dentro de los **ITS**, las áreas de aplicación de la **inteligencia computacional** orientadas al conductor se centran en los siguientes conceptos: **reconocimiento de patrones, caracterización y modelado**. Aunque son áreas de aplicación distintas, los estudios en general tienden a solaparse. Por ejemplo, las técnicas de reconocimiento de patrones pueden usarse como forma de extracción de características para una caracterización de conductores y a su vez esta caracterización puede usarse como base para su modelado. (Figura 4.6). Por ello, aunque nuestro interés pueda centrarse en el modelado de conductores, es necesario conocer el estado de las demás áreas.

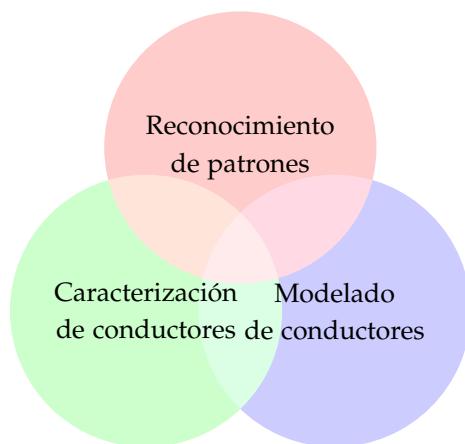


Figura 4.6: Principales áreas de aplicación de la **CI** en los **ITS**.

- En el **reconocimiento de patrones** las técnicas suelen trabajar en los temas de extracción de características y de predicción de comportamientos. La cantidad de datos que se pueden generar en un coche es tal que trabajar con la información en crudo es inviable (no digamos ya cuando los datos son extraídos de una flota de vehículos).
- La **caracterización de conductores** es interesante debido a que permite la identificación de perfiles de conducción y su clasificación de acuerdo a indicadores extraídos de su manera de conducir.
- El **modelado de conductores** nos permite la reproducción de comportamientos en simulación.

Las técnicas de **CI** más utilizadas en el modelado de conductores son

las **redes neuronales artificiales** y la **lógica borrosa**. Es comprensible dado que las primeras son una de las técnicas principales en la rama del **aprendizaje automático**, y la segunda por ser una manera sencilla y cercana a la manera de razonar del ser humano.

Sólo por la propia naturaleza de las técnicas, los modelos incluyen siempre más de un único comportamiento: al ser entrenados con datos de conductores reales y manejar la información con incertidumbre, “aprenden” a comportarse en diferentes regímenes (e.g. *free-flow* en *car-following*).

Las **redes neuronales artificiales** fueron el punto de entrada de la **inteligencia computacional** en la modelización de conductores. Los **perceptrones multicapa** y sus nuevas técnicas de entrenamiento se habían convertido en la nueva solución para todo aquel problema en el que hubiese disponible una colección de datos sobre los que entrenar estos modelos.

Las **redes neuronales artificiales** se han aplicado mucho sobre el campo de los **ITS** en general, no sólo en modelado de conductores sino en prácticamente todos los aspectos como la clasificación de conductores [Díaz Álvarez et al., 2014], los sistemas de conducción autónoma [Huval et al., 2015] o la predicción ([Dougherty et al., 1993, Chan et al., 2012, Naranjo et al., 2012] entre muchos otros.

El primer trabajo de la literatura sobre la aplicación de **red neuronal artificial** al modelado de conductores es [Fix and Armstrong, 1990], donde los autores desarrollan un controlador para imitar el comportamiento de un conductor.

En la misma década, en [Hunt and Lyons, 1994] se usaron **redes neuronales artificiales** aplicadas al entorno del vehículo para identificar el entorno y determinar cuándo y cómo se realiza un cambio de carril.

Los modelos hasta el momento hacen uso de datos extraídos de conductores reales pero desde entorno de simulación. El primer trabajo en usar datos reales de vehículos instrumentados para este cometido es [Jia et al., 2003]. A partir de las entradas correspondientes a velocidad relativa, espacio relativo, velocidad y velocidad deseada (para ello, clasifican al conductor de agresivo, normal, conservador) determinan la aceleración/deceleración del vehículo. No lo aplican a ningún simulador, sino que sólo comprueban que el modelo se ajusta a los valores esperados. Algunos trabajos en la misma línea que trabajan con arquitecturas diferentes son [Panwai and Dia, 2007, Khodayari et al., 2012], pero siempre sobre modelos longitudinales.

Otros trabajos, esta vez sin utilizar simuladores de tráfico, sino simuladores de conducción (concretamente **TORCS**) para la obtención de los datos reales son [Muñoz et al., 2010] y [Van Hoorn et al., 2009]. El primero modela el comportamiento del conductor con **perceptrones multicapa** entrenados con técnicas de *back-propagation*, mientras que el segundo utiliza técnicas de entrenamiento basadas en **algorit-**

mos genéticos multiobjetivo. Sin embargo, este tipo de modelos se encuentran más cercanos al nivel de control que al nivel táctico (ver Figura 4.1).

El trabajo de [Simonelli et al., 2009] también se apoya en datos extraídos de entornos reales. El interés de este estudio radica en que es el primero en realizar una comparativa entre el desempeño de una arquitectura *fast-forward* (e.g. *perceptrón multicapa*) frente a una *red neuronal recurrente* (e.g. *red de Elman*). El porqué del uso de *redes neuronales recurrentes* es por su capacidad de reconocer patrones dinámicos, los cuales son de esperar en este tipo de comportamientos.

El primer uso de la *lógica borrosa* fue contemporáneo al de las *redes neuronales artificiales*, y también aplicada a un modelo de *car-following*. Después de todo los modelos psico-físicos aparecieron debido a que la percepción del conductor no es absoluta, sino imprecisa.

Estos modelos parten de la hipótesis de que la información que maneja el conductor a la hora de tomar decisiones proviene de un análisis no demasiado detallado de la situación que le rodea; es decir, la percepción y el comportamiento humanos son estímulos percibidos de manera aproximada. Por tanto, el resultado debe ser fruto de un proceso de razonamiento que tenga en cuenta esa imprecisión en los estímulos, y la *lógica borrosa* es ideal para modelar la incertidumbre del mundo real y por tanto de las percepciones de los conductores.

[Kikuchi and Chakroborty, 1992] es el primer trabajo documentado sobre el tema. En él, los autores aplicaron la *lógica borrosa* sobre un modelo de aceleración de tipo *car-following*. Utilizaron el modelo **Gazis-Herman-Rothery (GHR)** (Ecuación 4.1) como base y determinaron las entradas al modelo como valores de pertenencia a conjuntos borrosa. Las entradas del modelo eran la distancia y velocidad relativa entre el vehículo modelado y el delantero y la variación en la aceleración del vehículo delantero. Como salida, el cambio en la tasa de aceleración sobre el vehículo modelado.

[McDonald et al., 1997, Wu et al., 2003] desarrollaron del modelo **Fuzzy Logic motorWay SIMulation (FLOWSIM)** con similares características pero incluyendo el comportamiento de cambio de carril, además en dos categorías: al carril **lento** (principalmente para evitar incordiar a los vehículos que se aproximan por detrás a velocidades superiores, usan dos variables, presión del vehículo trasero y satisfacción en el gap del carril destino) y al **rápido** (para ganar velocidad, variables: velocidad ganada con el cambio y oportunidad, es decir, seguridad y confort con el cambio). Otros trabajos en la línea son [Naranjo et al., 2006, Naranjo et al., 2007], donde los autores adaptan el comportamiento de un sistema basado en **FCS** a comportamientos como adelantamientos o situaciones tan extremas como un Stop&Go en autopista.

Los modelos hasta el momento se basaban en conjuntos borroso y reglas definidos *ad-hoc*. En [Chakroborty and Kikuchi, 2003] se intro-

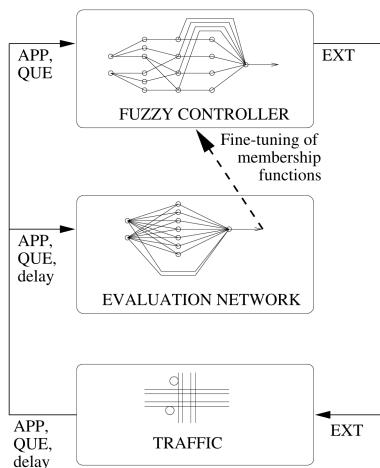


Figura 4.7: Un ejemplo de aproximación *neuro-fuzzy*, en este caso aplicado al control de señales de tráfico. El sistema de control borroso se implementa como una red neuronal artificial de tipo *feed-forward* en lugar de con una representación tradicional. Además, el sistema completo lleva integrado un subsistema basado en también en redes neuronales artificiales *feed-forward* que ajusta las funciones de pertenencia a través de entrenamiento por refuerzo. Fuente: [Bingham, 2001]

<sup>2</sup> Este trabajo se apoya en trabajos anteriores que separan las jerarquías en situaciones de MLC y DLC como [Yang and Koutsopoulos, 1996, Halati et al., 1997, Hidas, 2002]. Estos trabajos, no obstante, no pertenecen al área de la CI.

duce el concepto de personalización, ajustando los conjuntos borrosos de las variables de entrada y salida a valores extraídos de conductores reales. Este ajuste se realiza mediante la representación del sistema de control borroso como red neuronal artificial y posterior ajuste mediante entrenamiento de dicha red (*back-propagation*). A este trabajo le siguen muchas otras aproximaciones *neuro-fuzzy* como [Mbede et al., 2004, Zheng and McDonald, 2005].

Más adelante, en [Das and Bowles, 1999], dentro de su simulador Autonomous Agent SIMulation Package (AASIM) añaden los conceptos de MLC y DLC a los comportamientos de cambios de carril. En MLC las reglas tienen en cuenta la distancia al siguiente punto característico (e.g. una salida) y el número de cambios de carril necesarios. En DLC deciden si cambiar o no basándose en el nivel de satisfacción del conductor y en el nivel de congestión en los carriles adyacentes <sup>2</sup>.

Las redes neuronales artificiales y la lógica borrosa no son las únicas técnicas usadas para determinar comportamientos en conductores.

Por ejemplo, en [Maye et al., 2011] se presenta un modelo no supervisado, online y basado en redes bayesianas donde se infiere el comportamiento del conductor haciendo uso de una Intertial Measurement Unit (IMU) y una cámara. De la IMU se extraen datos que se separan en fragmentos para luego relacionarlos con las imágenes obtenidas de la cámara. Otro trabajo similar pueden ser [Van Ly et al., 2013] el cual se apoya también en la segmentación de los datos extraídos de una IMU pero con técnicas distintas (concretamente clustering basado en Support Vector Machines (SVMs) y en k-medias) y sin cámara.

En [Bando et al., 2013] describen otro modelo no supervisado, éste offline, basado en un modelo bayesiano no paramétrico para la clusterización, combinándolo con un modelo supervisado (Latent Dirichlet Allocation (LDA)) para la clusterización a más alto nivel. El trabajo de [Bender et al., 2015] usa una aproximación similar pero sin la segunda clusterización.

Otra aproximación es el de los Modelos Ocultos de Markov (HMMs, Hidden Markov Models). Por ejemplo, los trabajos [Kuge et al., 2000, Sekizawa et al., 2007] aplican entrenamiento supervisado sobre estos modelos para reconocer los eventos que están provocando los conductores (concretamente cambiando de un carril a otro). En [Hou et al., 2011] van un paso más allá desarrollando un modelo capaz de estimar si el conductor va a realizar un cambio a la derecha o a la izquierda a partir del ángulo de giro del volante (con una precisión de 0,95 en una ventana temporal de 1,5 segundos y de 0,83 en una ventana de 5 segundos).

Por último, [Aghabayk et al., 2013] presenta un modelo basado en LOcal LInear MOdel Tree (LOLIMOT) que son similares a una aproximación *neuro-fuzzy* del comportamiento. Intenta incorporar imperfecciones perceptuales en un modelo de *car-following*.

Estas técnicas no sólo se usan para modelar comportamientos complejos o modelos enteros. Algunos trabajos se ocupan de características o aspectos de un modelo concreto. Por ejemplo, los trabajos [Hatipkarasulu, 2002, Zheng et al., 2013] se ocupan exclusivamente del cálculo de tiempo de respuesta del conductor en modelos de *car-following*, el primero con **sistemas de control borroso** y el segundo con **redes neuronales artificiales**.



**PARTE II**

**MODELOS DE  
COMPORTAMIENTO**



## 5 Metodología

Los objetivos de la presente tesis se enfocan en el uso de técnicas pertenecientes al campo de la **inteligencia computacional** sobre datos de conductores reales para (i) incorporar comportamientos humanos en modelos de simulación (hipótesis H<sub>1</sub>) y (ii) que dichos comportamientos caracterizan a conductores concretos (hipótesis H<sub>2</sub>).

Para ello, se generarán modelos personalizados de conductor a partir de datos extraídos de entornos reales, se aplicarán dichos modelos al entorno de simulación multiagente **SUMO**, y por último se validarán los datos recogidos de los modelos generados contra los recogidos de los conductores reales. Es importante por tanto reparar en los siguientes detalles:

- Los datos extraídos del entorno real deben ser extraíbles también del entorno simulado y viceversa.
- Los parámetros a ajustar dependen completamente del entorno de simulación que son, en esencia, aceleración y cambio de carril.

Por tanto, los datos de entrada y de salida capturados del entorno real han de ser procesados para el entrenamiento de los modelos de tal manera que el modelo lo perciba lo más parecido posible a cómo los percibirá en el entorno de simulación.

En este capítulo se describe el esquema general del modelo a desarrollar, los entornos de obtención de datos, su posterior curación y los entornos de entrenamiento utilizados para la generación de los modelos.

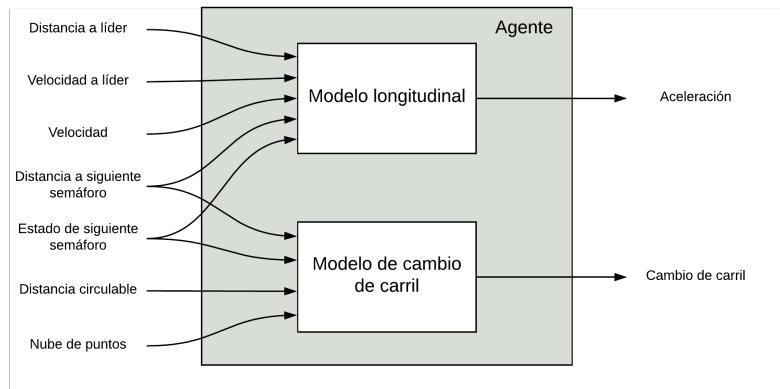
### 5.1 Perspectiva general del modelo de conducción

Como todo entorno de simulación, **SUMO** nos impone ciertas limitaciones a causa de su abstracción sobre el sistema a modelar en realidad. Una de las más decisivas es la del control lateral; aunque el simulador es continuo en el espacio, esta continuidad sólo existe en el desplazamiento longitudinal, en forma de distancia desde el comienzo del tramo. Esto produce que el comportamiento lateral sea modelado como saltos discretos entre carriles, sin ninguna continuidad durante el desplazamiento lateral.

Por ello, para el diseño y ajuste del modelo de conductor se ha decidido separar también los dos comportamientos principales que componen el nivel cognitivo táctico, es decir, un agente con un modelo de comportamiento que actúe sobre (i) la aceleración/deceleración, y (ii) la decisión y ejecución del cambio de carril, ambos a partir de los estímulos recibidos del exterior. Estos estímulos serán replicados en el simulador para que los agentes con los modelos entrenados a partir de estímulos del mundo real reciban estímulos similares en estos entornos.

Internamente, el modelo estará separado en los dos modelos básicos, el **modelo longitudinal** y el de **cambio de carril**. En la Figura 5.1 se describe el modelo, las entradas y salidas y el flujo de datos entre componentes.

Figura 5.1: Esquema general del modelo de conductor planteado en la tesis.



Estos modelos serán entrenados siguiendo un esquema supervisado, por lo que necesitamos datos reales a partir de los cuales se deberá ajustar su funcionamiento. La información que se considera suficiente para que los modelos desempeñen su función en el entorno de simulación es la siguiente:

- **Entorno.** El conductor, y por tanto el modelo, desarrolla su actividad y basa sus comportamientos, entre otros factores, en el entorno en el que se encuentra inmerso. Consideramos pues que el ajuste de la velocidad y sobre todo los cambios de carril, se ven influenciados por éste. Este entorno será representado como una nube de puntos alrededor del vehículo, como veremos posteriormente.
- **Velocidad actual del vehículo.** La velocidad del conductor influye en cómo va a alterar su velocidad en momentos posteriores. La velocidad máxima de la vía también influye, pero ésta se mantiene constante en el intervalo  $40 \text{ km h}^{-1}$  a  $50 \text{ km h}^{-1}$ , por lo que no se tiene en cuenta al considerarse una constante.
- **Distancia y diferencia de velocidad con el vehículo delantero.** Al igual que con la velocidad, el vehículo delantero juega un papel esencial en los cambios de aceleración. También se intuye que

puede influir en el comportamiento de cambio de carril en casos en los que el vehículo delantero circula muy lento.

- **Distancia circulable.** Es la forma de identificar en el simulador las siguientes salidas a tomar y los carriles cortados. Se considera que este tipo de información para cada carril es crucial a la hora de realizar cambios de carril, ya que además puede influir en otras maniobras tales como un adelantamiento.
- **Señales luminosas.** Es interesante contar con la información de este tipo de señales ya que pueden influir en diferentes patrones de aceleración o desaceleración. Otras señales también serían interesantes, pero el simulador en el momento del desarrollo de los modelos sólo ofrece el acceso a este tipo de señales.

## 5.2 Extracción de datos de conducción

Se ha hecho uso de un vehículo instrumentado, concretamente un Mitsubishi iMiEV (Figura 5.2).



Figura 5.2: El vehículo utilizado en los ensayos realizados.

Todos los dispositivos con los que se ha instrumentado el vehículo se conectan a un ordenador con sistema operativo GNU/Linux sobre Intel i7-7500U CPU con 16GB de memoria RAM. Al tener estos dispositivos funcionamientos muy diferentes, se ha optado por la creación de una aplicación basada en el framework ROS, del cual se da una breve visión general en el apéndice [Visión general de ROS](#).

### 5.2.1 Dispositivos usados para la captura de datos

A continuación se describen los dispositivos usados y su información generada.

**LiDAR** Se trata de un dispositivo que usa uno o más haces de luz pulsada para el cálculo de la distancia a los objetos donde éstos

impactan. Nuestro LiDAR en concreto modelo VLP-16 de la empresa Velodyne.

Está compuesto por un único haz láser con un movimiento continuo vertical y horizontal capturando información con una apertura de campo vertical de  $\pm 15^\circ$  distribuido a lo largo de 16 planos (dando como resultado una resolución fija de  $2^\circ$ ), y un campo horizontal de visión de  $360^\circ$  con una resolución dependiente de la velocidad de giro del láser, de  $0.1^\circ$  a  $0.4^\circ$  para 5 Hz a 20 Hz respectivamente. Estas medidas permiten una captura desde 75000 hasta 300000 puntos del entorno en función de la velocidad de giro a una distancia de hasta 100 m.

El acceso a la información se realiza a través del puerto Ethernet. Para la captura de datos se ha hecho uso del paquete de **Robot Operating System (ROS)** destinado para el trabajo con estos dispositivos denominado *velodyne\_pointcloud*<sup>1</sup>. Este nodo publica la nube de puntos en coordenadas cartesianas con el eje X definido en el sentido de los  $0^\circ$  y el eje Z en sentido ascendente.

El LiDAR instrumentado se encuentra anclado en la baca del vehículo con el eje X dispuesto en el sentido de conducción y el eje Z en sentido ascendente. Su posición es centrada en el vehículo y a una altura de 1.75 m.



Figura 5.3: LiDAR modelo VLP-16 de empresa Velodyne. Fuente: <http://velodynelidar.com/vlp-16.html>.

**GPS** El GPS es una tecnología usada para sincronización de tiempo y posicionamiento 3d (i.e. latitud, longitud y altitud) de objetos en el globo terráqueo. Es un dispositivo que funciona en un único sentido, es decir, el receptor recibe la información de los satélites, pero no envía ninguna.

El dispositivo GPS utilizado para los experimentos ha sido desarrollado en el Laboratorio de Electrónica e Instrumentación del **Instituto Universitario de Investigación del Automóvil (INSIA)**. Se trata de un dispositivo GPS con corrección diferencial que genera mensajes NMEA a una frecuencia de hasta 20 Hz. La posición de la antena receptora estará localizada en el centro del vehículo, justo debajo del LiDAR, situándose de esta forma en el origen de coordenadas del entorno capturado con éste último.

De todos los mensajes disponibles, los recuperados son los de los siguientes tipos:

- GGA. Geoposicionamiento del vehículo, el cual será usado para deducir valores no medibles directamente como, por ejemplo, la distancia al siguiente semáforo.
- VTG. Velocidad del vehículo, que se usará como el valor real de la velocidad tomada por el vehículo.

De la captura de datos se encargará el paquete de **ROS** denominado *nmea\_navsat\_driver*<sup>2</sup> el cual leerá por el puerto USB la información

originada en el [GPS](#).

**Bus CAN** Un bus es una topología caracterizada por tener un único canal de comunicaciones donde los dispositivos se conectan, vierten la información y la reciben. El Bus [CAN](#), o simplemente [CAN](#), es un protocolo de comunicaciones basado en esta topología utilizado, entre otras muchas áreas, para la comunicación entre los diferentes dispositivos que componen un vehículo.

El estándar [CAN](#) cubre únicamente las dos primeras capas del modelo [OSI](#)<sup>3</sup>, lo cual es suficiente para el acceso a la información del vehículo. Sin embargo, dependiendo del fabricante, los vehículos pueden contar con uno o más buses independientes, con acceso más o menos restringido y con identificadores de mensajes diferentes. Además, el acceso a esta información no suele ser accesible para el público en general, por lo que lo habitual, como ha sido en nuestro caso, es realizar una tarea previa de ingeniería inversa para identificar los mensajes correspondientes a los datos que se desean almacenar y su frecuencia de actualización.

El acceso a la información del bus se ha realizado a través del puerto USB. Para ello se ha utilizado el dispositivo CANUSB de la empresa LAWICEL AB, el cual se encarga de la traducción del protocolo CAN a una conexión estándar RS232. La captura de los paquetes se ha realizado a través de un nodo de [ROS](#) desarrollado para la elaboración de esta tesis.

Del [CAN](#) se ha recogido numerosa información, tanto para esta tesis como para más proyectos relacionados con ella. Para este experimento, no obstante, se ha utilizado únicamente la velocidad, y no directamente. La razón es que el [CAN](#) ofrece ésta como un número entero, por lo que los cambios de velocidad durante los recorridos ocurren de  $\pm 1 \text{ km h}^{-1}$ . Esta resolución es insuficiente para los cálculos de aceleración, por lo que se ha usado para validar la velocidad capturada por el [GPS](#)

<sup>3</sup> Las dos primeras capas son la capa física (la conexión el hardware a la red y la transmisión/recepción física de los datos) y la de enlace de datos (direcccionamiento físico, detección de errores y control de flujo). Sobre el resto de niveles no existen un estándar definido y por tanto existen múltiples protocolos de estas dependiendo del fabricante u organismo que lo implemente.

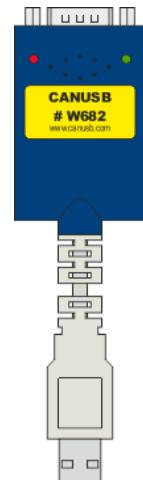


Figura 5.4: El dispositivo CANBUS de LACIWEL AB permite el acceso a través del protocolo RS 232 por el puerto USB al bus Controller Area Network (CAN).  
Fuente: <http://www.can232.com/>.

**Cámara** Para la obtención de algunos valores de indicadores derivados de los obtenidos directamente, necesitamos un proceso posterior de análisis sobre el recorrido, asociando las imágenes a los valores recogidos del resto de dispositivos, como posiciones o nube de puntos del [LiDAR](#). Para ello se ha hecho uso de una cámara que permite la visualización del recorrido desde el punto de vista del conductor.

La cámara utilizada es un Microsoft Kinect localizada tras el retrovisor interior del vehículo. Dicha cámara está orientada de tal manera que ofrece una visualización de la vía por la que circula el vehículo.

Entre otras capacidades, la cámara ofrece captura de imágenes VGA de  $640 \text{ px} \times 480 \text{ px}$  de resolución y a una frecuencia de 30 Hz. El dispositivo se conecta vía USB y de la captura se encarga un nodo de



Figura 5.5: La cámara Kinect desarrollada por Microsoft ofrece imágenes a color a una velocidad de 30 fps con una resolución de  $640 \text{ px} \times 480 \text{ px}$ .

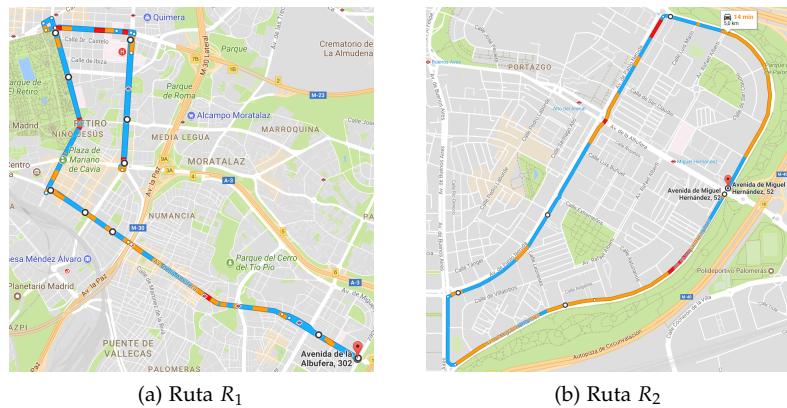
<sup>4</sup> [http://wiki.ros.org/freenect\\_stack](http://wiki.ros.org/freenect_stack).

ROS denominado *freenect*<sup>4</sup> que, entre otras cosas, publica las imágenes tomadas a dicha frecuencia.

### 5.2.2 Selección de rutas

Para la captura de datos se han propuesto dos rutas, en adelante  $R_1$  y  $R_2$ , consideradas equivalentes al tratarse de vías en entorno urbano, con tramos de entre uno y tres carriles a lo largo de su recorrido y con velocidades máximas establecidas entre los  $30 \text{ km h}^{-1}$  y los  $50 \text{ km h}^{-1}$ . La Figura 5.6 muestra estos recorridos en el mapa.

Figura 5.6: Mapa de los dos recorridos planteados para la captura de datos de conducción.



$R_1$  tiene una duración de recorrido estimada de 30 min y se utilizará como fuente de datos destinada al entrenamiento del modelo (conjuntos de entrenamiento y de validación).  $R_2$  por su lado tiene un tiempo estimado de recorrido de 15 min y sus datos tienen el propósito de servir de conjunto de test. Ambas fueron realizadas entre las 11:00 am y las 12:00 pm en días laborables, permitiendo una circulación con suficientes vehículos para requerir maniobras dependientes del entorno, pero sin demasiados como para impedir la circulación.

### 5.2.3 Selección de sujetos

Se han elegido un total de tres sujetos para los experimentos. Los tres pertenecen al grupo especificado en los supuestos del capítulo [Introducción](#).

Los sujetos tienen experiencia de conducción y han realizado el recorrido anteriormente a fin de basar sus comportamientos lo más posible al nivel táctico de conducción y disponer de una mayor libertad de acción. La razón es la siguiente; los comportamientos longitudinal (*car-following*) y lateral (*lane-change*) se asocian con el nivel cognitivo táctico.

Por un lado, al conocer los recorridos de antemano el comportamiento de planificación ya se ha cumplido. Por otro, no necesitan estar pendientes de las decisiones de un operador, ya que pueden limitar ciertas maniobras al no saber cuál será la siguiente ordenada por éste.

### 5.3 Preparación de los datos

Tras la captura se ha realizado una secuencia de pasos para dejar los datos preparados para el proceso de entrenamiento de los modelos. El resto de la sección detalla cada uno de éstos.

#### 5.3.1 Fusión de sensores

Como hemos visto anteriormente, cada uno de los dispositivos ofrece sus datos a una tasa de frecuencia diferente (con excepción del bus CAN, en el cual los datos van a frecuencias diferentes).

El primer paso en la preparación de los datos ha sido el de la fusión de estos. Afortunadamente cada uno de los mensajes almacenados que llegan desde nodos de ROS vienen con una marca temporal que podemos suponer sincronizada entre nodos de la misma aplicación. Por ello, la fusión se ha realizado en dos pasos:

1. Cálculo del primer elemento a fusionar de cada uno de los conjuntos de datos. Para ello, se han desecharido todos los primeros valores hasta encontrar la primera tupla de valores (uno por cada conjunto de datos) donde éstos están más próximos entre si.
2. Extracción iterativa de las tuplas más aproximadas. Iterativamente, se han ido extrayendo las tuplas más próximas a cada uno de los incrementos de la tasa deseada de sincronización  $f$ , siempre y cuando se encuentren dentro del intervalo  $f \pm \frac{f}{2}$ .

Este proceso se ha repetido con cada uno de los conductores para cada uno de los recorridos, dando como resultado seis conjuntos sincronizados con los datos en bruto sincronizados.

#### 5.3.2 Extracción de variables no observables directamente

Existen una serie de variables cuya extracción directa del entorno no es trivial. Ésta es una de las razones por las que se ha capturado con la cámara la visión del conductor en el recorrido.

El proceso de obtención ha sido manual, obteniendo las marcas temporales de las variables a capturar tras el visionado de las imágenes capturadas por la cámara. Estas variables son las siguientes:

*Cambio de carril* Se han identificado los cambios de carril como +1 si es hacia la izquierda o -1 si es hacia la derecha (el resto 0). Las marcas temporales de los cambios son aquellas desde el comienzo de la maniobra del cambio de carril hasta que el vehículo ha llegado a la mitad del cambio.

*Distancia y estado de semáforos* La distancia al semáforo ha sido obtenida a partir de la distancia euclídea entre la geoposición del origen de coordenadas y la del semáforo, por lo que es esperable cierto margen de error. Los estados se han extraído directamente del visionado de las imágenes, tomando este los valores  $g$ ,  $y$  y  $r$  dependiendo de si el semáforo se encuentra en verde, ámbar o rojo respectivamente.

*Distancia circulable en carriles* Al igual que con la distancia a los semáforos, ésta se ha calculado a partir de la distancia euclídea de la geoposición del origen de coordenadas a los puntos a partir de los cuales no se puede continuar por el recorrido especificado. Las distancias obtenidas se corresponden al carril izquierdo, al actual y al derecho.

*Distancia al obstáculo más cercano* Para el cálculo de esta variable, se ha procedido a capturar una región de interés de cada una de las nubes de puntos dentro de la cual identificar los posibles obstáculos existentes. Por la posición del LiDAR se ha decidido que ésta está acotada entre los intervalos  $(0,35,35)$ ,  $(-1,1)$  y  $(-1,5,0,5)$  para los ejes X, Y y Z respectivamente.

<sup>5</sup> DBSCAN [Ester et al., 1996] es un algoritmo de clusterización que identifica un número variable de conjuntos en un espacio  $n$ -dimensional.

Funciona a partir de la agregación de puntos en función de sus parámetros  $\epsilon$  y  $\mu$ .  $\epsilon$  es la distancia mínima a la que se deben encontrar dos puntos para considerarse pertenecientes al mismo clúster mientras que  $\mu$  determina el número mínimo de puntos que debe tener un clúster para ser considerado como tal.

Posteriormente, para la nube de puntos resultante se ha realizado un proceso de clusterización aplicando el algoritmo DBSCAN<sup>5</sup> con parámetros  $\epsilon = 0,5$  y  $\mu = 3$ . Este proceso identifica un número variable de clústers, tras el cual nos quedamos con el más cercano al vehículo.

Posteriormente y de forma manual, se ha realizado el recorrido mostrando las nubes de puntos correspondientes a las capturas superponiendo el centroide para eliminar los de aquellos frames que se corresponden con errores. De los restantes se ha calculado una distancia euclídea al origen de coordenadas.

### 5.3.3 Curación de datos

Tras obtener todas las variables principales, ya sean directamente de los sensores del coche o a través de un proceso manual generamos los conjuntos de datos para los comportamientos longitudinal y de cambio de carril. La tabla 5.1 describe qué variables son usadas en qué conjunto de datos.

## 5.4 Entrenamiento de modelos

Para los entrenamientos de los modelos se ha utilizado una máquina con procesador Intel®Core™i7-6700K a 4.00 GHz y 16 GiB de memoria. Los entrenamientos se han realizado sobre una GPU Titan XP cedida a nuestro grupo de investigación por parte de NVIDIA. El

Tabla 5.1: Resumen de los indicadores obtenidos tras los recorridos. Éstos incluyen tanto la información extraída directamente como aquella que ha requerido proceso manual. La parte inferior de la tabla describe las variables a predecir en los problemas de modelo longitudinal y de cambio de carril. Las variables “Distancia circulable” en realidad es una tupla que indica las distancias circulables en los carriles izquierdo, actual y derecho.

Variable	Longitudinal	Cambio de carril
Distancia circulable	✗	✓
Distancia al líder	✓	✗
Distancia a siguiente TLS	✓	✓
Estado de siguiente TLS	✓	✓
Nube de puntos	✗	✓
Velocidad	✓	✗
Velocidad al líder	✓	✗
Aceleración	✓	✗
<b>Cambio de carril</b>	✗	✓

sistema operativo utilizado ha sido un Debian GNU/Linux versión 9.4.

Para cada uno de los submodelos se han probado dos aproximaciones diferentes para comparar sus desempeños en las tareas por separado. Concretamente se han utilizado **FCSs** y **MLPs** para modelar el comportamiento longitudinal y **MLPs** y **CNNs** para modelar el comportamiento en cambio de carril.

Los modelos serán entrenados con un conjunto por tipo de modelo (longitudinal y cambio de carril) asociado al total de sujetos. Con estos modelos generales se comprobará si es posible aprender características de conducción de conductores y con qué técnicas y arquitecturas. Para validar este modelo se comprarán los datos reales de los conjuntos de test con datos de simulación generados en un entorno similar<sup>6</sup>.

Una vez decididas las arquitecturas de los modelos que se usarán para cada tipo, serán entrenadas con los datos de cada conductor en concreto. A partir de estos datos se determinará si los rasgos aprendidos se consideran suficientes para caracterizar a cada conductor por separado.

<sup>6</sup> Se ha valorado el uso de otros marcos de validación como el *KITTI Vision Benchmark Suite*. El problema es que la suite KITTI está orientada al análisis del entorno del vehículo y no al de la personalización, por lo que no tenemos manera de saber qué partes del conjunto pertenecen a unos u otros conductores.



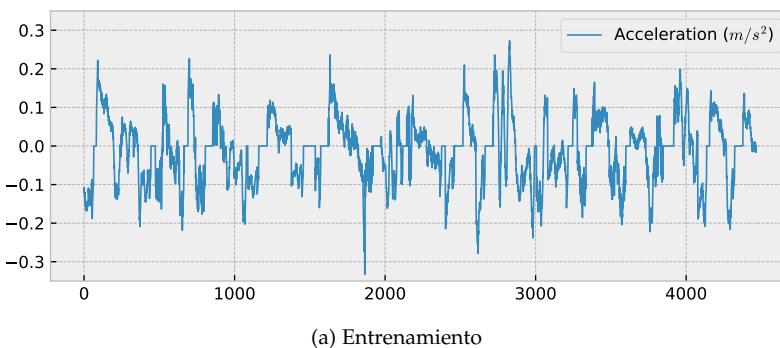
## 6 Comportamiento longitudinal

El comportamiento longitudinal es un problema de regresión sobre cómo ha de comportarse la aceleración del DVU en función de la información existente alrededor. El perfil de aceleración para el conjunto general de conductores tanto del conjunto de test como del de entrenamiento se ilustran en la figura 6.1.

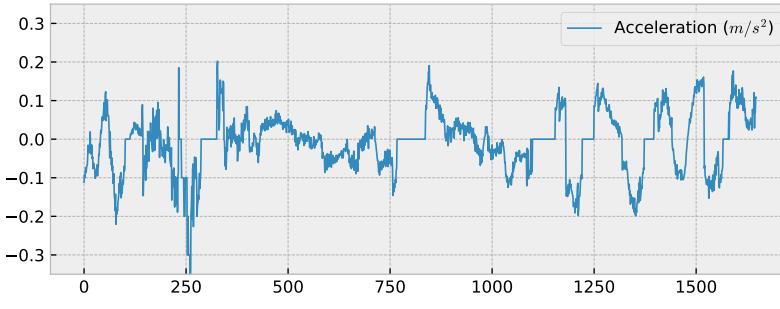
Por la naturaleza del problema se han seleccionado dos técnicas diferentes para el ajuste del modelo:

1. **MLP.** Al ser un problema de regresión, el uso de **perceptrones multicapa** en el comportamiento longitudinal está justificado a ser considerados éstos aproximadores universales [Hornik, 1991]<sup>1</sup>
2. **FCS.** La formulación de este problema se ajusta muy bien al funcionamiento de estos modelos, donde en función de las entradas y de acuerdo a una serie de reglas, el controlador toma una decisión para la salida. Además, los **sistemas de control borroso** tienen la

<sup>1</sup> El *Teorema de aproximación Universal* postula que un **MLP** con al menos una capa oculta es capaz de aproximar cualquier función si dispone de suficientes neuronas en ésta.



(a) Entrenamiento



(b) Test

Figura 6.1: Perfiles de aceleración durante los experimentos: (a) entrenamiento y (b) test. Estos son los conjuntos de datos que se intentarán ajustar con los modelos.

ventaja de que se puede explicar cómo funcionan, cosa que no es posible para un **perceptrón multicapa** con una o más capas ocultas.

Ambos modelos se ajustarán con un procedimiento basado en el descenso del gradiente denominado ADAM [Kingma and Ba, 2014]. Su aplicación a los **perceptrones multicapa** es directa (después de todo es una técnica desarrollada dentro de área de las **redes neuronales artificiales**), pero para los **sistemas de control borroso** es necesaria una representación que permita el uso de este método para su optimización. Esta representación es una de las aportaciones de esta tesis y se explica en el apéndice **Ajuste de sistemas de control borroso basado en descenso del gradiente**.

El error que se trata de minimizar es el cuadrado del **RMSE** entre los valores de aceleración del conjunto reales y los ajustados por el modelo, ya que es equivalente. Los resultados no obstante se presentan con el **RMSE** ya que éste es directamente interpretable en las medidas de la variable sobre la que se aplica (en nuestro caso,  $\text{m s}^{-2}$ ).

#### 6.0.1 Descripción de los datasets

Del conjunto de datos descrito en la Tabla 5.1 del capítulo **Metodología** seleccionamos aquellos indicadores de interés: *Distancia a líder*, *Distancia a siguiente TLS*, *Estado de siguiente TLS*, *Velocidad*, *Velocidad a líder* y *Aceleración* (este último como salida a ajustar por el modelo). Estos valores conformarán los conjuntos de entrenamiento y de test tanto para cada uno de los sujetos  $S_1$ ,  $S_2$  y  $S_3$ , como para el conjunto global de éstos  $S_A$ .

Dado que nuestros modelos se basan en un esquema *feed-forward*, existe el inconveniente de que para ellos es imposible mantener una memoria del orden en el que se están sucediendo las entradas. Sin embargo, contamos con las derivadas de la posición respecto al líder (la aceleración y la velocidad al líder) y la velocidad, por lo que consideramos que disponemos de información temporal suficiente para este problema en concreto.

El resultado de la generación de los datasets de entrenamiento y test para todos los sujetos se resume en la Tabla 6.1

Tabla 6.1: Descripción de los conjuntos de datos para el entrenamiento de los modelos.  $CF_{S_A}$  se corresponde con el modelo de conductor global, mientras que cada  $CF_{S_i}$  es la porción de datos correspondiente al sujeto  $S_i$ .

			Tamaño	
	Entradas	Salidas	Entrenamiento	Test
$CF_{S_1}$	7	1	1089	543
$CF_{S_2}$	7	1	1313	560
$CF_{S_3}$	7	1	2067	668
$CF_{S_A}$	7	1	4469	1771

## 6.1 Modelo basado en sistemas de control borroso

Se han realizado entrenamientos sobre arquitecturas con diferente número de particiones borrosas en las variables. Las arquitecturas que se han considerado más relevantes (tras un proceso de ensayo y error con diferente número de particiones borrosas) se describen en la Tabla 6.2.

	Arquitectura	Epochs	Entrenamiento	RMS	
				Validación	Test
$FCS_1$	2, 2, 2, 2, 2, 2	$10^5$	0,059	0,064	0,062
$FCS_2$	3, 3, 2, 2, 3, 3	$10^5$	0,073	0,079	0,080
$FCS_3$	4, 3, 2, 2, 3, 3	$10^5$	0,072	0,078	0,088
$FCS_4$	5, 5, 2, 2, 5, 5	$10^5$	0,063	0,068	0,109

Todas las arquitecturas mantienen el número de conjuntos borrosos asociados a las variables de estado del semáforo (verde, ámbar y rojo) a 2. La razón es debido a que dicha variable sólo puede tomar dos valores, 0 o 1, y en la inicialización de sus respectivas particiones, estas variables ya toman dichos valores con un grado de pertenencia de 1 en sus dominios.

Cada uno de los controladores se ha entrenado durante 250000 epochs. En un principio, el proceso de entrenamiento fue el habitual, ajustando todas las variables a la vez (particiones borrosas y pesos asociados a las reglas).

Sin embargo, tras varias pruebas, se ha comprobado que el ajuste de las variables que determinan las particiones borrosas parece suceder más rápido que el ajuste de los pesos asociados a las reglas. Por ello, en lugar de entrenar a la vez, se ha particionado el entrenamiento en secuencias sucesivas de entrenamiento de reglas y entrenamiento de particiones borrosas.

Concretamente, los 250000 epochs de entrenamiento han sido divididos en 250 iteraciones de:

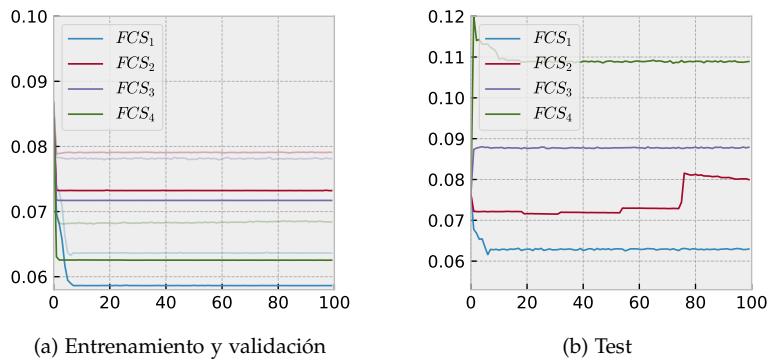
1. 800 epochs ajustando sólo las reglas con una tasa de entrenamiento de 0,01.
2. 200 epochs ajustando sólo las variables de las particiones borrosas con una tasa de entrenamiento de 0,001.

Empíricamente (y para este problema en concreto) se ha podido observar que el entrenamiento realizado de esta manera hace que el RMSE descienda más rápido en el mismo número de iteraciones.

En la Figura 6.2 se puede observar la evolución en general y un detalle de la disminución del error durante el entrenamiento de los controladores. El error en test se ofrece a título informativo, y no ha sido usado para determinar las arquitecturas seleccionadas.

Tabla 6.2: Resumen de las arquitecturas seleccionadas. La posición de cada número de la arquitectura indica a qué variable lingüística se refiere (*Distancia al líder*, *Distancia a siguiente TLS*, *TLS en verde*, *TLS en amarillo*, *TLS en rojo*, *Velocidad* y *Velocidad de aproximación al líder* respectivamente). Su valor se corresponde al número de conjuntos borrosos de la partición de cada una de ellas.

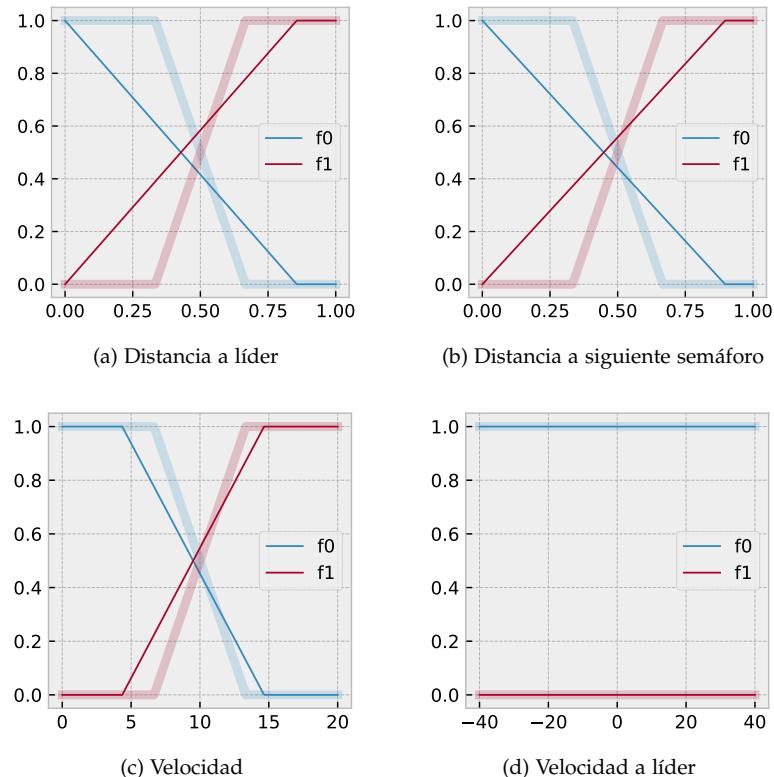
Figura 6.2: Visión en detalle de la evolución del error en los conjuntos de entrenamiento y validación (izquierda) y de test (derecha). Para cada arquitectura, el color más transparente se corresponde al error en el conjunto de validación. El error de test se muestra debido a que nos ofrece puede ofrecer intuición de qué forma aprende la red, pero su evolución no se ha considerado para determinar las arquitecturas.



Se puede observar que la arquitectura  $FCS_1$  es la que mejor error en test ha alcanzado. En las pruebas realizadas para el ajuste de controladores se ha observado además que los errores en entrenamiento y en test tienden a ser similares cuando los problemas son representables con un sistema de control borroso.

Éste será por tanto, el modelo seleccionado para la comparativa final. Las funciones de pertenencia ajustadas se muestran en la Figura 6.3 contrastadas con las versiones de antes de comenzar el entrenamiento. Las reglas, sin embargo, son muy numerosas para ser incluidas en el texto, ya que la mayoría de reglas posibles en el sistema (aproximadamente 60) tienen un peso asociado de más de 0,5.

Figura 6.3: Particiones borrosas después de la operación de ajuste en el modelo longitudinal basado en  $FCS_s$ , donde  $f_i$  se corresponde con la función de pertenencia que caracteriza al conjunto borroso  $i$ -ésimo de la variable. Las funciones de pertenencia correspondientes al estado del semáforo no se incluyen ya que no se vieron modificadas en ningún entrenamiento. Después de todo, estas variables eran binarias, y las particiones borrosas se inicializan de tal manera que dichos valores siempre alcanzan un valor de pertenencia 1. El de la variable *Velocidad a líder* es cuanto menos, curioso. Tras varias ejecuciones sobre esta y otras arquitecturas, ha mantenido este comportamiento, por lo que intuimos que la variable tiene un efecto mínimo o nulo sobre el comportamiento en el modelo longitudinal.



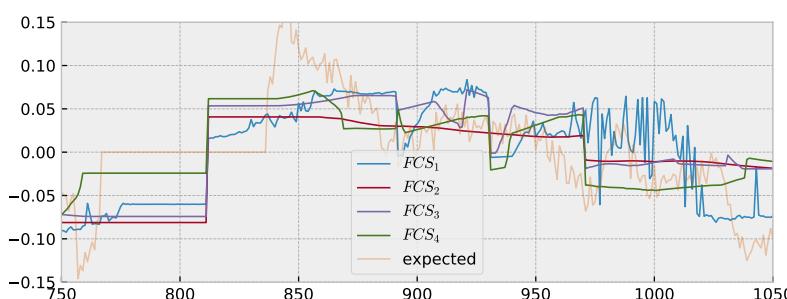
Un detalle que merece la pena destacar es el aprendizaje de la variable lingüística *Velocidad a Líder*. En todos los entrenamientos conducidos con las arquitecturas de la Tabla 6.2 el comportamiento ha sido el

mismo. Esto nos hace pensar que esta variable tiene un efecto mínimo o nulo sobre el comportamiento del modelo longitudinal.

En la Figura 6.4 se muestran los perfiles de la aceleración estimada por los controladores frente la aceleración real en cada momento.



(a) Perfil de aceleración



(b) Detalle entre frames de 750 a 1050

Figura 6.4: Comparación del perfil de aceleración real y el inferido por los modelos entrenados. En la visión general se puede observar el perfil real en color semi-transparente. Debajo, en el perfil, se amplia una pequeña sección del perfil para mostrar los diferentes ajustes de los modelos entrenados y cómo difieren del valor real.

En el detalle podemos ver de qué forma está ajustando el controlador con menor error en test frente al resto. Mientras que el  $FCS_1$  presenta picos de variación que aproximan el valor inferido al valor real, el resto de controladores mantiene valores constantes separados en escalones.

## 6.2 *Modelo basado en perceptrones multicapa*

Para determinar el modelo óptimo de **perceptrón multicapa** en comportamiento longitudinal, se han realizado entrenamientos sobre arquitecturas con diferente cantidad de neuronas y capas ocultas.

Las arquitecturas más ilustrativas de todas las probadas se resumen en la tabla 6.3.

Topología	Epochs	RMS			Test
		Entrenamiento	Validación	Test	
$MLP_1$	16	$10^5$	0,057	0,057	0,069
$MLP_2$	8,2	$10^5$	0,061	0,061	0,056
$MLP_3$	16,8	$10^5$	0,051	0,051	0,060
$MLP_4$	16,16,8	$10^5$	0,046	0,046	0,061

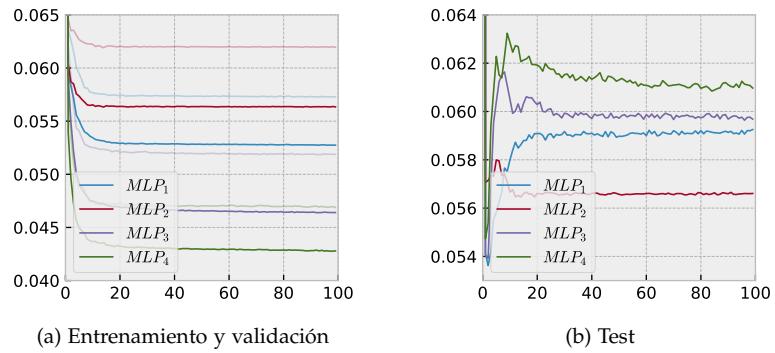
Tabla 6.3: Resumen de las arquitecturas de **perceptrones multicapa** para el modelo longitudinal. La posición de cada número de la topología indica el índice de la capa oculta y su valor el número de nodos (neuronas) que incluye dicha capa. Las arquitecturas seleccionadas en esta tabla son aquellas consideradas relevantes tras un proceso manual de ensayo y error.

<sup>2</sup> Se han utilizado también funciones de activación de tipo ReLU, pero las tasas de error tras el entrenamiento eran notablemente más altas por lo que se ha optado al final por el uso de activación basada en tangente hiperbólica.

El modelo de funciones de activación que se ha utilizado es de tipo tangente hiperbólica en todas las neuronas salvo en la última, que se ha utilizado una activación lineal <sup>2</sup>. Los pesos de la red han sido inicializados con una muestra aleatoria uniforme de valores reales en el intervalo  $(-0.25, 0.25)$ .

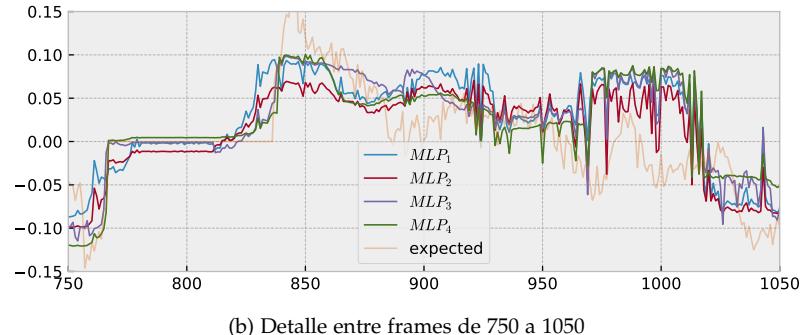
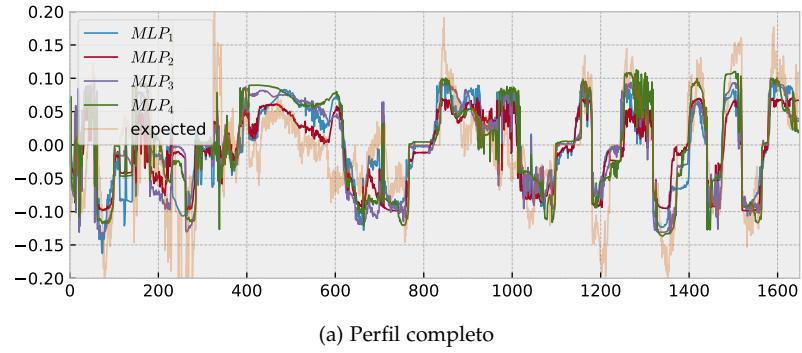
La Figura 6.5 muestra la evolución del RMSE durante el proceso de entrenamiento. También se ilustra el error de test durante el mismo, aunque éste no se ha utilizado para decidir las arquitecturas y es simplemente informativo.

Figura 6.5: Visión en detalle de la evolución del error en los conjuntos de entrenamiento y validación (izquierda) y de test (derecha). Para cada arquitectura, el color más transparente se corresponde al error en el conjunto de validación. El error de test se muestra debido a que nos ofrece puede ofrecer intuición de qué forma aprende la red, pero su evolución no se ha considerado para determinar las arquitecturas.



Estos errores se encuentran entre los  $0.05 \text{ m s}^{-2}$  y los  $0.07 \text{ m s}^{-2}$ , lo cual consideramos que es una aproximación aceptable. Una particularidad del problema ha sido la inestabilidad de los entrenamientos, esto es, la alta sensibilidad a los valores de inicialización de los parámetros. La intuición tras ver la evolución de los entrenamientos es que la función de error sufre de muchos mínimos locales o mesetas.

Figura 6.6: Comparación del perfil de aceleración real y el inferido por los modelos entrenados. En la visión general se puede observar el perfil real en semi-transparente. En el detalle, se amplía una pequeña sección del perfil para mostrar los diferentes ajustes de los modelos entrenados y cómo difieren del valor real.



Al contrastar los errores de test, podemos determinar que la arquitectura que parece que mejor generaliza es la  $MLP_2$  (arquitectura 7,8,2,1).

Una visión de detalle del ajuste de estas arquitecturas al conjunto de test se puede ver en la figura 6.6, donde se muestra el perfil de aceleración del conjunto de test y los perfiles de aceleración de las redes entrenadas.

A la vista de los resultados, y dado que las arquitecturas se ajustan bien a los datos reales, es razonable elegir el modelo  $MLP_2$  debido a que es el que aparentemente mejor generaliza los comportamientos del conjunto de conductores.

### 6.3 Comparación entre modelos

Las mejores arquitecturas de ambos modelos han sido la  $MLP_2$  para los **perceptrones multicapa** y  $FCS_1$  para los **sistemas de control borroso**. Los errores y el perfil de aceleración para ambos modelos se muestran en la Figura 6.7.

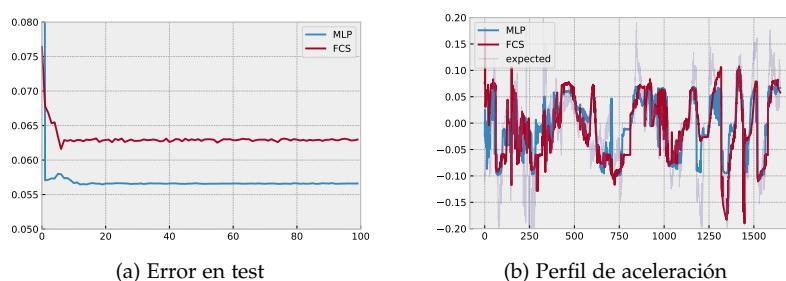


Figura 6.7: Comparación de la mejor arquitectura de **perceptrones multicapa** frente a la mejor arquitectura de **sistemas de control borroso**: (a) diferencia entre los errores cuadráticos medios de ambas arquitecturas y (b) perfiles de aceleración en el conjunto de test.

Aunque el modelo basado en **sistemas de control borroso** arroja un error bajo en test y parece que tiende a ajustarse al perfil de aceleración, parece que el problema es suficientemente complejo como para no poder representarse como un simple **sistema de control borroso**.

Además, el error arrojado por el **perceptrón multicapa** es sustancialmente menor y, por tanto, la arquitectura elegida para el modelo longitudinal será la  $MLP_2$ .



## 7 Comportamiento lateral

El problema del cambio de carril determina cuándo y cómo realiza los cambios de carril un conductor en un momento determinado. A priori la intuición sobre el problema es que muchos de los factores determinantes no son medibles en el mundo real y/o en el entorno simulado (e.g. estados de humor, condición física, eventos fortuitos, etcétera).

En un trabajo anterior [Díaz Álvarez et al., 2018] los autores tratamos de controlar muchos de estos factores dividiendo el problema en dos partes, la intención de cambio y la ejecución del cambio, y fijando el primero de tal manera que los conductores sólo cambiaban de carril cuando se les ordenaba. De esta manera se permitía el estudio de cómo diferentes perfiles de conductor ejecutaban de distintas formas los cambios de carril y de cómo este fenómeno es modelable.

En este trabajo se tratará de modelar, sin embargo, el proceso de cambio de carril como un todo, esto es, decidir en cada momento si cambiar a un carril (i.e. izquierda o derecha) o no hacerlo, a partir de las variables medibles del entorno real y simulado. El problema a resolver será, por tanto, de clasificación, donde se tratará de maximizar el número de aciertos entre los cambios de carril realizados y el modelo a ajustar. Las técnicas que se han seleccionado para tratar el cambio de carril son las siguientes:

1. **MLP.** Los **perceptrones multicapa**, más ahora en la época del *deep-learning* son una de las técnicas más usadas para problemas de clasificación. En esta época del *deep-learning* han aumentado su eficiencia varios órdenes de magnitud en capacidad de aprendizaje y, por tanto, en rendimiento a la hora de clasificar.
2. **CNN.** Otro de los grandes exponentes a la hora de clasificar, sobre todo trabajando con mapas de características  $n$ -dimensionales con las **redes convolucionales**. Al representar la evolución temporal del entorno circundante como una dimensión más en el espacio presentado a los modelos, podemos aplicar esta técnica para la identificación de características de manera, supuestamente, más eficiente.

Ambos modelos, al igual que con el modelo longitudinal, han sido entrenados ajustando sus parámetros con el algoritmo ADAM

<sup>1</sup> La inicialización clásica, de valores uniformes en un intervalo pequeño alrededor de 0 haría que la mitad de los valores cayesen por debajo de 0. Si la neurona tiene una función de activación de tipo ReLU, el peso no se vería modificado, ya que su derivada será siempre 0.

<sup>2</sup> Esta inicialización se basa en intentar mantener la misma desviación típica de gradientes en cada capa de la red. En [Glorot and Bengio, 2010] determinan que ésta debe ser  $\sigma^l = \frac{2}{\text{card}(W_{in}^l) + \text{card}(W_{out}^l)}$ , siendo  $\text{card}(W_{in}^l)$  y  $\text{card}(W_{out}^l)$  el número de entradas y de salidas de la capa, extrayendo posteriormente sus valores de una distribución aleatoria de la forma  $X^l \sim \mathcal{N}(\mu = 0, \sigma = \sigma^l)$

<sup>3</sup> La normalización usada ha sido la operación denominada softmax, definida como:

$$\text{softmax}(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (7.1)$$

[Kingma and Ba, 2014]. Las funciones de activación, sin embargo, varían, y por tanto la inicialización de las variables también <sup>1</sup>.

En este caso, ambos modelos hacen uso de neuronas de activación de tipo ReLU, salvo en la última capa que la activación es lineal. Para inicializar los pesos de las variables se hace uso del algoritmo Glorot [Glorot and Bengio, 2010] (también denominado Xavier) debido a su mejor desempeño, sobre todo en redes con funciones de activación tipo ReLU [Glorot et al., 2011] <sup>2</sup>.

Al tratarse además de un problema de clasificación donde las clases son mutuamente excluyentes, a la capa de salida de las redes se le ha aplicado una normalización <sup>3</sup> para transformar el vector de salida en un vector de probabilidades, siendo el cambio de carril seleccionado la salida con el valor de probabilidad más alto.

Para el cálculo del coste, en lugar de intentar maximizar la cantidad de aciertos o minimizar el número de falsos positivos o negativos, el error que se ha tratado de minimizar es la entropía cruzada, concepto del cual hablamos anteriormente en la sección 2.3.3 del capítulo Inteligencia Computacional. Este indicador es mucho más eficiente a la hora de trabajar con problemas de clasificación. Sin embargo, de cara a mostrar resultados en la tesis, el valor de ésta no ofrece información relevante más allá de “si decrece es bueno”. Por ellos se ofrecerán los valores de precisión y matrices de confusión que son más acordes con el problema en cuestión.

Antes de pasar a la descripción de los modelos, queda hablar del sesgo existente en los datos de entrenamiento. Debido a la naturaleza el problema, el número de ejemplos existentes de cambios de carril es significativamente menor al existente de no cambios de carril. Este sesgo hace que los modelos se entrenen rápidamente para marcar todos los ejemplos como “no cambio”, dificultando el ajuste posterior hacia cambios a izquierda o derecha.

Por tanto, debido a que (i) las limitaciones de la máquina no nos permiten operar con los conjuntos completos en cada epoch y (ii) existe un sesgo hacia predicciones de “no cambio”, en cada uno de los epochs, se usará un batch tamaño  $m = 6,000$  compuesto por una selección aleatoria de todos los ejemplos equidistribuidos entre las clases del problema (esto es, aproximadamente  $\frac{m}{3}$  para cada clase “cambio izda.”, “no cambio” y “cambio dcha.”).

## 7.1 Descripción de los datasets

Del conjunto de datos descrito en la Tabla 5.1 del capítulo Metodología seleccionamos aquellos indicadores de interés: Distancia circulable, Distancia a siguiente TLS, Estado de siguiente TLS, Nube de puntos y Cambio de carril (este último como salida a clasificar por el modelo). Estos valores conformarán los conjuntos de entrenamiento y de test tanto

para cada uno de los sujetos  $S_1$ ,  $S_2$  y  $S_3$ , como para el conjunto global de éstos  $S_A$ .

### 7.1.1 Representación de los datos

Las secuencias de las que se compone el conjunto de datos son, aparte de variables numéricas para cada indicador no espacial, una representación del entorno del vehículo como nube de puntos. Los límites técnicos y la representación en sí implican dos problemas principales:

1. Los modelos que utilizamos en esta tesis se basan en un número fijo de entradas, y la nube de puntos contiene un número variable de éstos, dependiendo del número de obstáculos y su distancia a éstos.
2. La nube de puntos se origina a través de un dispositivo mecánico que funciona con coordenadas esféricas a una resolución horizontal de  $0.2^\circ$  y vertical de  $2^\circ$  con una tasa de refresco de 10 Hz. Esto implica que la superficie del sector circular que no se cubre a largas distancias sea muy extenso, por lo que el espacio según nos alejamos del origen va siendo cada vez más disperso.

Para el primer caso, se ha optado por representar el entorno como un mapa de profundidad, ilustrado en la Figura 7.1. Un mapa de profundidad representa el entorno como una imagen de un sólo canal donde cada píxel representa la distancia a un sector esférico del espacio original.

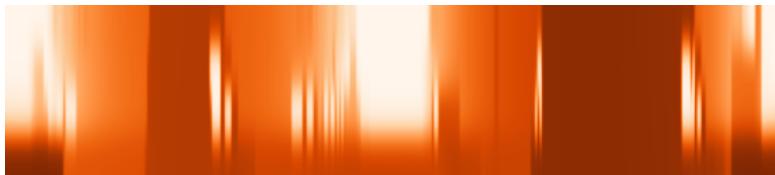


Figura 7.1: Un mapa de profundidad asociado a la nube de puntos original. Un color más oscuro implica más proximidad al obstáculo. La imagen ha sido sometida a un proceso de filtrado gaussiano por motivos de ilustración, pero no se realiza durante los procesos de entrenamiento e inferencia de los modelos.

Dado que el **LiDAR** describe el entorno de manera discreta con valores constantes de elevación y azimut<sup>4</sup>, se podría llegar a definir una biyección entre el conjunto de puntos original y los píxeles del mapa de profundidad, por lo que la información disponible en ambas es equivalente. Sin embargo, en nuestro caso no necesitamos una representación tan fiel del entorno porque:

- La resolución horizontal produce un mapa de profundidad de 1800 columnas. Esta resolución es extremadamente grande, y requeriría el uso de modelos con muchos parámetros, pudiendo caer fácilmente en un problema de *over-fitting*.

<sup>4</sup> El **azimut** se refiere al ángulo de elevación existente desde el suelo.

- La apertura vertical del **LiDAR** genera puntos en planos que no son relevantes para el problema. Esto es, planos muy bajos que impactan en el vehículo o muy altos que no impactan con el entorno considerado de interés.

Por estas razones, los mapas de profundidad se generarán de una manera más compacta, usando una resolución horizontal de  $1^\circ$  y los seis canales que van desde los  $-7^\circ$  hasta los  $3^\circ$ , lo que nos da un mapa de profundidad con una resolución de  $6 \times 360$  con un canal representando la distancia al punto de impacto más cercano contenido en el sector esférico que representa cada posición del mapa.

Para el segundo caso, se ha definido un radio de interés de 25 m, ya que se ha considerado para el proceso de cambio de carril como un entorno de influencia lo suficientemente amplio para establecer un límite entre información y no relevante para el cambio de carril.

Con estos datos, los mapas de profundidad serán normalizados al intervalo  $[0, 1] \in \mathbb{R}$  invertido, esto es, los valores más cercanos al vehículo serán más próximos a 1 mientras que los valores más alejados estarán más próximos a 0.

#### 7.1.2 Generación artificial de datos

Este problema de Los problemas con alta variabilidad de sus entradas suelen ser complejos y requerir de conjuntos de datos de tamaños bastante grandes para poder identificar patrones, como lo son por ejemplo los problemas de reconocimiento de imágenes.

En nuestro caso, el modelo de cambio de carril tiene como entrada una nube de puntos, la cual representa en un espacio de 3 dimensiones un conjunto muy limitado de puntos, con la dificultad añadida de que el **LiDAR** tiene de base un error de 3 cm. Como el espacio sobre el que trabajar es tan complejo, se requerirían modelos con muchos parámetros, pero al disponer de pocos ejemplos, podríamos caer muy fácilmente en problemas de *over-fitting*.

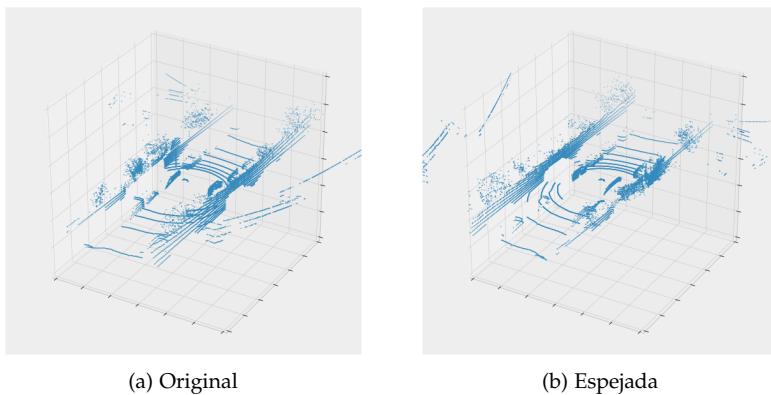
Por ello, se ha optado por realizar un proceso de generación de datos artificiales a partir de los datos existentes. De esta manera, ayudaremos al modelo a entrenar con casos similares y que de esta manera generalice mejor. La generación artificial se ha realizado sobre los datos recogidos en la ruta  $R_1$ , ya que es la que nos proporciona la información para entrenar el modelo y es por tanto en el único conjunto que cobra sentido este proceso. Concretamente hemos hecho uso de dos técnicas, primero un *mirroring* sobre todas las filas del conjunto y diez aplicaciones de la técnica *shaking* sobre el nuevo conjunto con los datos originales y simétricos.

La aplicación de estas técnicas requiere además que los nuevos datos generados mantengan una coherencia temporal. Por ello, cada uno se mantendrá en una secuencia independiente.

A continuación se pasan a describir los procesos de generación de datos artificiales introducidos previamente.

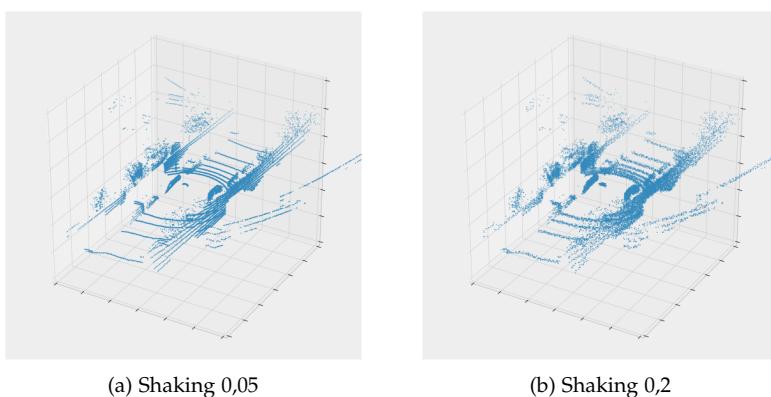
*mirroring* Se parte de la suposición de que los procesos cognitivos que producen determinados comportamientos (en nuestro caso, el cambio de carril) son los mismos independientemente de un cambio a la izquierda o hacia la derecha<sup>5</sup>.

Por tanto, para cada fila generaremos una nueva nube de puntos a partir de una simetría respecto al plano XY (recordemos que el eje X determina el sentido del movimiento del vehículo). De esta manera, modificando las variables pertinentes<sup>6</sup>, de cada ejemplo obtenemos uno nuevo. En la figura 7.2 se ilustra un ejemplo de este proceso sobre una nube de puntos arbitraria dentro del conjunto de ejemplos.



La principal ventaja de esta aproximación es que es posible doblar el tamaño del conjunto de datos sin añadir más ruido al ya existente.

*shaking* Esta técnica, a diferencia del *mirroring* sí puede llegar a tener impacto en la precisión de los datos. Partiendo de una nube de puntos original<sup>7</sup>, se aplica un desplazamiento aleatorio ( $\delta x, \delta y, \delta z$ ) sobre cada punto tal y como se describe en [Díaz Álvarez et al., 2018]. La Figura 7.3 ilustra dos procesos de shaking con diferentes desplazamientos sobre la nube original mostrada en la figura 7.2



<sup>5</sup> Es una suposición que en estudios sucesivos se puede tratar de refutar. Sin embargo, en el estadio actual de la investigación, nos parece razonable asumir que los procesos cognitivos en ambas situaciones son equivalentes.

<sup>6</sup> Es decir, invirtiendo los cambios de carril y la distancia recorrible en carriles izquierdo y derecho.

Figura 7.2: Un ejemplo de una nube de puntos (a) original, y (b) tras aplicarle el proceso de mirroring. Para cada fila, tras un proceso de mirroring y una inversión de las variables simétricas en función de la conducción (cambio de carril y distancia recorrible) se genera una nueva fila válida para el conjunto de datos.

<sup>7</sup> Debido a que la aplicación iterativa de este proceso sobre la misma nube de puntos acumula los errores entre iteraciones haciéndola ininteligible.

Figura 7.3: Un ejemplo de dos nubes de puntos tras pasar el proceso de shaking con un desplazamiento ( $\delta x, \delta y, \delta z$ ) de (a) (0,05, 0,05, 0,05) y de (b) (0,2, 0,2, 0,2) sobre la nube de puntos original. Para cada fila, tras un proceso de shaking disponemos de una nueva fila ligeramente diferente de la original, añadiendo ruido y, presumiblemente, generalización al modelo tras el entrenamiento.

Las especificaciones técnicas del LiDAR usado en los experimentos garantizan un error por debajo de los 3 cm de radio, por lo que un desplazamiento aleatorio para cada punto menor o igual que este valor no añadiría más ruido del existente. Sin embargo, en el experimento se ha optado sin embargo por aplicar un desplazamiento de  $\delta x = \delta y = \delta z = 0,05m$  en cada eje, ligeramente superior al proporcionado por el LiDAR. De esta forma se pretende la incorporación de ruido sobre el entorno original para aumentar la capacidad de generalización del modelo.

### 7.1.3 Incorporación de información temporal

Las **red convolucional (CNN, convolutional neural network)** son también redes que funcionan con un esquema *feed-forward*, y por tanto tenemos el mismo inconveniente que vimos en el anterior capítulo acerca de mantener una noción temporal en nuestros modelos. En el caso del modelo longitudinal, se consigue este efecto incluyendo variables derivadas de la posición (i.e. velocidad y aceleración). Para el caso del entorno circundante hay que realizar un proceso similar.

En cada una de las filas poseemos una representación del entorno en forma de mapa de profundidad que describe el entorno, por lo que al modelo le alimentaríamos únicamente con la situación en un instante  $t$  de tiempo. Esto no ayuda al modelo a descubrir patrones temporales como la velocidad o la aceleración puesto que no tiene información de eventos anteriores. Para solventar esta limitación, los conjuntos de datos serán transformados para que se alimenten con una ventana temporal de parámetros de entrada.

Comenzaremos definiendo el *momento*  $t_i$  como aquel ejemplo situado a  $t - \frac{i}{10}s$  en el pasado (ver Figura 7.4).

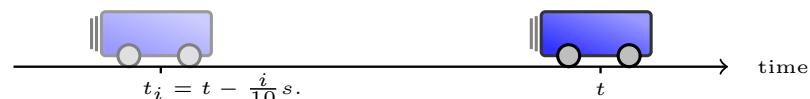


Figura 7.4: Un momento  $t_i$  se define como el estado en el que se encontraba el vehículo en  $t - \frac{i}{10}s$ . Por tanto,  $t_0$  es el momento actual.

<sup>8</sup> Dichos experimentos trabajan sobre el proceso de ejecución de cambio de carril, donde se asume que la maniobra involucra al córtex visual y al córtex prefrontal. Estos procesos cognitivos tienen un tiempo de respuesta de entre 0.2 s y 1.2 s [Buzsáki et al., 2012].

Los momentos que se tendrán en cuenta en los datasets finales serán  $t_0$ ,  $t_{10}$ ,  $t_{20}$ , correspondientes al momento actual, 1 s anterior y 2 s anteriores respectivamente. Estos valores no son arbitrarios, sino que se han elegido de acuerdo a los experimentos realizados en [Díaz Álvarez et al., 2018]<sup>8</sup>. Además, al incluir tres momentos temporales, ayudamos a los modelos a intuir patrones correspondientes a la primera y segunda derivadas de la posición (velocidad y aceleración).

Tras este ajuste, los conjuntos de datos están finalizados. Sus propiedades quedan descritas en la Tabla 7.1.

	Entradas	Salidas	Tamaño Entrenamiento	Tamaño Test	Cambio carril izda. Entrenamiento	Cambio carril izda. Test	Cambio carril dcha. Entrenamiento	Cambio carril dcha. Test
$LC_{S_1}$	8653	3	68990	2057	820	43	820	13
$LC_{S_2}$	8653	3	84810	2886	925	27	925	17
$LC_{S_3}$	8653	3	82060	2890	995	51	995	23
$LC_{S_A}$	8653	3	248930	7833	2740	121	2740	53

## 7.2 Modelo basado en perceptrones multicapa

Los entrenamientos han sido realizados sobre diferentes topologías, siendo las más representativas las descritas en la tabla 7.2. La tasa de dropout utilizada en los entrenamientos ha sido de 0,1.

Tabla 7.1: Descripción de los conjuntos de datos para el entrenamiento de los modelos. Los datos de entrenamiento incluyen todos los valores generados tras los procesos de *mirroring* y *shaking*.

Topología	Epochs	Precisión		
		Entrenamiento	Validación	Test
$MLP_1$	64, 64	$10^5$	0,33	0,33
$MLP_2$	1024, 512, 128	$10^5$	0,33	0,33
$MLP_3$	1024, 512, 256, 128	$10^5$	0,33	0,33

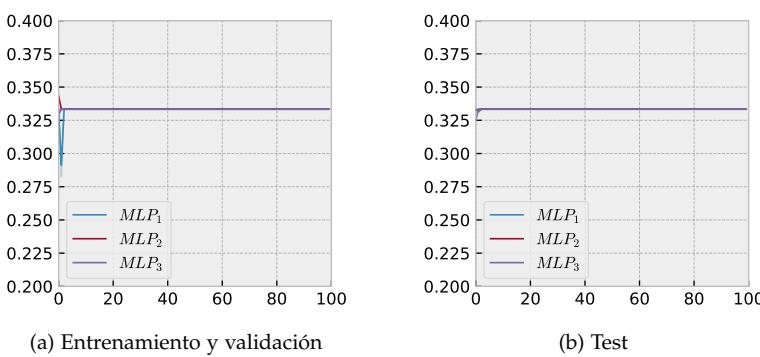
La figura 7.5 nos muestra la evolución de la precisión alcanzada por los modelos a lo largo del entrenamiento, tanto en entrenamiento como en el posterior test.

Cada una de las redes se ha entrenado durante  $10^5$  epochs. A la vista de los resultados, la capacidad de predicción de este tipo de redes con el conjunto de datos propuesto no supera el valor de predecir de forma aleatoria <sup>9</sup>. Además, en ningún caso las redes desarrolladas han sido capaces de superar ese valor (sí el caso del entrenamiento). Por tanto, el uso de MLP aquí es descartado definitivamente.

Tabla 7.2: Resumen de las arquitecturas de **perceptrones multicapa** para el modelo de cambio de carril. La posición de cada número de la topología indica la capa, siendo su valor el número de nodos (neuronas) que incluye dicha capa. Las arquitecturas seleccionadas en esta tabla son aquellas consideradas relevantes tras un proceso manual de ensayo y error.

<sup>9</sup> Los ejemplos le son propuestos de forma equiprobable, por lo que al ser 3 clases, la probabilidad de acertar de manera aleatoria es de  $\frac{1}{3}$ .

Figura 7.5: Comparación de las precisiones alcanzadas por los modelos de tipo **perceptrón multicapa**.



## 7.3 Modelo basado en redes convolucionales

Las arquitecturas de los modelos que mejores resultado han arrojado tras las pruebas se describen en la Tabla 7.3. La tasa de *dropout* utilizada en los entrenamientos ha sido de 0,1.

La descripción de los parámetros de las arquitecturas es la siguiente:

	Topología	Epochs	Precisión		
			Entrenamiento	Validación	Test
$CNN_1$	c16-4-18-v d128	$10^5$	0,589	0,576	0,573
$CNN_2$	c64-5-36-v, c256-3-5-v, d256-d128-d16	$10^6$	0,506	0,531	0,518
$CNN_3$	c16-3-18-v, c32-3-18-v, c64-2-18-v, d128	$10^6$	0,561	0,569	0,554

Tabla 7.3: Resumen de las arquitecturas de [redes convolucionales](#) para el modelo de cambio de carril. La posición de cada número de la topología indica la capa, siendo su valor el número de nodos (neuronas) que incluye dicha capa. Las arquitecturas seleccionadas en esta tabla son aquellas consideradas relevantes tras un proceso manual de ensayo y error.

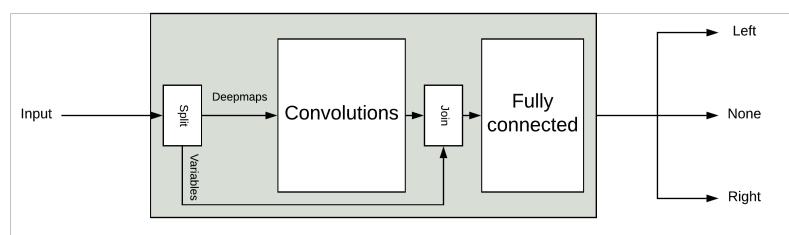
- $cF-W-H-x$ . Esta nomenclatura se corresponde con una capa de convolución de  $F$  filtros donde cada uno tiene un tamaño de  $W \times H$ .  $x$  puede tomar el valor  $v$  (de *valid*) o  $s$  (de *same*), dependiendo de si se desea que el filtro se limite a las dimensiones de la entrada o si por el contrario se prefiere que sobrepase los límites para que las convoluciones den como resultado una salida del mismo tamaño que la entrada original, respectivamente.
- $dN-r$ . Esta capa se corresponde con las capas *fully-connected* que trabajan sobre el espacio de las características extraídas en las convoluciones. En esta nomenclatura,  $N$  se corresponde con el número de neuronas que tiene la capa y  $r$  con la tasa de *dropout* aplicada a sus neuronas.

Existen una serie de particularidades del proceso de entrenamiento que hay que destacar. La primera y principal es la modificación de la arquitectura básica de una [CNN](#) para atender a variables que no obedecen al esquema de mapa  $n$ -dimensional. Una [CNN](#) basa su funcionamiento en una fase de extracción de características seguida de una fase de inferencia sobre las características extraídas de la imagen. En nuestro caso, sin embargo, necesitamos alimentar a la red con más información.

La decisión tomada ha sido entender estas variables (*Distancia circulable*, *Distancia a siguiente TLS* y *Estado de siguiente TLS*) como características del modelo ya extraídas. Para ello, al alimentar el modelo, los mapas de profundidad y estas variables son separadas al comenzar el entrenamiento. Los mapas de profundidad se pasan por el flujo normal de funcionamiento y una vez se han extraído las características de estos mapas se combinan con las variables para continuar con el flujo normal (ver Figura 7.6)

La siguiente modificación ha sido la alteración del mapa de profun-

Figura 7.6: Esquema de modificación de red [convolucional](#) para incluir características no espaciales. Las variables se separan al principio de la convolución para que éstas trabajen con las imágenes. Una vez este proceso se ha completado, las características de las imágenes y las variables de entrada separadas se vuelven a combinar para continuar el flujo normal de funcionamiento.



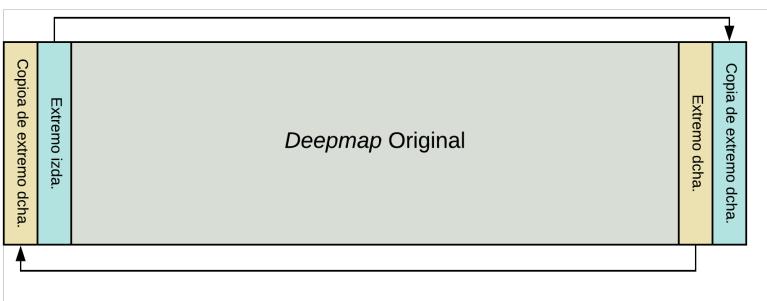


Figura 7.7: Padding artificial en los extremos del mapa de profundidad. Para no perder información que puede ser relevante, los mapas se aumentan artificialmente al alimentar a la red, de tal manera que a los extremos izquierdo y derecho se les aumenta con parte de los extremos derecho e izquierdo con un número de píxeles igual a la mitad del tamaño horizontal del los filtros que los recorrerán.

didad justo al comienzo de la alimentación. Al ser los mapas de profundidad una representación cilíndrica de las distancias, el extremo izquierdo y el derecho de la imagen son, en realidad, un punto de corte donde las convoluciones perderán información. Para evitar esta limitación, las imágenes con las que se alimenta la red son aumentadas artificialmente en sus bordes con el extremo contrario de la imagen (ver Figura 7.7), para que la convolución opere sobre la información que de otro modo se perdería.

La última modificación es la de la aplicación de una tasa de aprendizaje adaptativa al proceso de entrenamiento. Esta adaptación puede basarse en muchos factores, pero en nuestro caso la tasa se basa en una disminución suave de un valor máximo a un valor mínimo de manera que en los primeros estadios del entrenamiento la tasa sea suficientemente alta como para converger rápidamente a zonas de interés, pero que a la vez, según el entrenamiento avanza, se eviten los problemas de saltos alrededor de mínimos. La tasa de aprendizaje en cada epoch quedará determinada por la siguiente ecuación:

$$\alpha_i = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min}) \cdot e^{\frac{i}{\alpha_d}} \quad (7.2)$$

Donde  $\alpha_i$  será la tasa de aprendizaje a aplicar en el epoch  $i$ ,  $\alpha_{\min}$  y  $\alpha_{\max}$  las cotas inferior y superior que tomará la tasa de aprendizaje y  $\alpha_d$  la tasa de decrecimiento. En la Figura 7.8 se ilustra la gráfica de la función usada para entrenar las arquitecturas de esta sección.

Un último apunte antes de pasar a mostrar las gráficas de los entrena-

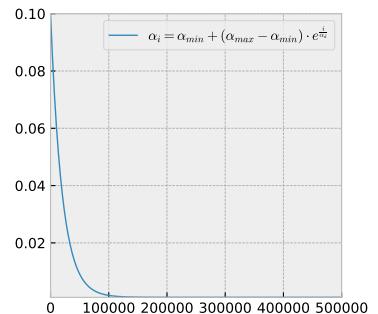
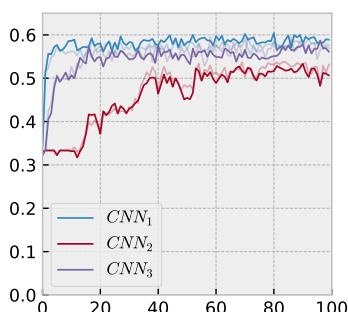
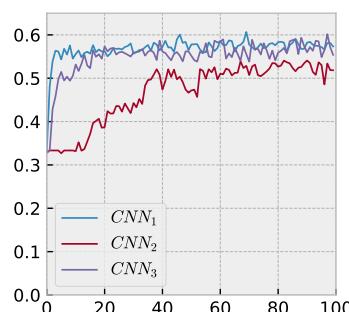


Figura 7.8: Gráfica de la tasa de aprendizaje adaptativo por epoch usada para entrenar **redes convolucionales**. En nuestros entrenamientos, los valores de los parámetros son  $\alpha_{\min} = 0,1$  y  $\alpha_{\max} = 0,001$  y  $\alpha_d = 20000$ .



(a) Entrenamiento y validación



(b) Test

Figura 7.9: Comparación de las precisiones alcanzadas por los modelos de tipo **CNN**.

mientos. No se ha incluido ninguna capa de *pooling* en las arquitecturas, sino que se ha preferido usar las capas de convolución sin añadir padding para ese efecto, tal y como proponen en [Howard et al., 2017].

La evolución de la precisión a lo largo del entrenamiento para las diferentes arquitecturas se muestra en la Figura 7.9.

Se puede observar que se ha necesitado una red relativamente grande para superar el límite impuesto por la clasificación aleatoria. Esto contrasta con los resultados obtenidos en [Díaz Álvarez et al., 2018], donde los valores de clasificación eran mucho más altos. La razón por la que se intuye esta disparidad de datos es que es precisamente la operación de **decisión de cambio de carril** la que es más compleja de modelar, a diferencia de la de **ejecución de cambio de carril**, en la que esta decisión está ya tomada.

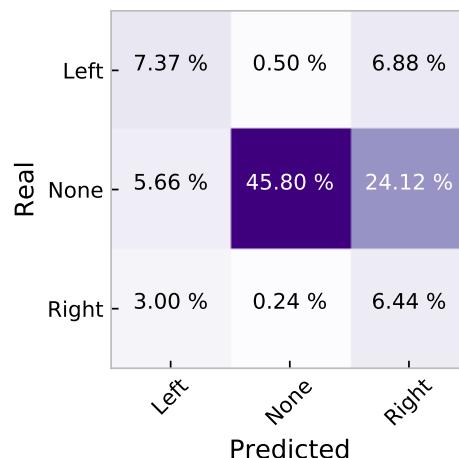
Probablemente con otra serie de variables se pueda aumentar la precisión de estos modelos, pero en este experimento estamos limitados al conjunto de variables observables en los entornos real y de simulación.

#### 7.4 Comparación entre modelos

En este caso, de las dos técnicas propuestas para los modelos, los **perceptrones multicapa** ha sido descartados completamente al ser sus clasificaciones iguales a la de la selección aleatoria. Por tanto, el modelo que usaremos será el *CNN<sub>2</sub>*, ya que es la única arquitectura que supera este límite.

En la figura 7.10 se expone la matriz de confusión asociada a esta arquitectura para los valores existentes en el conjunto de test.

Figura 7.10: Matriz de confusión del conjunto de test para el modelo de cambio de carril seleccionado.



## 8 Modelos específicos de conductores

En los capítulos anteriores se han decidido los modelos candidatos que conformarán el modelo de comportamiento de conductor general. Ahora que sabemos cuáles son las técnicas y arquitecturas que mejor se adecúan a este problema con los datos que disponemos, pasaremos a probarlas con los datos de los conductores específicos comprobando si:

- Los errores y precisiones se mantienen en el orden del modelo global.
- Los modelos capturan las particularidades de cada sujeto de estudio.

Para ello, se comparará primero el modelo longitudinal global con los modelos entrenados de los sujetos. Posteriormente, se compararán los modelos de los sujetos contra los conjuntos de test de cada uno de ellos.

### 8.1 Modelo longitudinal

La arquitectura que se ha entrenado para cada uno de los sujetos ha sido la  $MLP_2$ . En la Figura 8.1 se muestra la evolución del  $RMSE$  en test para cada uno de los sujetos tras entrenar los modelos con los parámetros indicados en el anterior capítulo.

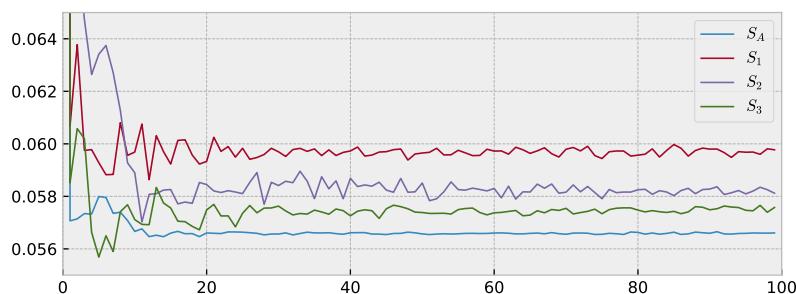


Figura 8.1: Comparativa de la evolución del error de test entre los sujetos de la arquitectura seleccionada para el modelo longitudinal. El  $RMSE$  en test en el caso del sujeto  $S_2$  es muy alto (mayor que 0,2) comparado con el error en el resto de sujetos y en el global.

Se puede observar que el error en test de los sujetos es ligeramente mayor que el del conjunto global. Esto se puede explicar por la mayor capacidad de generalización de  $S_A$  al haber sido entrenado con un conjunto de datos mayor. Los errores se resumen en la Tabla 8.1.

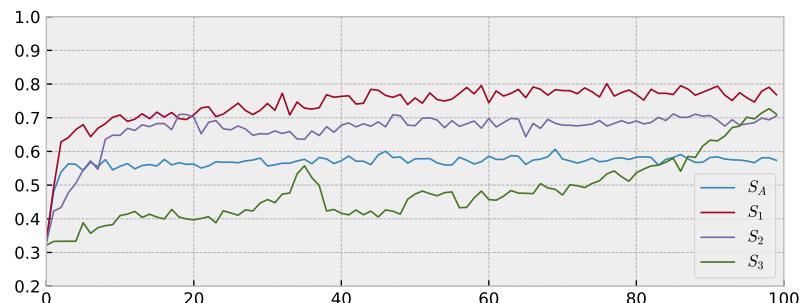
Tabla 8.1: Resumen de los valores de para los modelos específicos de comportamiento longitudinal.

	Entrenamiento	Validación	Test
$S_A$	0,056	0,062	0,057
$S_1$	0,045	0,039	0,060
$S_2$	0,048	0,047	0,058
$S_3$	0,055	0,054	0,058

## 8.2 Control lateral

En el caso del modelo del control lateral, la arquitectura con la que se han entrenado los modelos específicos es al  $CNN_1$  (topología  $c16-4-18-v, d128$ ). En la Figura 8.2 se muestra la evolución del en test para cada uno de los sujetos tras entrenar los modelos con los parámetros indicado en el anterior capítulo.

Figura 8.2: Comparativa de la evolución de la precisión en test entre los sujetos de la arquitectura seleccionada para el modelo de cambio de carril.



La precisión alcanzada en los modelos específicos entrenados con esta arquitectura es mayor que la alcanzada por el modelo global. Tiene sentido ya que el modelo global se ha entrenado con el conjunto de datos global y por tanto no obedece exactamente al comportamiento de ningún perfil en concreto, mientras que en los modelos específicos sí. En la Tabla 8.2 se exponen los valores de la precisión de este modelo.

Tabla 8.2: Resumen de los valores de precisión para los modelos específicos de cambio de carril.

	Entrenamiento	Validación	Test
$S_A$	0,588	0,576	0,573
$S_1$	0,805	0,763	0,768
$S_2$	0,683	0,708	0,706
$S_3$	0,727	0,706	0,710

### 8.3 Personalización en modelos de conducción específicos

Las anteriores secciones han mostrado cómo las topologías seleccionadas en los capítulos [Comportamiento longitudinal](#) y [Comportamiento lateral](#) son adecuadas para modelar los comportamientos específicos de cada conductor, además del conjunto global.

Lo interesante es comprobar si estas topologías son capaces de capturar las diferencias entre conductores. Para ello, comprobaremos cómo se comportan cada uno de los conductores en todos los conjuntos de test.

En el caso del modelo longitudinal, el [RMSE](#) de cada uno de los conductores se mantiene más bajo que el del resto de los sujetos en su propio conjunto de test. En la Figura 8.3 se muestra gráficamente el perfil de aceleración de cada uno de los conductores sobre cada uno de los perfiles reales.

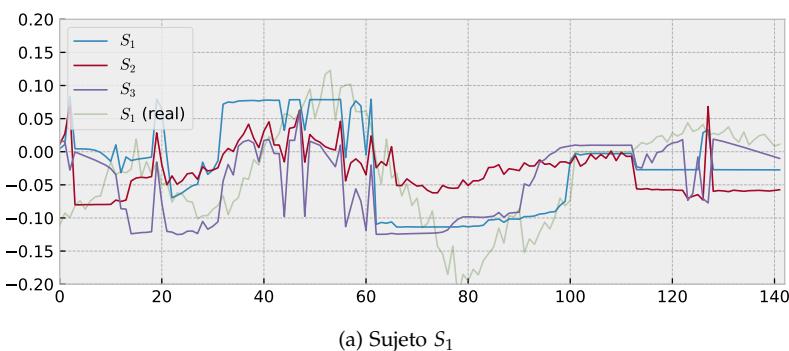
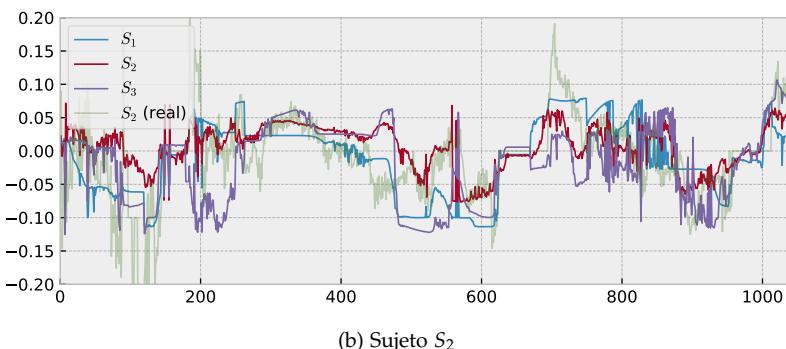
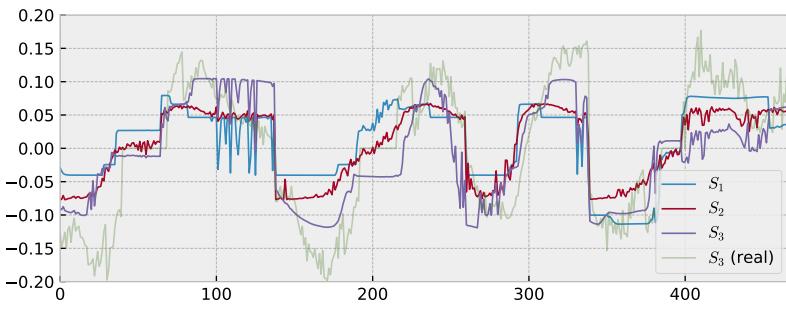
(a) Sujeto  $S_1$ (b) Sujeto  $S_2$ (c) Sujeto  $S_3$ 

Figura 8.3: Perfiles de aceleración de los diferentes sujetos sobre el resto. Se puede observar que de todos ellos, los perfiles de los sujetos originales se aproximan más a sus respectivos perfiles que los del resto.

Los valores de error cruzados entre sujetos se resumen en la tabla 8.3.

Tabla 8.3: Comparación de los errores de aceleración en los diferentes modelos longitudinales. Las filas se corresponden con los recorridos mientras que las columnas se corresponden con los modelos que se han intentado ajustar a ellas.

En el caso del modelo de cambio de carril, la comparativa la deberíamos hacer con respecto a las tasas de acierto en sus respectivos conjuntos de test. La Tabla 8.4 muestra los índices de precisión de los sujetos sobre cada uno de los conjuntos de test de los demás.

Tabla 8.4: Comparación de la precisión para los diferentes modelos de cambio de carril. Las filas se corresponden con los recorridos mientras que las columnas se corresponden con los modelos que se han intentado ajustar a ellas.

	$S_1$	$S_2$	$S_3$
$S_1$	0,059	0,074	0,070
$S_2$	0,064	0,058	0,067
$S_3$	0,065	0,065	0,057

## 9 Implementación en simulador

El entorno de simulación que nos ofrece **SUMO** nos permite recoger la práctica totalidad de información propuesta en la Tabla 5.1 del capítulo **Metodología** con la excepción del entorno.

El problema es que la representación que nos ofrece **SUMO** de éste es un conjunto posiciones y tipologías de elementos, de tal manera que podemos acceder a información rápidamente pero de forma muy limitada. Dicho de otro modo, en un entorno real no podemos saber, por ejemplo, que existe un coche en la posición  $(x, y, z)$  de unas determinadas dimensiones<sup>1</sup>.

Por lo tanto, la solución por la que se ha optado es por la implementación de un **LiDAR** en el vehículo simulado de tal manera que ofrece una nube de puntos de las mismas características que las capturadas por el **LiDAR** físico y situado en la misma posición del vehículo. Este **LiDAR**, a una tasa de 10 Hz, realizará las siguientes operaciones:

1. Captura de posición de todos los semáforos situados a un radio  $r$  del centro del **LiDAR** y transformación de éstos a prisma con base cuadrada sitiada a altura 0 m y con altura de 3 m.
2. Captura de posición de todos los vehículos situados a un radio  $r$  del centro del **LiDAR** y transformación de éstos a prismas con las dimensiones que especifiquen sus propiedades.
3. Cálculo de la nube de puntos colisionando contra los prismas generados.

En el vehículo se han implantado los modelos longitudinales y de cambio de carril dentro de un conductor virtual tanto para el sujeto global como para cada uno de los sujetos del experimento y se ha realizado un recorridos de características similares al recorrido del que se obtuvieron los datos del conjunto de test (ver Figura 9.1).

La forma por la que se comprobará si los comportamientos son similares es la propuesta en [Díaz Álvarez et al., 2014] para el comportamiento longitudinal. En la Tabla 9.1 se resumen estos valores para el modelo de conducción global y los conductores específicos. La leyenda de indicadores es la siguiente:

- $V$ . Velocidad instantánea del vehículo.

<sup>1</sup> En realidad sí sería posible conseguir esa información con cierto grado de certeza aplicando técnicas de reconocimiento de patrones a las nubes de puntos. Dichas técnicas son propensas a errores, más aun con la escasa resolución a largas distancias del **LiDAR** usado en los experimentos, por lo que no han sido consideradas para este trabajo.

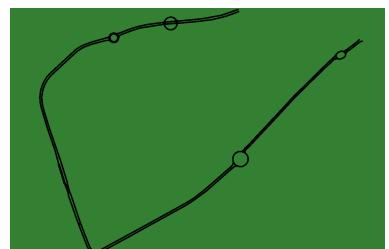


Figura 9.1: Circuito generado para recoger los datos de los vehículos circulando con los modelos longitudinal y de cambio de carril implantados.

<sup>2</sup> El **jerk** es la derivada de la aceleración.

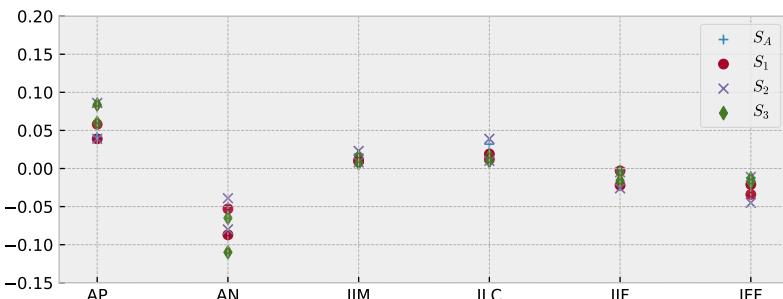
- **AP/AN.** Aceleración instantánea positiva y negativa del vehículo.
- **JIM.** *Jerk*<sup>2</sup> positivo con aceleración positiva, el cual se corresponde con la situación en la que el vehículo está iniciando la marcha (*jerk* de inicio de marcha).
- **JLC.** *Jerk* positivo con aceleración negativa, caso que se da cuando se está llegando a la velocidad deseada, por lo que aunque la velocidad sigue aumentando, lo hace en una tasa cada vez más decreciente (*jerk* de llegada a crucero).
- **JIF.** *Jerk* negativo con aceleración negativa, cuando el conductor inicia una maniobra de reducción de velocidad (*jerk* de inicio de frenada).
- **JFF.** *Jerk* positivo con aceleración negativa, que se corresponde con el momento en el que el conductor está terminando una maniobra de frenada o de reducción de velocidad (*jerk* de fin de frenada).

Tabla 9.1: Resumen de los indicadores provenientes de los datos de los conductores en los recorridos frente a los valores. En el caso del conjunto global, se ha elegido la media (redondeada a entero en el caso de cambio de carril) de los tres sujetos como estimador de los valores que correspondan para un supuesto conductor medio en ese recorrido. Los valores de los cambios de carril son el numero exacto y no el número de frames que transcurren durante los mismos. Los valores de cambio de carril correspondientes a las simulaciones son la media y la varianza de tres ejecuciones sobre el mismo escenario.

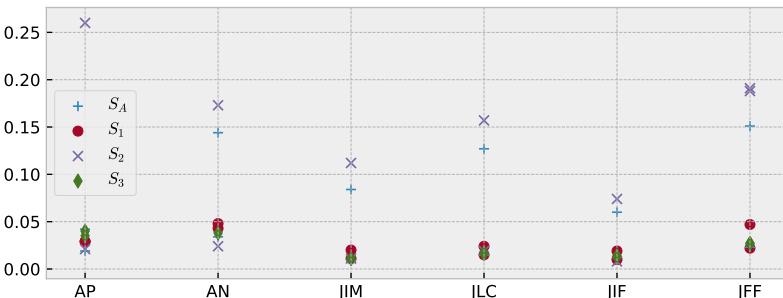
Por otro lado, para el comportamiento del modelo de cambio de carril se usará el número de éstos que se han producido durante el recorrido, ya que podemos considerar las condiciones similares (i.e. flujo de tráfico moderado en ambas direcciones, misma distancia de recorrido y aproximadamente el mismo número de carriles y semáforos durante el recorrido).

La Tabla 9.1 ofrece mucha información, por lo que la comentaremos por partes.

		$S_A$		$S_1$		$S_2$		$S_3$	
		$\mu$	$\sigma$	$\mu$	$\sigma$	$\mu$	$\sigma$	$\mu$	$\sigma$
$V$	R.	3,868	3,515	5,823	5,165	4,246	3,583	2,415	1,834
	S.	3,401	3,003	5,816	4,996	4,111	3,515	2,577	2,050
$AP$	R.	0,082	0,201	0,039	0,029	0,086	0,260	0,084	0,040
	S.	0,042	0,019	0,058	0,029	0,039	0,021	0,060	0,035
$AN$	R.	-0,088	0,144	-0,087	0,048	-0,080	0,173	-0,110	0,041
	S.	-0,050	0,034	-0,053	0,043	-0,039	0,024	-0,065	0,038
$JIM$	R.	0,019	0,084	0,011	0,011	0,023	0,112	0,015	0,013
	S.	0,007	0,009	0,009	0,020	0,008	0,011	0,007	0,011
$JLC$	R.	0,032	0,127	0,019	0,015	0,039	0,157	0,016	0,016
	S.	0,009	0,018	0,012	0,024	0,010	0,019	0,010	0,017
$JIF$	R.	-0,023	0,060	-0,022	0,019	-0,026	0,074	-0,016	0,014
	S.	-0,005	0,010	-0,003	0,010	-0,005	0,008	-0,005	0,013
$JFF$	R.	-0,035	0,151	-0,021	0,022	-0,045	0,191	-0,018	0,027
	S.	-0,012	0,023	-0,034	0,047	-0,011	0,188	-0,013	0,027
$LC$	R.	6	1,414	7	-	4	-	7	-
	S.	2	0	2	0	1	0,471	2	0,816
$RC$	R.	3,333	0,942	2	-	4	-	4	-
	S.	1,667	1,700	0,667	0,471	2,333	1,886	0,667	0,943



(a) Medias



(b) Desviaciones típicas

Figura 9.2: Media y desviación típica de cada uno de los indicadores planteados para el modelo longitudinal. Arriba, las medias y abajo las desviaciones típicas.

### 9.1 Comportamiento en el modelo longitudinal

Hemos decidido mostrar los datos de aceleración y *jerk* en el formato de la Figura 9.2. Para cada uno de los indicadores se muestran las marcas de cada sujeto de estudio. Cada sujeto tiene una marca diferente asignada para poder ver la diferencia entre ambas.

En el caso de las medias, todos los valores de los sujetos con sus respectivos modelos están muy próximos entre si salvo, quizás, los valores de la aceleración negativa. Lo mismo ocurre con las desviaciones típicas salvo por dos de ellos, el sujeto  $S_2$  y el sujeto genérico  $S_A$ .

Del sujeto genérico es lógico si contamos con que su modelo fue entrenado con los datos del sujeto  $S_2$ . Si volvemos atrás, al capítulo [Modelos específicos de conductores](#), en la Figura 8.3 se puede observar que el perfil de aceleración del sujeto  $S_2$  es muy irregular, y por tanto es probable que el modelo falle al generalizar esos casos tan específicos.

Del resto de sujetos, sin embargo, no se encuentran demasiadas discrepancias, sino que se mantienen en un nivel similar al del resto de variables.

Podemos concluir que los valores entre simulación y realidad son muy similares entre si, y que por tanto que los conductores simulados se pueden considerar similares a los reales en este comportamiento en concreto.

## 9.2 Comportamiento en el modelo de cambio de carril

En el caso del modelo de cambios de carril, se pueden ver indicios de que el número de cambios de carril en la realidad se mantienen en el mismo rango aproximado que el número de cambios de carril de los modelos estimados, tanto para el general como para los específicos.

Otro detalle es que el número de cambios de carril es menor siempre en el modelo estimado. La razón creo que puede deberse a la baja proporción de cambios de carril respecto a los ejemplos donde éste no existe. Este hecho es muy patente en el caso de los cambios de carril a derecha (fila *RC* de los indicadores de al tabla 9.1).

## **PARTE III**

## **CONCLUSIONES**



## *10 Conclusiones*

La aplicación de técnicas de la **inteligencia computacional** sobre el área de la conducción y, concretamente, al área del comportamiento es una línea de investigación muy prometedora. Es posible la extracción de comportamientos específicos a partir de datos reales de conducción, y, aunque no se ha conseguido emular con un cierto grado de satisfacción conductores en concreto, sí que se ha conseguido incorporar el factor indeterminado de la conducción en modelos basados en redes neuronales.

Las implicaciones de estos resultados son variadas. Aun con las limitaciones que nos imponen los micro-simuladores, es posible modelar estos dos comportamientos a partir de datos reales con precisión suficiente como para decir que los agentes se comportan de manera más humana.

El resto del capítulo entra en detalle acerca de los objetivos planteados y su consecución, una discusión sobre los resultados obtenidos y un planteamiento acerca de algunas posibles líneas futuras de trabajo a partir de la presente tesis.

### *10.1 Hipótesis planteadas*

Al comienzo de la tesis se plantearon una serie de objetivos materializados en dos hipótesis que se han tratado de demostrar a lo largo de esta tesis.

La hipótesis H<sub>1</sub> planteaba que la aplicación de técnicas pertenecientes al campo de la **inteligencia computacional** con datos de conductores reales sí permite la creación de modelos de conducción con características humanas. A tenor de los resultados obtenidos, se concluye que se pueden desarrollar modelos con la técnica propuesta para incorporar conductores en simulaciones que incorporen comportamientos no deducibles directamente con un modelo lineal.

Sin embargo, la hipótesis H<sub>2</sub> proponía que estas técnicas eran suficientes para modelar un conductores en concreto dentro de entornos simulados. Si bien es cierto que los resultados indican que los modelos generados para conductores diferentes son diferentes entre sí, no creemos que exista suficientes indicios para afirmar que los

comportamientos modelos simulados son suficientemente fieles al comportamiento de los conductores en el mundo real.

Por tanto, podemos concluir que es posible incorporar características humanas sobre los modelos de conductor para hacer las simulaciones más fieles a la realidad. Sin embargo, a la hora de modelar conductores en particular, aunque se ha podido extraer comportamiento personalizado, consideramos que no es suficiente para ser usado como fuente para la comparación objetiva de dos conductores.

Somos optimistas, no obstante, a que esta conclusión es debida a la dificultad de hacer corresponder los datos de la realidad con los datos disponibles desde el simulador, y por tanto el uso de simuladores que ofrezcan un entorno más parecido a la realidad permitirán que la aplicación de estas técnicas sí permitan la comparación entre diferentes conductores.

## *10.2 Sobre los datos, los sensores y su utilidad*

Para minimizar la variación de las variables de entrada, los recorridos fueron planteados con ciertos factores a tener en cuenta (e.g. mismos horas, día de la semana, clima, velocidades máximas de vía, etcétera) para mantener un conjunto de datos lo más parecido posible entre sujetos, y así minimizar en la medida de lo posible la incorporación de imprecisiones o sesgos a los conjuntos de datos. Los dispositivos además fueron configurados de la forma más precisa posible, y su sincronización establecida a sus tasas de refresco más bajas para acercar lo máximo posible sus valores entre sí, a fin de minimizar el desplazamiento temporal entre medidas.

Con todo, la captura en este tipo de entornos es una tarea compleja, y más tratándose de tráfico. Los recorridos no están exentos de situaciones fortuitas no predecibles<sup>1</sup> y los dispositivos no son lo suficientemente precisos para capturar los datos de interés. En nuestro caso concreto, consideramos que:

- La precisión de un único **LiDAR** de 16 capas no es el dispositivo más adecuado para capturar las características del entorno necesarias para modelar un comportamiento humano. Una separación de 2° por capa hace que a 15 m las capas estén separadas algo más de 0.5 m, lo cual es insuficiente para muchos usos<sup>2</sup>
- En el caso del **GPS**, aun con alta precisión y corrección diferencial implementada, es arriesgado su uso como único dispositivo de localización y para determinar velocidades.

Acerca de los datos, los mapas de profundidad obtenidos han demostrado ser una herramienta válida para la representación de un entorno a pesar de su baja resolución.

<sup>1</sup> Los recorridos fueron repetidos para todos los sujetos en dos ocasiones debido entre otras cosas al tiempo y a accidentes en las rutas planteadas.

<sup>2</sup> Por ejemplo, sobre una persona de 1.80 m a una distancia de 20 m alcanzarían únicamente 2 haces a lo sumo. Quizá la incorporación de más dispositivos de este tipo en configuraciones con haces cruzados podrían ayudar a esta tarea.

La técnica del *mirroring* es muy usada en el dominio de reconocimiento de imágenes. Sin embargo, y aunque en nuestro caso está justificada<sup>3</sup>, puede ser contraproducente en el caso de querer extraer información del comportamiento de conductores en entornos extra-urbanos, ya que en éstos, tanto el carril por el que se circula como los cambios sí se pueden considerar dependientes del sentido de la circulación.

Por otro lado, la técnica propuesta del *shaking* ha sido extremadamente útil en la generación artificial de datos y además no sufre del problema del *mirroring* de los entornos extra-urbanos. Consideramos que esta técnica, en conjuntos de datos donde existe un cierto grado de error o imprecisión, es capaz de incrementar su tamaño ayudando a mejorar la generalización y, por tanto, a la calidad de los resultados obtenidos.

### *10.3 Sobre las técnicas de inteligencia computacional*

Las técnicas utilizadas ([red convolucional](#), [sistema de control borroso](#) y [perceptrón multicapa](#)) dependen mucho de las variables de los datos, y no sólo de la cantidad de éstos. De nada sirve ofrecer datos de una variable que no tiene impacto sobre el resultado que se quiere alcanzar. La selección de variables realizada en esta tesis se ha basado en la disponibilidad de éstos en ambos entornos (real y simulación), tratando de obtener de formas alternativas los más críticos (e.g. implementando un [LiDAR](#) dentro de un simulador, calculando la distancia que se puede recorrer en los carriles).

La aplicación del descenso del gradiente para el ajuste de un [sistema de control borroso](#) permite de forma rápida encontrar un controlador óptimo para un problema. Sin embargo el problema del modelo longitudinal no es directamente tratable con un controlador, al menos no con las variables planteadas y el conjunto de datos. Sin embargo, creemos que aumentando la precisión de los datos obtenidos y con un nuevo conjunto de variables más cercano al entorno real, el desempeño de éstos aumentaría notablemente.

Para el trabajo con datos del entorno, se ha podido comprobar que las [redes convolucionales](#) han generalizado mejor los resultados que los [perceptrones multicapa](#). Después de todo el entorno se ha representado con una imagen, dominio en las que estas redes destacan por su capacidad de clasificación. Además, al incorporar la ventana temporal como canales de la imagen, toda información está localizada topológicamente cerca con sus momentos anteriores, por lo que es de esperar que esta representación sea capaz de converger más rápidamente hacia una solución. Esto, junto con la modificación de inyección de datos de entrada en capas posteriores a la extracción de características, hace que las [redes convolucionales](#) se hayan comportado mejor en su tarea de clasificación.

<sup>3</sup> La normativa indica que el carril por el que se debe circular es el que más convenga para su destino, a diferencia de vías como una autopista.

Concluimos por tanto que las **redes convolucionales**, en tareas donde es importante la topología espacial y temporal de las entradas, es extremadamente eficiente a la hora de modelar eventos donde sus ventanas temporales están localizadas lo suficientemente cerca en el tiempo como para que sus patrones se desplacen ligeramente, mejorando mucho en precisión y capacidad de caracterización de las entradas a los **perceptrones multicapa**.

Somos optimistas en que la implementación de estas técnicas en simuladores más potentes y realistas mejorarán los resultados obtenidos, y por ello es una de las líneas futuras de investigación propuestas.

#### *10.4 Sobre el entorno de simulación*

**SUMO** es un entorno de simulación de tráfico muy potente a la hora de simular escenarios de tráfico complejo. Sin embargo, para comportarse de esta manera, tiene que simplificar otros aspectos. El realismo se queda en la faceta de tráfico, ya que en la faceta de conductor no existe apenas complejidad. El entorno en el que circulan los vehículos no es realista (e.g. no hay edificios o suelo), los carriles son vías por las que circulan los vehículo, los cambios de carril son una “*teleportación*”, y así un largo etcétera.

En la captura de datos ha sido necesario un proceso previo de muchos de esos datos para convertirlos a algo lo más parecido posible a lo que sería recuperable bajo el entorno simulado, y esas concesiones son las que hacen que se pierda fidelidad en el conjunto de datos capturado.

La conclusión a la que hemos llegado tras trabajar intensivamente es que el entorno es muy adecuado para trabajar otros aspectos de los **ITS** tales como comportamientos colaborativos, comunicaciones, planificación vial, cálculo de rutas óptimas, etcétera. Para trabajar el comportamiento de un conductor, sin embargo, necesitamos otras aproximaciones, como por ejemplo el simulador 3DCoAutoSim [Olaverri-Monreal et al., 2018], desarrollado por el *Intelligent Technologies in Smart Cities* de la *Fachhochschule Technikum Wien* y con el que se ha tenido la oportunidad de trabajar en la estancia realizada durante el doctorado.

# *11 Resultados y líneas de investigación futuras*

En este capítulo se comentarán una serie de líneas de trabajo futuras que se consideran de interés tras el trabajo realizado. Posteriormente se enumerarán las publicaciones realizadas durante la tesis como consecuencia directa o indirecta de la tesis.

## *11.1 Propuestas de líneas de investigación futuras*

### *11.1.1 Entornos de simulación más realistas*

Para simular conductores en entornos de simulación es necesario que estos sean lo suficientemente potentes como para presentar un escenario realista al agente que va a modelar al conductor.

Sera muy interesante evaluar las mismas técnicas en otros entornos de simulación. Algunas opciones sería entornos de conducción de tipo [TORCS](#) adaptados a la simulación de tráfico, simuladores de conducción que funcionasen conectándose a simuladores de entornos de tráfico (como por ejemplo el anteriormente citado 3DCoAutoSim [[Olaverri-Monreal et al., 2018](#)]) o, más reciente, el uso de videojuegos con entornos de tráfico <sup>1</sup>.

La aplicación de estas técnicas en el modelado de agentes en estos entornos propiciaría la investigación en otras áreas más orientadas a factores humanos, ya que el comportamiento de los conductores sería más realista.

### *11.1.2 Modelado de diferentes actores y escenarios*

En esta tesis se trabaja sobre todo con el comportamiento global de comportamiento de conductores. Existen casos específicos de comportamiento que merecería la pena explorar por separado, como pueden ser intersecciones más o menos complejas, rotundas, incorporaciones a vías, etcétera.

Además, los conductores de turismos no son los únicos agentes existentes en el sistema que es el tráfico. En general se tiende a prestar

<sup>1</sup> Está habiendo una tendencia en los últimos años de hacer aprender a agentes a jugar a todo tipo de videojuegos a partir de la lectura en crudo de la pantalla y actuando sobre los controles del juego. Uno de éstos juegos titulado Grand Theft Auto V incorpora una simulación de una ciudad entera con peatones, señales de tráfico, vehículo y un punto de vista de cámara desde el interior del vehículo. Dos lecturas interesantes sobre este fenómeno son [[Richter et al., 2016](#)] y ?? donde ahondan en la recuperación de datos a partir de videojuegos y de hasta qué punto pueden llegar a reemplazar daos reales.

poca atención a los vehículos pesados, pero su conducción y comportamiento es muy diferente a la de un turismo. Existen también motos, bicicletas, pero también peatones.

Otro objetivo de modelado sería los sistemas de generación de emisiones de los vehículos. Actualmente, los simuladores incorporan modelos de emisión<sup>2</sup> para simular de qué manera los vehículos producen emisiones. Los sistemas más completos (como **SUMO**) implementan las tipologías descritas en el **HBEFA** [De Haan and Keller, 2004], pero no dejan de ser modelos que generalizan tipologías de vehículos y se alejan de la realidad. Creemos que las emisiones de los vehículos en particular son fácilmente modelables a partir de sus datos reales con técnicas de **CI**.

### 11.1.3 Redes Neuronales Recurrentes

El estudio se ha limitado a redes de tipo *feed-forward*, representando la evolución del tiempo como ventanas de datos temporales. Este tipo de redes no ven más allá de su ventana temporal, y puede no ser suficiente para determinados modelos.

Se propone el uso de arquitecturas de **redes neuronales recurrentes** para incorporar capacidad de comprender la continuidad temporal en el flujo de datos que percibe el agente. Quizá de esta manera se podrían captar características tales como la percepción temporal de velocidades de aproximación a obstáculos o impaciencia en función de otros factores.

## 11.2 Diseminación de resultados

### 11.2.1 Revistas

1. Díaz-Álvarez, A., Clavijo, M., Jiménez, F., Talavera, E., & Serradilla, F. (2018). *Modelling the human lane-change execution behaviour through Multilayer Perceptrons and Convolutional Neural Networks*. Transportation Research Part F: Psychology and Behaviour, 2018. **(Q2)**.
2. Olaverri-Monreal, C., Errea-Moreno, J., & Díaz-Álvarez, A. (2018). *Implementation and Evaluation of a Traffic Light Assistance System Based on V2I Communication in a Simulation Framework*. Journal of Advanced Transportation, 2018. **(Q2)**.
3. Talavera, E., Díaz-Álvarez, A., Jiménez, F., & Naranjo, J. E. (2018). *Impact on Congestion and Fuel Consumption of a Cooperative Adaptive Cruise Control System with Lane-Level Position Estimation*. Energies, 11(1), 194. **(Q2)**.
4. Jiménez, F., Naranjo, J. E., Serradilla, F., Pérez, E., Hernández, M. J., Ruiz, T., ... & Díaz, A. (2016). *Intravehicular, short-and long-range*

<sup>2</sup> Normalmente, las emisiones incluyen también el consumo de gasolina, diésel y electricidad, así como el ruido que emiten.

- communication information fusion for providing safe speed warnings.*  
*Sensors, 16(1), 131. (Q1).*
5. Díaz-Álvarez, A., Serradilla-García, F., Anaya-Catalán, J. J., Jiménez-Alonso, F., & Naranjo-Hernández, J. E. (2015). *Estimación de la autonomía de un vehículo eléctrico según el estilo de conducción.* DYNA-Ingeniería e Industria, 90(3). (Q4).
  6. Alvarez, A. D., Garcia, F. S., Naranjo, J. E., Anaya, J. J., & Jimenez, F. (2014). *Modeling the driving behavior of electric vehicles using smartphones and neural networks.* IEEE Intelligent Transportation Systems Magazine, 6(3), 44-53. (Q3).

#### 11.2.2 Congresos

1. Clavijo, M., & Díaz, A., Serradilla, F., Jiménez F., Naranjo, J.E. (2017, July). *Deep learning application for 3D LiDAR odometry estimation in autonomous vehicles.* Connected and Automated Transport, 2018 Transport Research Arena (TRA).Internacional.
2. Clavijo, M., Serradilla, F., Naranjo, J.E., Jiménez F., & Díaz, A. (2017, July). *Application of Deep Learning to Route Odometry Estimation from LiDAR Data.* Advances in Vehicular Systems, Technologies and Applications, 2017 The Sixth International Conference on (pp. 60-65). Internacional.
3. Felipe, J., Amarillo, J. C., Naranjo, J. E., Serradilla, F., & Díaz, A. (2015, September). *Energy consumption estimation in electric vehicles considering driving style.* In Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on (pp. 101-106). IEEE.



## **PARTE IV**

## **APÉNDICES**



## A *Ajuste de sistemas de control borroso basado en descenso del gradiente*

Como vimos en la sección [Acrlongplspfcs](#) del capítulo [Inteligencia Computacional](#), un [sistema de control borroso](#) se compone de cuatro bloques diferenciados: la fuzzificación, el bloque de reglas, la inferencia y la defuzzificación. De estos bloques, el proceso manual de ajuste se resume siempre en dos pasos<sup>1</sup>:

1. Elección de las particiones borrosas de las variables lingüísticas.
2. Definición de las reglas borrosas.

Las soluciones de ajuste que existen en la literatura suelen funcionar ajustando automáticamente uno de los dos puntos, manteniendo fijo el otro (e.g. ajustar las particiones fijas las reglas). Nosotros representaremos el controlador como un grafo computacional de manera que sus gradientes sean sencillos de ajustar<sup>2</sup>.

Se han tomado una serie de decisiones de diseño para facilitar el desarrollo del controlador, aunque son fácilmente modificables. Éstas son:

- Para cada partición borrosa se limitarán las funciones de pertenencia a: una línea descendente para caracterizar al primer conjunto borroso, una línea ascendente para caracterizar el último y trapezoides para caracterizar al resto.
- Las  $t$ -norma y  $t$ -conorma serán el máximo y el mínimo respectivamente. La  $t$ -norma se usará como operador asociado al AND lógico y a la operación de implicación, mientras que la  $t$ -conorma se usará como operador asociado al OR lógico y a la acumulación.
- El controlador será de tipo *Takagi-Sugeno* de orden 0, y por tanto se representarán las funciones de salida como conjuntos borrosos de tipo singleton. La función de defuzzificación será la la operación CoGS<sup>3</sup>.

<sup>1</sup> Por supuesto implica más ajustes como la elección de las funciones de fuzzificación, las  $t$ -normas y  $t$ -conormas, de la función de defuzzificación, pero estas operaciones no tienen tanto impacto en el desempeño del controlador como las particiones borrosas y las reglas.

<sup>2</sup> Como veremos más adelante, el tiempo de ajuste con esta representación es muy rápido, lo que abre la posibilidad de desarrollar un algoritmo que incluya este para el ajuste de los metapárametros del controlador, como por ejemplo los tamaños de las particiones.

<sup>3</sup> La operación CoGS (*Center of Gravity for Singletons*) se define como:

$$CoGS = \sum_{i=1}^n w_i \cdot o_i \quad (A.1)$$

Es decir, la suma ponderada de los valores de salida.

- El tamaño de las particiones borrosas no variará dinámicamente a lo largo del entrenamiento, siendo éste un meta-parámetro de configuración del controlador.
- El controlador tendrá una única variable de salida.

### A.1 Representación de un FCS como grafo computacional

Sea  $\mathcal{V} = V_1, V_2, \dots, V_n$  el conjunto ordenado de variables lingüísticas de nuestro controlador, y sea  $O$  la variable lingüística de salida. Cada una de estas variables tendrá un número prefijado de conjuntos  $\mathcal{N} = N_{V_1}, N_{V_2}, \dots, N_{V_n}$  para las variables de entrada y  $N_O$  para la variable de salida.

El controlador recibirá una matriz bidimensional de rango  $(m, n)$  y devolverá un vector de rango  $(m)$ , siendo  $m$  el número de ejemplos que queremos calcular a la vez.

Representaremos el **sistema de control borroso** como grafo computacional. De esta manera, podremos (i) aplicar la función de manera más eficiente y (ii) calcular fácilmente los gradientes de las funciones parciales para aplicar la técnica del descenso del gradiente propagando el error de la salida.

El proceso de desarrollo será el siguiente: primero, construiremos cada uno de los bloques fundamentales del **sistema de control borroso** como un grafos computacionales; después construiremos el controlador conectando entre sí estos grafos.

#### A.1.1 Bloque de fuzzificación

El grafo computacional asociado a este bloque tomará una matriz bidimensional de rango  $(m, n)$  (la matriz de entrada) que contendrá para cada ejemplo una lista de los valores que toma cada una de las variables de entrada. La salida de este grafo será una matriz bidimensional de rango  $(m, \sum_{i=1}^n N_i)$ , donde se almacenarán los grados de pertenencia de esos valores acada conjunto borroso de la variable que les corresponda (Figura A.1).

$$\left( \begin{array}{ccc} x_{V_1}^1 & x_{V_2}^1 & \dots \\ x_{V_1}^2 & x_{V_2}^2 & \dots \\ x_{V_1}^3 & x_{V_2}^3 & \dots \\ x_{V_1}^4 & x_{V_2}^4 & \dots \end{array} \right) \rightarrow \left( \begin{array}{cccc} \mu_{V_1}^1(x_{V_1}^1) & \mu_{V_1}^2(x_{V_1}^1) & \mu_{V_2}^1(x_{V_2}^1) & \mu_{V_2}^2(x_{V_2}^1) & \dots \\ \mu_{V_1}^1(x_{V_1}^2) & \mu_{V_1}^2(x_{V_1}^2) & \mu_{V_2}^1(x_{V_2}^2) & \mu_{V_2}^2(x_{V_2}^2) & \dots \\ \mu_{V_1}^1(x_{V_1}^3) & \mu_{V_1}^2(x_{V_1}^3) & \mu_{V_2}^1(x_{V_2}^3) & \mu_{V_2}^2(x_{V_2}^3) & \dots \\ \mu_{V_1}^1(x_{V_1}^4) & \mu_{V_1}^2(x_{V_1}^4) & \mu_{V_2}^1(x_{V_2}^4) & \mu_{V_2}^2(x_{V_2}^4) & \dots \end{array} \right)$$

Figura A.1: El bloque de fuzzificación transformará los valores de sus respectivos dominios al de cada conjunto borroso. La matriz generada tendrá tantas columnas como conjuntos borrosos posee cada variable.

Para ello, se le aplicará a cada columna de la matriz de entrada tan-

tas operaciones (funciones de pertenencia) como conjuntos borrosos posea. Concretamente se aplicarán operaciones de línea descendente, trapezoidal y ascendente.

Los grafos de las líneas **ascendente** y **descendente** tienen una forma parecida. El grafo asociado a la fórmula de la línea descendente se muestra en la Figura A.2, que es la que se correspondería con el último conjunto de una partición borrosa. Como se puede ver, las variables ajustables son  $a$  y  $\delta b$ . Estas definen el intervalo  $(a, a + \delta b) \in \mathbb{R}$  donde los valores  $f(X)$  ascienden de 0 a 1 (o descenden de 1 a 0 en el caso de la recta ascendente).

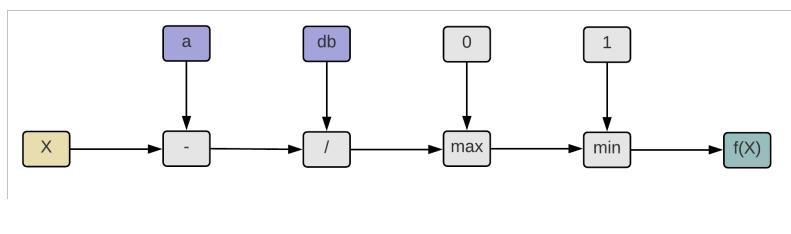


Figura A.2: Ilustración del grafo computacional para la función de pertenencia línea ascendente. La fórmula que describe es  $\mu(x) = \min(\max(\frac{x-a}{\delta b}, 0), 1)$ , quedando acotada  $\mu(x)$  en el intervalo  $[0, 1] \in \mathbb{R}$ . El grafo computacional para la línea descendente es similar y se corresponde con la fórmula  $\mu(x) = \min(\max(\frac{a-x}{\delta b+1}, 0), 1)$ .

El **trapeo** se define a partir de los parámetros  $(a, \delta b, \delta c, \delta d) \in \mathbb{R}$ , que definen los intervalos  $I_1 = (a, a + \delta b)$ ,  $I_2 = (a + \delta b, a + \delta b + \delta c)$  y  $I_3 = (a + \delta b + \delta c, a + \delta b + \delta c + \delta d)$ .  $I_1$  es el intervalo donde la función de pertenencia aumenta su valor de 0 a 1,  $I_2$  el intervalo superior del trapeo donde la función vale 1 e  $I_3$  donde la función comienza a descender su valor de 1 a 0. El grafo asociado a esta función se ilustra en la figura A.3

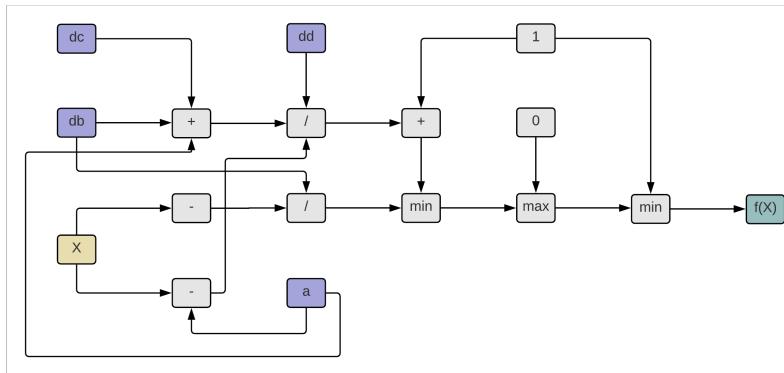


Figura A.3: Ilustración del grafo computacional para la función de pertenencia trapezoidal. La fórmula que describe es  $\mu(x) = \min(\max(\min(\mu_{Lasc}, \mu_{Ldesc}), 0), 1)$ , esto es, una línea ascendente y otra descendente, estando ambas acotadas en el intervalo  $[0, 1] \in \mathbb{R}$ .

Sabiendo los grafos computacionales de cada una de las funciones de pertenencia, podemos definir el grafo asociado a la partición borrosa de una variable lingüística. Suponiendo que la variable  $V_i$  está dividida en  $N_{V_i}$  conjuntos borrosos, la partición estará compuesta de:

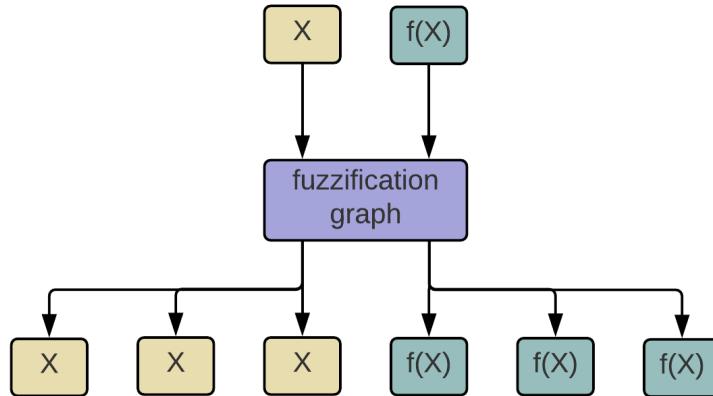
- Un primer conjunto definido como una pendiente descendente.
  - $N_{V_i} - 2$  conjuntos definidos como trapezoides.
  - Un último conjunto definido como una pendiente ascendente.

Este grafo tiene que definir una serie de variables para que nuestro algoritmo de entrenamiento las ajuste correctamente. Estas variables están directamente relacionadas con los parámetros de las funciones de pertenencia descritas anteriormente. Hemos decidido por tanto establecer estas variables como los espacios existentes entre los puntos característicos de las funciones.

Al estar definidas de esta manera, logramos (i) que la suma de todas las pertenencias en cada punto del eje  $X$  sea 1 y (ii) que cada pequeña variación del gradiente de una de las variables tiene el potencial de provocar una variación en el resto de variables.

Por último, un **sistema de control borroso** define un número de variables de entrada. Nuestro grafo de fuzzificación estará definido de tal manera que para una matriz de entrada  $m \times l$ , siendo  $m$  cada tupla de valores a inferir y  $l$  cada una de las variables lingüísticas, generará una matriz de la forma  $m \times \sum_{i=1}^l |l_i|$ , siendo  $|S|$  el número de conjuntos borrosos que contiene la variable lingüística  $l_i$  (Figura A.4)

Figura A.4: Siendo el número de conjuntos borrosos de las variables lingüísticas  $X_1$  e  $X_2$  3, el grafo de fuzzificación transformará los dos valores de entrada  $x \in X_1$  e  $y \in X_2$  en seis valores, los correspondientes a los valores de pertenencia de  $x$  e  $y$  a cada uno de los conjuntos borrosos de  $X_1$  y  $X_2$  respectivamente



#### A.1.2 Bloque de inferencia

Este bloque tomará una matriz bidimensional de rango  $(m, \sum_{i=1}^n N_i)$ , es decir, la matriz de salida del bloque de fuzzificación, y generará una matriz bidimensional de rango  $(m, N_O)$  que contendrá los valores borrosos de salida (uno por cada conjunto borroso de salida). Para ello, hará uso de un bloque de reglas en las que basará su inferencia.

Este bloque de reglas es el que se tratará de ajustar. La representación será la de una matriz  $(v_i + 1)$ -dimensional, siendo  $v_i = \sum_{i=1}^l |l_i|$  el número total de entradas borrosas que llegan al bloque. La dimensión adicional se corresponde a la variable lingüística de salida.

Dicho de otro modo, cada posible conjunto borroso de salida (cada valor dentro del eje correspondiente a la salida) se corresponderá con

una matriz  $v_i$ -dimensional resultado del producto cartesiano de las variables de entradas. Esto es, cada una de las posibles combinaciones de reglas, a la que podemos asociar un valor. En la Figura A.5 se muestra un ejemplo con las variables obtenidas en el ejemplo de la Figura A.4 tras aplicarles la  $t$ -norma.

$$\begin{pmatrix} \mu_1^A \\ \mu_2^A \\ \mu_3^A \end{pmatrix} \times \begin{pmatrix} \mu_1^B \\ \mu_2^B \\ \mu_3^B \end{pmatrix} = \begin{pmatrix} T(\mu_1^A, \mu_1^B) \\ T(\mu_1^A, \mu_2^B) \\ T(\mu_1^A, \mu_3^B) \\ T(\mu_2^A, \mu_1^B) \\ T(\mu_2^A, \mu_2^B) \\ T(\mu_2^A, \mu_3^B) \\ T(\mu_3^A, \mu_1^B) \\ T(\mu_3^A, \mu_2^B) \\ T(\mu_3^A, \mu_3^B) \end{pmatrix}$$

Figura A.5: El producto cartesiano de las variables borrosas de entrada genera todas las posibles combinaciones entre los conjuntos borrosos de las variables lingüísticas. Aplicándosele la  $t$ -norma a cada una de estas combinaciones tenemos todas las posibles reglas que se pueden definir en este [sistema de control borroso](#).

Al estar definida la  $t$ -conorma y la acumulación con el mismo operador, una regla de tipo OR es equivalente a dos reglas de tipo AND, ya que la acumulación de sus resultados es equivalente. Por tanto, al aplicar la  $t$ -norma a estas combinaciones tenemos todas las posibles combinaciones de reglas. Pero hasta aquí no tenemos ningún ajuste.

Si aplicamos el producto de Hadamard a una matriz de pesos con la misma dimensión y acotamos sus valores a  $[0, 1] \in \mathbb{N}$ , tenemos una manera de ajustar qué reglas son las más relevantes y cuáles no. Esta representación tiene un problema: estos dos valores definen una función escalón donde el error no se propaga al ser su gradiente 0. Sin embargo, si en lugar de un valor natural, el peso toma un valor real y le aplicamos una operación sigmoidal, el valor se mantendrá entre  $(0, 1) \in \mathbb{R}$  con la ventaja de que el gradiente no se estanca, y en un proceso posterior se pueden descartar las reglas cuyos valores superen ciertos rangos establecidos.

A la salida de la inferencia, tras aplicar la acumulación, disponemos de tantos valores borrosos como conjuntos borrosos tiene la salida.

### A.1.3 Bloque de defuzzificación

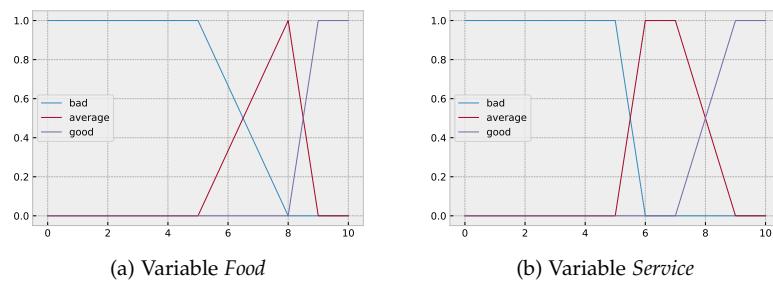
Este bloque tiene la particularidad de que no posee ninguna variable que ajustar. Se trata simplemente de una operación que toma una matriz bidimensional de rango  $(m, N_O)$  con los valores borrosos de salida y devuelve un vector de rango  $(m)$  con los valores en el dominio de la variable lingüística de salida.

### A.2 Ejemplo de ajuste de un sistema de control borroso

Se realizará el ajuste de un **sistema de control borroso** a partir de un conjunto de entrenamiento con unos valores determinados. Éstos se habrán obtenido a su vez de un **sistema de control borroso** de ejemplo para comprobar su correcto funcionamiento.

El ejemplo en concreto se trata del problema clásico de la asignación de propinas, donde la propina viene determinada en función de las variables calidad de la comida y calidad del servicio. Las particiones de las variables lingüísticas del controlador (denominadas *food* y *service*) se muestran en la Figura A.6.

Figura A.6: Particiones borrosas del controlador de ejemplo. Estas particiones junto con las reglas definirán la superficie de la función que modela nuestro controlador de ejemplo.

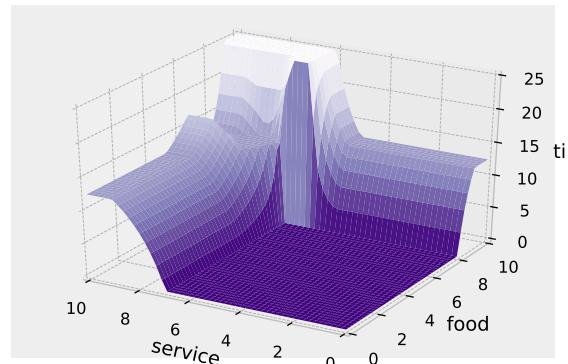


Las reglas que determinan su comportamiento son las siguientes:

$$\begin{aligned}
 &\text{service IS good} \rightarrow \text{tip IS high} \\
 &\text{food IS good} \rightarrow \text{tip IS high} \\
 &\text{service IS good} \wedge \text{food IS average} \rightarrow \text{tip IS low} \\
 &\text{service IS average} \wedge \text{food IS good} \rightarrow \text{tip IS high} \\
 &\text{service IS bad} \rightarrow \text{tip IS low} \\
 &\text{food IS bad} \rightarrow \text{tip IS low}
 \end{aligned} \tag{A.2}$$

El controlador es una función con dos variables que describen la superficie mostrada en la Figura A.7.

Figura A.7: Las reglas y las particiones definen esta función. De ella obtendremos un conjunto de puntos aleatorio para comprobar si nuestra representación de **sistema de control borroso** como grafo computacional es capaz de ajustarse a ésta con la técnica del descenso del gradiente.



Recogeremos una muestra aleatoria uniforme de 1500 puntos de entre los 62500 que conforman la representación de la superficie del

controlador de ejemplo. Posteriormente se creará un controlador borroso y se entrenará aplicando el algoritmo de descenso del gradiente ADAM [Kingma and Ba, 2014].

El proceso de entrenamiento consistirá en la ejecución del algoritmo de propagación del error durante 2500 epochs con una tasa de aprendizaje de 0,01, la cual se ha considerado suficiente para el ejemplo en cuestión.

Durante el entrenamiento, las superficies evolucionan hacia una representación aproximada de lo que es el controlador original. En la Figura A.8 podemos observar cómo evoluciona la superficie que representa el controlador ajustado.

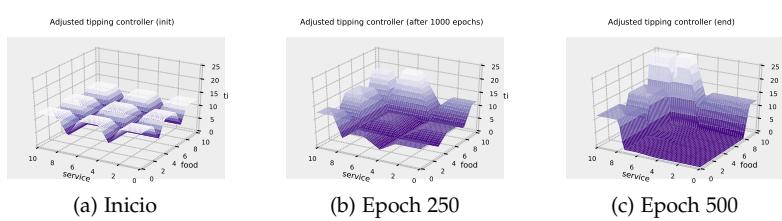


Figura A.8: Particiones borrosas del sistema de control borroso de ajustado. Estas figuras muestran la superficie del controlador ajustado (a) al comienzo del proceso de entrenamiento, (b) tras 250 iteraciones sobre el conjunto de entrenamiento y (c) al final del proceso del entrenamiento.

Este proceso demuestra que es posible obtener sistemas de control borroso ajustados a un patrón de datos con un grado alto de precisión. La ventaja de este método radica en que es posible por tanto explicar mediante reglas el por qué de las predicciones que hace el modelo, a diferencia de otras técnicas como las red neuronal artificial.



## B Visión general de ROS

Robot Operating System (ROS) es un framework para el desarrollo de aplicaciones relacionadas con automática y robótica. Abstacta una serie de servicios normalmente provistos por el sistema operativo además de ofrecer librerías y herramientas para facilitar el desarrollo de aplicaciones.

Está desarrollado desde el año 2007 y licenciado bajo la licencia [Berkeley Software Distribution \(BSD\)](#)<sup>1</sup>. Entre sus múltiples colaboradores destacan la Universidad de Stanford como desarrolladores originales, el instituto de investigación Willow Garage como desarrolladores principales desde el año 2008 y la [Open Source Robotics Foundation \(OSRF\)](#)<sup>2</sup> como actuales mantenedores y responsables de [ROS](#).

El framework de [ROS](#) se compone en realidad de dos partes:

- El **núcleo**, el cual entre otras cosas incluye todas las bibliotecas de utilidades y [APIs](#) para crear los nodos y las librerías del sistema a desarrollar, herramientas para gestionar el funcionamiento de éste y el nodo principal que se encarga de las comunicaciones entre los diferentes nodos.
- El **sistema de paquetes**, compuesto por toda la base de datos de nodos y herramientas desarrolladas para [ROS](#). Toda esta paquetería está desarrollada siguiendo el estándar que propone [ROS](#) para el desarrollo de utilidades, y está compuesto por paquetes de tipos de mensaje, controladores y drivers de dispositivos, codificación y decodificación de datos, etcétera.

El verdadero potencial de [ROS](#) es su infraestructura de comunicación, basada en el protocolo de comunicación [TCP/IP](#). Al estar basada en un patrón de comunicación *publish-subscribe* se logra no sólo mantener a los diferentes componentes del sistema muy débilmente desacoplados, sino que además permite su uso en un sistema distribuido a lo largo de varias máquinas independientes sin necesidad de modificar la implementación de los paquetes específicos del sistema desarrollado.

Hay una serie de conceptos dentro de [ROS](#) que es necesario conocer para poder comprender de qué manera un sistema está construido. Estos son:

<sup>1</sup> Se trata de una licencia de software libre permisivo, lo que entre otras cosas permite que el desarrollo de software enlazado al framework sin forzar a que éste también requiera una licencia libre.

<sup>2</sup> La [OSRF](#) fue una iniciativa lanzada desde el instituto de investigación Willow Garage en el año 2012.

- **Nodo.** Es el componente principal de un sistema desarrollado sobre [ROS](#). Los nodos tienen un funcionamiento en principio independiente de los demás, realizan las tareas para las que han sido programados, se comunican con componentes externos al sistema (e.g. hardware) y entre si a través de los mecanismos de comunicación ofrecidos por el [framework](#). [ROS](#) Proporciona un [Application Programming Interface \(API\)](#) para facilitar el desarrollo de nodos en los lenguajes C/C++ y [Python](#).
- **Mensaje.** Es cada uno de los paquetes de información enviado entre nodos. Es una estructura de datos definida como tuplas de la forma (nombre, tipo) donde los tipos pueden ser o bien los básicos definidos por ros (e.g. valores enteros, de coma flotante, cadenas o timestamps) o bien valores compuestos como listas de tipos básicos u otros mensajes, permitiendo así la creación de mensajes complejos a partir de otros más simples.
- **Topic.** Un topic es el “tipo de mensaje” que la infraestructura de comunicación basado en *publish-subscribe* de [ROS](#) usa para el envío y recepción de información entre nodos. Un nodo puede publicar mensajes en uno o más *topics* y suscribirse también a uno o más *topics*. De esta manera, todos los mensajes de un topic que envía un nodo serán recibidos por todos los nodos suscritos a dicho *topic*.
- **Service.** El mecanismo de comunicación basado en *publish-subscribe* es asíncrono y basado únicamente en el envío de información. Cuando la comunicación requiere ser síncrona o en modo *request-response* se hace uso de servicios. Un mismo servicio sólo puede ser ofrecido por un único nodo, ofreciendo un [API](#) de acceso al que se accederá de forma síncrona.
- **Master.** Este componente se lanza en cualquier sistema desarrollado en [ROS](#). Se trata de un nodo que se ocupa de establecer y gestionar la infraestructura de comunicaciones, ofrecer parámetros globales y en general de mantener coherente el sistema en funcionamiento.

## *Referencias*

- [Aghabayk et al., 2013] Aghabayk, K., Forouzideh, N., and Young, W. (2013). Exploring a local linear model tree approach to car-following. *Computer-Aided Civil and Infrastructure Engineering*, 28(8):581–593.
- [Ahmed, 1999] Ahmed, K. I. (1999). Modeling Drivers ' Acceleration and Lane Changing Behavior. *Transportation*, Ph.D:189.
- [Al-Shihabi and Mourant, 2001] Al-Shihabi, T. and Mourant, R. R. (2001). A framework for modeling human-like driving behaviors for autonomous vehicles in driving simulators. In *Proceedings of the fifth international conference on Autonomous agents*, pages 286–291. ACM.
- [Alexiadis et al., 2004] Alexiadis, V., Colyar, J., Halkias, J., Hranac, R., and McHale, G. (2004). The next generation simulation program. *ITE Journal (Institute of Transportation Engineers)*, 74(8):22–26.
- [Balmer et al., 2004] Balmer, M., Cetin, N., Nagel, K., and Raney, B. (2004). Towards truly agent-based traffic and mobility simulations. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 1:60–67.
- [Bando et al., 2013] Bando, T., Takenaka, K., Nagasaka, S., and Taniguchi, T. (2013). Unsupervised drive topic finding from driving behavioral data. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 177–182. IEEE.
- [Barlovic et al., 1998] Barlovic, R., Santen, L., Schadschneider, A., and Schreckenberg, M. (1998). Metastable states in cellular automata for traffic flow. *Eur. Phys. J. B*, 5:793–800.
- [Behrisch et al., 2011] Behrisch, M., Bieker, L., Erdmann, J., and Krajzewicz, D. (2011). Sumo—simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind.
- [Bender et al., 2015] Bender, A., Agamennoni, G., Ward, J. R., Worrall, S., and Nebot, E. M. (2015). An unsupervised approach for inferring driver behavior from naturalistic driving data. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3325–3336.

- [Bexelius, 1968] Bexelius, S. (1968). An extended model for car-following.
- [Bingham, 2001] Bingham, E. (2001). Reinforcement learning in neurofuzzy traffic signal control. *European Journal of Operational Research*, 131(2):232–241.
- [Brilon and Wu, 1999] Brilon, W. and Wu, N. (1999). Evaluation of cellular automata for traffic flow simulation on freeway and urban streets. *Traffic and Mobility*.
- [Buzsáki et al., 2012] Buzsáki, G., Llinas, R., Singer, W., Berthoz, A., and Christen, Y. (2012). *Temporal coding in the brain*. Springer Science & Business Media.
- [Casas et al., 2011] Casas, J., Perarnau, J., and Torday, A. (2011). The need to combine different traffic modelling levels for effectively tackling large-scale projects adding a hybrid meso/micro approach. *Procedia-Social and Behavioral Sciences*, 20:251–262.
- [Chaib-draa and Levesque, 1994] Chaib-draa, B. and Levesque, P. (1994). Hierarchical model and communication by signs, signals, and symbols in multi-agent environments. *on Modelling Autonomous Agents in a Multi- . . . .*
- [Chakroborty and Kikuchi, 2003] Chakroborty, P. and Kikuchi, S. (2003). Calibrating the membership functions of the fuzzy inference system: Instantiated by car-following data. *Transportation Research Part C: Emerging Technologies*, 11(2):91–119.
- [Chan et al., 2012] Chan, K. Y., Dillon, T. S., Singh, J., and Chang, E. (2012). Neural-Network-Based Models for Short-Term Traffic Flow Forecasting Using a Hybrid Exponential Smoothing and Levenberg-Marquardt Algorithm. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):644–654.
- [Chandler et al., 1958] Chandler, R. E., Herman, R., and Montroll, E. W. (1958). Traffic Dynamics: Studies in Car Following. *Operations Research*, 6(2):165–184.
- [Clymer, 2002] Clymer, J. (2002). Simulation of a vehicle traffic control network using a fuzzy classifier system. In *Proceedings 35th Annual Simulation Symposium. SS 2002*, pages 285–291. IEEE Comput. Soc.
- [Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.
- [Das and Bowles, 1999] Das, S. and Bowles, B. (1999). Simulations of highway chaos using fuzzy logic. *18th International Conference of the North American Fuzzy Information Processing Society - NAFIPS (Cat. No.99TH8397)*, pages 130–133.

- [Das et al., 1999] Das, S., Bowles, B. A., Houghland, C. R., Hunn, S. J., and Zhang, Y. (1999). Microscopic simulations of freeway traffic flow. In *Proceedings of the Thirty-Second Annual Simulation Symposium IEEE Computer Society*, pages 79–84. IEEE Comput. Soc.
- [De Haan and Keller, 2004] De Haan, P. and Keller, M. (2004). Modelling fuel consumption and pollutant emissions based on real-world driving patterns: the hbefa approach. *International journal of environment and pollution*, 22(3):240–258.
- [Dia, 2002] Dia, H. (2002). An agent-based approach to modelling driver route choice behaviour under the influence of real-time information. *Transportation Research Part C: Emerging Technologies*, 10(5):331–349.
- [Díaz Álvarez et al., 2018] Díaz Álvarez, A., Clavijo, M., Jiménez, F., Talavera, E., and Serradilla García (2018). Modelling the human lane-change execution behaviour through multilayer perceptrons and convolutional neural networks. *Transportation Research Part F: Psychology and Behaviour*.
- [Díaz Álvarez et al., 2014] Díaz Álvarez, A., Serradilla García, F., Naranjo, J. E., Anaya, J. J., and Jiménez, F. (2014). Modeling the driving behavior of electric vehicles using smartphones and neural networks. *IEEE Intelligent Transportation Systems Magazine*, 6(3):44–53.
- [Dijkstra, 1972] Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM*, 15(10):859–866.
- [Dingus et al., 2006] Dingus, T. A., Klauer, S. G., Neale, V. L., Petersen, A., Lee, S., Sudweeks, J., Perez, M., Hankey, J., Ramsey, D., Gupta, S., et al. (2006). The 100-car naturalistic driving study, phase ii-results of the 100-car field experiment. Technical report, United States. National Highway Traffic Safety Administration.
- [Directive, 2010] Directive, E. (2010). 40 / eu european parliament and of the council of 7 july 2010 on the framework for the deployment of intelligent transport systems in the field of road transport and for interfaces with other modes of transport. *L207/1*, 6.
- [Dougherty et al., 1993] Dougherty, M. S., Kirby, H. R., and Boyle, R. D. (1993). The use of neural networks to recognise and predict traffic congestion. *Traffic engineering & control*, 34(6):311–4.
- [Dresner and Stone, 2004] Dresner, K. and Stone, P. (2004). *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems : AAMAS 2004 : New York City, New York, USA : July 19-23, 2004*. IEEE Computer Society.
- [Du and Swamy, 2006] Du, K.-L. and Swamy, M. N. (2006). *Neural networks in a softcomputing framework*. Springer Science & Business Media.

- [Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.
- [Ehlert and Rothkrantz, 2001] Ehlert, P. a. M. and Rothkrantz, L. J. M. (2001). A reactive driving agent for microscopic traffic simulation. *Modelling and Simulation 2001*, pages 943–949.
- [Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.
- [Finin et al., 1994] Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. *Proceedings of the third international conference on Information and knowledge management - CIKM '94*, pages 456–463.
- [Fix and Armstrong, 1990] Fix, E. and Armstrong, H. (1990). Modeling human performance with neural networks. *1990 IJCNN International Joint Conference on Neural Networks*, pages 247–252 vol.1.
- [Fritzsche and Ag, 1994] Fritzsche, H.-t. and Ag, D.-b. (1994). A model for traffic simulation. *Traffic Engineering & Control*, 35(5):317–321.
- [Galis and Rao, 2000] Galis, A. and Rao, S. (2000). Application of agent technology to telecommunication management services. In *On The Way To Information Society*. IOS Press.,
- [Gazis et al., 1959] Gazis, D. C., Herman, R., and Potts, R. B. (1959). Car-Following Theory of Steady-State Traffic Flow. *Operations Research*, 7(4):499–505.
- [Gipps, 1981] Gipps, P. G. (1981). A behavioural car-following model for computer simulation.
- [Gipps, 1986] Gipps, P. G. (1986). A model for the structure of lane-changing decisions. *Transportation Research Part B*, 20(5):403–414.
- [Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.
- [Glorot et al., 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323.
- [Guériaud et al., 2015] Guériaud, M., Billot, R., El Faouzi, N.-E., Hassas, S., and Armetta, F. (2015). Multi-agent dynamic coupling for cooperative vehicles modeling. In *AAAI*, pages 4276–4277.
- [Halati et al., 1997] Halati, A., Lieu, H., and Walker, S. (1997). CORSIM-corridor traffic simulation model. *Traffic Congestion and Traffic Safety in the*.

- [Hatipkarasulu, 2002] Hatipkarasulu, Y. (2002). *A variable response time lag module for car following models using fuzzy set theory*. PhD thesis, Louisiana State University.
- [Hebb, 1949] Hebb, D. O. (1949). The organization of behavior: A neurophysiological approach.
- [Hidas, 2002] Hidas, P. (2002). Modelling lane changing and merging in microscopic traffic simulation. *Transportation Research Part C: Emerging Technologies*, 10(5-6):351–371.
- [Hinton, 2006] Hinton, G. E. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507.
- [Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- [Hopfield, 1982] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.
- [Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.
- [Hou et al., 2011] Hou, H., Jin, L., Niu, Q., Sun, Y., and Lu, M. (2011). Driver Intention Recognition Method Using Continuous Hidden Markov Model. *International Journal of Computational Intelligence Systems*, 4(3):386–393.
- [Howard et al., 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.
- [Hunt and Lyons, 1994] Hunt, J. G. and Lyons, G. D. (1994). Modelling dual carriageway lane changing using neural networks. *Transportation Research Part C*, 2(4):231–245.
- [Huval et al., 2015] Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., Cheng-Yue, R., Mujica, F., Coates, A., and Ng, A. Y. (2015). An Empirical Evaluation of Deep Learning on Highway Driving. *arXiv*, pages 1–7.
- [Imprailou et al., 2016] Imprailou, M.-I. M., Quddus, M., Pitfield, D. E., and Lord, D. (2016). Re-visiting crash-speed relationships: A new perspective in crash modelling. *Accident Analysis & Prevention*, 86:173–185.
- [Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456.

- [Jaworowski, 2004] Jaworowski, Z. (2004). Solar cycles, not  $\text{CO}_2$ , determine climate. *21ST CENTURY SCIENCE AND TECHNOLOGY*, 16(4):52–65.
- [Jia et al., 2003] Jia, H., Juan, Z., and Ni, A. (2003). Develop a car-following model using data collected by 'five-wheel system'. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 1:346–351.
- [Jin et al., 2016] Jin, J., Ma, X., Koskinen, K., Rychlik, M., and Kosonen, I. (2016). Evaluation of fuzzy intelligent traffic signal control (fits) system using traffic simulation. In *Transportation Research Board 95th Annual Meeting*, volume 16.
- [Jin, 2006] Jin, W. (2006). A kinematic wave theory of lane-changing vehicular traffic. eprint. *arXiv: math*, 503036.
- [K. et al., 1996] K., A., M., B.-A., H., K., and R., M. (1996). Models of freeway lane changing and gap acceptance behavior. *Transportation and traffic theory*, 13:501–515.
- [Kerner et al., 2008] Kerner, B., Klenov, S., and Brakemeier, A. (2008). Testbed for wireless vehicle communication: A simulation approach based on three-phase traffic theory. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 180–185. IEEE.
- [Khodayari et al., 2012] Khodayari, A., Ghaffari, A., Kazemi, R., and Braunstingl, R. (2012). A modified car-following model based on a neural network model of the human driver effects. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 42(6):1440–1449.
- [Kikuchi and Chakroborty, 1992] Kikuchi, S. and Chakroborty, P. (1992). Car-following model based on fuzzy inference system. *Transportation Research Record*, pages 82–82.
- [Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [Kiszka et al., 1985a] Kiszka, J. B., Kochańska, M. E., and Sliwińska, D. S. (1985a). The influence of some fuzzy implication operators on the accuracy of a fuzzy model-part i. *Fuzzy sets and systems*, 15(2):111–128.
- [Kiszka et al., 1985b] Kiszka, J. B., Kochańska, M. E., and Sliwińska, D. S. (1985b). The influence of some fuzzy implication operators on the accuracy of a fuzzy model-part ii. *Fuzzy sets and systems*, 15(3):223–240.
- [Klir and Yuan-Yu, 1997] Klir, G. J. and Yuan-Yu, H. (1997). *Fuzzy set theory: theory & applications*. na.
- [Kohonen, 1998] Kohonen, T. (1998). The self-organizing map. *Neurocomputing*, 21(1-3):1–6.

- [Krajzewicz et al., 2012] Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. (2012). Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138.
- [Krajzewicz et al., 2002] Krajzewicz, D., Hertkorn, G., Rössel, C., and Wagner, P. (2002). Sumo (simulation of urban mobility) – an open-source traffic simulation. In *Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM20002)*, pages 183–187.
- [Krauss et al., 1997] Krauss, S., Wagner, P., and Gawron, C. (1997). Metastable states in a microscopic model of traffic flow. *Physical Review E*, 55(5):5597–5602.
- [Kuge et al., 2000] Kuge, N., Yamamura, T., Shimoyama, O., and Liu, A. (2000). A Driver Behavior Recognition Method Based on a Driver Model Framework. *Structure*, 109(Idm):469–476.
- [Laval and Daganzo, 2006] Laval, J. A. and Daganzo, C. F. (2006). Lane-changing in traffic streams. *Transportation Research Part B: Methodological*, 40(3):251–264.
- [Lerner et al., 2010] Lerner, N., Jenness, J., Singer, J., Klauer, S., Lee, S., Donath, M., Manser, M., and Ward, N. (2010). An exploration of vehicle-based monitoring of novice teen drivers. *Final Report. NHTSA, Report No. DOT HS*, 811:333.
- [Liu and Li, 2013] Liu, R. and Li, X. (2013). Stability analysis of a multi-phase car-following model. *Physica A: Statistical Mechanics and its Applications*, 392(11):2660–2671.
- [Lofti A., 1965] Lofti A., Z. (1965). Fuzzy sets. *Journal of Information and Control*, 8(3):338–353.
- [Margolus, 1995] Margolus, N. (1995). Cam-8: a computer architecture based on cellular automata. *arXiv preprint comp-gas/9509001*.
- [Maye et al., 2011] Maye, J., Triebel, R., Spinello, L., and Siegwart, R. (2011). Bayesian on-line learning of driving behaviors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4341–4346. IEEE.
- [Mbede et al., 2004] Mbede, J. B., Ma, S., Toure, Y., Graefe, V., and Zhang, L. (2004). Robust neuro-fuzzy navigation of mobile manipulator among dynamic obstacles. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 5051–5057. IEEE.
- [McCarthy et al., 1956] McCarthy, J., Minsky, M., Rochester, N., and Shannon, C. (1956). Dartmouth conference. In *Dartmouth Summer Research Conference on Artificial Intelligence*.
- [McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

- [McDonald et al., 1997] McDonald, M., Wu, J., and Brackstone, M. (1997). Development of a fuzzy logic based microscopic motorway simulation model. *IEEE Conference on*.
- [Michon, 1985] Michon, J. A. (1985). A critical view of driver behavior models: what do we know, what should we do? In *Human behavior and traffic safety*, pages 485–524. Springer.
- [Minderhoud, 1999] Minderhoud, M. (1999). *Supported Driving:Impacts on Motorway Traffic Flow*. PhD thesis, Delft University of Technology.
- [Minski and Papert, 1969] Minski, M. L. and Papert, S. A. (1969). Perceptrons: an introduction to computational geometry. MA: MIT Press, Cambridge.
- [Muñoz et al., 2010] Muñoz, J., Gutierrez, G., and Sanchis, A. (2010). A human-like torcs controller for the simulated car racing championship. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 473–480. IEEE.
- [Munoz et al., 2001] Munoz, L., Gomes, G., Yi, J., Toy, C., Horowitz, R., and Alvarez, L. (2001). Integrated meso-microscale traffic simulation of hierarchical ahs control architectures. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 82–87. IEEE.
- [Nagel and Schreckenberg, 1992] Nagel, K. and Schreckenberg, M. (1992). A cellular automaton model for freeway traffic.
- [Nagel et al., 1998] Nagel, K., Wolf, D. E., Wagner, P., and Simon, P. (1998). Two-lane traffic rules for cellular automata: A systematic approach. *Physical Review E*, 58(2):1425–1437.
- [Naranjo et al., 2006] Naranjo, J. E., González, C., García, R., and De Pedro, T. (2006). Acc+ stop&go maneuvers with throttle and brake fuzzy control. *IEEE Transactions on intelligent transportation systems*, 7(2):213–225.
- [Naranjo et al., 2012] Naranjo, J. E., Jiménez, F., Serradilla, F. J., and Zato, J. G. (2012). Floating car data augmentation based on infrastructure sensors and neural networks. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):107–114.
- [Naranjo et al., 2007] Naranjo, J. E., Sotelo, M. A., Gonzalez, C., Garcia, R., and De Pedro, T. (2007). Using fuzzy logic in automated vehicle control. *IEEE intelligent systems*, 22(1):36–45.
- [Ng, 2004] Ng, A. Y. (2004). Feature selection, l<sub>1</sub> vs. l<sub>2</sub> regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM.
- [OICA, 2015] OICA (2015). Motorization rate 2014 – worldwide. <http://www.oica.net/category/vehicles-in-use/>. Último acceso en 2015-06-16.

- [Olaverri-Monreal et al., 2018] Olaverri-Monreal, C., Errea-Moreno, J., and Díaz-Álvarez, A. (2018). Implementation and evaluation of a traffic light assistance system based on v2i communication in a simulation framework. *Journal of Advanced Transportation*, 2018.
- [Oreskes, 2018] Oreskes, N. (2018). The scientific consensus on climate change: How do we know we're not wrong? In *Climate Modelling*, pages 31–64. Springer.
- [Osogami et al., 2012] Osogami, T., Imamichi, T., Mizuta, H., Morimura, T., Raymond, R., Suzumura, T., Takahashi, R., and Ide, T. (2012). Ibm mega traffic simulator. *IBM Res., Tokyo, Japan, IBM Res. Rep. RT0896*.
- [Pakkenberg and Gundersen, 1997] Pakkenberg, B. and Gundersen, H. J. (1997). Neocortical neuron number in humans: effect of sex and age. *The Journal of comparative neurology*, 384(2):312–20.
- [Panwai and Dia, 2007] Panwai, S. and Dia, H. (2007). Neural agent car-following models. *IEEE Transactions on Intelligent Transportation Systems*, 8(1):60–70.
- [Pipes, 1953] Pipes, L. (1953). An operational analysis of traffic dynamics. *Journal of applied physics*.
- [Poslad, 2007] Poslad, S. (2007). Specifying Protocols for Multi-Agent Systems Interaction. *ACM Transactions on Autonomous and Adaptive Systems*, 2(4, Article 15):25.
- [Qian, 1999] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151.
- [Quaassdorff et al., 2016] Quaassdorff, C., Borge, R., Pérez, J., Lumbreras, J., de la Paz, D., and de Andrés, J. M. (2016). Microscale traffic simulation and emission estimation in a heavily trafficked roundabout in madrid (spain). *Science of The Total Environment*, 566:416–427.
- [Ramón y Cajal, 1888] Ramón y Cajal, S. (1888). *Estructura de los centros nerviosos de las aves*. Madrid Nicolas Moya.
- [Ramón y Cajal, 1904] Ramón y Cajal, S. (1904). *Textura del Sistema Nervioso del Hombre y de los Vertebrados*, volume 2. Madrid Nicolas Moya.
- [Rasmussen, 1986] Rasmussen, J. (1986). *Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering*. North-Holland.
- [Reuschel, 1950] Reuschel, A. (1950). Fahrzeugbewegungen in der Kolonne. *Osterr. Ing. Archiv.*, 4(1):193–215.
- [Richter et al., 2016] Richter, S. R., Vineet, V., Roth, S., and Koltun, V. (2016). Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, pages 102–118. Springer.

- [Robinson et al., 2007] Robinson, A. E., Hammon, P. S., and de Sa, V. R. (2007). Explaining brightness illusions using spatial filtering and local response normalization. *Vision research*, 47(12):1631–1644.
- [Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, DTIC Document.
- [Russell et al., 2003] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (2003). *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River.
- [Rutkowski, 2008] Rutkowski, L. (2008). *Computational intelligence: methods and techniques*. Springer Science & Business Media.
- [Sasoh and Ohara, 2002] Sasoh, A. and Ohara, T. (2002). Shock Wave Relation Containing Lane Change Source Term for Two-Lane Traffic Flow. *Journal of the Physical Society of Japan*, 71(9):2339–2347.
- [Sekizawa et al., 2007] Sekizawa, S., Inagaki, S., Suzuki, T., Hayakawa, S., Tsuchida, N., Tsuda, T., and Fujinami, H. (2007). Modeling and recognition of driving behavior based on stochastic switched ARX model. *IEEE Transactions on Intelligent Transportation Systems*, 8(4):593–606.
- [Shiose et al., 2001] Shiose, T., Onitsuka, T., and Taura, T. (2001). Effective information provision for relieving traffic congestion. In *Proceedings Fourth International Conference on Computational Intelligence and Multimedia Applications. ICCIMA 2001*, pages 138–142. IEEE.
- [Shoham, 1993] Shoham, Y. (1993). Agent-oriented programming. *Artificial intelligence*, 60(1):51–92.
- [Siddique and Adeli, 2013] Siddique, N. and Adeli, H. (2013). *Computational intelligence: synergies of fuzzy logic, neural networks and evolutionary computing*. John Wiley & Sons.
- [Simonelli et al., 2009] Simonelli, F., Bifulco, G. N., and Martinis, V. D. (2009). Human-Like Adaptive Cruise Control Systems through a Learning Machine Approach. *Applications of Soft Computing*, pages 240–249.
- [Sparmann, 1978] Sparmann, U. (1978). Spurwechselvorgänge auf zweispurigen bab-richtungsfahrbahnen. *FORSCH STRASSENBAU U STRASSENVERKEHRSTECH* 263, 263.
- [Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.
- [Sugeno and Kang, 1988] Sugeno, M. and Kang, G. (1988). Structure identification of fuzzy model. *Fuzzy sets and systems*, 28(1):15–33.

- [Suzumura and Kanezashi, 2012] Suzumura, T. and Kanezashi, H. (2012). Highly scalable x10-based agent simulation platform and its application to large-scale traffic simulation. *Proceedings of the 2012 IEEE/ACM 16th*.
- [Takagi and Sugeno, 1993] Takagi, T. and Sugeno, M. (1993). Fuzzy identification of systems and its applications to modeling and control. In *Readings in Fuzzy Sets for Intelligent Systems*, pages 387–403. Elsevier.
- [Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*.
- [Toledo et al., 2003] Toledo, T., Koutsopoulos, H. N., and Ben-Akiva, M. (2003). Modeling Integrated Lane-Changing Behavior. *Transportation Research Record*, 1857(1):30–38.
- [Toledo et al., 2007] Toledo, T., Koutsopoulos, H. N., and Ben-Akiva, M. (2007). Integrated driving behavior modeling. *Transportation Research Part C: Emerging Technologies*, 15(2):96–112.
- [Tordeux et al., 2011] Tordeux, A., Lassarre, S., and Roussignol, M. (2011). A study of the emergence of kinematic waves in targeted state car-following models of traffic. <http://cybergeo.revues.org>.
- [Trask et al., 2015] Trask, A., Gilmore, D., and Russell, M. (2015). Modeling order in neural word embeddings at scale. *arXiv preprint arXiv:1506.02338*.
- [Turing, 1950] Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.
- [Van Hoorn et al., 2009] Van Hoorn, N., Togelius, J., Wierstra, D., and Schmidhuber, J. (2009). Robust player imitation using multiobjective evolution. In *2009 IEEE Congress on Evolutionary Computation*, pages 652–659. IEEE.
- [Van Ly et al., 2013] Van Ly, M., Martin, S., and Trivedi, M. M. (2013). Driver classification and driving style recognition using inertial sensors. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 1040–1045. IEEE.
- [Wagner et al., 1997] Wagner, P., Nagel, K., and Wolf, D. E. (1997). Realistic multi-lane traffic rules for cellular automata. *Physica A: Statistical Mechanics and its Applications*, 234(3–4):687–698.
- [Wegener et al., 2008] Wegener, A., Piórkowski, M., Raya, M., Hellbrück, H., Fischer, S., and Hubaux, J.-P. (2008). Traci: an interface for coupling road traffic and network simulators. In *Proceedings of the 11th communications and networking simulation symposium*, pages 155–163. ACM.

- [Wei et al., 2000] Wei, H., Lee, J., Li, Q., and Li, C. (2000). Observation-Based Lane-Vehicle Assignment Hierarchy: Microscopic Simulation on Urban Street Network. *Transportation Research Record*, 1710(1):96–103.
- [Weidemann and Reiter, 1992] Weidemann, R. and Reiter, U. (1992). Microscopic Traffic Simulation, The Simulation System-Mission. *University Karlsruhe, Germany*, 2:54.
- [Widrow and Hoff, 1960] Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. Technical report, STANFORD UNIV CA STANFORD ELECTRONICS LABS.
- [Wiedemann, 1974] Wiedemann, R. (1974). Simulation des strassenverkehrsflusses. *Institute for Traffic Engineering, University of Karlsruhe*.
- [Wolpert and Macready, 1997] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.
- [Wooldridge et al., 1995] Wooldridge, M., Jennings, N. R., Adorni, G., Poggi, A., Allen, J. F., Bates, J., Bell, J., BELNAP, N., PERLOFF, M., Bratman, M. E., Israel, D. J., Pollack, M. E., Brooks, R., Brooks, R. A., Bussmann, S., Demazeau, Y., Castelfranchi, C., Chaib-Draa, B., Moulin, B., Mandiau, R., Millot, P., Chang, E., Chapman, D., Chellas, B. F., Cohen, P. R., Levesque, H. J., Cohen, P. R., Perrault, C. R., Cutkosky, M., Engelmore, R., Fikes, R., Genesereth, M., Gruber, T., Mark, W., Tenenbaum, J., Weber, J., Downs, J., Reichgelt, H., Emerson, E. A., Halpern, J. Y., Fagin, R., Halpern, J. Y., Vardi, M. Y., Fisher, M., Gasser, L., Gasser, L., Braganza, C., Herman, N., Genesereth, M. R., Ketchpel, S. P., Georgeff, M. P., Greif, I., Guha, R. V., Lenat, D. B., Haas, A. R., Halpern, J. Y., Halpern, J. Y., Moses, Y., Halpern, J. Y., Vardi, M. Y., Hayes-Roth, B., Hewitt, C., Huang, J., Jennings, N. R., Fox, J., Jennings, N. R., JENNINGS, N. R., Jennings, N., Varga, L., Aarnts, R., Fuchs, J., Skarek, P., Kaelbling, L. P., Kraus, S., Lehmann, D., Kripke, S. A., Maes, P., Maes, P., McCabe, F. G., Clark, K. L., Mukhopadhyay, U., Stephens, L. M., Huhns, M. N., Bonnell, R. D., Müller, J. P., Pischel, M., Thiel, M., Newell, A., Simon, H. A., Norman, T. J., Long, D., PAPAZOGLOU, M. P., LAUFMANN, S. C., SELLIS, T. K., Perlis, D., Perlis, D., Perloff, M., Poggi, A., Reichgelt, H., Sacerdoti, E. D., Searle, J. R., Shoham, Y., Thomas, B., Shoham, Y., Schwartz, A., Kraus, S., Thomason, R. H., Varga, L., Jennings, N. R., Cockburn, D., Vere, S., Bickmore, T., Wavish, P., Graham, M., Weerasooriya, D., Rao, A., Ramamohanarao, K., and Wooldridge, M. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(02):115.
- [Wu et al., 2003] Wu, J., Brackstone, M., and McDonald, M. (2003). The validation of a microscopic simulation model: A methodological case study. *Transportation Research Part C: Emerging Technologies*, 11(6):463–479.

- [Wymann et al., 2013] Wymann, B., Espi, E., Guionneau, C., Dimitrakakis, C., and Sumner, A. (2013). TORCS : The open racing car simulator e. at <http://torcs.> . . . , pages 1–4.
- [Yang and Koutsopoulos, 1996] Yang, Q. and Koutsopoulos, H. N. (1996). A microscopic traffic simulator for evaluation of dynamic traffic management systems. *Transportation Research Part C: Emerging Technologies*, 4(3 PART C):113–129.
- [Zadeh, 1975] Zadeh, L. A. (1975). The concept of a linguistic variable and its application to approximate reasoning—i. *Information sciences*, 8(3):199–249.
- [Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.
- [Zhang et al., 2011] Zhang, J., Wang, F.-Y., Wang, K., Lin, W.-H., Xu, X., and Chen, C. (2011). Data-Driven Intelligent Transportation Systems: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1624–1639.
- [Zheng et al., 2013] Zheng, J., Suzuki, K., and Fujita, M. (2013). Car-following behavior with instantaneous driver-vehicle reaction delay: A neural-network-based methodology. *Transportation Research Part C: Emerging Technologies*, 36:339–351.
- [Zheng and McDonald, 2005] Zheng, P. and McDonald, M. (2005). Application of fuzzy systems in the car-following behaviour analysis. In *International Conference on Fuzzy Systems and Knowledge Discovery*, pages 782–791. Springer.



# *Índice alfabético*

- k*-medias, 68  
ADAS, 57  
algoritmos genéticos, 11, 67  
approaching, 63  
aprendizaje automático, 11, 15, 66  
autómata celular, 44, 46–50  
autoencoder, 14, 22
- back-propagation, 23, 24, 66, 68  
BDI (Belief-Desire-Intention), 41  
car-following, 47, 54, 60–64, 66–69, 78  
clustering, 14, 21, 68, 80  
conjunto borroso, 31, 32, 34–37, 67, 68, 85, 121–124  
control lateral, IX, 73  
courtesy yielding function, 64  
critical gap, 62, 63
- DBSCAN, 80  
deep belief networks, 14, 17  
deep learning, 22  
discretionary lane change, 62, 64, 68  
driver-vehicle object, 59  
driver-vehicle unit, 43, 50–52, 59, 83
- epoch, 85  
fast-forward, 67  
feed-forward, 21, 84, 96, 116  
free-flow, 62, 63, 66
- gap-acceptance, 62, 63  
GNU General Public License, 53, 54
- hard computing, 13  
hard-computing, 13
- inertial measurement unit, 68  
ITS, 1, 2, 57, 64–66, 114  
KITTI, 81
- lógica borrosa, 9, 11, 30–35, 38, 66–68  
lane-change, IX, XI, 5, 6, 48, 54, 59–64, 66, 67, 73–75, 78–81, 96, 102, 104–106, 108  
lane-selection, 61  
latent dirichlet model, 68  
LiDAR, 75, 76, 80, 93, 94, 96, 105, 112, 113  
local response normalization, 27  
Long-Short Term Memory, 22
- mandatory lane change, 62, 64, 68  
merging, 61  
micro-simulación, 5  
modelos ocultos de Markov, 68
- neuro-fuzzy, 68  
non free-lunch theorem, 15
- perceptrón multicapa, 66, 67, 81  
perceptron multicapa, IX, XI, 22, 23, 25, 27, 28, 83, 84, 87, 89, 97, 100, 113, 114
- procesamiento de lenguaje natural, 12
- Python, 52, 54, 55, 130
- red de convolución, IX, XI, 22, 28, 81, 97, 113, 114

red neuronal artificial, IX, XI, 5, 9, 10, 15, 17, 18, 21, 23, 29, 38, 66–69, 84, 127  
redes neuronales recurrentes, 14, 22, 67, 116  
ReLU, 17, 21, 88, 92  
RMSE, 23, 84, 101, 102  
  
sistema de control borroso, IX, XI, 32, 34–36, 50, 67–69, 81, 83–86, 89, 113, 121, 122, 124, 126, 127  
sistema experto, 11  
sistemas de recomendación, 11  
sistemas multiagente, IX, XI, 2, 4, 5, 41, 45, 48–51, 57, 58  
sobre-especialización, 28  
soft-computing, 13  
stop-and-go, 63  
sub-especialización, 28  
SUMO, 52–54, 73, 105, 114, 116  
Support Vector Machines, 68  
  
TORCS, 66  
TraaS, 55  
TraCI, 54, 55  
  
umbral perceptual, 61, 62  
  
v2i, 50, 51  
v2v, 50, 51

## Cómo citar esta tesis

Si deseas citar esta tesis, lo primero gracias. Me alegra de que te sirva para tu investigación. Si lo deseas, incluye el siguiente código en bibtex:

```
@phdthesis{diazalvarez2018-phd,  
    author = {Alberto Díaz Álvarez},  
    abstract = {XXX},  
    pages = {XXX},  
    title = {Modelado de comportamiento de conductores con técnicas de Inteligencia Computacional},  
    url = {XXX},  
    year = {5 de mayo de 2018}  
}
```

## Acerca del código fuente

La presente tesis lleva consigo numerosas horas de programación y, por tanto, muchísimas líneas de código. Éste se encuentra en formato electrónico como datos adjuntos a la memoria y no como capítulo o anexo a ésta, una forma más manejable para su consulta y a la vez respetuosa con el medio ambiente. No obstante sí es posible que existan pequeños fragmentos de código para apoyar explicaciones. En caso de necesitar los fuentes y no estar disponibles los datos anexos a la memoria, puedes contactar directamente conmigo en [alberto.diaz@upm.es](mailto:alberto.diaz@upm.es).