

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 en Inteligencia Artificial  
**Dr. Felipe Jiménez Alonso**  
Doctor en Ingeniería Mecánica



hoja según tribunal



Alberto Díaz Álvarez

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

Tesis doctoral, 5 de abril de 2018

Revisores: Rev1, Rev2 y Rev3

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.





*De momento nada de dedicatorias, mejor TODOs:*

- *Cambiar el glosario para que esté en español.*

*Si pasamos la tesis a inglés, pasamos también los términos del glosario.*



## *Resumen*

Aquí el abstract en español



## *Abstract*

Aquí el abstract en inglés



# *Índice general*

<b>Introducción</b>	<b>25</b>
Motivación . . . . .	26
Objetivos . . . . .	28
Estructura de la tesis . . . . .	30
<b>I Estado de la cuestión</b>	<b>31</b>
<b>Inteligencia Computacional</b>	<b>33</b>
De Inteligencia Artificial a Inteligencia Computacional . . . . .	33
El rol del aprendizaje en la Inteligencia Computacional . . . . .	37
Artificial Neural Networks . . . . .	41
Lógica Difusa . . . . .	54
El paradigma de los Agentes Inteligentes . . . . .	62
<b>Simulación de tráfico</b>	<b>69</b>
Clasificación de simuladores de tráfico . . . . .	70
Modelos de microsimulación . . . . .	73
Software de simulación . . . . .	77
Entorno seleccionado: SUMO . . . . .	79
<b>Modelos de comportamiento</b>	<b>83</b>
Comportamientos modelados . . . . .	85
Modelado de conductores clásico . . . . .	86
Modelado basado en . . . . .	90
Papers a añadir o al menos relevantes . . . . .	94

<b>II Modelos de comportamiento</b>	<b>95</b>
<b>Metodología</b>	<b>97</b>
Perspectiva general del modelo de conducción . . . . .	97
Extracción de datos de conducción . . . . .	99
Preparación de los datos . . . . .	102
Entrenamiento de modelos . . . . .	105
<b>Comportamiento longitudinal</b>	<b>107</b>
Modelo Sistema de Control Difuso . . . . .	108
Modelo Perceptrón Multicapa . . . . .	109
Comparación entre modelos . . . . .	111
<b>Comportamiento de cambio de carril</b>	<b>113</b>
Descripción de los datasets . . . . .	114
Modelo Perceptrón Multicapa . . . . .	119
Modelo Convolutional Neural Network (CNN) . . . . .	120
Comparación entre modelos . . . . .	122
<b>Modelos específicos de conductores</b>	<b>123</b>
<b>Implementación en simulador</b>	<b>125</b>
Implementación en entorno de simulación . . . . .	125
Validación del modelo . . . . .	125
<b>III Resultados y conclusiones</b>	<b>127</b>
<b>Resultados</b>	<b>129</b>
<b>Conclusiones</b>	<b>131</b>
Aportaciones . . . . .	131
Futuras líneas de investigación . . . . .	131
<b>Ajuste de controlador difuso basado en descenso del gradiente</b>	<b>133</b>
Sistema de Control Difuso como grafo computacional . . . . .	134
Ejemplo de ajuste de un controlador difuso . . . . .	139

<b>Visión general de ROS</b>	<b>141</b>
<b>Sistemas desarrollados</b>	<b>143</b>
Outrun . . . . .	143
Modelos de comportamiento . . . . .	143
Otro software desarrollado . . . . .	143



# Índice de figuras

1.	Censo de conductores según género y edad . . . . .	29
2.	Experimento mental de la <i>Habitación China</i> , por John Searle. . . . .	34
3.	Diferentes objetivos perseguidos por la Inteligencia Artificial. . . . .	36
4.	Separación clásica de conjuntos de datos en tareas de Machine Learning. . . . .	38
5.	Ciclo de aplicación de soluciones basadas en Inteligencia Computacional . . . . .	41
6.	Capacidad de los modelos en función de la cantidad de datos . . . . .	41
7.	Ilustración de una sección del neocórtez humano . . .	42
8.	Modelo de neurona artificial de McCulloch y Pitts . .	42
9.	Funciones de activación: sigmoidal y tangente hiperbólica. . . . .	43
10.	Funciones de activación: ReLU y Leaky-ReLU. . . . .	44
11.	Diferencias entre topologías de redes <i>feed-forward</i> y <i>recurrentes</i> . . . . .	46
12.	Estructura general de una Convolutional Neural Network	49
13.	Descripción de la capa de convolución . . . . .	50
14.	Ejemplo de las operaciones de <i>pooling</i> . . . . .	51
15.	Ejemplo de la operación de dropout en tres epochs sucesivos. . . . .	53
16.	Ejemplo de grafo computacional. . . . .	53
17.	Derivadas parciales sobre un grafo computacional. . .	54
18.	Gráfica de funciones de pertenencia triangular y trapezoidal. . . . .	56
19.	<i>t</i> -normas del mínimo y del producto algebraico. . . . .	57

20.	<i>t</i> -normas del máximo y de la suma algebraica. . . . .	57
21.	Diferencias entre <i>modus ponens</i> clásico y <i>modus ponens generalizado</i> . . . . .	58
22.	Esquema de un sistema de control difuso . . . . .	59
23.	Representación de conjuntos de reglas como matriz . .	60
24.	Ejemplo de reglas en controlador Takagi-Sugeno . . . .	62
25.	Esquema de agente y sus propiedades. . . . .	63
26.	Arquitectura básica de un agente. . . . .	65
27.	Diferencias entre un agente sin y con modelo de entorno.	65
28.	Arquitecturas de agente según su comportamiento. . .	66
29.	Diferencias entre colaboración y competitividad de agentes. . . . .	68
30.	Aspectos medibles del tráfico . . . . .	70
31.	Clasificación de simuladores según granularidad . . .	71
32.	Alternativas a la clasificación por granularidad . . . .	72
33.	Ejemplo de simulador basado en Autómata Celular . .	72
34.	Ejemplo de modelo lineal en un espacio continuo . . .	73
35.	Ejemplo de efecto de ondas de choque en simulación de tipo Nagel-Schreckenberg . . . . .	74
36.	Simulación de comportamiento en intersección basada en un . . . . .	75
37.	Captura de pantalla del simulador MovSim . . . . .	76
38.	Características obligatorias y deseables del simulador a elegir . . . . .	77
39.	Captura de pantalla del simulador SUMO . . . . .	79
40.	Ejemplo de forma de envío de mensajes a través de TraCI	80
41.	Arquitectura de la plataforma TraaS . . . . .	81
42.	Los tres niveles jerárquicos de conducción según [Michon, 1985]	83
43.	Diferentes modelos de aceleración . . . . .	85
44.	Operaciones involucradas en el proceso de cambio de carril . . . . .	85
45.	Nomenclaturas a usar en los modelos de conducción .	86
46.	Evolución de los tres tipos generales de modelo de <i>car-following</i> . . . . .	87

47. Ejemplo de cambio de carril obligatorio frente a cambio de carril discrecional . . . . .	88
48. Efecto de la distancia en el tipo de cambio de carril según el modelo de [Gipps, 1986] . . . . .	89
49. Estructura del modelo de comportamiento propuesto por [Toledo et al., 2007] . . . . .	90
50. Principales áreas de aplicación de la en los Sistema Inteligente de Transporte (ITS, Intelligent Transport System) . . . . .	91
51. La inexactitud se tiene en cuenta en los modelos de la . . . . .	92
52. Ejemplo de aproximación <i>neuro-fuzzy</i> al control de señales de tráfico . . . . .	93
53. Esquema general del modelo de conductor planteado . . . . .	98
54. El vehículo utilizado en los ensayos realizados. Se trata de un Mitsubishi iMiEV instrumentado con un LIDAR anclado en la baca superior, un GPS anclado en el techo, un puerto de acceso directo al Bus CAN y una cámara Microsoft Kinect tras el espejo retrovisor. . . . .	99
55. El Light Detection and Ranging (LiDAR) VLP-16 de Velodyne tiene un FOV horizontal de 360° y vertical de ±15°, permitiendo una captura de todo el entorno circundante a una frecuencia de hasta 20 Hz. Fuente: <a href="http://velodynelidar.com/vlp-16.html">http://velodynelidar.com/vlp-16.html</a> . . . . .	100
56. El GPS permite el posicionamiento 3d sobre en la tierra gracias a la comunicación unidireccional de satélites situados en órbita. Fuente: Wikimedia Commons. . . . .	100
57. 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: <a href="http://www.can232.com/">http://www.can232.com/</a> . . . . .	101
58. La cámara Kinect desarrollada por Microsoft ofrece imágenes a color a una velocidad de 30 fps con una resolución de 640 px × 480 px. . . . .	101
59. Dos recorridos para la captura de datos de conducción . . . . .	102
60. Perfiles de aceleración a ajustar por los modelos. Conjuntos de entrenamiento y de test . . . . .	107
61. Evolución del error en test de los controladores difusos ajustados . . . . .	108
62. Evolución del error en entrenamiento en los Perceptrón Multicapa para las arquitecturas seleccionadas . . . . .	110

63.	Evolución del error en el conjunto de test durante el entrenamiento . . . . .	110
64.	Comparación del perfil de aceleración real y el inferido por los modelos entrenados . . . . .	111
65.	Comparación entre los dos tipos de modelo longitudinal	111
66.	Perfiles de cambios de carril en los conjuntos de entrenamiento y test . . . . .	113
67.	Ejemplo de un mapa de profundidad . . . . .	115
68.	Ejemplo de la técnica de <i>mirroring</i> . . . . .	117
69.	Ejemplo de la técnica de <i>shaking</i> . . . . .	117
70.	Momento $t_i$ en el conjunto de datos . . . . .	118
71.	Evolución del error en test de los controladores difusos ajustados . . . . .	119
72.	Evolución del error en entrenamiento en los Perceptrón Multicapa para las arquitecturas seleccionadas . . . . .	121
73.	Evolución del error en el conjunto de test durante el entrenamiento . . . . .	121
74.	Comparación del perfil de aceleración real y el inferido por los modelos entrenados . . . . .	122
75.	Comparación entre los dos tipos de modelo longitudinal	122
76.	Ejemplo de operación en el bloque de fuzzificación . .	134
77.	Funciones de pertenencia definidas como posición y desplazamientos. . . . .	135
78.	Grafo computacional de la función de pertenencia para la línea ascendente . . . . .	135
79.	Grafo computacional de la función de pertenencia para el trapecio . . . . .	135
80.	Representación de una partición difusa para su ajuste .	136
81.	Ejemplo de operación de fuzzificación como grafo computacional . . . . .	137
82.	Producto cartesiano de variables difusas de entrada .	138
83.	Particiones difusas del controlador de ejemplo . . . .	139
84.	Superficie de la función que modela el controlador difuso de ejemplo . . . . .	140
85.	Evolución del controlador difuso de acuerdo al conjunto de datos extraido del . . . . .	140

86. Descripción de los componentes de un sistema desarrollado para Robot Operating System (ROS). . . . . 142



## *Índice de cuadros*

1.	Tabla comparativa de los simuladores seleccionados . . . . .	79
2.	Resumen de los indicadores obtenidos tras los recorridos	104
3.	Descripción de los conjuntos de datos . . . . .	108
4.	Resumen de las arquitecturas Sistema de Control Difuso para el modelo longitudinal . . . . .	108
5.	Resumen de las arquitecturas Perceptrón Multicapa para el modelo longitudinal . . . . .	109
6.	Descripción de los conjuntos de datos . . . . .	118
7.	Resumen de las arquitecturas Perceptrón Multicapa para el modelo de cambio de carril . . . . .	119
8.	Resumen de las arquitecturas Perceptrón Multicapa para el modelo longitudinal . . . . .	120



# *Introducción*

La **Inteligencia Artificial (AI)** 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 y de muy poca, debido a las altas expectativas que generan los nuevos avances 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 **AI** es común diferenciar dos tipos de aproximaciones a la hora de representar el conocimiento: el enfoque **clásico**, que postula que el conocimiento como tal se puede reducir a un conjunto de símbolos con operadores para su manipulación, y el enfoque de la **Inteligencia Computacional (IC)**, 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 “emerja” de la interacción de éstos.

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 **AI** clásica y la **bottom-up** la típica usada en la **IC**. Revisaremos las diferencias entre conceptos de diferentes autores en el capítulo **Inteligencia Computacional**.

Uno de los campos de aplicación es el de los **ITS**. Éstos se definen como un conjunto de aplicaciones orientadas a gestionar el transporte en todos sus aspectos (e.g. conducción eficiente, diseño de automóviles, 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 [par, 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 tarea 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 (e.g. el consumo estimado para una ruta en concreto). Sin embargo, la evaluación en algunos casos puede no ser posible debido a limitaciones como, por ejemplo, el tiempo, el dinero o la peligrosidad del escenario.

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 complejas, alejadas de la realidad, además con pocas o ninguna opciones 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 [IC](#). 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.

## 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 arreglarlo, 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<sup>1</sup>.

<sup>1</sup> Uno puede argumentar que el parque automovilístico se recicla con nuevos vehículos eléctricos categorizados “de consumo o”. La triste realidad es que estos vehículos consumen la electricidad generada actualmente de una mayoría de centrales de combustibles fósiles y nucleares. Además, mientras que en países desarrollados el crecimiento ha sido en torno al 4-7%, en países subdesarrollados, donde no existe aun infraestructura para la recarga de vehículos eléctricos, dicho crecimiento ha superado el 120%.

- 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.
- La emisión de gases está correlacionada con el aumento de la temperatura del planeta. Con el ritmo actual, dependiendo de la fuente estamos o bien llegando obien ya hemos sobrepasado un punto de no retorno con consecuencias catastróficas para la vida en el planeta.
- La conducción eficiente afecta directamente a factores correlacionados con el número de accidentes de tráfico. Un factor de sobra conocido es el de la velocidad, relacionado no sólo con el número sino con la gravedad de los accidentes ([Imprilou 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 centrarse en el problema de cómo hacer de la conducción una actividad más eficiente y segura. Por ello, la **conducción eficiente** o *eco-driving* se define como la aplicación de una serie de reglas de conducción con el objetivo de reducir el consumo de combustible, independientemente del tipo (e.g. electricidad, gasolina, gas natural, ...).

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 claro 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 fácil, dado que su comportamiento se ve condicionado por numerosos 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 en una situación, únicamente porque su comportamiento ha sido condicionado por las malas reacciones de otros conductores 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 de 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.

### *Objetivos*

El objetivo de esta tesis doctoral es la de demostrar las hipótesis 1 y 2, quedando dicha demostración dentro de los límites impuestos por los supuestos y restricciones indicados más adelante.

**Hipótesis 1 (H<sub>2</sub>):** *La aplicación de técnicas pertenecientes al campo de la IC sobre datos de conductores reales para la obtención de 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 IC con datos extraídos de un entorno de microsimulación de espacio continuo y tiempo discreto basado en sistemas multiagentes permitirá modelar, de manera fiel a la realidad, el comportamiento de conductores reales.*

Por tanto, el objetivo de la tesis es el de simular el comportamiento de conductores en entornos de micro-simulación a partir de su comportamiento en entornos reales usando técnicas de IC. Para ello se consideran los siguientes objetivos específicos:

- Estudiar y aplicar técnicas de la IC sobre el área de la conducción.
- Realizar un estudio naturalista de conducción <sup>2</sup> sobre conductores reales para:

<sup>2</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 velocidad de captura 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.

1. Generar modelos personalizados de conductor a partir de los datos de conducción obtenidos.
  2. Aplicar modelos de conductores a entornos de simulación multagiente.
  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.

### 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.
- Los datos de los que extraer el comportamiento se corresponderán con lecturas realizadas durante el día, con buena visibilidad y sin lluvia.
- El tipo de vehículo sobre el que modelar el comportamiento será el *utilitario*.
- El conductor a modelar pertenecerá al grupo más representativo de conductores. Esto se corresponde con varón de 35 a 39 años (ver figura 1).

### Restricciones

- Los modelos generados se corresponderán a entornos urbanos.
- Reduciremos el comportamiento del conductor a los de circulación en línea y en cambio de carril <sup>3</sup>.
- La resolución máxima del modelo creado será de 10 Hz.
- En el caso de los modelos que hacen uso de *Artificial Neural Network*, no se pueden explicar las razones del comportamiento inferido.

<sup>3</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**.

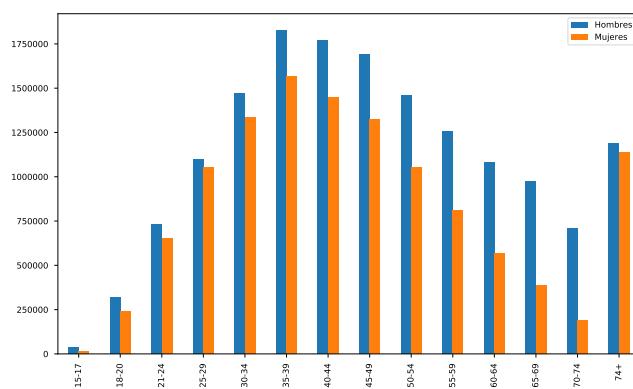


Figura 1: Último censo de conductores según género segmentado por edades. Fuente: Dirección General de Tráfico ([dgt.es](http://dgt.es)).

### *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, donde en los capítulos ??, ?? y ?? se introduce el problema a resolver propuesto en la hipótesis, se desarrolla el modelo de conducción personalizado y se muestran la configuración de las simulaciones.

Por último en los capítulos **Resultados** y **Conclusiones** se exponen los resultados y las conclusiones respectivamente extraídos de la tesis. Además, tras las conclusiones se indican una serie de posibles líneas futuras de trabajo consideradas interesantes tras la realización de la tesis.

## Estado de la cuestión



# *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 qué es la **Inteligencia Computacional (IC)**. Dependiendo del autor, se la considera desde un sinónimo de la **Inteligencia Artificial (AI)** hasta un campo completamente diferenciado.

En esta tesis hablaremos desde el punto de vista mayoritario, la **IC** como rama de la **AI** que agrupa 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.

Por ello, la primera sección del capítulo ofrecerá una visión histórica de la aparición del concepto para justificar el por qué de esta definición. El resto del capítulo introducirá la importancia que tiene el aprendizaje dentro de este área, describirá las técnicas de **Redes Neuronales Artificiales (ANNs, Artificial Neural Networks)** y la **Lógica Difusa** necesarias para sentar las bases del posterior desarrollo de la tesis y dará nociones de qué son los agentes y por qué este punto de vista es útil para nosotros.

## *De Inteligencia Artificial a Inteligencia Computacional*

Es difícil precisar el comienzo del interés del ser humano por la 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 el 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>4</sup>. Estas ideas saltaron al mundo

<sup>4</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. [y Cajal, 1888] y [Ramón and Cajal, 1904]).

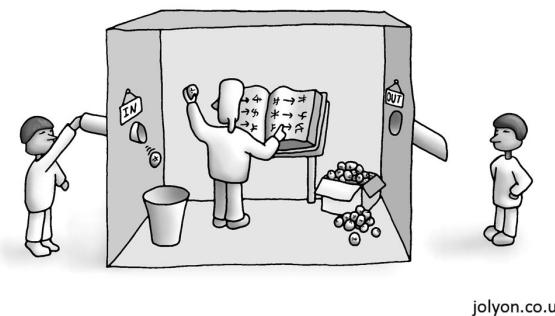
Figura 2: La *Habitación China* de John Searle es un experimento mental por el que se trata de demostrar la invalidez del Test de Turing. Partiendo de un Test de Turing donde la máquina ha aprendido a hablar chino. Reemplazamos la máquina por un humano sin idea de chino pero con un manual de correspondencias de ideogramas. Cuando una persona le manda mensajes en chino, esta otra responde usando el manual, por lo que podemos afirmar que la persona, y por tanto la máquina, no saben chino.

<sup>5</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>6</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>7</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), 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>8</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.



jolyon.co.uk

de la computación hacia mediados del siglo XX, cuando Warren S. McCulloch y Walter Pitts publicaron su trabajo “*A logical calculus of the ideas immanent in nervous activity*” [McCulloch and Pitts, 1943]<sup>5</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>6</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>7</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 el área nació con entidad propia tras acuñarlo John McCarthy como **Inteligencia Artificial**.

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 permitían 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*” [Minsky 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 **ANNs** 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>8</sup>.

El interés por el campo volvió de nuevo a principios de los 80 con la aparición en escena de los primeros *Sistemas Expertos*, los cuales se consideran como el primer caso de éxito en la *AI* ([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 empezaron a perder interés frente al nuevo avance del conexionismo <sup>9</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 de el primero.

Junto con el resurgir del conexionismo, otras técnica alineadas como la *Lógica Difusa* o los *Algoritmos Genéticos (GAs, Genetic Algorithms)* también ganaban popularidad, y entre ellas retroalimentaban los exitos 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 *AI* (e.g *Aprendizaje Automático (ML, Machine Learning)*, *Sistema de Recomendación (RS, Recommender System)*, o *Procesamiento de Lenguaje Natural (NLP, Natural Language Processing)*). Es evidente que la *AI* 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 *AI*. El resumen se puede observar en la figura 3.

Volviendo al tema de 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 y que surje de la propia *AI* parece lógico mantenerla como un subconjunto y no como un nuevo campo del conocimiento humano. Sin embargo, algunos autores abogan por que la *IC* es un campo diferenciado de la *AI*.

Podemos definir la *Inteligencia Computacional* como la “rama de la *AI* que aporta soluciones a tareas específicas de forma inteligente a partir

<sup>9</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 intelectual, emerge de modelos a más bajo nivel. Por ello, algunas voces se alzan contra lo que se consideraba el *enfoque incorrecto* de la *AI*. 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.

*del aprendizaje mediante el uso de datos experimentales".* A diferencia de la aproximación clásica de la **AI**, se buscan aproximaciones a las soluciones y no las soluciones exactas. Esto es debido a que muchos problemas son de naturaleza compleja, ya sea por la relación entre sus múltiples variables, a la falta de información o a la imposibilidad de traducirlos a lenguaje binario.

Figura 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].

<b>Thinking Humanly</b>	<b>Thinking Rationally</b>
"The exciting new effort to make computers think ... <i>machines with minds</i> , in the full and literal sense." (Haugeland, 1985)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985)
"[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Hellman, 1978)	"The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
<b>Acting Humanly</b>	<b>Acting Rationally</b>
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990)	"Computational Intelligence is the study of the design of intelligent agents." (Poole et al., 1998)
"The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"AI ... is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

<sup>10</sup> <http://cis.ieee.org/>

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>10</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 IC."

Durante estos años ganaba popularidad también el concepto del **Soft Computing** en contraposición con el **Hard Computing**. 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 **IC** algunos autores consideran ambos términos equivalentes. Nosotros consideramos que el **Soft Computing** es un punto de vista de la computación a diferencia de la **IC**, la cual es un área de específica dentro de la **AI** hace uso de métodos incluidos en el concepto **Soft Computing**.

Hoy en día la **IC** 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.

**Hard Computing** y **Soft Computing** son la forma de referirse a la computación convencional frente al **Soft Computing**. 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 tiende a llegar a soluciones aproximadas de manera más rápida. Se basa en modelos aproximados, emergencia de algoritmos y modelos estocásticos.

## *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 **IC** 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 **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.** Supongamos 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, es decir el *algoritmo de entrenamiento* se encargaría 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 brutos sus relaciones y extrae patrones, sin 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 mapas autoorganizados (SOM), los autoencoders o las redes de creencia profunda (DBN de 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 pertenecientes a ambos paradigmas en forma de aproximación híbrida 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 la optimización es éstos.

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 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 determinado conjunto) y de **regresión**, esto es, ajustar las salidas de un modelo para ajustarse lo máximo posible al valor real del sistema modelado.

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

La terminología general que se usa en la **IC** no es demasiado compleja (con la salvedad de los nombres de técnicas y algoritmos). La Figura 4 muestra un esquema de esta terminología.

Figura 4: Los diferentes tipos de conjuntos de datos existentes.



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>11</sup>.

Para problemas que requieren soluciones del área del **ML**, este conjunto se suele partir en varios subconjuntos <sup>12</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. Durante el entrenamiento, dependiendo del modelo o modelos que se están explorado, lo más

<sup>11</sup> Desgraciadamente este 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>12</sup> Evidentemente disjuntos.

común es realizar una serie de modificaciones en metavariables 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 ajuste 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 metavariables a lo que le dicta el conjunto de validación. En este punto surge el conjunto de test o **test set**<sup>13</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*.

### *Problemas del aprendizaje en la Inteligencia Computacional*

El entrenamiento de modelos en la IC adolece de dos principales problemas que, además, no tienen una solución general (*non free-lunch theorem* [Wolpert and Macready, 1997]) ya que dependen tanto de la estructura de los datos sobre los que vamos a tratar como de los hiperparámetros<sup>14</sup> del modelo. Estos son la sobreespecialización y la subespecialización<sup>15</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 **sobre-especialización**, 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 el problema y por lo tanto generaliza demasiado. Hablaremos de casos concretos y soluciones más adelante.

### *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 definirá los eslóganes de todas las empresas del ámbito en el que se mueven. Palabras con fuerza como Big Data,

<sup>13</sup> En la literatura existe cierta confusión entre estos dos términos. Históricamente el conjunto de dato 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 evaluar el 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>14</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>15</sup> En la literatura también se habla de *high variance* o *over-fitting* y de *high bias* o *under-fitting* para referirse a la sobre-especialización y subespecialización respectivamente.

Cloud Computing, Web Services o SaaS que visten muy bien logos de corporaciones y frases de consultores vendehúmos, pero que lo más normal es que sean conceptos ya existentes como proceso de datos masivos, computación distribuida o ejecución remota.

Al margen de esta pequeña crítica, 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*, 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.

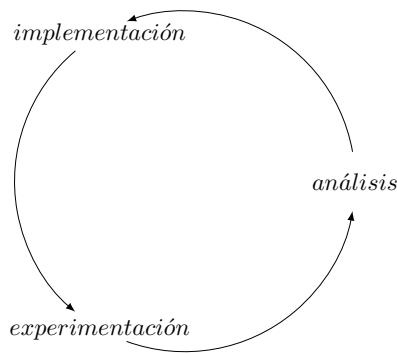
El *deep learning* es una de las palabras con la que últimamente se inunda la prensa, pero lo cierto es que desde la aparición del término, parece que los avances dentro de éste no tienen límites. En esta tesis se considera al *deep learning* a la nueva generación de enfoques en la que confluyen varios 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 proporcional a una mayor velocidad en el ciclo de experimentación de modelos (ver Figura 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.
- **Algoritmos más eficientes.** Nuevos elementos como las funciones de activación *Rectified Linear Unit (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 redonda, 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, redes de creencia, ...), donde se origina este término, se refiere a una red con más capas de lo habitual<sup>16</sup> (normalmente más de dos o tres). Este tipo de capas, históricamente ha sido más difícil de entrenar, debido entre otras cosas a las técnicas de entrenamiento (donde los

<sup>16</sup> De aquí surge el nombre de **shallow network** o red superficial, en contraposición a **deep network** o red profunda.

Figura 5: Ciclo de aplicación de soluciones basadas en **Inteligencia Computacional**. Los sistemas inteligentes se conciben, se implementan y se prueba. En la fase de experimentación se determina cómo de bien o de mal lo está haciendo y cuales son las decisiones a tomar para el nuevo ciclo. Cuandto 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.



parámetros tendían a diluirse según se aumentaba el número de capas entre la entrada y la salida) o a la disponibilidad de conjuntos de datos, los cuales al no ser muy grandes, las redes grandes tendían a sobre especializarse. La Figura 6 introduce, con una ilustración un tanto informal, las capacidades del deep learning frente a las de los modelos entrenados antes de esta época.

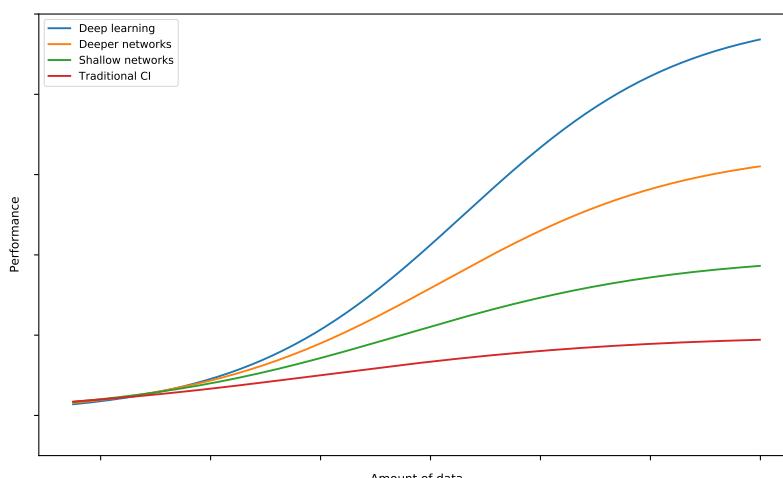


Figura 6: La enorme cantidad de datos junto con la capacidad computacional y la mejora de las técnicas de entrenamiento hacen posible que en la actualidad, con las técnicas asociadas al contexto del deep learning, los modelos entrenados sean más eficientes inteligentes. Imagen adaptada de la charla *How scale is enabling deep learning* de Andrew Y. Ng, accesible <https://youtu.be/LcfLo7YP804>.

Los factores antes mencionados han hecho posible la operación sobre redes más grandes y profundas con cantidades de ordenes de magnitud muy superiores. El resultado en la actualidad es que tenemos modelos que mejoran mucho los modelos anteriores, y no parece haber cota superior en su capacidad de aprendizaje <sup>17</sup>.

### *Artificial Neural Networks*

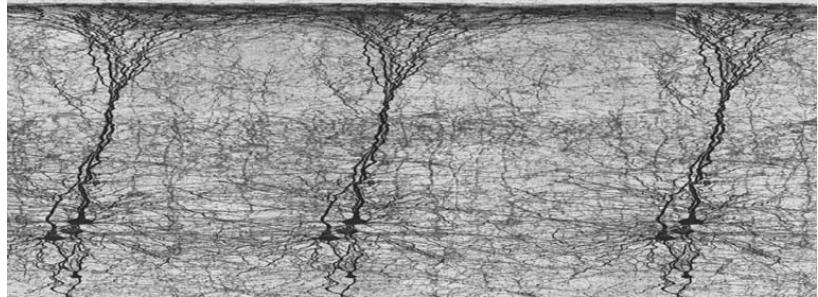
Son herramientas que tratan de replicar las funciones cerebrales de un ser vivo de una manera muy fundamental, esto es, desde sus componentes más básicos, las neuronas. Para ello se basan en estudios de neurobiología y de ciencia cognitiva moderna del cerebro humano <sup>18</sup>.

Una ANN es independiente del problema a solucionar. Se la puede

<sup>17</sup> Dentro de un contexto de aplicación específica, creemos que aún estamos muy lejos de crear vida inteligente.

<sup>18</sup> Aún apoyándose en la topología y funcionamiento del cerebro humano para realizar el símil, lo cierto es que dichos modelos distan aún de considerarse *cerebros artificiales*. La red neuronal más compleja hasta la fecha es la propuesta en [Trask ANDREWTRASK et al., ], con alrededor de 160,000 parámetros a ser ajustados (podemos abstraernos y pensar en ellos como conexiones entre neuronas). Si comparamos esta cifra sólo con las del neocórtex (figura 7) hace que, tecnológicamente hablando, nos quedemos con la sensación de estar aún a años luz de aproximarnos a la complejidad de un cerebro humano.

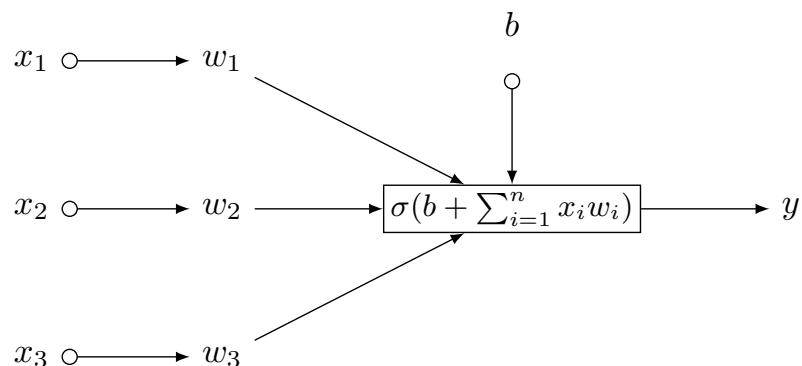
Figura 7: Sección del neocórtex humano, región asociada a las capacidades cognitivas y que supone alrededor de un 76 % del volumen total del cerebro humano. Está distribuido en 6 capas y miles de columnas que las atraviesan, cada una con alrededor de 10,000 neuronas y un diámetro de 0,5mm. Como dato 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/>.



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 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) interconectados 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].

Figura 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.



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 artificial. Sin embargo, este modelo es muy limitado. La verdadera potencia de las redes surge del tanto del uso de funciones de activación no lineales como de la agrupación de estas neuronas en estructuras más complejas.

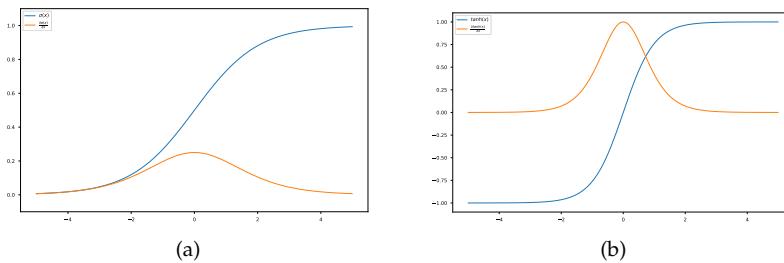
## Funciones de activación

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>19</sup>.

Las funciones de activación clásicas usadas en redes neuronales han sido la función sigmoidal (eq. 1) y la tangente hiperbólica (eq. 2). Por un lado, son funciones no lineales que mantienen normalizados las activaciones de las neuronas, y por otro, son derivables a lo largo de todo el dominio de los reales, siendo su derivada además fácilmente computable. En la Figura 9 se muestra una representación gráfica de éstas funciones junto con su derivada.

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

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

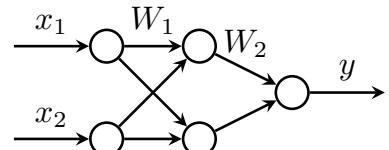


En general, la tangente hiperbólica es superior a la sigmoidal en todos sus aspectos. Computacionalmente es menos costoso el cálculo de una función sigmoidal, pero con la potencia de cómputo actual esta diferencia puede considerarse despreciable. Además de tener una forma similar a la sigmoidal, la tangente 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 sigmoidales en lugar de tangentes hiperbólicas. Por ejemplo, en un problema de clasificación, mantener en la capa de salida funciones de activación sigmoidales 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

<sup>19</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:

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

Por tanto:

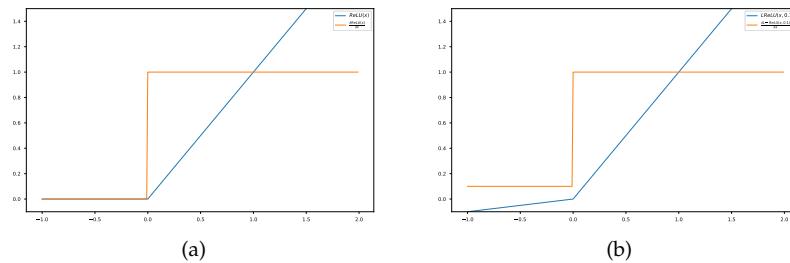
$$y = f(W_2 * f(W_1 * \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

Figura 9: Las funciones de activación sigmoidal y tangente hiperbólica. Ambas se han usado como funciones de activación no lineales gracias a que sus derivadas son siempre positivas. Sin embargo, el vector columna de la salida de la capa oculta y el vector de los bias de la capa de salida son fáciles de computar, ya que son el resultado de la multiplicación del 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.

frecuentemente como *vanishing gradient* en la literatura. Es una de las razones por las que en la fase de inicialización de una red neuronal se tendía a valores en torno al 0 en los pesos. Unos valores altos hacían que las entradas netas fuesen muy grandes ralentizando el aprendizaje.

Figura 10: La función de activación (a) 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 no nos indica hacia dónde ha de descender el error.



Uno de los avances dentro del Deep Learning (ver ??) fue el uso de un tipo de función de activación denominada **Rectified Linear Unit (ReLU)** (eq. 3). Ésta posee una forma muy simple pero es increíblemente efectiva. 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). En la figura 7

$$ReLU(x) = \max(0, x) \quad \frac{dReLU(x)}{dx} \approx \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \quad (3)$$

$$L - ReLU_\epsilon(x) = \max(0, x) \quad \frac{dL - ReLU_\epsilon(x)}{dx} \approx \begin{cases} \epsilon & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \quad (4)$$

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 al ésta en la que el algoritmo de descenso del gradiente se comporta de forma muy similar. Por ejemplo, la representación en la Figura 10, la derivada es 1 en el punto  $x = 0$ .
- Sin límite superior y derivada 0 para  $x < 0$ . Estas características pueden ser una ventaja o una desventaja. Las activaciones muy fuertes representan relaciones entre elementos muy próximos entre sí, aunque tienen el riesgo de anular el impacto del resto de entradas, pudiendo ralentizar el aprendizaje. Por otro lado, en el momento que el gradiente de una neurona se hace 0, ésta “muere”,

pudiendo ser una desventaja ya que la red dispone de menos neuronas para representar un modelo, como una ventaja, al ajustarse el modelo al número de neuronas necesarias. En general, las desventajas en estos aspectos aparecen cuando el factor de aprendizaje es notoriamente alto.

Por último, la función de activación *Leaky ReLU* (eq. 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 10. Su uso no está muy extendido, pero existen casos en los que su uso está justificado.

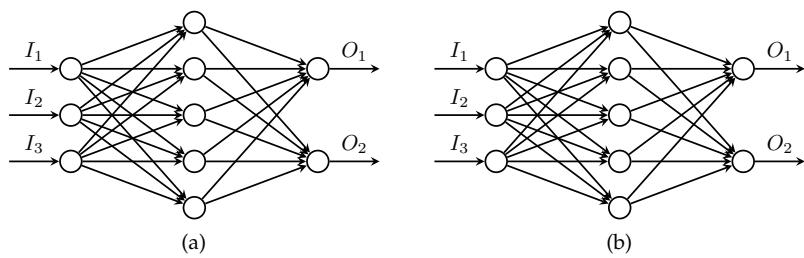
### *Estructura y clasificación de redes neuronales*

Las *ANN* reciben ese nombre debido a que son sistemas formados por multitud de neuronas artificiales simples. Dependiendo de cómo su topología y configuración, éstas serán más adecuadas para unos u otros problemas (e.g. unas serán más adecuadas para regresión, clasificación, predicción, *clustering*).

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:

- Redes **feed-forward** (o prealimentadas). Sus representación como grafo no presenta ningún ciclo (por tanto ninguna retroalimentación entre neuronas, como se ilustra en la figura 11). Es la topología más usada en aplicaciones prácticas debido a su sencillez y su efectividad. En ellas el flujo de información sigue un camino desde un conjunto de neuronas denominadas *entradas* hasta otro conjunto de neuronas denominado *salidas*. No es requisito que las neuronas se agrupen en capas, aunque suele ser la estructura común. A las redes de más de dos/tres capas ocultas (i.e. las capas que se encuentran entre la capa de neuronas de entrada y la capa de neuronas de salida) 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 *Mapas Auto-Organizados (SOM)* [Kohonen, 1998].
- Redes **recurrentes**. Éstas, a diferencia de las prealimentadas, tienen al menos un ciclo dentro de su representación, de tal manera que el flujo de información de salida de una neurona puede llegar a afectar a su propio estado. Aunque estas topologías representan de

Figura 11: Diferencias entre los grafos que representan las ANNs de tipos (a) *feed-forward* y (b) recurrentes. Las redes 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.**TODO!**CAMBIAR LA FIGURA DE LA RECURRENTE



una forma más fiel las bases biológicas de las ANN, históricamente han sido más complejas a la hora de operar y entrenar debido a sus relaciones entre nodos. Sin embargo, durante la última década han aparecido nuevas técnicas 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 topologías Perceptrón Multicapa y CNN, 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*

### *Perceptrones multicapa*

En un perceptrón multicapa, 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 las primera y últimas capas de la red se las denomina respectivamente capa de entrada y de salida, mientras que las capas intermedias son denominadas capas ocultas. Hemos visto algunas ilustraciones a lo largo del presente capítulo, pero en la Figura ?? se puede ver en detalle este tipo de red.

La forma de calcular la salida de un perceptrón multicapa es la siguiente. 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 dimensiones  $(l - 1, l)$ . 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 siguiente función que se muestra en la ecuación 5:

$$\mathbf{s}^l = f(W^l \mathbf{s}^{l-1} + \mathbf{b}^l) \quad (5)$$

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 calcular la salida es

incorporando las entradas en la capa de entrada e ir iterando hasta la capa de salida.

El algoritmo base de aprendizaje en un perceptrón multicapa de denomina *back-propagation* [Rumelhart et al., 1985] y se basa en la regla delta [Widrow and Hoff, 1960] para el perceptrón simple. Éste 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 como aumenta o disminuye esta 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, cuando mayor es un error, más ha contribuido). 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 lo determina a partir del error de las sucesivas capas apoyándose en el descenso del gradiente. Es decir:

Supongamos que nos encontramos en la capa  $l$  para la cual conocemos el error de salida que denotaremos como el vector columna  $\delta\mathbf{s}^l$ , 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:

$$\delta\mathbf{e} = \delta\mathbf{s}^l \circ f'(W^l \cdot \mathbf{s}^{l-1} + \mathbf{b}^l) \quad (6)$$

Nótese que  $\circ$  denota al producto de Hadamard (elemento a elemento). También es interesante fijarse en que  $W^l \cdot \mathbf{s}^{l-1} + \mathbf{b}^l$  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. 7b y 7c) y el error de salida (eq. 7a) de la capa anterior de la siguiente manera:

$$\delta\mathbf{s}^{l-1} = W^{l\top} \cdot \delta\mathbf{e} \quad (7a)$$

$$\delta W^l = \delta\mathbf{e} \cdot \delta\mathbf{s}^{l-1} \quad (7b)$$

$$\delta\mathbf{b}^l = \delta\mathbf{e} \quad (7c)$$

El valor  $\delta\mathbf{s}^{l-1}$  será el valor de entrada para la capa anterior, mientras que los valores  $\delta W^l$  y  $\delta\mathbf{b}^l$  serán los gradientes calculados. La forma más común de aplicar el error es la que se muestra en las ecuaciones 8a y 8b:

$$W^l = W^l + \alpha \delta W^l \quad (8a)$$

<sup>20</sup> Existen dos términos asociados al error en ANNs: el error y el 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 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:

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

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. Para tareas de clasificación, una función de error muy útil que ha remplazado a la función *hit* (relación de aciertos-fracasos) es la entropía cruzada, que tiene la siguiente forma:

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

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 *hiy*, 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.

$$\mathbf{b}^l = \mathbf{b}^l + \alpha \delta \mathbf{b}^l \quad (8b)$$

Esta es la forma original del algoritmo de back propagation. Al valor  $\alpha$  que aparece en las ecuaciones se le denomina factor de aprendizaje 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 disminuyen al ser capaz el algoritmo de “saltar” pequeños baches.
- **Adagrad ??, RMSProp ?? y Adadelta** [Zeiler, 2012]. Los tres se basan en el mismo principio. Al factor de entrenamiento de la red se le añade la existencia de un factor de entrenamiento 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. Es el más utilizado en los últimos años.

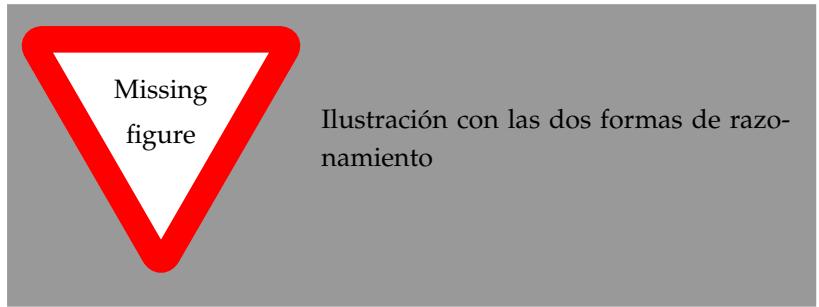
### *Redes de convolución*

Tal y como su nombre indica, las **CNNs** son redes que usan convoluciones para su funcionamiento. Aunque también está estructurada en capas, su funcionamiento es diferente. Para comenzar, su estructura se puede dividir 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 *predicción*). En la Figura 12 podemos ver una ilustración del esquema general de una **Convolutional Neural Network** donde se identifican ambas regiones.

La región de inferencia es en esencia un **Perceptrón Multicapa**. A éste 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,

Figura 12: Estructura general de una [Convolutional Neural Network](#). Se pueden observar sombreadas de distinto color la relación existente entre las regiones de identificación de patrones y la de predicción. La primera actúa sobre la entrada extrayendo características consideradas relevantes mientras que la segunda traeja sobre esas características en lugar de sobre todo el espacio de entrada.



normalmente 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 I 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:

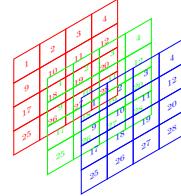
$$R_w = M_w - F_w + 1 \quad (9)$$

$$R_h = M_h - F_h + 1 \quad (10)$$

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.

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:

<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.

<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.

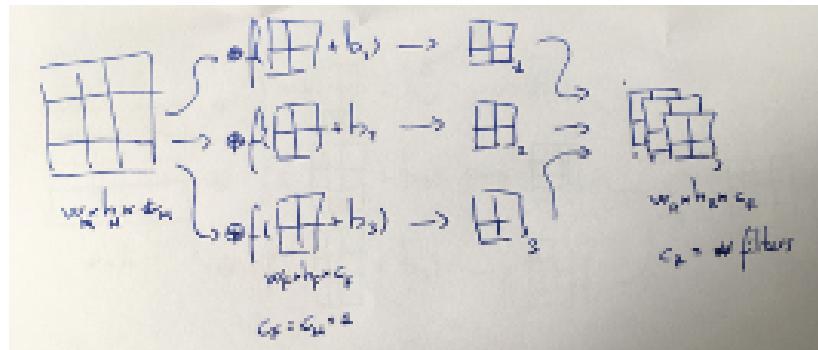
$$P_w = \frac{F_w - 1}{2} \quad (11)$$

$$P_h = \frac{F_h - 1}{2} \quad (12)$$

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

*Capas de convolución* Las capa más importante en una CNN 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. La figura ?? ilustra esto.

Figura 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 tanto, son los valores del filtro junto con los bias los parámetros que el algoritmo de aprendizaje debe aprender.



El proceso de aprendizaje en una red de convolución 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:

$$\delta W_f = \sum_{w=0}^{F_h} \sum_{h=0}^{F_w} S^{l*} \times \delta C_{w,h} \quad (13a)$$

$$\delta b_f = \sum_{w=0}^{F_h} \sum_{h=0}^{F_w} \delta C_{w,h} \quad (13b)$$

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 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 los mismos principios que los Perceptrones Multicapa.

*Pooling y normalización* Junto con las capa 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 discretización 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.

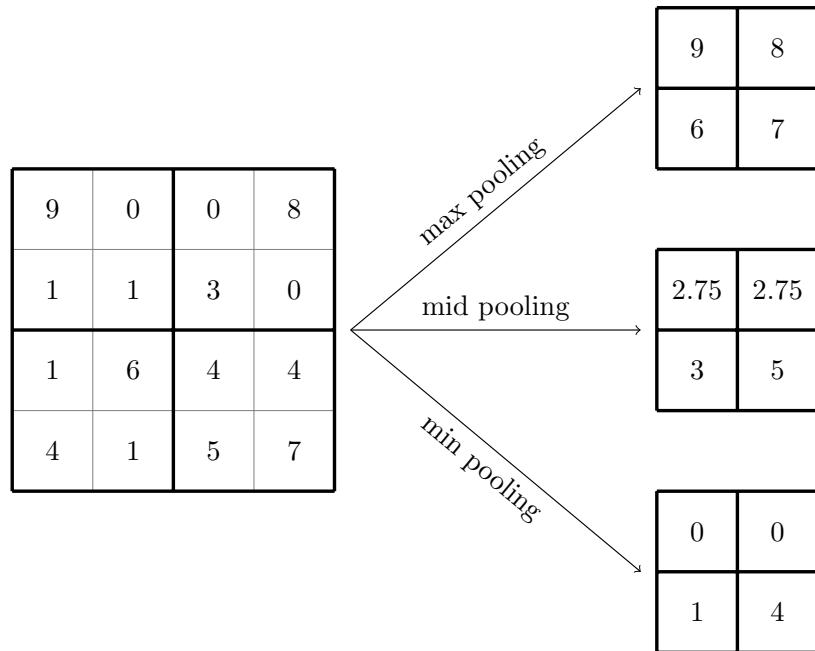


Figura 14: Ejemplo de las operaciones de *pooling* sobre una matriz de entrada de dos dimensiones. El filtro de tamaño  $2 \times 2$  recorre la matriz en saltos de 2 celdas horizontales y 2 verticales extrayendo de cada región filtrada el mayor valor, el menor o el valor medio dependiendo de si la operación es *max-pooling*, *min-pooling* o *mid-pooling*.

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 no solapables devolviendo un valor para cada ventana y reduciendo por tanto el tamaño de la entrada. 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. En la Figura ?? se ilustra un ejemplo de la operación de **max-pooling**<sup>24</sup>.

La normalización es otra operación, esta 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 CNN 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"

### *Solucionando los problemas de entrenamiento*

Anteriormente hablábamos de los problemas a los que se enfrentan los procesos de entrenamiento en la [IC](#). En líneas generales, los problemas de subespecialización pueden darse por tres razones diferentes:

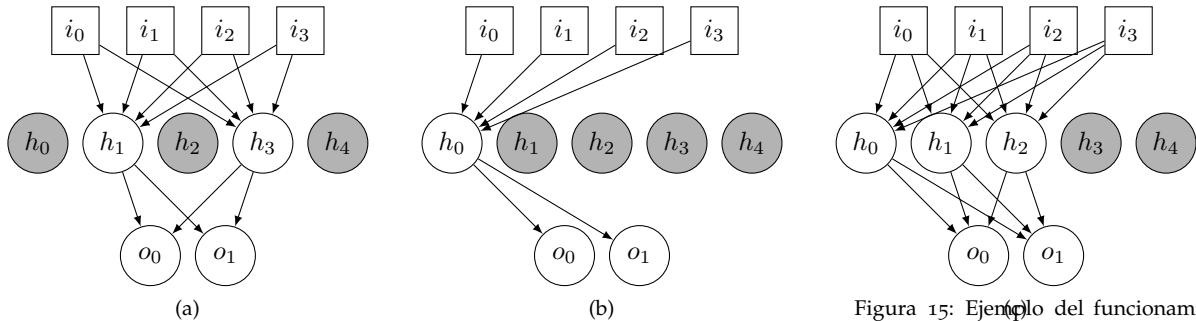
- La red no es lo suficientemente grande. Puede ocurrir que el conjunto de datos requiera de más propiedades o parámetros. La solución es incrementar el número de capas o de neuronas por capa a fin de que el modelo acabe aprendiendo al menos los datos del conjunto de entrenamiento.
- El modelo no ha 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 entrernamiento o las funciones de activación están predisponiendo al modelo a aprender más lentamente.
- La topología del modelo no se adecúaa los datos. Puede ocurrir que el modelo que estamos tratando de aprender aprenda mucho mejor en una topologías que en otras. Por ejemplo, para aprender a clasificar imágenes, las [CNNs](#) funcionan mejor, entre otra serie de razones, porque mantiene una coherencia espacial entre los píxeles de la imagen desde el principio, cosas que con un [Perceptrón Multicapa](#) 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 quizá 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, por lo que el modelo no generaliza 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. Este suele ser el caso más común, y es que la red tiene tantos parámetros que al final a 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 ésta, la idea es 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 15 describe un ejemplo de funcionamiento en tres epochs de un perceptrón multicapa con una tasa de dropout del 0,5.

Figura 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.

### Grafos computacionales

En la actualidad el concepto de grafo computacional se relaciona directamente con las redes neuronales, 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 al área de la IC, sino que son simplemente una vía para representar las operaciones en forma de grafo, de tal manera que es fácil ver cuál es el flujo de los datos y las operaciones que se realizan sobre ellos.

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 16 describe un posible grafo computacional para un perceptrón simple.

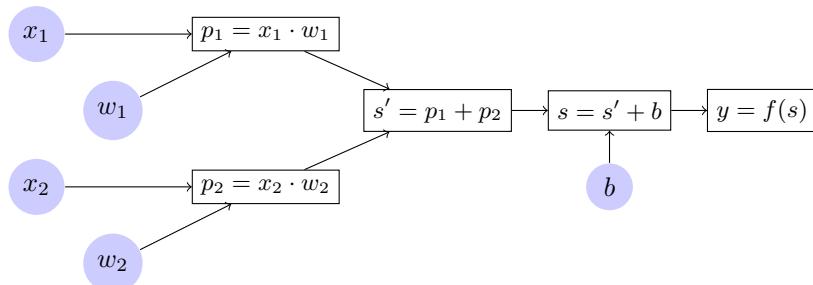
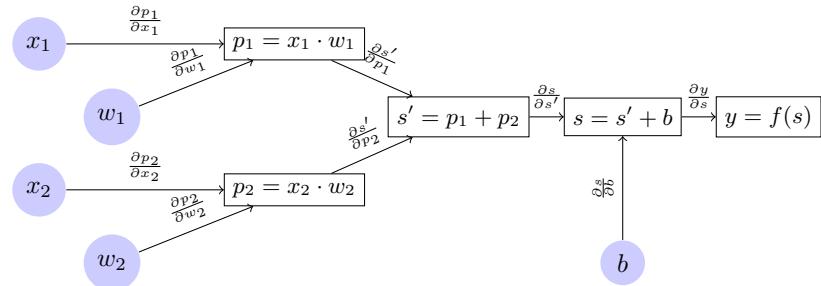


Figura 16: Ejemplo de grafo computacional para un perceptrón simple de dos neuronas de entrada y una de salida.

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 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 17 de derecha a izquierda).

Figura 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.



Com 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.

### Lógica Difusa

<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 (Cantor, finales del siglo XIX, buscar referencia) 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, llenos de imprecisiones. Además tienden a ser nombrados cualitativamente, no quantitativamente, y cuando existe una correspondencia, ésta suele estar marcada por la subjetividad de los términos. La lógica difusa 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 que 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 Difusa 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 difuso, cómo se usan como mecanismo de razonamiento y su utilidad dentro del área de sistemas de control.

### *Ampliando la teoría de conjuntos tradicional*

La teoría de conjuntos difusos 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 Difusa), los conjuntos difusos 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 difusos es el de **variable lingüística**, las cuales sirven de base para toda la teoría porque sus posibles valores son, precisamente, conjuntos difusos. 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 difusos) se hereda a lo largo de todo el proceso de deducción, de manera similar a como lo hacen los humanos.

*Conjuntos difusos* Una posible definición de conjunto difuso podría ser dada  $X$  una colección de elementos. Se define al **conjunto difuso**  $F$  como un conjunto ordenado de pares de la forma  $F = (x, \mu_F(x)) | x \in X$ , 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 únicamente a un conjunto difuso del dominio de  $X$ <sup>28 29</sup>.

La teoría de conjuntos clásica también define los conjuntos de acuerdo a una función, en este caso denominada *función característica*. Las ecuaciones 14a y 14b muestran las diferencias entre los valores de  $f$  siendo ésta una función característica de la teoría de conjuntos clásica o una función de pertenencia de la teoría de conjuntos difusos respectivamente.

$$f(x) = \begin{cases} 0 & \text{if } x \notin F \\ 1 & \text{si } x \in F \end{cases} \quad (14a)$$

<sup>27</sup> La Lógica Difusa 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 difusos 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.

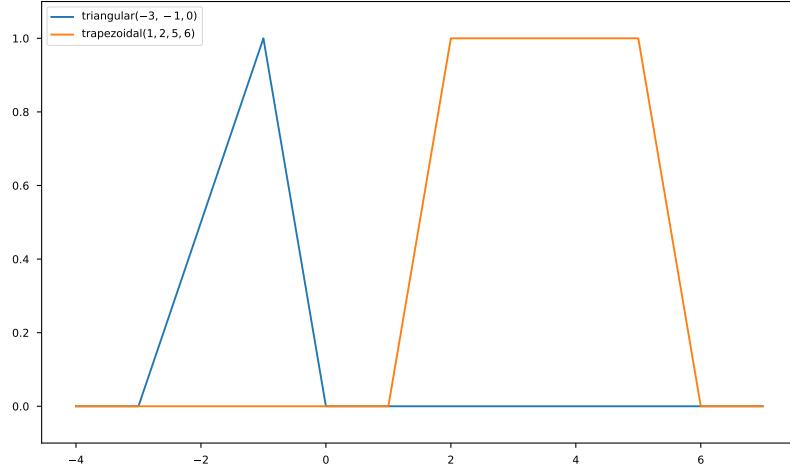
$$f(x) = \mu_F(x) \quad (14b)$$

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

Mientras que un conjunto tradicional se basa en si pertenece o no, un conjunto difuso 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), pero lo habitual es definirlas como funciones triangulares o trapezoidales. Un ejemplo de estas funciones se ilustra en la figura 77.

Figura 18: Las funciones de pertenencia triangular y trapezoidal son las dos funciones más usadas a la hora de definir conjuntos difusos, 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.



Las funciones de pertenencia caracterizan a los conjuntos difusos. Su notación habitual es la que se presenta en la ecuación ??, 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.

$$F = (x, \mu_F(x)) | x \in U \quad (15)$$

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

*Operaciones entre conjuntos* Las mismas nociones de operaciones en teoría de conjuntos son aplicables a la teoría de conjuntos difusos. Una operación entre conjuntos difusos es aquella que genera un nuevo conjunto difuso a partir de uno o más conjuntos difusos de entrada definidos sobre el mismo universo de discurso. Son las operaciones  $t$ -norma,  $t$ -conorma y complemento.

La  $t$ -norma es la generalización del operador conjunción en lógica clásica. Son un conjunto de aplicaciones de la forma  $T : [0, 1] \times$

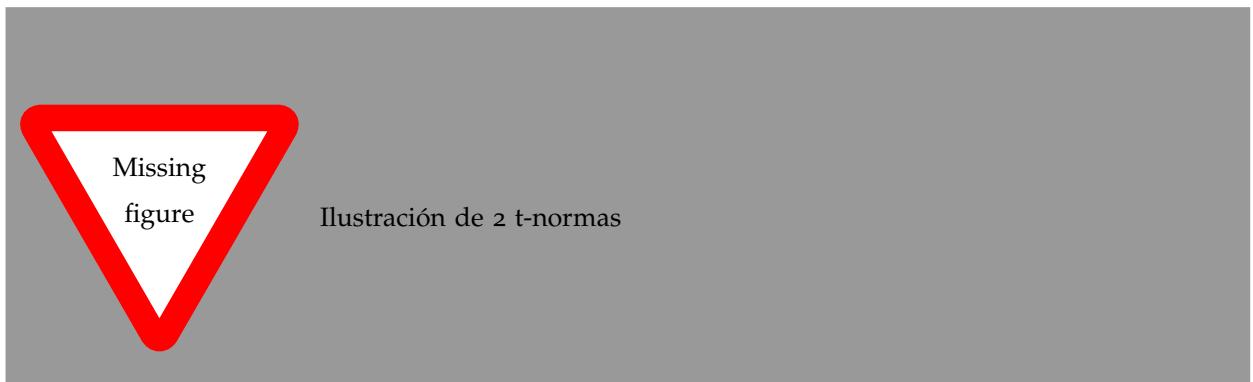


Figura 19: Las  $t$ -normas de mínimo y de producto algebraico son dos representantes de un conjunto muy amplio de funciones.

$[0, 1] \rightarrow [0, 1]$  tales que cumplen las propiedades *comutativa*, *asociativa*, *monótona* e *identidad*<sup>31</sup>. Dos de las operaciones más usadas como  $t$ -norma se ilustran en la Figura 19. Existen muchas más, pero en controladores difusos se suelen usar el mínimo o, en su defecto, el producto.

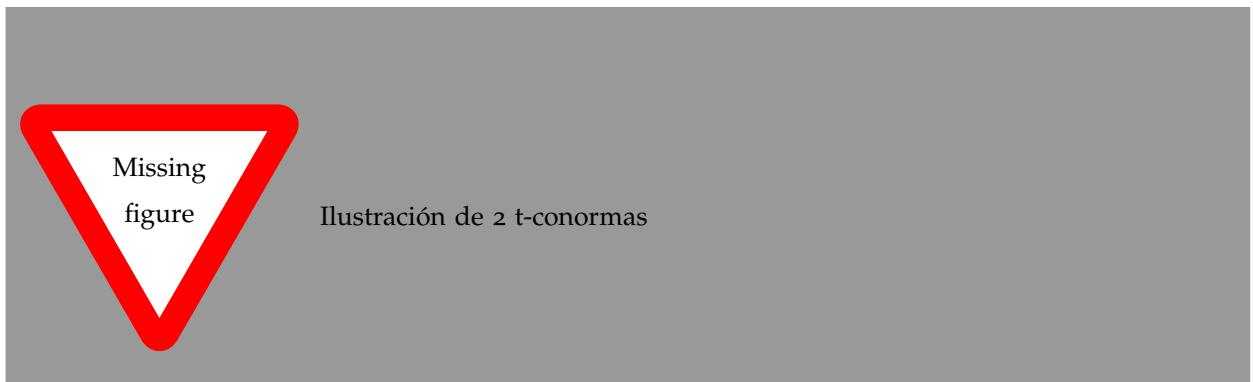
La  $t$ -conorma (o  $s$ -norma), por otro lado, generaliza el concepto de unión. Una  $t$ -conorma  $s$  se define a partir de una  $t$ -norma  $t$  como  $S = 1 - T(1 - x, 1 - y)$ , es decir, la función complementaria a la  $t$ -norma. De esta forma se consigue que las Leyes de Morgan se cumplan de una forma generalista. Dos ejemplos de  $t$ -conormas se ilustran en la Figura 20

Las propiedades que ha de cumplir una  $t$ -conorma son las mismas que las de la  $t$ -norma<sup>32</sup>.

<sup>31</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:  $\exists 1 | T(x, 1) = x$ .

<sup>32</sup> De hecho, son deducibles a partir de la  $t$ -norma con una salvedad, la identidad, que se define como  $\exists 0 | S(x, 0) = x$ .



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 com-

Figura 20: Las  $t$ -conormas de máximo y de suma algebraica son las complementarias a  $t$ -normas mostradas previamente, y como ocurre en estas, existen tantas  $t$ -conormas como  $t$ -normas definidas.

<sup>33</sup> Las propiedades son:

- Equivalencia con teoría de conjuntos clásica en el caso de usar conjuntos *crisp*, es decir,  $f(1) = 0 \text{ y } f(0) = 1$ .
- 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>34</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,  $f_\lambda(x) = \frac{1-x}{1-\lambda \cdot x}$ . A partir de ambas se puede definir el complemento de Zadeh variando el valor de  $\lambda$ .

plementario  $F^C$  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>33</sup>.

El operador más común para el complemento es el probabilístico, denominado también *complemento de Zadeh* en **Lógica Difusa**. 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>34</sup>.

### Razonamiento

El razonamiento o inferencia es el proceso de obtener consecuencias a partir de hechos (premisas). En **Lógica Difusa**, el proceso de inferencia lógica se amplía a través de relaciones difusas, trasladando valores de verdad aproximados.

Toda regla difusa está compuesta por un antecedente y un consecuente. Tanto el uno como el otro están compuestos de sentencias que asignan valores de variables difusas (conjuntos difusos) a éstas. La forma más común de representar las reglas difusas 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 Figura ?? detalla los componentes de estas reglas.

La implicación lógica  $A \implies B$  es equivalente a  $\neg A \vee B$ , pero en lógica difusa 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>35</sup>.

**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 difusos. En la Figura 21

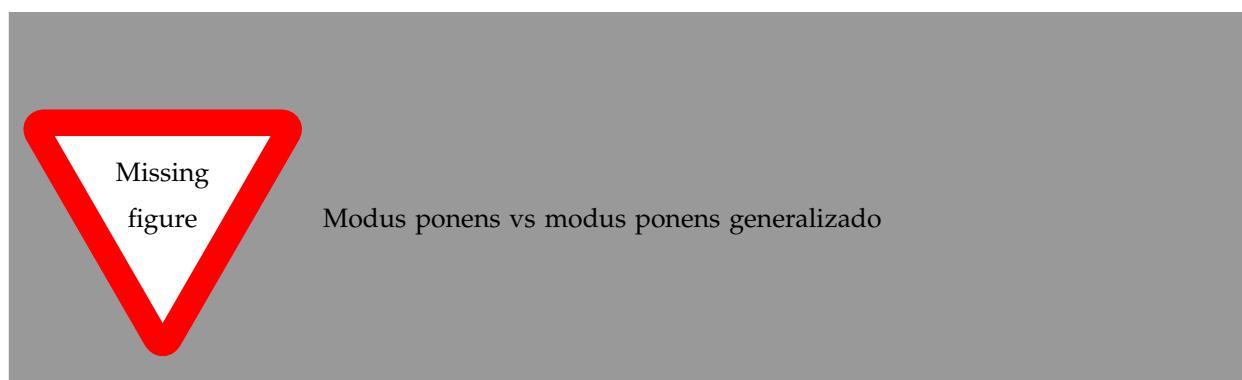


Figura 21: Diferencias entre *modus ponens* clásico y *modus ponens generalizado*. Debido a que los valores de las reglas de inferencia difusa son conjuntos difusos, la conclusión de un silogismo es por tanto un conjunto difuso, no un valor absoluto de verdad o mentira.

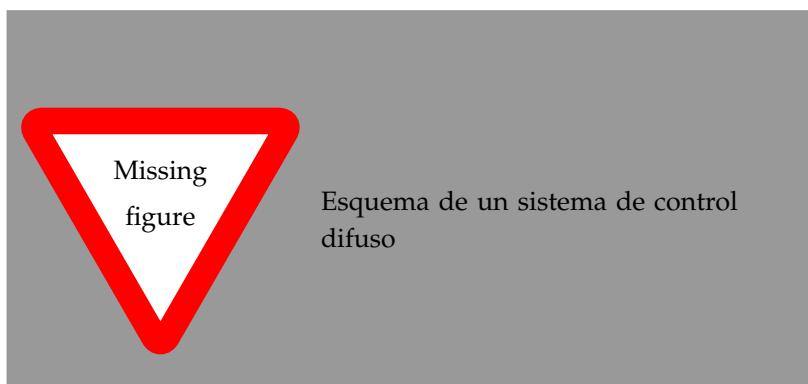
La *regla composicional de inferencia* es el método para determinar qué grado de asignamos aun consecuente a partir de las premisas de los antecedentes en un proceso de *modus ponens generalizado*, o dicho de otro modo, cómo obtener la función de pertenencia del conjunto resultado a partir de las funciones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

Sin entrar demasiado en detalle, cada regla de un sistema difuso de la forma *IF x is A THEN y is B* se puede representar como una función *I* como  $I(\mu_A(x), \mu_B(y))$  [La regla composicional de Zadeh].  
REVISAR ESTO PORQUE CREO QUE EN LA TESIS DE MASTER ESTÁ MAL O RARO.

### *Sistemas de control difuso*

Los () son el caso de éxito de la lógica difusa que más resultados ha cosechado tanto a nivel académico como a nivel industrial. Se trata sistemas que utilizan el razonamiento difuso para inferir una respuesta a partir de un conjunto de entradas<sup>36</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).

Éste se puede definir como un componente que asigna salidas de dominios no difusos (*crisp*) a entradas de dominios también no difusos (*crisp*) de forma no lineal haciendo uso de lógica difusa. La arquitectura de estos sistemas está compuesta por cuatro bloques: *fuzzificador*, *bloque de reglas*, *razonador* y *defuzzificador*. En la Figura 22 se describe el esquema de un sistema de control difuso general.



<sup>36</sup> Dado que el control difuso puede extenderse a casi cualquier problema de control tradicional hace que se haya usado en multitud de campos, no sólo en el industrial. Esto hace que el mismo concepto pueda encontrarse con muchos nombres diferentes, como Sistemas de Lógica Difusa, Sistemas de Inferencia Difusa, Sistemas Expertos Difusos, Sistemas Basados en Reglas Difusas o simplemente Sistemas Difusos. [COGER LAS REFERENCIAS DE LA TESIS DE MASTER]

Figura 22: Un sistema de control difuso es un componente compuesto por cuatro bloques principales: fuzzificador, bloque de reglas, razonador y defuzzificador.

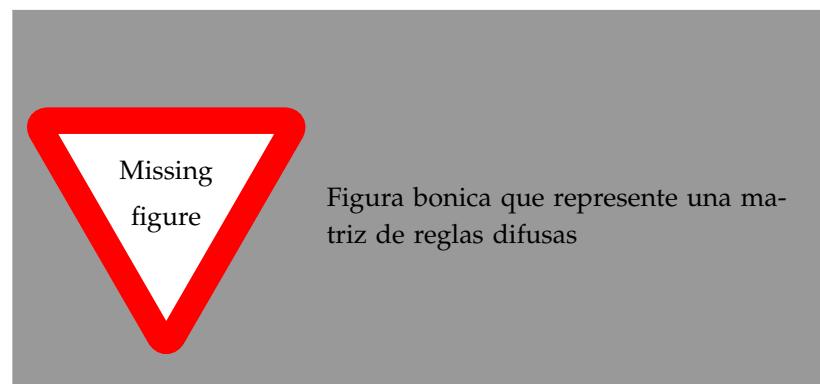
*Fuzzificador* Este bloque recibe las entradas *crisp* al sistema (los cuales pertenecerán al dominio sobre el que las variables están definidas) y las transforma a valores de pertenencia de sus respectivos conjuntos difusos.

*Bloque de reglas* Define el conjunto de reglas que gobiernan el proceso deductivo del sistema de control difuso. 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  $m$  matriz  $n$ -dimensionales (una por cada conjunto difuso de salida), donde cada dimensión se corresponde con una variable lingüística, con los conjuntos difusos de éstas como índices y con los valores que toman esas intersecciones como los valores de esa salida. En la Figura 23<sup>37</sup>.

<sup>37</sup> En el desarrollo de la tesis veremos un caso en el que esta representación es útil a la hora de inferir un controlador difuso a partir de un conjunto de entrenamiento. Figura 23: Un ejemplo de representación matricial de un conjunto de reglas difusas. Las reglas activas se simbolizan como un 1 (o como un valor real en caso de querer aplicar pesos).



*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 difusos 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 difuso 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.

$$f(x) = \frac{\int_F x \cdot \mu_F(x) \cdot dx}{\mu_F(x)} \quad (16a)$$

$$f(x) = \frac{\sum_F x \cdot \mu_F(x) \cdot dx}{\sum_F \mu_F(x)} \quad (16b)$$

Existen diferentes algoritmos para la transformación de un conjunto difuso (expresado a partir de su función de pertenencia  $\mu_F$ ) a un valor *crisp*. Suelen ser comunes en conjuntos de salida continuos el COA (*centroid of area*, ecuación 16a) y en conjuntos de tipo singleton su versión *media ponderada* (ecuación 16b) <sup>38</sup>.

<sup>38</sup> En el trabajo [Defuzzification: criteria and classification] se realiza una clasificación muy completa de una gran cantidad de algoritmos de defuzzificación.

### Tipos de Sistemas de Control Difuso

Los sistemas de control difuso 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 difuso 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 difusas del tipo *IF...THEN...* obtenidas directamente de la experiencia de los operarios.

Su estructura es la básica presentada previamente (ver Figura 22).

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

*Controlador difuso tipo Takagi-Sugeno-Kang* Este tipo de controlador difuso fue propuesto por Takagi, Sugeno y Kang en los trabajos [Fuzzy identification of systems and its applications to modeling and control, Structure identification of fuzzy model], 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* (ver Figura 24).

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  $f(x, y) = c$  (constante) 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>39</sup>.

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

<sup>39</sup> Los controladores de tipo Takagi-Sugeno-Kang de orden 0 son equivalentes a controladores de tipo Mamdani cuyos conjuntos difusos de salida son de tipo singleton.

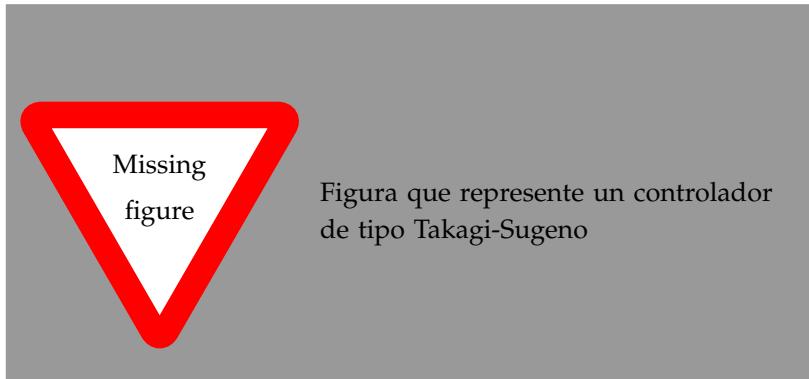


Figura 24: En un controlador de tipo Takagi-Sugeno-Kang se prescinde del bloque de defuzzificación y los consecuentes de las reglas de inferencia son reemplazados por funciones. Por término general, estas funciones son polinomios.

### *El paradigma de los Agentes Inteligentes*

En la figura 3 se mostraban los cuatro objetivos perseguidos por la [AI](#). En uno de ellos en particular se la entiende como el estudio del conseguir que entidades (e.g. sistemas, software, ...) actúen de la manera más inteligente posible.

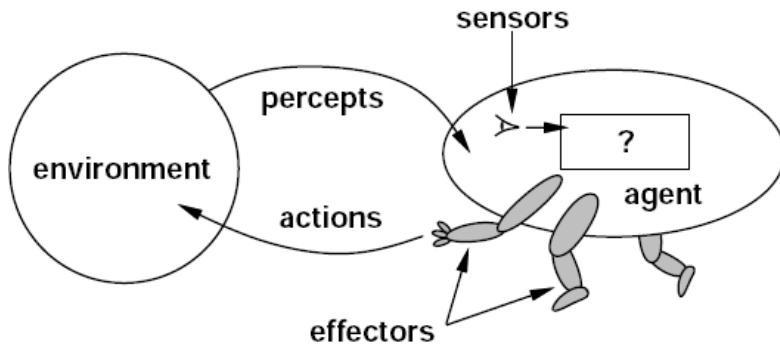
Este concepto es una metáfora denominada **agente**<sup>40</sup> en el campo de la [AI](#), pero que se ha extendido a lo largo de todo el área de la computación, especialmente en las áreas de los sistemas distribuidos o de las simulaciones basadas en sistemas multiagentes.

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 coínciden dentro de la literatura que exponemos a continuación (ver Figura 25):

- Operan siempre en un **entorno**, ya sea éste 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.
- Pueden ser **sociales**, es decir, tienen la capacidad de comunicarse con otras entidades (e.g. otros agentes) para llevar a cabo sus objetivos.

Nosotros hablaremos de estas entidades desde el punto de vista de

<sup>40</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 impri me 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.



la [AI](#), y las denominaremos indistintamente *agentes*, *agentes inteligentes* o *agentes racionales*<sup>41</sup>. Por tanto usaremos la siguiente definición:

“Un agente es una entidad física o virtual que realiza una acción<sup>42</sup> 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 [IC](#) van de la mano. Esto es debido a que su definición funciona a la perfección para las técnicas de la [IC](#), 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 [ANN](#), [Lógica Difusa](#), ...). Por ello desde mediados de los años 1990 el concepto de agente inteligente ha ganado tanta popularidad<sup>43</sup>.

### 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.

Figura 25: Esquema de un agente y sus propiedades. Aunque no existe una definición comúnmente aceptada de agente, sí que existe una serie de propiedades que los que los identifican. Es autónomo, opera realizando acciones sobre un entorno dependiendo de las percepciones que le llegan de éste y tiene la capacidad de comunicarse con el resto de elementos, incluidos otros agentes.[TODO!CAMBIAR LA IMAGEN Y VER CÓMO SE PUEDE INCLUIR EL CONCEPTO DE COLABORACIÓN](#)

<sup>41</sup> El término preferido en esta tesis es precisamente este último, **agentes racionales**, dado que captura 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, esto es, 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 entre qué es y qué no es un inteligente cae dentro de los dominios de la filosofía.

<sup>42</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 impri me 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.

<sup>43</sup> Tanto es así que en algunos trabajos se define el objetivo de la [AI](#) 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.

- **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>.

### *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 26 muestra el esquema de las partes principales de un agente. En general, todo 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 en el que éste se mueve.

<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.

<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 26: Arquitectura básica de un agente. Aunque existen múltiples arquitecturas diferentes, todas se basan en la misma estructura. El agente percibe el entorno, lo interpreta y toma la decisión de cómo actuar sobre él.



2. De acuerdo a cómo recordamos el entorno (llamémoslo **modelo del entorno**), el agente genera una **interpretación del entorno** tal y como supone el agente que es. Esto es, 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 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 27):

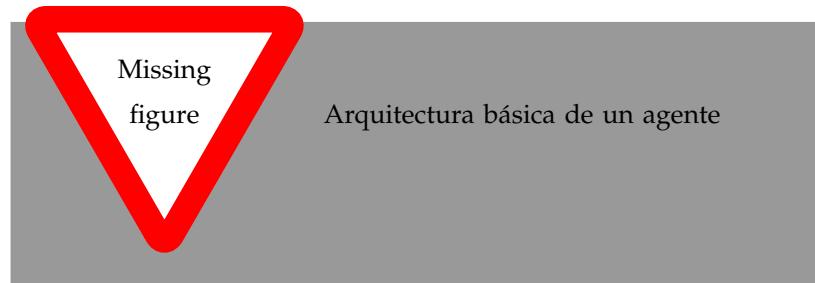
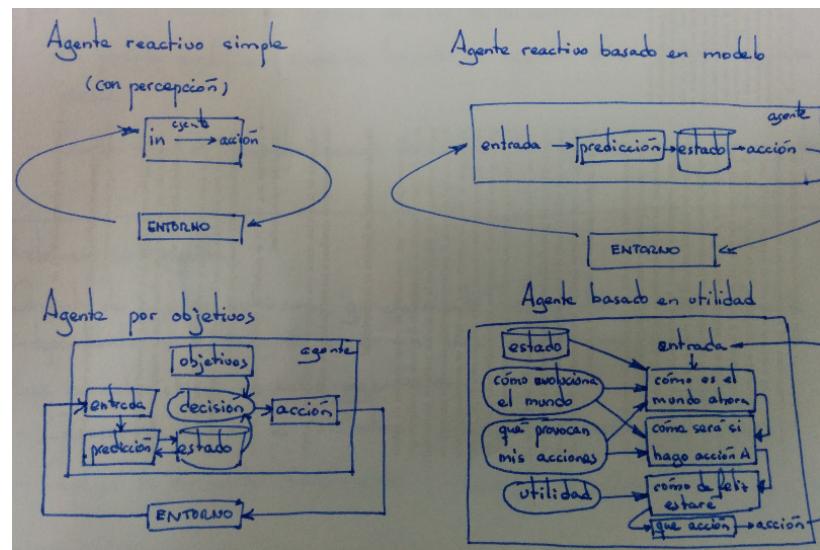


Figura 27: Ilustración de la diferencia entre un agente sin modelo de entorno y uno con modelo de entorno. Cada acción realizada por el agente con modelo de entorno tiene en cuenta el estado del entorno en momentos pasados. El agente sin modelo de entorno actúa tal y como interpreta el entorno en cada momento, como si sufriese de amnesia.

- **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 como *agentes reactivos* o *simple-reflex agents* ([Russell et al., 2003]). Sin embargo, los términos *reactivo* o *reflex* para algunos autores se refieren a la forma de inducción de acciones a partir de percepciones, y por ello preferimos la denominación *sin modelo de entorno*.
- **Con modelo de entorno.** El agente genera su interpretación más detallada del entorno a partir de las percepciones que llegan desde los sensores y de el histórico del entorno que mantiene. Otras formas de llamarlo es *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.

Figura 28: Distintas arquitecturas de agentes en función del comportamiento. Dependiendo de las acciones a realizar, se identifican tres tipos, los reactivos que aplican una acción sin proceso deductivo y los basados en modelo y utilidad (en algunos contextos denominados deliberativos) que basan su comportamiento en alguna forma de deducción.



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 28):

- **Reactivos.** 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.** 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.** Éstos 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 BDI (Believe-Desire-Intention) o los agentes lógicos (i.e. el entorno se representa con reglas lógicas y se infiere mediante métodos como por ejemplo deducción lógica o prueba de teoremas). Sin embargo, éstos pueden definirse en los términos aquí expuestos (figuras 26, 27 y 28).

### *Sistema 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 [AI](#), 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 [Sistema Multiagente](#), 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 [Sistema Multiagente](#) hace natural su implementación en este tipo de arquitecturas.

Desde el punto de vista de la [AI](#) 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.

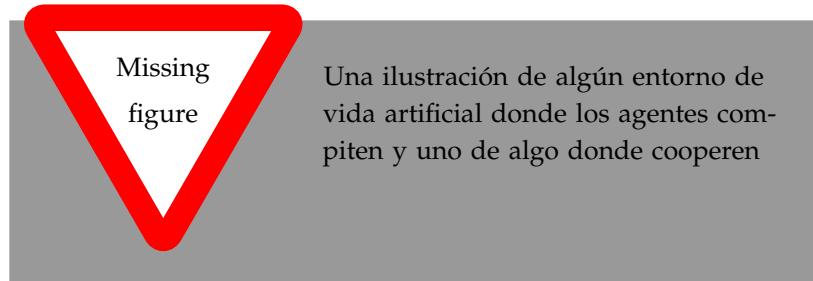
LA COMUNICACIÓN ENTRE AGENTES, se trata de una característica clave en un [Sistema Multiagente](#), ya que para denominarse de esta manera dos o más agentes deben interactuar (i.e. comunicarse) entre sí. Esta interacción puede implementarse de diversas maneras<sup>46</sup> y siempre toman una o las dos formas siguientes (figura 29):

- **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 poseerla uno o varios agentes que hacen uso de más agentes para ir avanzando la solución.

<sup>46</sup> Las formas clásicas de comunicación son el de 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.

- **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.

Figura 29: La comunicación entre agentes puede ser de dos tipos: *colaborativa*, donde los agentes tratan de llegar a una solución intercambiándose información y *competitiva*, donde los agentes compiten unos contra otros en un entorno.



# Simulación de tráfico

El tráfico es un sistema de comportamiento tan caótico que extraer modelos de su funcionamiento es una tarea prácticamente imposible. Por un lado, la cantidad de variables existentes es innumerable y en muchos casos con relaciones no detectables a primera vista. Por otro, 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 en 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 DVU, conductor y vehículo para referirse al mismo concepto. En caso de no ser así, se indicará de manera explícita.

El entorno de simulación **TORCS**, usado en multitud de concursos e investigaciones, se trata de un juego. Los juegos son un *sandbox* perfecto ya que presentan una abstracción de complejidad acotada sobre el dominio que trabajar.



Algunos trabajos interesantes que usan como base **TORCS** para emular comportamientos de conductores reales (conduciendo en el simulador) son [Muñoz et al., 2010], donde se usan perceptrones multicapa entrenados con técnicas de *back propagation* y [Van Hoorn et al., 2009], donde también se usan perceptrones, pero esta vez entrenados mediante Algoritmo Genético multiobjetivo. Sin embargo, este tipo de modelos se encuentran más cercanos al nivel de control que al nivel táctico (ver figura 42).

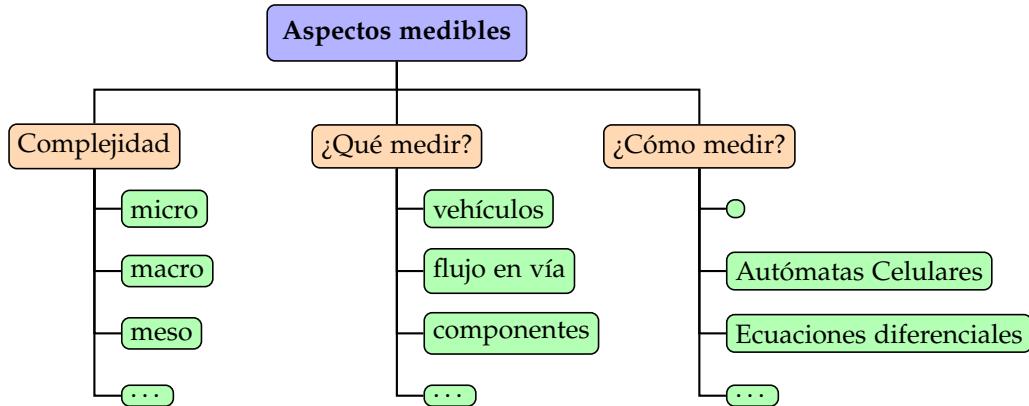


Figura 30: Los aspectos medibles del problema del tráfico son muy diversos, y dependen del nivel de granularidad (i.e. complejidad) al que se quiere llegar, de qué queremos medir y de cómo lo queremos hacer.

### *Clasificación de simuladores de tráfico*

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

- 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 millones).
- 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.

### *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, los cuales se ilustran en la figura 31:

- **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>47</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 requeriría 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 **sub-micro** y de tipo **meso**, de los cuales se muestra un ejemplo en la figura 32.

Los **sub-micromodelos** 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 **mesosimuladores** (e.g. [Munoz et al., 2001] o [Casas et al., 2011]) nacen para amortiguar los problemas inherentes a la complejidad en los micromodelos y a la falta de resolución en los macromodelos.

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

<sup>47</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.) 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]).

Figura 31: 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.

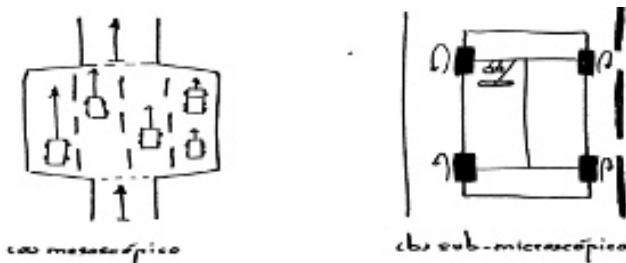
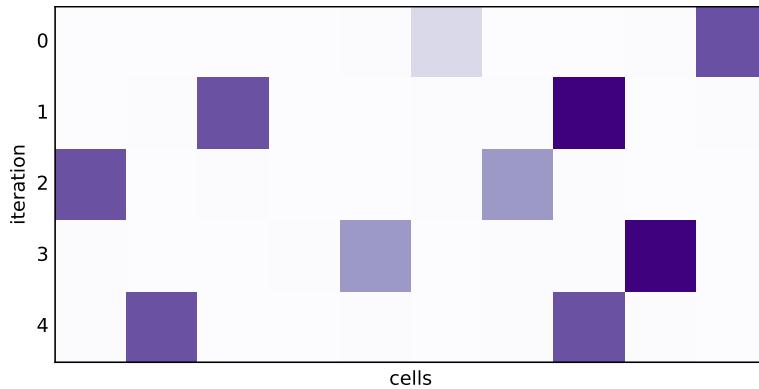


Figura 32: Otras aproximaciones alternativas de modelos en función de la complejidad. Ejemplo de mesosimulación como ventana de microsimulación dentro de un flujo de tráfico en un macrosimulador (e.g. [Munoz et al., 2001]) y ejemplo de submicrosimulación donde se modelan componentes internos de un vehículo.

### 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 simuladores, el tipo de simulador según una de ellas tiende a determinar en gran medida los tipos en las demás.

Figura 33: Simulador de tráfico basado en CA. 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 2 vehículos donde en eje  $x$  representa la posición en la vía y el eje  $y$  el momento temporal (iteración) de la vía. Fuente: simulador nagel-scherckenberg-demo (Ver capítulo Sistemas desarrollados).

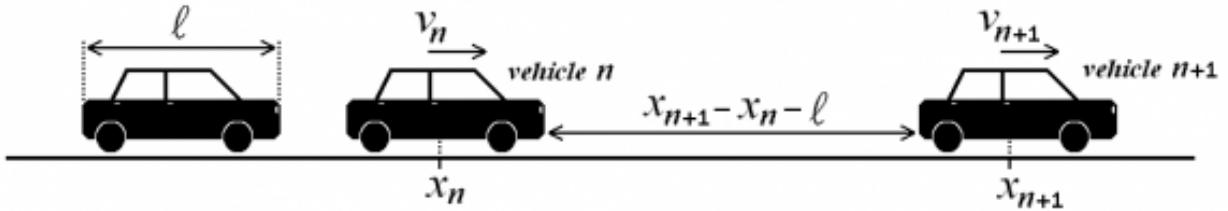


En el caso de la dimensión **espacio**, la clasificación diferencia las simulaciones que se mueven por un espacio discreto o por uno continuo:

- **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 (CAs, Cellular Automata)** (figura 33).
- **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 34).

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 (aunque existen excepciones) de longitud fija durante toda la simulación.



Los simuladores basados en **CAs** son también simuladores típicos discretos, ya que cada posición en el espacio se va calculando para cada intervalo discreto de tiempo (figuras 33 y 35).

- **Tiempo continuo.** En estos simuladores el tiempo es un factor más para un modelo de ecuaciones diferenciales. La figura 34 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 y no una situación aproximada en una separación discreta del espacio. Esto nos dirige hacia simuladores de **espacio continuo**. Por otro lado, realizamos 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; por tanto, la elección en la dimensión tiempo ha de ser de **eventos discretos**.

### *Modelos de microsimulación*

Los simuladores que se basan en un modelo de granularidad micro están en su mayoría implementados en dos tipos de paradigma: Autómatas Celulares y .

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.

#### *Microsimulación basada en Autómatas Celulares*

Un **CA** es una colección ordenada de celdas (*células*) ordenadas 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>48</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

Figura 34: 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].

<sup>48</sup> Existen arquitecturas diseñadas para operar de esta manera, esto es, arquitecturas basadas en **CA** (e.g. [Margolus, 1993]). 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.

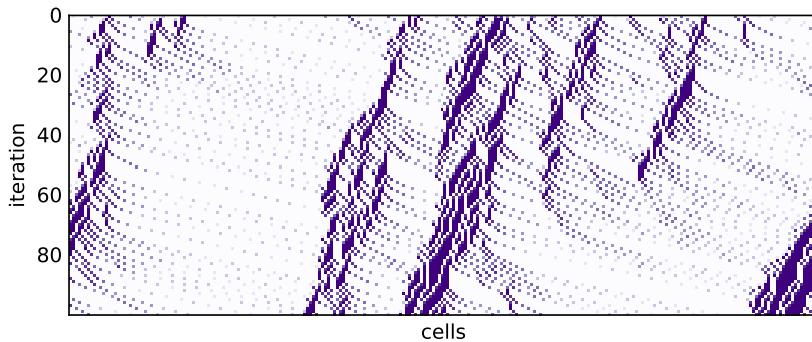


Figura 35: Aparición de retenciones en una autopista de 250 celdas usando el modelo Nagel-Scherckenberg. La densidad de ocupación es de 50 coches en la vía, la velocidad máxima es de  $5c/\Delta t$  y la probabilidad de frenada es de  $p = 0,5$ . Se puede observar en la figura cómo se desplazan las olas del atasco a lo largo de las 100 iteraciones. Fuente: simulador nagel-scherckenberg-demo (Ver capítulo [Sistemas desarrollados](#)).

**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 1 s, 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.

de las células vecinas y del mismo algoritmo de modificación al que responden todas y cada una de las células.

Estos modelos de microsimulación, debido a la propia naturaleza de los CA, se encuentran clasificados como simuladores de tiempo y espacio discreto, y se usan debido a su facilidad de implementación y a su eficiencia, ya que son fácilmente paralelizable.

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 35 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]).

### *Microsimulación basada en sistemas multiagentes*

Los modelos basados en Autómatas Celulares, aunque interesantes, 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  $27km/h$ . Es más, en una situación favorable, cualquier vehículo puede realizar una aceleración de 0 a  $162km/h$  en tan sólo 6 segundos. Por tanto, no ofrecen una visión demasiado realista ni fiable en caso de querer realizar estudios muy detallados de tráfico a nivel micro.

Por otro lado, en un cada uno de los agentes tiene su propia en-

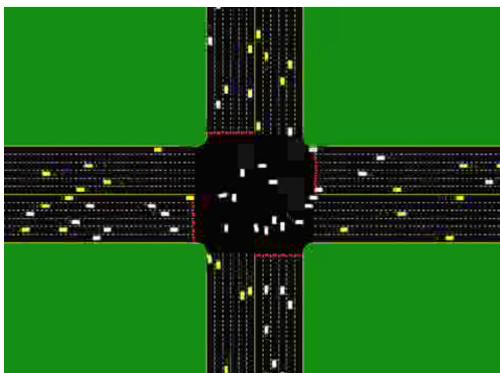


Figura 36: Simulación de comportamiento en intersección basada en un . En ésta, cada uno de los vehículos representa a un vehículo real que posee un controlador para hacerlo autónomo. Modelar este caso de estudio con una arquitectura basada en permite centrarse en el diseño del agente en concreto (i.e. 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/>).

tidad 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 entra 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 I. Por ello, este apartado únicamente hará una pequeña introducción a estudios existentes y aplicaciones de simuladores basados en este modelo.

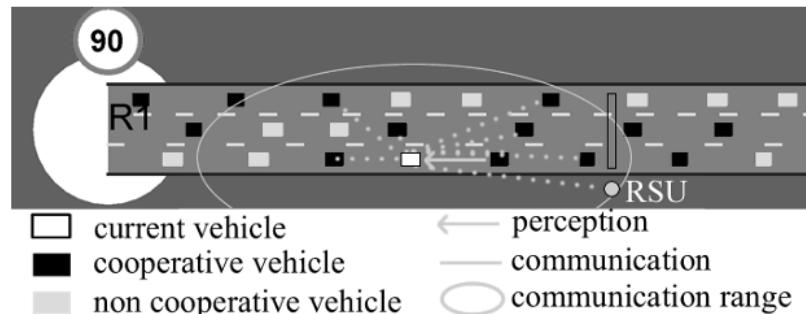
A diferencia de los **CAs**, los **Sistemas Multiagente** pueden emplazarse en un entorno virtual que represente un espacio continuo y no discreto. Esto permite modelar con mayor fidelidad magnitudes físicas asociadas a cada agente (e.g. posición y velocidades actuales, dimensiones del vehículo, masa, 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 .

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 36. 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). 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 **CA**, 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** ≡ 1 agente, dando así una enorme cantidad de posibilidades a experimentar. Por ejemplo en [Das et al., 1999] se hace uso de sistemas difusos 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.

Figura 37: 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 carros. Fuente: [Gu et al., 2015].



Los estudios en materia de simuladores de tráfico 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 por 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 intravehícuulares, [Vehicle-to-Vehicle \(V2V\)](#) y [Vehicle-to-Infrastructure \(V2I\)](#) respectivamente. El modelo de [Sistema 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 37). 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 si (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, gestionando 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.

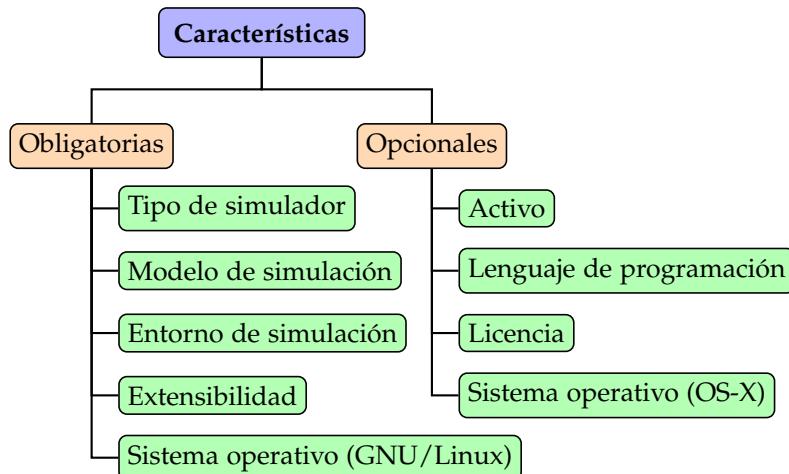


Figura 38: Características obligatorias y deseables del simulador donde implementar nuestros modelos personalizados de conductor.

### *Software de simulación*

Para la realización de esta tesis es necesario contar con un paquete de simulación que permita modelar un 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 38):

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 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 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.

#### *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 el cuadro 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.

	AORTA	MatSIM	MitSIM	MovSim	SUMO
Activo	✗	✓	✗	✓	✓
Lenguajes de programación	Scala	Java	C++	Java	C++ y Python
Licencia					
Propietaria	✗	✗	✗	✗	✗
OSS	✓	✓	✓	✓	✓
GPL	✓	✓	✗	✓	✓
Extensibilidad					
Código fuente	✓	✓	✓	✓	✓
API	✗	✗	✗	✗	✓
Sistemas Operativos					
GNU/Linux	✓	✓	✓	✓	✓
OS X	✓	✓	✗	✓	✓
Windows	✓	✓	✗	✓	✓

Cuadro 1: Tabla comparativa donde se contrastan las características de los si-

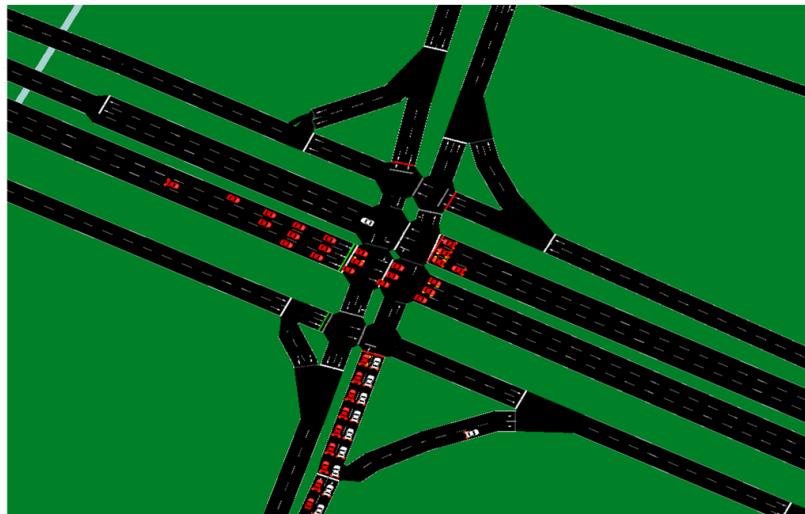


Figura 39: Captura de pantalla del simulador **SUMO**. Además de software de simulación propiamente dicho, **SUMO** provee de una interfaz gráfica que permite una visualización general, de zonas y de elementos en concreto a la vez que permite la variación de configuración de la simulación durante el desarrollo de la misma. **TODO!** Meter una imagen de nuestras simulaciones cuando estén, a poder ser sin color. Hacer el alto más pequeño

## 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 39) 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.
- 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).

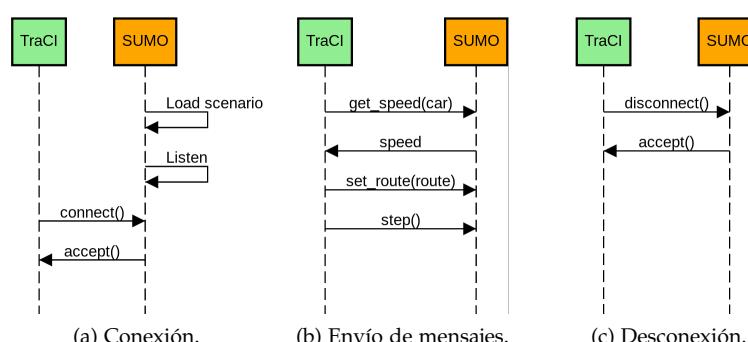
**SUMO** usa como modelo por defecto de *car-following* el modelo de Stefan Krauß [Jin et al., 2016], debido a su simplicidad y su velocidad de ejecución y como modelo de cambio de carril el modelo de Gipps [Krajzewicz et al., 2002]. No obstante, se encuentran paraseleccionar otros modelos como el **Intelligent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

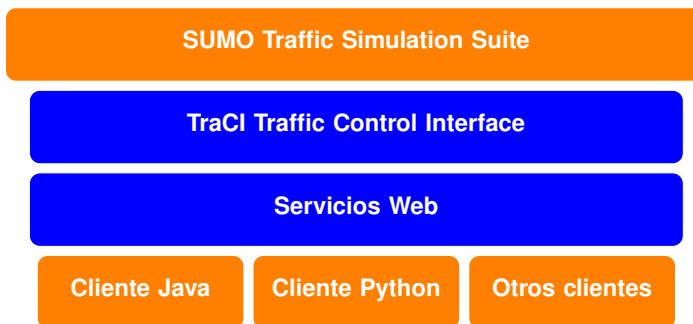
### La interfaz **TraCI**

**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.

Figura 40: **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.



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 40 (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 envíe una señal de desconexión (figura 40 (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 40 (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>49</sup>. El homólogo de la librería de abstracción de Python pero para el lenguaje Java. Está desarrollada por un tercero.
- **TrasS**<sup>50</sup>. Una plataforma ofrecida como SaaS que proporciona una interfaz de servicios web bajo protocolo SOAP para abstraer el protocolo en mensajes HTTP (figura 41).

Figura 41: 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.

<sup>49</sup> <https://github.com/egueli/TrACI4J>.

<sup>50</sup> <http://traas.sourceforge.net/cms/>.



# 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 , 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 **Sistemas Avanzados de ayuda a la Conducción (ADASs, Advanced Driver Assistance Systems)**.

El modelo de [Michon, 1985] define tres niveles de abstracción para las tareas que requieren procesos cognitivo, 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 42).

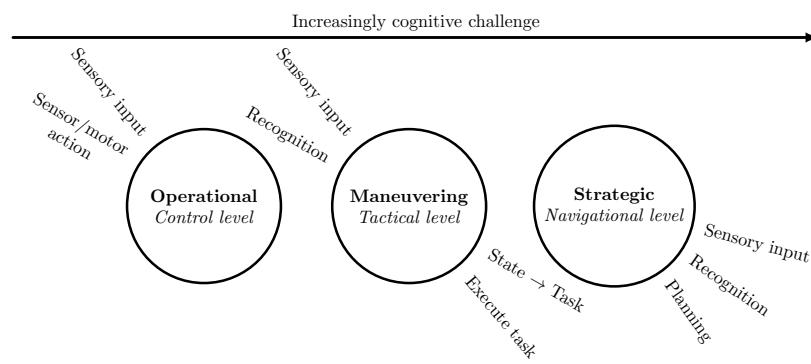


Figura 42: Los tres niveles jerárquicos que describen la tarea de conducción según [Michon, 1985]: *estrategia* (i.e. las decisiones generales), la *maniobra* (i.e. decisiones durante la conducción de más corto plazo) y *control* (i.e. automatismos).

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.

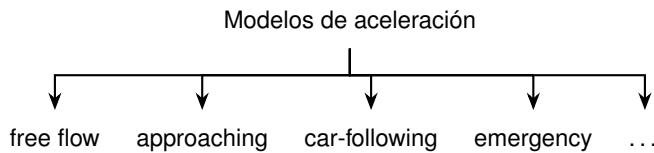
El comportamiento de un conductor al volante tiene una relación directa con el nivel de abstracción táctico. Se puede entender como el nivel encargado de planificar acciones a corto plazo para conseguir objetivos a corto plazo. Las tareas de control son automáticas e influyen poco o nada en la toma de 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<sup>51</sup>.

<sup>51</sup> No obstante algunos trabajos han demostrado que en ocasiones la planificación de la ruta sí afecta a decisiones normalmente asociadas al 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 . 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 , alrededor de los años 90, 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 utilizadas es de todo tipo. 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. utilitarios o vehículos de grandes dimensiones) y magnitudes físicas (tamaño, velocidad máxima). Dependiendo del trabajo, algunos autores añaden más conocimiento.



miento 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 .

### *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 (figuras 44 y 43): el de la **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.

El tráfico real, sin embargo, no está compuesto por un sólo carril, sino por varios. Los modelos de **cambio de carril** (figure ??) tienen como objetivo identificar cuándo el conductor desea cambiar de carril y realizar dicho cambio, ya sea porque 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).

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 carga de tráfico de la vía en la que se ejecutan. Por un lado, si la carga de tráfico es de ligera, mejora la velocidad media del flujo de la

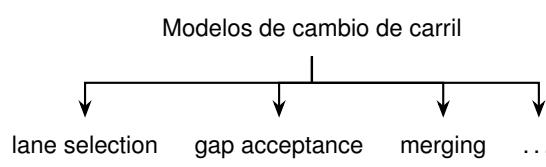


Figura 43: Tras la aparición de los modelos psico-físicos se comprobó que los umbrales en las percepciones y por tanto el comportamiento podía variar dependiendo de las situaciones. Por ello, el *car-following* no era más que una entre diferentes clases o regímenes de aceleración. Algunos de los regímenes 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 44: El cambio de carril 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. Otros autores pueden llegar a añadir operaciones más especializadas.

Figura 45: Representación de los vehículos en una vía junto con la nomenclatura a usar durante el resto de la tesis.



Ilustración estupenda y maravillosa de los vehículos en la vía y cómo están relacionados unos con otros.

vía. Sin embargo, según aumenta la carga de tráfico, 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* ([\[Laval and Daganzo, 2006\]](#)).

### *Nomenclatura*

Para no llevar a equívocos, en la figura 45 se ilustran los actores típicos en una situación de tráfico junto con los nombres y su rol. A continuación los explicamos:

- **Lag car.**
- **Lead car.**

### *Modelado de conductores clásico*

El **Modelo GHR**, presentado en [[Chandler et al., 1958](#)], es el modelo más conocido antes de la introducción del modelo de Gipps. Desarrollado a finales de los años 50 dentro de la *General Motors* (por ello también se le conoce como modelo **Generalized Model (GM)**), se caracteriza por el uso del concepto *estímulo → respuesta*, 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:

$$a(t) = cv^m(t) \frac{\Delta v(t - \tau)}{\Delta x^l(t - \tau)} \quad (17)$$

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.

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](#)]. Un vehículo está en una situación car-following cuando su velocidad está condicionada por el vehículo que se encuentra frente a él. En el primer modelo concreto ([\[Pipes, 1953\]](#)), el comportamiento responde a tratar de mantener un espacio variable en función de la velocidad.

Este modelo se puede considerar de una clase que denominaremos **mantenimiento de medida** 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 parachoque delantero o trasero.

Más adelante, a finales de la década se presentó el modelo **Gazis-Herman-Rothery (GHR)** (ver ecuación 17), el cual sirvió como base para el desarrollo de muchos modelos posteriores. Ese modelo dio origen a una nueva clase de modelos de conducción, los de tipo **estímulo → respuesta**.

Figura 46: 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

**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 del efecto de segundos coches delanteros [Bexelius, 1968].

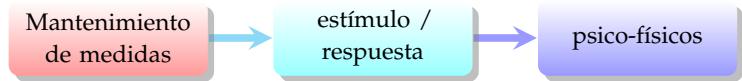
Los métodos de estas dos clases tienen un problema principal: suponen que el conductor es capaz de percibir incluso el más ínfimo cambio en las variables observadas, cuando la realidad no es así. Por ello, a mediados de los años 70 apareció una nueva clase de modelos de *car-following*, denominados posteriormente 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>52</sup>. El segundo, la diferenciación entre la selección de carril o *lane-selection* y la ejecución del cambio o *merging*.

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 actúan 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.

En general se basan en una fórmula que determina si el cambio es viable o no en función de una serie de parámetros entre los que se incluye el hueco del carril destino. En la ecuación 18 se describe el modelo típico de un modelo de *gap acceptance*.



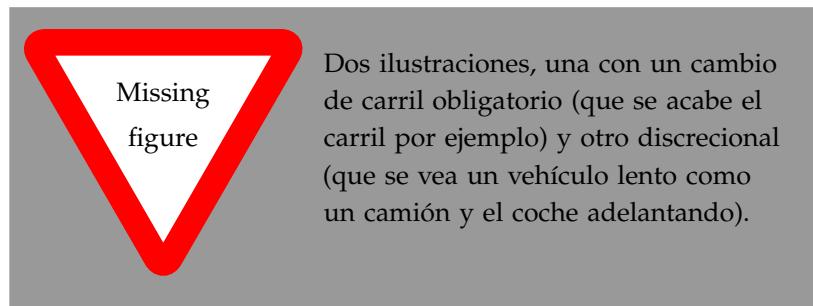
<sup>52</sup> En [Sparmann, 1978] el autor entiende de la derecha como el carril lento y la izquierda como el carril rápido, y es como se entiende dentro del contexto de esta tesis. Sin embargo en otros países esta correspondencia es al revés.

El modelo típico del **gap acceptance** responde a la ecuación 18, donde en un momento  $t$ , el cambio a un carril  $l$  es viable ( $f_{gl}(t) = 1$ ) o no ( $f_{gl}(t) = 0$ ) dependiendo de si el espacio en el carril destino  $g_l(t)$  es mayor o menor que un "hueco crítico" (en inglés critical gap)  $g_l^{crit}(t)$ .

$$f_{gl}(t) = \begin{cases} 0 & \text{si } g_l(t) < g_l^{crit}(t) \\ 1 & \text{si } g_l(t) \geq g_l^{crit}(t) \end{cases} \quad (18)$$

Por otro lado, existen autores que definen factores de influencia que modifican el modelo típico. Algunos ejemplos de factores pueden ser la velocidad absoluta del vehículo ([Gips, 1986, K. et al., 1996]), el tipo de cambio (**Mandatory Lane Change (MLC)** o **Discretionary Lane Change (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, ?]).

Figura 47: Los cambios de carril se clasifican como aquellos necesarios para continuar con la conducción (obligatorios) y aquellos útiles para mejorar la situación de conducción (discretionales).



Dos ilustraciones, una con un cambio de carril obligatorio (que se acabe el carril por ejemplo) y otro discrecional (que se vea un vehículo lento como un camión y el coche adelantando).

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 43).

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 **MLC**) cuando los vehículos se ven obligados a abandonar el carril que ocupan. Por otro lado, pueden ser **discretionales** cuando el cambio obedece a motivaciones más relacionadas con la mejora del confort o de la situación actual de conducción (ver figura 47).

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

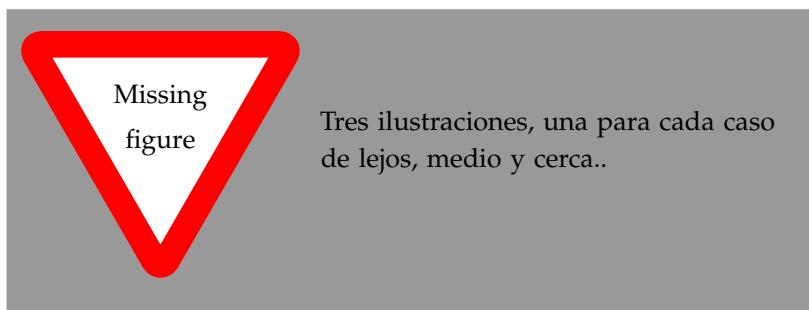


Figura 48: En el modelo de [Gipps, 1986], la distancia al punto (i.e. cerca, media distancia y lejos) determina el grado de obligatoriedad del cambio.

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). 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).

El modelo de Gipps sin embargo, sufre de dos problemas clave. Muchos de estos problemas fueron heredados por posteriores trabajos que basaban su funcionamiento en éste.

**UN CAMBIO DE CARRIL NO SÓLO INVOLUCRA** al conductor que lo ejecuta. En [?], 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 presenta 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 cual 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.

En el trabajo de [Weidemann and Reiter, 1992] 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: (i) **free-flow**, donde el comportamiento no se ve afectado por el del vehículo delantero), (ii) **car-following**, donde el comportamiento sí se ve influenciado por el vehículo delantero, obligando a disminuir la velocidad deseada en el conductor en cuestión, (iii) **approaching**, situación intermedia entre las dos anteriores y (iv) **emergency**, donde la situación es crítica (e.g. colisión inminente) obligando, normalmente, a respuestas extremas.

Esto es sólo un ejemplo, y diferentes autores identifican diferentes situaciones dependiendo del alcance del trabajo como por ejemplo el *close-following* o el *stop-and-go* [Toledo et al., 2003, Liu and Li, 2013]).

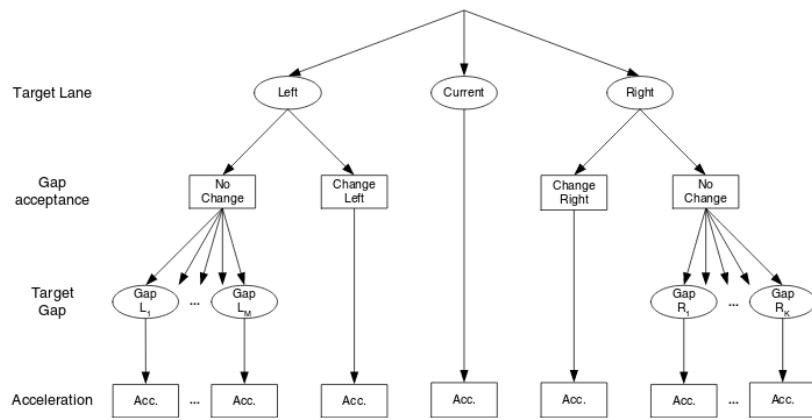


Figura 49: Estructura del modelo de comportamiento de los vehículos propuesto por [Toledo et al., 2007]. Este modelo se basa en el concepto de “objetivo a corto plazo” para elaborar un “plan a corto plazo” apoyándose, para ello, en un árbol de decisión. Aunque mantiene la clasificación, es probabilístico y existe opción de realizar un **DLC** en lugar de un **MLC** aún en situaciones donde acciones de 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. Fuente: [Toledo et al., 2007].

ADEMÁS, AL USAR ÁRBOLES SECUENCIALES, los factores son evaluados uno detrás de otro hasta encontrar una situación favorable, en cuyo caso el resto de factores no son evaluados. Por ejemplo, tal y como está formado el modelo de Gibbs, y algunos basados en éste como [?], 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. Ejemplos de avances en esta línea de trabajo pueden ser [Toledo et al., 2003, Toledo et al., 2007, Wei et al., 2000] (figura 49).

### *Modelado basado en*

Desde mediados de los años 90 empezó a crecer el interés por aplicar la a los **ITS**. Hay dos razones por las que esto es así: la primera, los éxitos cosechados por la , los cuales atrajeron a investigadores de multitud de áreas incluida ésta. La segunda, el rápido desarrollo de la tecnología <sup>53</sup>, que posibilita la existencia de conjuntos de datos masivos con la capacidad de explotarlos y aprender de ellos.

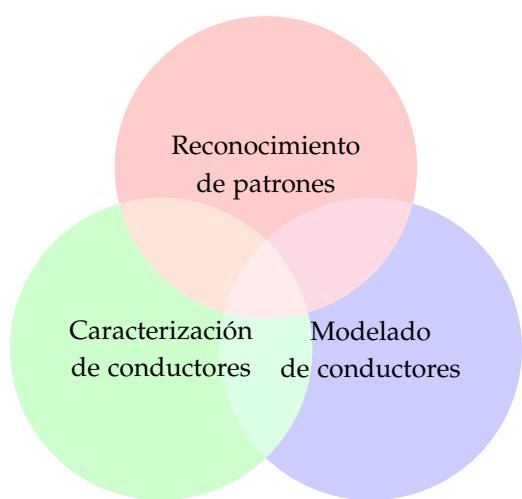
Algunos autores ([Zhang et al., 2011]) se atreven a afirmar incluso que el futuro de las **ITS** son las técnicas de la **IC**, y que los resultados que se puedan cosechar de técnicas basadas en el desarrollo convencional de sistemas es marginal comparado con las que se obtendrán con el nuevo enfoque.

Dentro de las **ITS**, las áreas de aplicación de la **IC** se centran en los siguientes conceptos: **reconocimiento de patrones**, **caracterización de conductores** y **modelado de conductores**. Es necesario mencionar que aunque se trate de áreas distintas, éstas suelen retroalimentarse, y es difícil encontrar estudios que se centren en una única área sin tocar el resto (figura 50). Por ello, aunque nuestro interés pueda centrarse en el modelado de conductores, es necesario conocer el estado de las demás áreas.

- En el **reconocimiento de patrones** las técnicas suelen trabajar en

<sup>53</sup> La tecnología es cada vez más barata, más precisa y con más funcionalidades. En la última década hemos vivido una explosión de dispositivos de todo tipo: teléfonos y relojes con GPS, acelerómetros y giroscopio, ordenadores completamente funcionales del tamaño de una moneda, sensores RADAR y LIDAR para uso amateur además de profesional y así un largo etcétera.

Figura 50: Las principales áreas de aplicación de la en los ITS son el reconocimiento de patrones, la caracterización de conductores y la modelización de los mismos. 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.



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 el trabajo a través de información en curdo es inviable (no digamos ya cuando los datos son extraídos de una flota de vehículos).

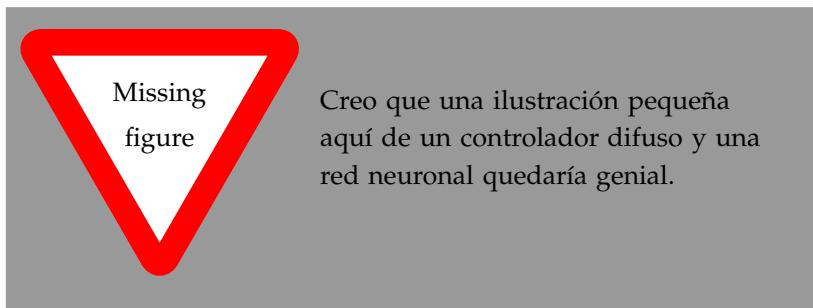
- La **caracterización de conductores** es interesante debido a que permite la identificación de perfiles de conducción y su clasificación de acuerdo a indicadores extraídos de su manera de conducir.
- El **modelado de conductores** nos permite la reproducción de comportamientos en simulación.

Las técnicas de IC más utilizadas en el modelado de conductores son las Redes Neuronales Artificiales y la . 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*).

**LAS REDES NEURONALES ARTIFICIALES** fueron el punto de entrada de la IC en la modelización de conductores. Los perceptrones multicapa y sus nuevas técnicas de entrenamiento se habían convertido en la nueva solución universal para todo aquel problema del que se dispusiese de datos.

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



Creo que una ilustración pequeña aquí de un controlador difuso y una red neuronal quedaría genial.

Figura 51: Al trabajar con métodos como las Redes Neuronales Artificiales o la , la inexactitud y la incertidumbre son ciudadanos de primera clase y forman parte de los modelos.

El primer trabajo de la literatura sobre la aplicación de [ANNs](#) 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 desarrolló uso de [ANN](#) aplicadas al entorno del vehículo para identificar el entorno y determinar cuándo y cómo realiza el conductor 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, sólo que los valores se ajustan. Otros trabajos similares son [[Panwai and Dia, 2007](#), [Khodayari et al., 2012](#)].

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 recurrente (e.g. red de Elman). El por qué del uso de redes recurrentes es por su capacidad de reconocer patrones dinámicos, los cuales son de esperar en este tipo de comportamientos.

EL PRIMER USO DE LA fue contemporáneo al de las [ANN](#), y también aplicada a un modelo de *car-following*. Después de todo los modelos psicofísicos 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 difusa 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 difusa sobre un modelo de aceleración de tipo *car-following*. Utilizaron el modelo GHR (Ecuación 17) como base y determinaron las entradas al modelo como valores de pertenencia a conjuntos difusos. Las entradas del modelo eran las 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 L0gic 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).

Los modelos hasta el momento se basaban en conjuntos difusos y reglas definidos *ad-hoc*. En [Chakroborty and Kikuchi, 2003] se introduce el concepto de personalización, ajustando los conjuntos difusos de las variables de entrada y salida a valores extraídos de conductores reales. Este ajuste se realiza mediante la representación del controlador difuso como red neuronal y posterior ajuste mediante entrenamiento de dicha red (*back-propagation*). A este trabajo le siguen muchas otras aproximaciones *neuro-fuzzy* como [?, 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 adjacentes<sup>54</sup>.

**LAS REDES NEURONALES ARTIFICIALES Y LA 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

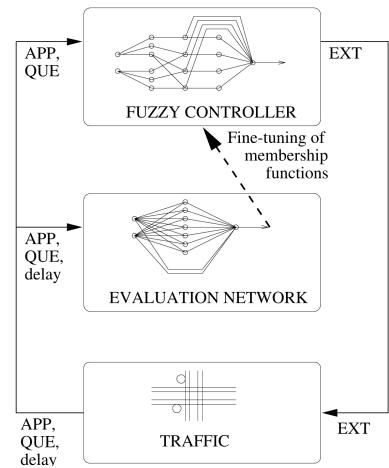


Figura 52: Un ejemplo de aproximación *neuro-fuzzy*, en este caso aplicado al control de señales de tráfico. El se implementa como una Red Neuronal Artificial de tipo *feed-forward* en lugar de con una representación tradicional. Además, el sistema completo lleva integrado un subsistema basado en también en Redes Neuronales Artificiales *feed-forward* que ajusta las funciones de pertenencia a través de entrenamiento por refuerzo. Fuente: [Bingham, 2001]

<sup>54</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, ?]. Estos trabajos, no obstante, no pertenecen al área de la **IC**.

en Support Vector Machine (SVM) 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 modelos 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 LLocal 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, 2003, Zheng et al., 2013] se ocupan exclusivamente del cálculo de tiempo de respuesta del conductor en modelos de *car-following*, el primero con controladores difusos y el segundo con ANNs.

#### *Papers a añadir o al menos relevantes*

Analysis of Recurrent Neural Networks for Probabilistic Modeling of Driver Behavior (de Morton J, Wheeler T, Kochenderfer M). Explora redes recurrentes para estudiar modelos de aceleración. Review of microscopic lane-changing models and future research opportunities. Su propio nombre dice de qué va. A review of intelligent driving style analysis systems and related artificial intelligence algorithms. Este lo mismo sirve para completar algún hueco que haya quedado en el estado del arte. Es relativamente reciente.

# Modelos de comportamiento



# *Metodología*

Los modelos de conducción propuestos se basarán en datos extraídos de entornos reales. Tras su generación, se probará posteriormente en el entorno de simulación **SUMO** para comprobar su desempeño. Es importante por tanto reparar en los siguientes detalles:

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

Por tanto, los parámetros de entrada y de salida han de ser procesados para el entrenamiento de los modelos de tal manera que el modelo funcione correctamente 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 transformación y los entornos de entrenamiento utilizados para la generación de los modelos.

## *Perspectiva general del modelo de conducción*

El entorno de simulación con el que trabajamos, **SUMO** impone ciertas limitaciones, siendo una de las más decisivas el funcionamiento “por carril”. Esto es, aunque el simulador sea continuo en el espacio, su recorrido longitudinal funciona en una dimensión, siendo el cambio de carril el salto de un carril a otro.

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 con 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 53 se describe el modelo, las entradas y salidas y el flujo de datos entre componentes.

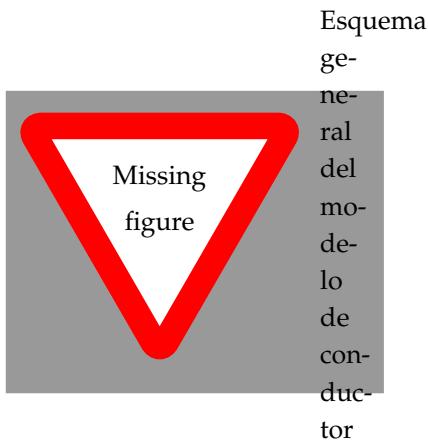


Figura 53: Esquema general del modelo de conductor planteado en la tesis. En éste se puede ver cómo se distribuyen los estímulos de entrada entre los diferentes componentes del modelo y las salidas del sistema, que irán conectadas a los actuadores pertinentes.

Estos modelos serán entrenados siguiendo un esquema supervisado, por lo que necesitamos datos reales a partir de los cuales deberá ajustar su funcionamiento. La información que se considera suficiente para que los modelos desempeñen su función en el entorno de simulación y la razón por la cual se ha tenido en cuenta 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. Se considera por tanto que el ajuste de la velocidad y, sobre todo los cambios de carril, se ven influenciados por éste.
- **Velocidad actual del vehículo y velocidad máxima del carril.** Tanto la velocidad en la vía como del vehículo en un momento concreto influye en cómo el conductor va a modificar su aceleración en momentos posteriores.
- **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 va muy lento
- **Siguiente salida y carriles cortados.** Se considera que este tipo de información es crucial a la hora de realizar cambios de carril, ya que además de lo obvio, puede influir en otras maniobras tales como un adelantamiento.
- **Señales luminosas.** Es interesante contar con este tipo de señales ya que pueden influir en diferentes patrones de aceleración (e.g. estamos cerca y cambia de color a ámbar) o desaceleración (e.g.

semáforo en rojo). 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.

### *Extracción de datos de conducción*

Se ha hecho uso de un vehículo instrumentado con sensores para capturar la mayor cantidad posible de información especificada en la introducción. El vehículo en cuestión es un Mitsubishi iMiEV (Figura 54) y los dispositivos un LiDAR, un GPS, el Bus CAN y una cámara.



Figura 54: El vehículo utilizado en los ensayos realizados. Se trata de un Mitsubishi iMiEV instrumentado con un LiDAR anclado en la baca superior, un GPS anclado en el techo, un puerto de acceso directo al Bus CAN y una cámara Microsoft Kinect tras el espejo retrovisor.

Todos los dispositivos se conectan a un ordenador con sistema operativo GNU/Linux sobre Intel i7-7500U CPU con 16GB de memoria RAM siguiendo el esquema que se muestra en la Figura ???. Al ser éstos dispositivos con capacidades 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 III. A continuación se describen los dispositivos usados y su información generada.

**LiDAR** Un LiDAR es un dispositivo que usa uno o más haces de luz pulsada para el cálculo de la distancia a los objetos donde éstos impactan.

Nuestro dispositivo en concreto es el modelo VLP-16 de la empresa Velodyne.

Está compuesto por un único haz láser con un movimiento continuo vertical y horizontal capturando información con un vertical de  $\pm 15^\circ$  distribuido a lo largo de 16 planos (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 de 75000



Figura 55: El **LiDAR** VLP-16 de Velodyne tiene un FOV horizontal de 360° y vertical de ±15°, permitiendo una captura de todo el entorno circundante a una frecuencia de hasta 20 Hz. Fuente: <http://velodynelidar.com/vlp-16.html>.

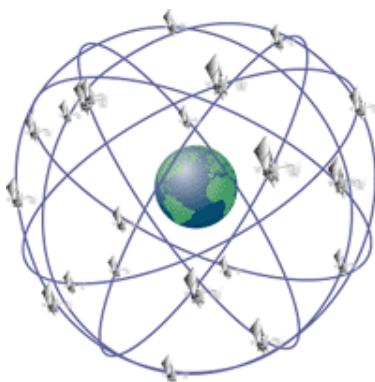


Figura 56: El **GPS** permite el posicionamiento 3d sobre en la tierra gracias a la comunicación unidireccional de satélites situados en órbita. Fuente: Wikimedia Commons.

a 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 de un nodo de **ROS** para este tipo de dispositivos **TODO!** Explicarlo mejor e indicar el nombre del paquete y su dirección. Este nodo publica en un topic específico la nube de puntos en coordenadas cartesianas con el eje X definido en el sentido de los 0° y el eje Z en sentido ascendente.

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

**GPS** El **GPS** es una tecnología usada para sincronización de tiempo y posicionamiento 3d (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.

De todos los mensajes disponibles, los recuperados son lo 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 un nodo de **ROS** **TODO!** Explicarlo mejor e indicar el nombre del paquete y su dirección 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 **Controller Area Network (CAN)**, o simplemente **CAN**, es un protocolo de comunicaciones basado en esta topología utilizado, entre otras muchas áreas, la comunicación entre los diferentes dispositivos que componen un vehículo.

El estándar CAN cubre únicamente las dos primeras capas del modelo OSI<sup>55</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 general, por lo que lo normal es realizar una tarea previa de ingeniería inversa para identificar los mensajes correspondientes a los datos que se desean almacenar y su frecuencia.

Del bus 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 la aceleración, por lo que se ha usado para validar la velocidad capturada por el GPS.

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 LACIWEL 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 esta tesis y descrito brevemente en el apéndice *Sistemas desarrollados*.

**Cámara** Para la obtención de algunos valores de indicadores no existentes 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 entorno. Para ello se ha hecho uso de una cámara para 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 ROS denominado *freenect*<sup>56</sup> publicando en un topic las imágenes generadas.

### Selección de rutas

Para la captura de datos se han propuesto dos rutas, en adelante  $R_1$  y  $R_2$ , consideradas equivalentes. Se tratan 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 59 muestra los recorridos en el mapa.

<sup>55</sup> Las dos primeras capas son la capa física (la conexión hardware a la red y la transmisión/recepción física de los datos) y la de enlace de datos (direccionalamiento 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.

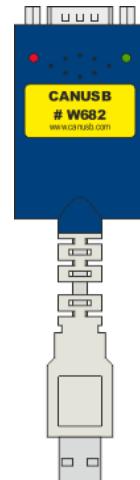


Figura 57: El dispositivo CANBUS de LACIWEL AB permite el acceso a través del protocolo RS 232 por el puerto USB al bus CAN. Fuente: <http://www.can232.com/>.



Figura 58: 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>56</sup> [http://wiki.ros.org/freenect\\_stack](http://wiki.ros.org/freenect_stack).

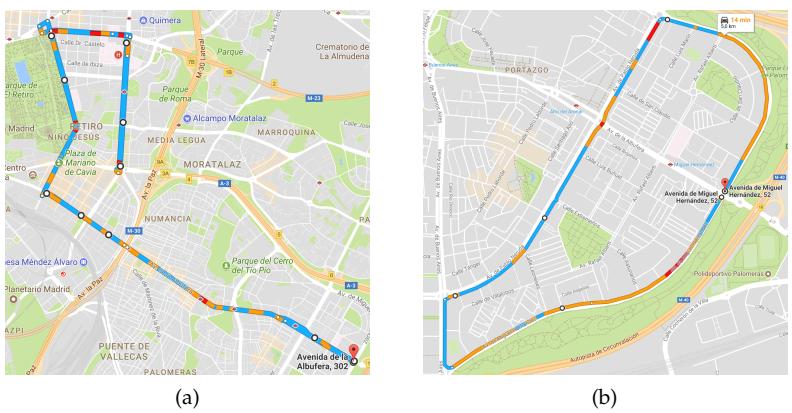


Figura 59: Los dos recorridos realizados para la captura de datos de conducción, ambos en entorno urbano. (a)  $R_1$  tiene una duración estimada de 30 min y sirve para la captura de los datos de entrenamiento. (b)  $R_2$  tiene una duración estimada de 15 min y sus datos serán utilizados para los conjuntos de test.

$R_1$  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 11:00am y las 12:00pm en días laborables, permitiendo una circulación con suficientes vehículos para requerir maniobras dependientes del entorno, pero sin demasiados como para impedir la circulación.

### 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](#), es decir varones dentro del rango de 35 a 39 años.

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<sup>57</sup>.

**TODO!** Quizá debería decir algo más?

### Preparación de los datos

**TODO!** Revisar las variables porque han cambiado. Están en el paso 03 de los scripts.

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.

### 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).

<sup>57</sup> Dado que los comportamientos de *car-following* y *lane-change* se asocian con el nivel cognitivo táctico, se ha querido reducir el impacto del operador decidido durante el experimento hacia dónde o no ir. De esta manera los conductores son libres de realizar el movimiento que deseen y de anticipar maniobras con más libertad.

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.

#### *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, siendo estos marcados como  $+1$  si es un cambio hacia la izquierda o  $-1$  si es un cambio a la derecha, y por tanto, todos aquellos momentos en los que no hay cambio de carril se marcan tendrán un valor de 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.

*Velocidad máxima de la vía* Se han añadido las velocidades máximas de las vías indicando las marcas temporales en las que el conductor entra o sale de cada uno de los tramos.

*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 geoposición 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.

*Distancia a recorrer 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>58</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>58</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.

### 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 2 describe qué variables son usadas en qué conjunto de datos.

Cuadro 2: 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	✓	✗
Cambio de carril	✗	✓

### *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. El 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 **Sistemas de Control Difuso** y **Perceptrones Multicapa** para modelar el comportamiento longitudinal y **Perceptrones Multicapa** y **CNNs** para modelar el comportamiento en cambio de carril.

Los modelos serán entrenados con los conjuntos asociados al total de sujetos (esto es,  $CT_{SA}$  para el modelo longitudinal y  $LT_{SA}$  para el modelo de cambio de carril). Tras ello, y una vez decididos los modelos que se usarán para el modelo de conducción, se entrenarán dichas arquitecturas para los sujetos por separado.



## 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 6o.

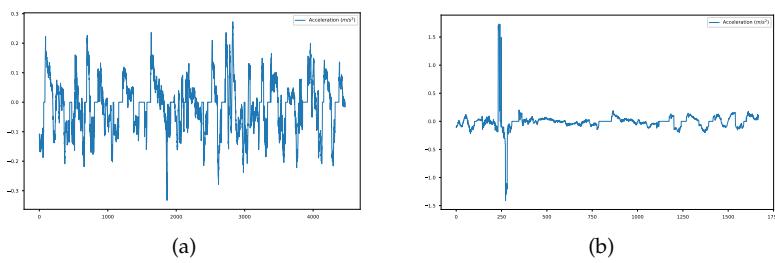


Figura 6o: Perfiles de aceleración de los conjuntos de entrenamiento y test para ajustar por los modelos.

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

1. **Perceptrón Multicapa.** 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>59</sup>
2. **Sistema de Control Difuso.** 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 Difuso tienen la ventaja de que se puede explicar cómo funcionan, cosa que no es posible para un Perceptrón Multicapa con una o más capas ocultas.

<sup>59</sup> El Teorema de aproximación Universal postula que un Perceptrón Multicapa con al menos una capa oculta es capaz de aproximar cualquier función si dispone de suficientes neuronas en ésta.

Ambos modelos se entrenaran ajustando sus parámetros con un procedimiento basado en el descenso del gradiente denominado ADAM [Kingma and Ba, 2014]. Su aplicación a los Perceptrón Multicapa es directa, pero para Sistemas de Control Difuso 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 controlador difuso basado en descenso del gradiente.

### Descripción de los datasets

Del conjunto de datos 2 descrito en el capítulo Metodología seleccionaremos aquellos indicadores de interés y generaremos un conjunto de entrenamiento y test para cada uno de los sujetos y un conjunto de entrenamiento y test para el total de conductores.

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 y la velocidad (la aceleración y la velocidad al líder respectivamente), por lo consideramos que disponemos de información temporal suficiente para este problema en concreto.

Cuadro 3: Descripción de los conjuntos de datos para el entrenamiento de los modelos.

Nombre	Entradas	Salidas	Tamaño (training)	Tamaño (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

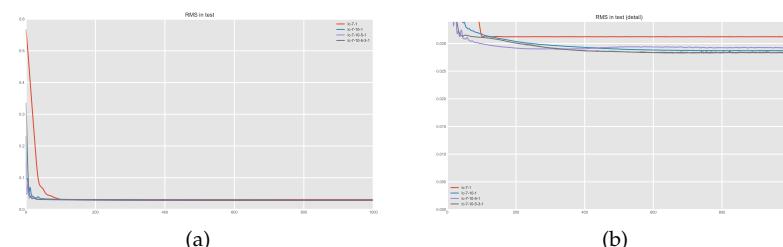
### Modelo Sistema de Control Difuso

Se han realizado entrenamientos sobre arquitecturas con diferente número de particiones difusas en las variables. Las arquitecturas que se han considerado más relevantes (tras un proceso de ensayo y error con diferente número de particiones difusas) se describen en la tabla 4.

Cuadro 4: Resumen de las arquitecturas de *Sistema de Control Difuso* para el modelo longitudinal. La posición de cada número de la topología indica a qué variable lingüística se refiere (*Distancia al líder*, Distancia a siguiente semáforo, Semáforo en verde, Semáforo en amarillo, Semáforo en rojo, Velocidad y velocidad de aproximación al líder respectivamente), siendo su valor el número de conjuntos difusos que configuran la partición difusa de la variable. Las arquitecturas seleccionadas en esta tabla son aquellas consideradas relevantes tras un [Figura con visión general y detalle](#) de la evolución del error en test de los diferentes controladores difusos ajustados.

Nombre	Arquitectura	Epochs	RMS		
			Training	Validation	Test
$FCS_1$	X, X, X	$10^5$	0.XXXXX	0.XXXXX	0.XXXXX
$FCS_2$	X, X, X	$10^5$	0.XXXXX	0.XXXXX	0.XXXXX
$FCS_3$	X, X, X	$10^5$	0.XXXXX	0.XXXXX	0.XXXXX
$FCS_4$	X, X, X	$10^5$	0.XXXXX	0.XXXXX	0.XXXXX

Cada uno de los controladores se ha entrenado durante 250,000 epochs. En la Figura 71 se puede observar la evolución en general y un detalle de la disminución del error en test de los controladores.



El proceso de entrenamiento seguido no ha sido el del ajuste de todas las variables en su conjunto. La razón principal de esto es que el

ajuste de las variables que determinan las particiones difusas parece suceder órdenes de magnitud más rápido que el ajuste de los pesos asociados a las reglas.

Por tanto, y para este problema en concreto, El entrenamiento se realiza iterativamente alternando conjuntos de epochs dedicados a las reglas y conjuntos de epochs dedicados a las variables de las reglas difusas.

En lugar de eso se ha partitionado el entrenamiento en secuencias sucesivas de entrenamiento de reglas y entrenamiento de particiones difusas. Concretamente, los 250,000 epochs de entrenamiento se corresponden a 250 iteraciones de 800 epochs ajustando sólo las reglas seguidos de 200 epochs ajustando las variables de las particiones difusas.

Empíricamente (y en 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.

Tras el entrenamiento, el controlador difuso que menor error en test arroja es el  $FCS_2$ <sup>60</sup>. Por tanto, éste será el modelo seleccionado para la comparativa final.

### Modelo Perceptrón 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 8.

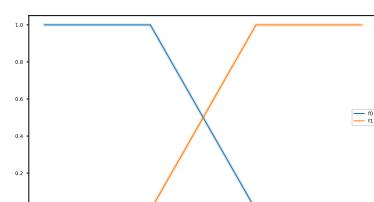
Nombre	Topología	Epochs	Dropout	RMS		Test
				Training	Validation	
$MLP_1$	7, 16, 1	$10^5$	0,1	0,052741	0,057301	0,059253
$MLP_2$	7, 8, 4, 1	$10^5$	0,1	0,056341	0,061951	0,056607
$MLP_3$	7, 16, 8, 1	$10^5$	0,1	0,046404	0,051878	0,059681
$MLP_4$	7, 16, 16, 8, 1	$10^5$	0,1	0,042789	0,046876	0,040971

El modelo de neuronas 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 en todas las neuronas salvo en la neurona de salida que se ha utilizado una activación lineal<sup>61</sup>. Los pesos de la red han sido inicializados con una muestra aleatoria uniforme de valores reales en el intervalo  $(-0,25, 0,25)$ .

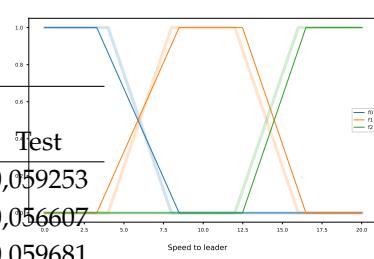
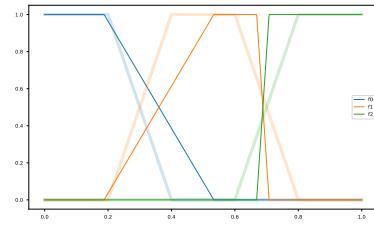
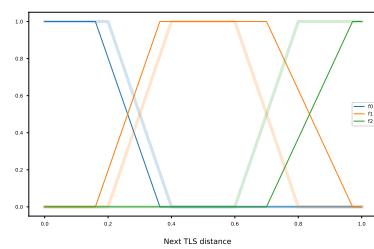
El error que se trata de minimizar es el error cuadrático medio entre los valores de aceleración del conjunto reales y los ajustados por el modelo. La Figura 72 muestra la evolución de este error durante el proceso de entrenamiento.

Estos errores se encuentran entre los  $0,05 \text{ m s}^{-2}$  y los  $0,07 \text{ m s}^{-2}$ , lo

<sup>60</sup> Para el controlador  $FCS_2$ , la forma de sus particiones difusas queda, para las variables binarias sin apenas ajuste:



El resto de particiones sí se han visto modificadas:



<sup>61</sup> Cuadro 5: Resumen de las arquitecturas de **Perceptrón Multicapa** para el modelo longitudinal. 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 **Entrenamiento** y **Prueba** para evaluar el estado inicial de la partición, y en **opaco** la forma de ésta tras el proceso de entrenamiento.

<sup>62</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.

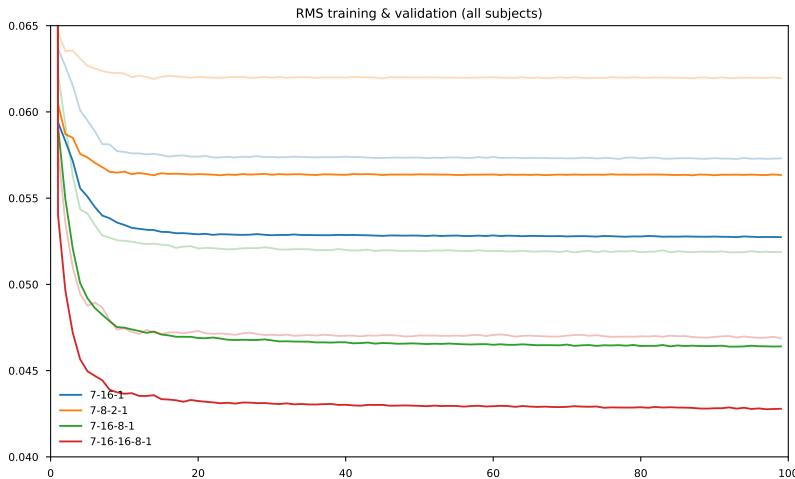
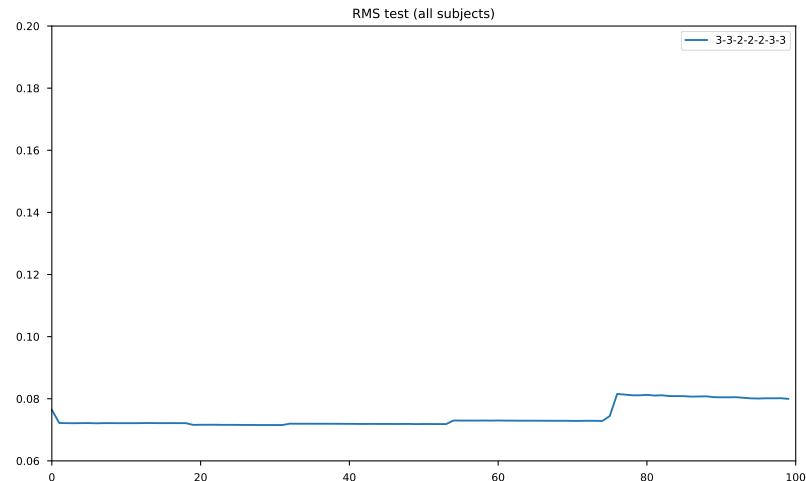


Figura 62: Visión en detalle de la evolución del error en los conjuntos de entrenamiento y validación. Para cada arquitectura, el color más transparente se corresponde al error en el conjunto de validación..

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 del problema tiene muchos mínimos locales o mesetas.

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), como podemos ver en la figura 73.

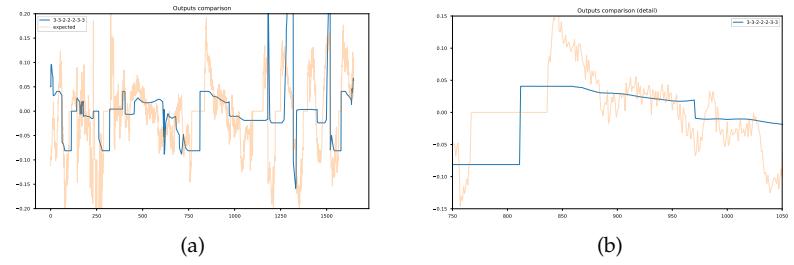
Figura 63: Visión en detalle de la evolución del error en el conjunto de test. Aunque no se ha considerado para determinar las arquitecturas, sí se ha recogido la información de la evolución del error en el conjunto de test debido a que nos ofrece puede ofrecer intuición de qué forma aprende la red. Por lo que podemos observar, para el problema en cuestión las redes más potentes tienden a sobre-entrenarse.



Una visión de detalle del ajuste de estas arquitecturas al conjunto de test se puede ver en la figura 74, 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 razonablemente bien, es razonable elegir el modelo  $MLP_2$  debido a que es el que aparentemente mejor generaliza los comportamientos

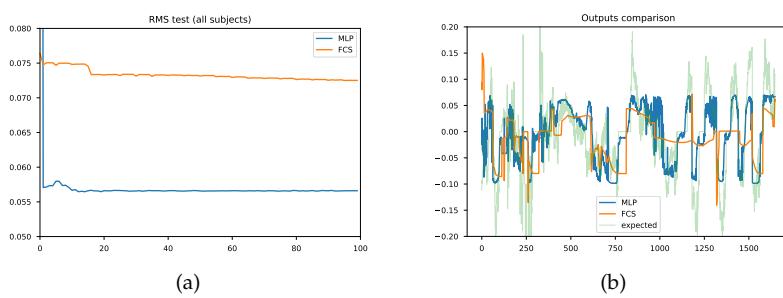
Figura 64: Comparación del perfil de aceleración real y el inferido por los modelos entrenados. En la visión general se puede observar, en transparente, el perfil real. A la derecha se amplía una pequeña sección del perfil para mostrar los diferentes ajustes de los modelos entrenados y cómo difieren del valor real.



del conjunto de conductores.

### Comparación entre modelos

Las mejores arquitecturas de ambos modelos han sido la  $MLP_2$  para los Perceptrones Multicapa y  $FCS_x$  para los Sistemas de Control Difuso. Los errores y el perfil de aceleración para ambos modelos se muestran en la figura 75.



Aunque el modelo basado en Sistema de Control Difuso 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 controlador difuso.

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

Figura 65: Comparación de la mejor arquitectura Perceptrón Multicapa frente a la mejor arquitectura Sistema de Control Difuso: (a) diferencia entre los errores cuadráticos medios de ambas arquitecturas y (b) perfiles de aceleración en el conjunto de test.



## Comportamiento de cambio de carril

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[?] se trataron de controlar muchos de estos factores dividiendo el problema en dos partes, la intención de cambio y la ejecución del cambio, fijando el primero de manera que los conductores sólo cambiaban de carril cuando se les ordenaba y permitiendo estudiar, de esta manera, cómo diferentes perfiles de conductores ejecutan de diferente forma el cambio de carril y 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 de ajuste será por tanto un problema de clasificación donde se tratará de maximizar el número de aciertos entre los cambios de carril realizados y el modelo a ajustar. La figura 66.

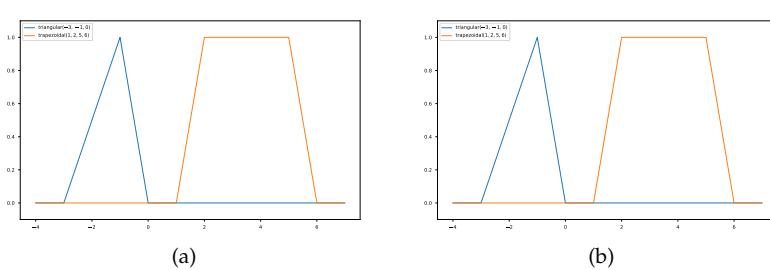


Figura 66: Perfiles de cambio de carril en los conjuntos de entrenamiento y de cambios de carril en el conjunto de test.

Las técnicas que se han seleccionado para tratar el cambio de carril son las siguientes:

1. **Perceptrón Multicapa.** 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 **CNNs**. Al representar la evolución temporal del entorno circundante como un espacio tridimensional, podemos aplicar esta técnica para la identificación de características de manera, supuestamente, más eficiente.

Ambos modelos, al igual que ocurría con el modelo longitudinal, han sido entrenados ajustando sus parámetros con el algoritmo ADAM [Kingma and Ba, 2014]. Las funciones de activación, sin embargo, varían, y por tanto la inicialización de las variables también<sup>62</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>63</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>64</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.

Antes de pasar a la descripción de los modelos, queda hablar del sesgo existente en los datos de entrenamiento. Debido a la naturaleza del 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 para entrenar de tamaño  $m$  compuesto por una selección aleatoria de todos los ejemplos equidistribuida entre las clases de los ejemplos (esto es, aproximadamente  $\frac{m}{3}$  para cada clase “cambio izda.”, “no cambio” y “cambio dcha.”).

### *Descripción de los datasets*

Del conjunto de datos ?? descrito en el capítulo **Metodología** seleccionaremos aquellos indicadores de interés y generaremos un conjunto de entrenamiento y test para cada uno de los sujetos y un conjunto de entrenamiento y test para el total de conductores.

<sup>62</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>63</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 **card**( $W_{in}^l$ ) y **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>64</sup> La normalización usada ha sido la operación denominada softmax, definida como:

$$\text{softmax}(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (19)$$

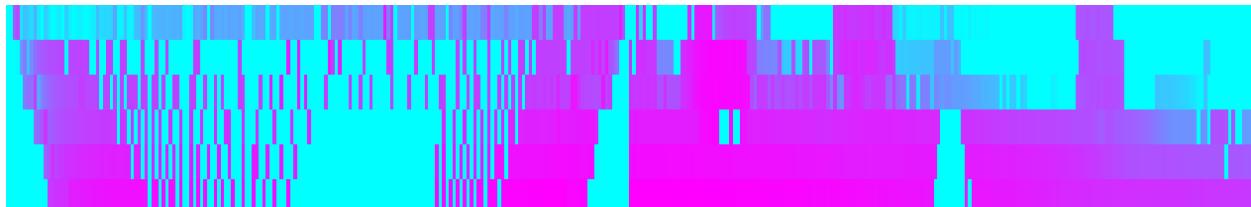
### *Representación de los datos*

Las secuencias de las que se compone el conjunto de datos están compuestas, aparte de variables numéricas, de 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 al origen.
2. La nube de puntos se origina a través de un dispositivo mecánico que funciona con coordenadas esféricas a una resolución horizontal de  $0.2^\circ$ <sup>65</sup> y vertical de  $2^\circ$ . 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.

<sup>65</sup> Funcionando a 10 Hz.

Para el primer caso, se ha optado por representar el entorno como un mapa de profundidad, ilustrado en la Figura 67. 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.



Dado que el LIDAR describe el entorno de manera discreta con valores constantes de elevación y azimuth, se puede 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*.
- 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.

Figura 67: Un ejemplo del mapa de profundidad asociado a la nube de puntos original **TODO!** sacar un mapa de profundidad de los nuevos, porque se veían mejor.

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<sup>66</sup>.

<sup>66</sup> TODO! Determinar si tiene sentido explicar aquí el proceso de generación del mapa de profundidad (el algoritmo).

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 lo suficientemente amplio para determinar un 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.

### *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*.

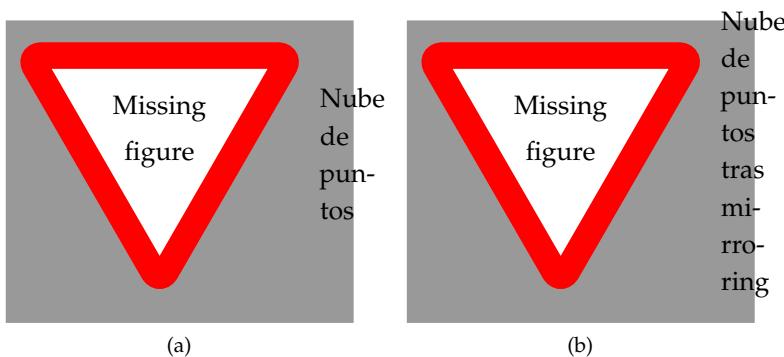
Por ello, se ha optado por realizar un proceso de generación de datos artificiales a partir de los datos existentes. De esta manera, ayudaremos al modelo a entrenar con casos similares y que de esta manera generalice mejor. La generación artificial se ha realizado sobre los datos recogidos en la ruta  $R_1$ , 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 las nuevas porciones de datos generadas mantengan una coherencia temporal. Por tanto, aunque pertenezcan al mismo conjuntos de datos, cada uno se mantiene en una secuencia independiente.

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

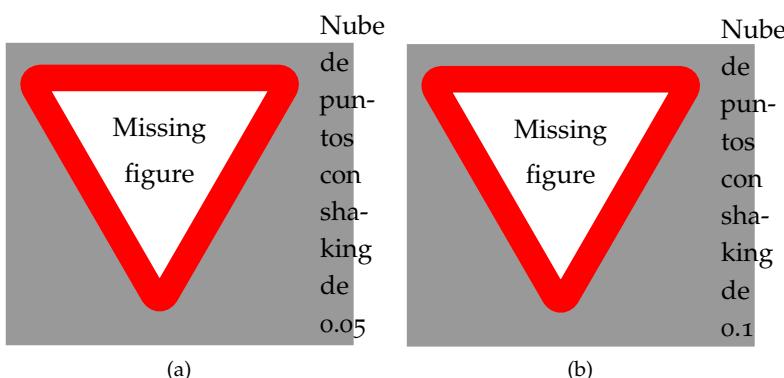
*mirroring* Se parte de la suposición de que los procesos cognitivos que producen determinados comportamientos (en nuestro caso, el cambio de carril) son los mismos independientemente de un cambio a la izquierda o hacia la derecha<sup>67</sup>.

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>68</sup>, de cada ejemplo obtenemos uno nuevo. En la figura 68 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 afectar añadir más ruido al existente en los mismos.

*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>69</sup>, se aplica un desplazamiento aleatorio ( $\delta x, \delta y, \delta z$ ) sobre cada punto tal y como se describe en [?]. La figura 69 ilustra dos procesos de shaking con diferentes desplazamientos sobre la nube original mostrada en la figura 68



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

<sup>67</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>68</sup> Es decir, invirtiendo los cambios de carril y la distancia recorrible en carriles izquierdo y derecho.

Figura 68: 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>69</sup> Debido a que la aplicación iterativa de este proceso sobre la misma nube de puntos acumula los errores entre iteraciones haciéndola ininteligible.

Figura 69: 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.

<sup>70</sup> La intuición de este proceso

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 <sup>70</sup>.

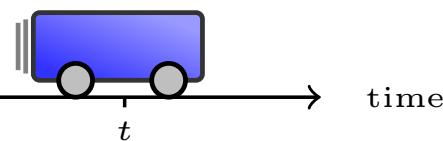
### Incorporación de información temporal

Las CNN 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 70). Los momentos que se tendrán en cuenta en los datasets finales serán  $t_0, t_{10}$ , 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 [?] <sup>71</sup>.

<sup>71</sup> Dichos experimentos se corresponden con 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 ejecución de entre 0.2s, 1.2s [2], por lo que es común que la ventana temporal elegida sea aceptable. Además, al incluir tres momentos temporales, es  $\frac{3}{10}s$  posible que el modelo reconozca una intuición de la velocidad, la aceleración e incluso el jerk a partir de imágenes estáticas. Figura 70: 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.



Tras este ajuste, los conjuntos de datos están finalizados. Sus propiedades quedan descritas en la Tabla 6.

Cuadro 6: Descripción de los conjuntos de datos para el entrenamiento de los modelos.

Nombre	Entradas	Salidas	Tamaño (training)	Tamaño (test)
$LC_{S_1}$	8653	3	68991	2058
$LC_{S_2}$	8653	3	84811	2887
$LC_{S_3}$	8653	3	82061	2891
$LC_{S_A}$	8653	3	248933	7836

## Modelo Perceptrón Multicapa

Los entrenamientos han sido realizados sobre diferentes topologías, siendo las más representativas las descritas en la tabla ??.

Nombre	Topología	Epochs	Dropout	Training	Precisión	Cuadro 7: Resumen de las arquitecturas de Perceptrón Multicapa para el modelo de cambio de carril. La posición de Test Training Validation indica la posición de los pesos en la topología.
				Training	Validation	Test
MLP <sub>1</sub>	X, X, X	10 <sup>5</sup>	0,1	0.XXX	0.XXX	0.XXX 0.XXX 0.XXX
MLP <sub>2</sub>	X, X, X	10 <sup>5</sup>	0,1	0.XXX	0.XXX	0.XXX 0.XXX 0.XXX
MLP <sub>3</sub>	X, X, X	10 <sup>5</sup>	0,1	0.XXX	0.XXX	0.XXX 0.XXX 0.XXX
MLP <sub>4</sub>	X, X, X	10 <sup>5</sup>	0,1	0.XXX	0.XXX	0.XXX 0.XXX 0.XXX

Se han entrenado más redes, pero no se han incluido al no aportar información relevante en la descripción del proceso de entrenamiento.

Cada una de las redes se ha entrenado durante XXX epochs. Debido a las limitaciones de memoria RAM en la máquina de entrenamiento, los epochs no incluyen todo el conjunto de entrenamiento en cada pasada, sino que se limitan a *batches* de YYY elementos seleccionados aleatoriamente del conjunto total de datos.

. En la Figura 71 se puede observar la evolución en general y un detalle de la disminución del error en test de los controladores.

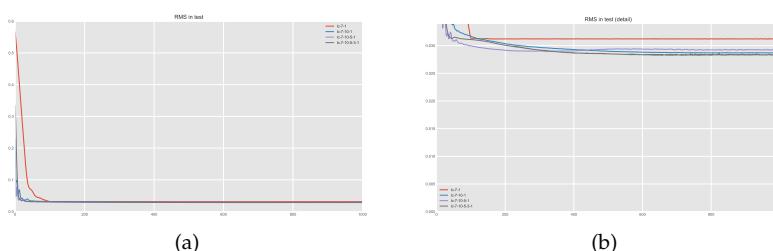


Figura 71: Visión general y detalle de la evolución del error en test de los diferentes controladores difusos ajustados.

El proceso de entrenamiento seguido no ha sido el del ajuste de todas las variables en su conjunto. La razón principal de esto es que el ajuste de las variables que determinan las particiones difusas parece suceder órdenes de magnitud más rápido que el ajuste de los pesos asociados a las reglas.

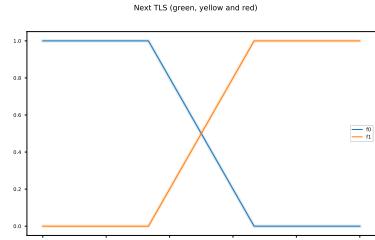
Por tanto, y para este problema en concreto, El entrenamiento se realiza iterativamente alternando conjuntos de epochs dedicados a las reglas y conjuntos de epochs dedicados a las variables de las reglas difusas.

En lugar de eso se ha particionado el entrenamiento en secuencias sucesivas de entrenamiento de reglas y entrenamiento de particiones difusas. Concretamente, los 250,000 epochs de entrenamiento se corresponden a 250 iteraciones de 800 epochs ajustando sólo las reglas seguidas de 200 epochs ajustando las variables de las particiones difusas.

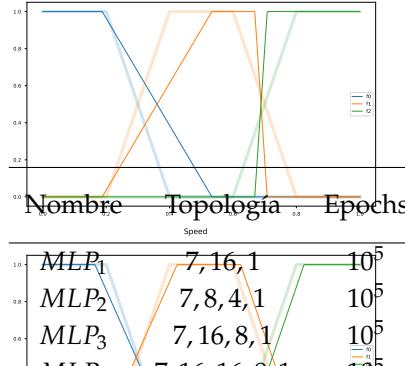
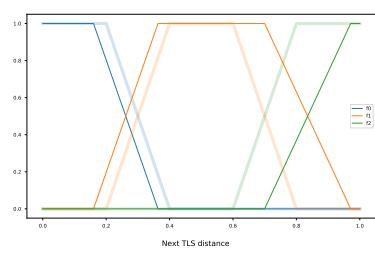
Empíricamente (y en este problema en concreto) se ha podido observar que el entrenamiento realizado de esta manera hace que el

RMSE descienda más rápido en el mismo número de iteraciones.

<sup>72</sup> Para el controlador  $FCS_2$ , la forma de sus particiones difusas queda, para las variables binarias sin apenas ajuste:



El resto de particiones sí se han visto modificadas:



En transparente se puede ver el estado inicial de la partición, y en opaco la forma de ésta tras el proceso de entrenamiento.

<sup>73</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.

Tras el entrenamiento, el controlador difuso que menor error en test arroja es el  $FCS_2$ <sup>72</sup>. Por tanto, éste será el modelo seleccionado para la comparativa final.

### Modelo CNN

**TODO!**Indicar que, en este modelo concreto, las imágenes de los deepmap se han modificado dinámicamente en el proceso de alimentación de la red de tal manera que los extremos derecho e izquierdo de éstas se corresponden con los bordes izquierdo y derecho respectivamente para mantener la coherencia espacial en la horizontal.

**TODO!**Indicar que a la red de convolución se la alimenta con todos los datos, pero que las convoluciones se realizan únicamente con imágenes, y el resto de parámetros se introducen en la fase de inferencia

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 8.

Nombre	Topología	Epochs	Dropout	RMS		
				Training	Validation	Test
MLP <sub>1</sub>	7, 16, 1	$10^5$	0,1	0,052741	0,057301	0,059253
MLP <sub>2</sub>	7, 8, 4, 1	$10^5$	0,1	0,056341	0,061951	0,056607
MLP <sub>3</sub>	7, 16, 8, 1	$10^5$	0,1	0,046404	0,051878	0,059681
MLP <sub>4</sub>	7, 16, 16, 8, 1	$10^5$	0,1	0,042789	0,046876	0,060971

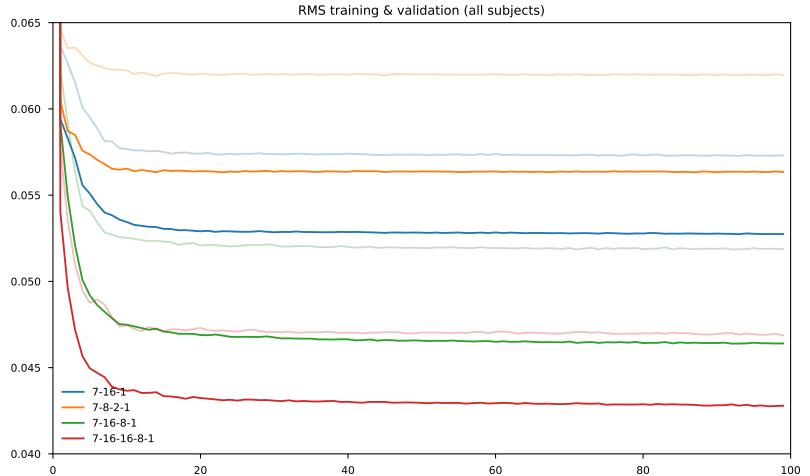
Cuadro 84 Resumen de las arquitecturas de Perceptrón Multicapa para el modelo longitudinal. La posición de cada número de la topología indica la capa, siendo su valor el número de nodos (neuronas) que incluye dicha capa. Las arquitecturas seleccionadas en esta tabla son aquellas consideradas relevantes tras un proceso manual de ensayo y error.

El modelo de neuronas 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 en todas las neuronas salvo en la neurona de salida que se ha utilizado una activación lineal<sup>73</sup>. Los pesos de la red han sido inicializados con una muestra aleatoria uniforme de valores reales en el intervalo  $(-0,25, 0,25)$ .

El error que se trata de minimizar es el error cuadrático medio entre los valores de aceleración del conjunto reales y los ajustados por el modelo. La Figura 72 muestra la evolución de este error durante el proceso de entrenamiento.

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 del problema tiene muchos mínimos locales

Figura 72: Visión en detalle de la evolución del error en los conjuntos de entrenamiento y validación. Para cada arquitectura, el color más transparente se corresponde al error en el conjunto de validación..



o mesetas.

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), como podemos ver en la figura 73.

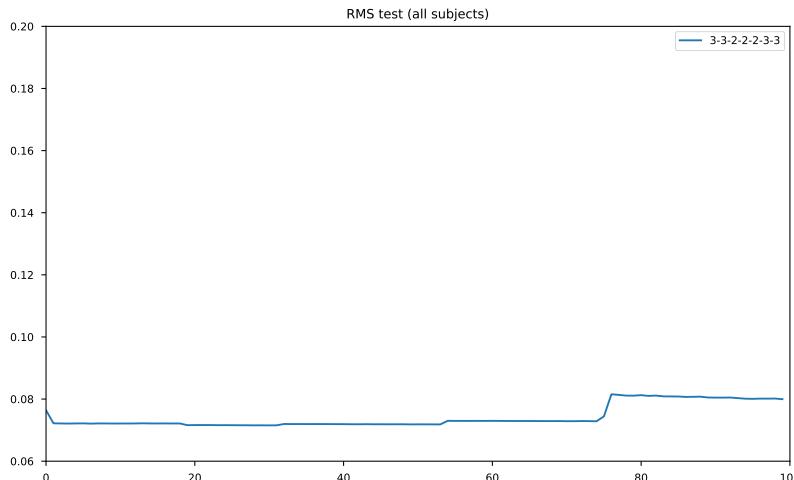
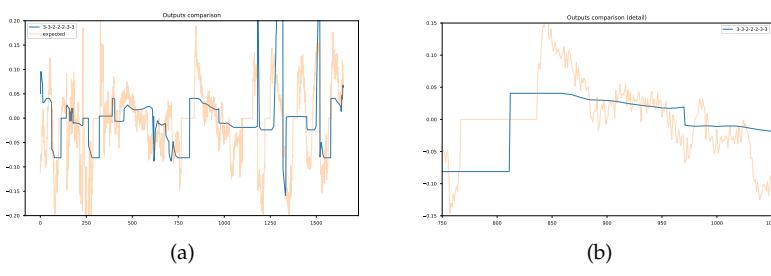


Figura 73: Visión en detalle de la evolución del error en el conjunto de test. Aunque no se ha considerado para determinar las arquitecturas, sí se ha recogido la información de la evolución del error en el conjunto de test debido a que nos ofrece puede ofrecer intuición de qué forma aprende la red. Por lo que podemos observar, para el problema en cuestión las redes más potentes tienden a sobre-entrenarse.

Una visión de detalle del ajuste de estas arquitecturas al conjunto de test se puede ver en la figura 74, donde se muestra el perfil de aceleración del conjunto de test y los perfiles de aceleración de las redes entrenadas.

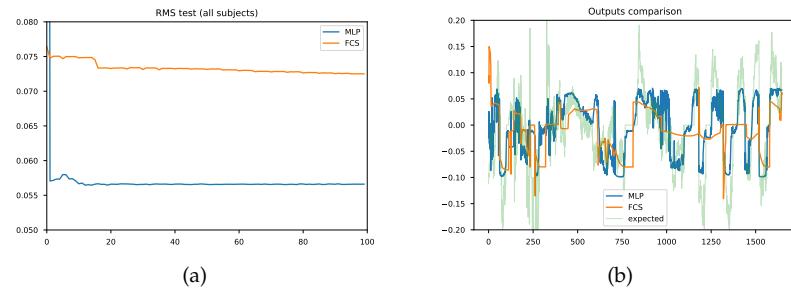
A la vista de los resultados, y dado que las arquitecturas se ajustan razonablemente bien, es razonable elegir el modelo  $MLP_2$  debido a que es el que aparentemente mejor generaliza los comportamientos del conjunto de conductores.



*Comparación entre modelos*

Las mejores arquitecturas de ambos modelos han sido la  $MLP_2$  para los **Perceptrones Multicapa** y  $FCS_x$  para los **Sistemas de Control Difuso**. Los errores y el perfil de aceleración para ambos modelos se muestran en la figura 75.

Figura 75: Comparación de la mejor arquitectura **Perceptrón Multicapa** frente a la mejor arquitectura **Sistema de Control Difuso**: (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 **Sistema de Control Difuso** 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 controlador difuso.

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

## *Modelos específicos de conductores*



## *Implementación en simulador*

### *Implementación en entorno de simulación*

**TODO!**Hablar de la implementación del lidar, de cómo se realiza la transformación someramente, un par de gráficas de la nube de puntos del entorno y del mapa de profundidad. Los vídeos para la presentación.

### *Validación del modelo*

**TODO!**Para validar el modelo habrá que realizar una batería contra los conjuntos de test. Creo que la tasa de error en el problema de ajuste de la aceleración vale con un RMS, pero en el caso del cambio de carril habrá que indicar una tasa de acierto. Ésta debería tener en cuenta si cambiamos de carril dentro del tiempo estipulado para ello, en lugar de acertar directamente, pero si no pues ya lo apañaremos.



## Resultados y conclusiones



## *Resultados*

Mínimo de resultados, los datasets, porque además en materia de cambio de carril hay poco.

Frase para poner en los resultados a raíz de los que estoy obteniendo. Quizá si hablo un poco más de este caso relleno más y parece más consistente: La configuración de la salida en el caso de los cambios de carril es cuanto menos curiosa. Cuando la salida se realiza de forma que el no-cambio de carril se encuentra en un extremo (e.g. 0, 1, 2 siendo 0 el no cambio y 2 el cambio), el entrenamiento devuelve modelos peores que en el caso en el que el no-cambio de carril se encuentra en el medio (e.g. -1, 0, 1). Me aventuro a predecir que en los casos en los que la salida al problema a modelar son los estados con transiciones entre ellos dentro del problema, mantener la relación de cercanía entre transiciones de estado mejora los resultados del entrenamiento y del modelo aprendido (en nuestro caso, -1, 0, 1 mantiene una distancia de 1 transición, mientras que en el caso 0, 1, 2 hay una distancia que no se corresponde con la distancia en transiciones). Esta predicción es una generalización de la experiencia obtenida en estos experimentos y requiere de más estudio.

Hablar también de las funciones de activación en el mlp del car following. Las relu se comportan como el culo, seguramente por el tema de que sólo se activan cuando los valores son positivos.

Hablar de que da la impresión que el problema del car following sufre de planicies o de mínimos locales, porque a la hora de entrenar, en bastantes casos con una arquitectura los valores se estancaban.



# *Conclusiones*

¿Pierde rendimiento el sistema cuando se aplican los modelos a escenarios significativamente diferentes de los escenarios de test? Si sí, un trabajo futuro y algo para escribir en conclusiones sería hablar de este defecto y de cómo subsanarlo.

## *Aportaciones*

### *Futuras líneas de investigación*

¿A lo mejor se podría tirar por el campo de las V2X desde esta tesis?

¿Entrar en el tema de la mesosimulación?

Intersection model (lo he visto nombrar por primera vez en [http://elib.dlr.de/89233/1/SUMO\\_Lane\\_change\\_model\\_T.pdf](http://elib.dlr.de/89233/1/SUMO_Lane_change_model_T.pdf))  
Habrán creado también un concepto así rotondas?

No sé, pero me parece lógico tratar de realizar una disociación entre vehículo y conductor en lugar de contemplar el binomio vehículo/conductor como uno sólo. Es más, creo que resultaría interesante evaluar comportamientos de conductores sobre diferentes tipologías de vehículos. La librería de todas formas debería soportar esta disociación (y se debería indicar).

Hemos dejado fuera comportamientos interesantes de estudiar: cruces, rotundas (Driving behavior at a roundabout: An hierarchical Bayesian regression analysis), ...

Parece que se presta poca atención sobre el tema de vehículos pesados (creo que he encontrado en total un par de referencias), y su forma de funcionar es diferente. Puede ser interesante de cara a perfeccionar los simuladores con esta tipología de vehículos y de cara a ser de utilidad a empresas de transporte.

Los cambios de carril no son inmediatos, toman en torno a los 3 segundos, y no he visto que se tenga en cuenta. Todo se centra en la decisión del cambio de carril, pero a la hora de ejecutar van a saco. Quizá habría que prestarle un poco más de atención a este comportamiento.

Para evaluar la efectividad de determinadas técnicas se mira el comportamiento en nivel macro tanto del modelo real como del modelo simulado. De hecho es lo que haré en esta tesis. Sin embargo, no parece que sea el modo más correcto de evaluar la precisión de los modelos. Quizá habría que rebuscar más por este lado para ver cómo se comportan en nivel micro modelos reales y modelos simulados.

At first glance, the obtained outputs of the final stage of the experiments verify that: 1. It is possible to adjust the problem of lane-change execution to a specific driving profile on a different circuit after enough training with a high certainty. 5. As could be expected, a model trained for a profile performs worse when validated against data from a different one, even while driving in the same route with the same driving conditions (due to its different driving styles). So, the developed models intrinsically distinguish driving profiles. 6. The comparison realized between CNN and MLP show that the former ones can generalize much better than the latter in this problem, although they require more training. In fact, by the results obtained in Table , it can be observed that, whereas the training accuracy has slightly decreased in all the architectures, in the case of CNN2– the network has experienced a remarkable increase in the validation accuracy. This fact reinforces the belief that the problem required a network of this size with a moderately-to-high regularization factor to ensure the reduction of overspecialization. Beyond these results, another result has been obtained that is worth highlighting: the difference on how CNNs and MLPs differentiate between subjects. A new training process was run after the one in the paper to see if the resulting values coincide, and the results were similar. It seems that our models based on CNNs differentiate better between profiles than the ones based on MLPs. The reasons for this behaviour may have to do with the way in which CNNs recognize patterns as opposed with MLPs, but further research is required. Wrapping up, the use of CNNs for tasks where the input topology is space-dependent (meaning that the position of the required patterns in the space are relevant) is extremely effective to model temporal events when their windows are narrow and surpasses the precision of the MLPs without loss of time. The last result obtained is the adequacy of the proposed shaking technique to the problem. It has proven to be extremely useful in artificially augmenting the size of the dataset and therefore helps lowering the overfitting problem. We think that, for situations like this one, where the point clouds have errors or where a degree of error or inaccuracy is allowed, this technique can help improving dramatically the size of the datasets to increase the quality of the results.

# *Ajuste de controlador difuso basado en descenso del gradiente*

Como vimos en la sección I del capítulo [Inteligencia Computacional](#), un [Sistema de Control Difuso](#) se compone de cuatro bloques diferenciados: la fuzzificación, el bloque de reglas, la inferencia y la defuzzificación. De éstos bloques, el proceso manual de ajuste se resume siempre<sup>74</sup> en dos pasos:

1. Elección de las particiones difusas de las variables lingüísticas.
2. Definición de las reglas difusas.

Las soluciones de ajuste que existen en la literatura suelen funcionar ajustando automáticamente uno de los dos puntos manteniendo constante el otro (e.g. ajustar las particiones manteniendo fijas las reglas). Nosotros representaremos el controlador como un grafo computacional de manera que sus gradientes sean sencillos de ajustar<sup>75</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 difusa se limitarán las funciones de pertenencia a: una línea descendente para caracterizar al primer conjunto difuso, 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 difusos de tipo singletón. La función de defuzzificación será la la operación CoGS<sup>76</sup>.
- El tamaño de las particiones difusas no variará dinámicamente a lo largo del entrenamiento, siendo este un meta-parámetro de configuración del controlador.

<sup>74</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 difusas y las reglas.

<sup>75</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>76</sup> La operación CoGS (*Center of Gravity for Singletons*) se define como:

$$CoGS = \sum_{i=1}^n w_i \cdot o_i \quad (20)$$

Es decir, la suma ponderada de los valores de salida.

$$\left( \begin{array}{c|c|c} x_{V_1}^1 & x_{V_2}^1 & \dots \\ x_{V_1}^2 & x_{V_2}^2 & \dots \\ x_{V_1}^3 & x_{V_2}^3 & \dots \\ x_{V_1}^4 & x_{V_2}^4 & \dots \end{array} \right) \rightarrow \left( \begin{array}{c|c|c|c|c} \mu_{V_1}^1(x_{V_1}^1) & \mu_{V_1}^2(x_{V_1}^1) & \mu_{V_2}^1(x_{V_2}^1) & \mu_{V_2}^2(x_{V_2}^1) & \dots \\ \mu_{V_1}^1(x_{V_1}^2) & \mu_{V_1}^2(x_{V_1}^2) & \mu_{V_2}^1(x_{V_2}^2) & \mu_{V_2}^2(x_{V_2}^2) & \dots \\ \mu_{V_1}^1(x_{V_1}^3) & \mu_{V_1}^2(x_{V_1}^3) & \mu_{V_2}^1(x_{V_2}^3) & \mu_{V_2}^2(x_{V_2}^3) & \dots \\ \mu_{V_1}^1(x_{V_1}^4) & \mu_{V_1}^2(x_{V_1}^4) & \mu_{V_2}^1(x_{V_2}^4) & \mu_{V_2}^2(x_{V_2}^4) & \dots \end{array} \right)$$

Figura 76: El bloque de fuzzificación transformará los valores de sus respectivos dominios al dominio de pertenencia a cada conjunto difuso. La matriz generada tendrá tantas columnas como conjuntos difusos posea cada variable.

- El controlador tendrá una única variable de salida.

### *Sistema de Control Difuso 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 Difuso** 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 Difuso** como un grafos computacionales; después construiremos el controlador conectando entre sí estos grafos.

### *Bloque de fuzzificación*

El grafo computacional asociado a este bloque tomará una matrix 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 difuso de la variable que les corresponda (Figura 76).

Para ello, se le aplicará a cada columna de la matriz de entrada tantas operaciones (funciones de pertenencia) como conjuntos difusos posea. Concretamente se aplicarán operaciones de línea descendente, trapezoidal y ascendente. En la Figura 77 se ilustran las tres funciones

de pertenencia junto con las variables que determinan su funcionamiento

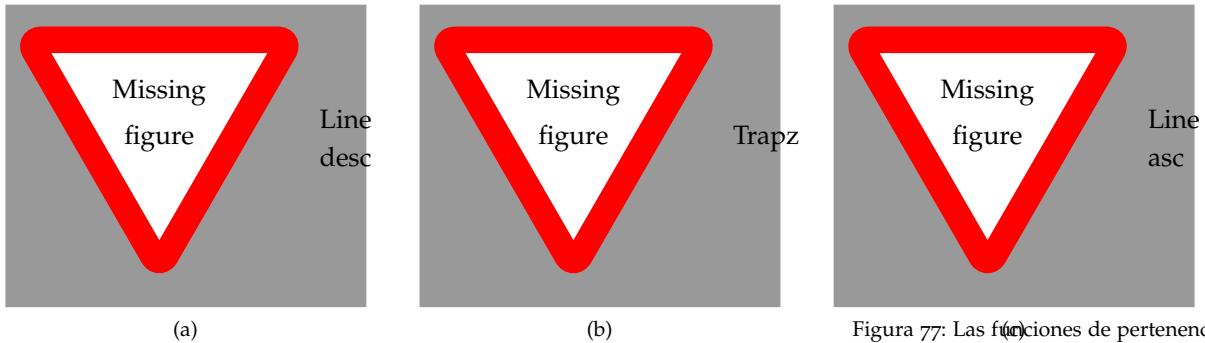


Figura 77: Las funciones de pertenencia (a) línea descendente, (b) trapezoidal y (c) línea ascendente. Éstas se definen como un punto de inicio seguido de tantos desplazamientos como sea necesario.

Las líneas **ascendente** y **descendente** tienen una forma parecida. El grafo asociado a la fórmula de la línea descendente se muestra en la figura 78, que es la que se correspondería con el último conjunto de una partición difusa. Como se puede ver, las variables ajustables son  $a$  y  $\delta a$ . Estas definen el intervalo  $(a, a + \delta b) \in \mathbb{R}$  donde los valores  $f(X)$  ascienden de 0 a 1 (o desciden de 1 a 0 en el caso de la recta ascendente).

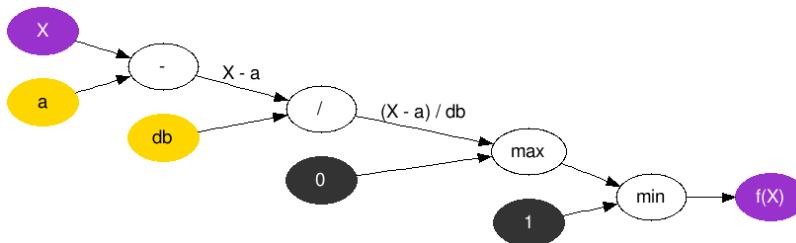


Figura 78: Ilustración del grafo computacional para la función de pertenencia línea ascendente. La fórmula que describe es  $\mu(x) = \min(\max(\frac{x-a}{\delta b}, 0), 1)$ , quedando acotada  $\mu(x)$  en el intervalo  $[0, 1] \in \mathbb{R}$ . El grafo computacional para la línea descendente es similar y se corresponde con la fórmula  $\mu(x) = \min(\max(\frac{a-x}{\delta b+1}, 0), 1)$ .

El **trapecio** 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 trapezio 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 79

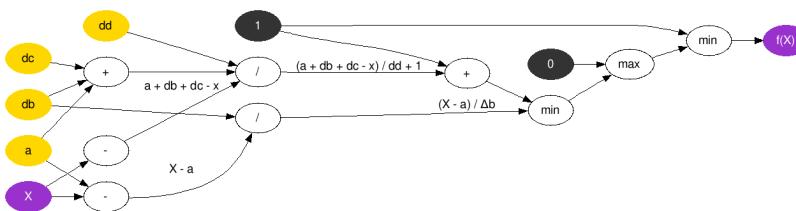


Figura 79: Ilustración del grafo computacional para la función de pertenencia trapezoidal. La fórmula que describe es  $\mu(x) = \min(\max(\min(\mu_{L_{asc}}, \mu_{L_{desc}}), 0), 1)$ , esto es, una línea ascendente y otra descendente, estando ambas acotadas en el intervalo  $[0, 1] \in \mathbb{R}$ .

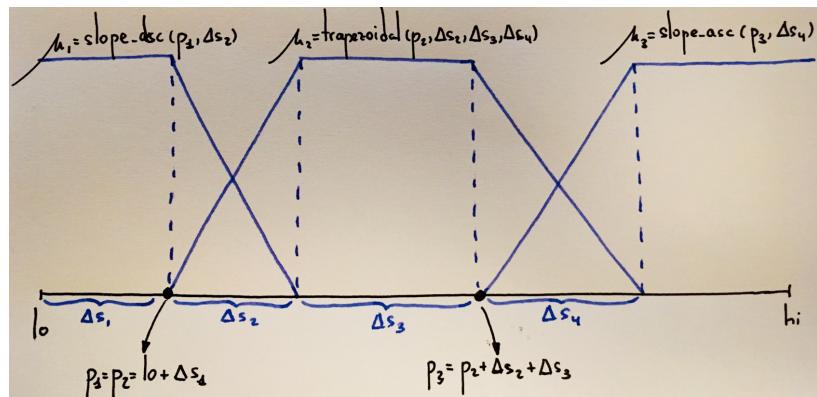
Sabiendo los grafos computacionales de cada una de las funciones de pertenencia, ya podemos definir el grafo asociado a la partición difusa de una variable lingüística. Suponiendo que la variable  $V_i$

está dividida en  $N_{V_i}$  conjuntos difusos, la partición de ésta estará compuesta de:

- Un primer conjunto definido como una pendiente descendente.
- $N_{V_i} - 2$  conjuntos definidos por una función de pertenencia de tipo trapezoidal.
- 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. En la figura 8o se puede observar de qué manera están relacionadas las variables de desplazamiento y los parámetros de las funciones de pertenencia.

Figura 8o: Ilustración de una partición difusa para una variable lingüística con  $n = 3$  conjuntos difusos en el dominio  $[l, h]$ . Al ser  $n = 3$ , la partición queda definida como un punto de origen  $l$  y  $2(n - 1)$  variables correspondientes a desplazamientos consecutivos, los cuales son los que definen los parámetros de las funciones de pertenencia de cada uno de los conjuntos difusos. A partir de este punto, los grafos computacionales son más complejos y su ilustración no aporta nada, por lo que los conceptos se ilustrarán de forma diferente.



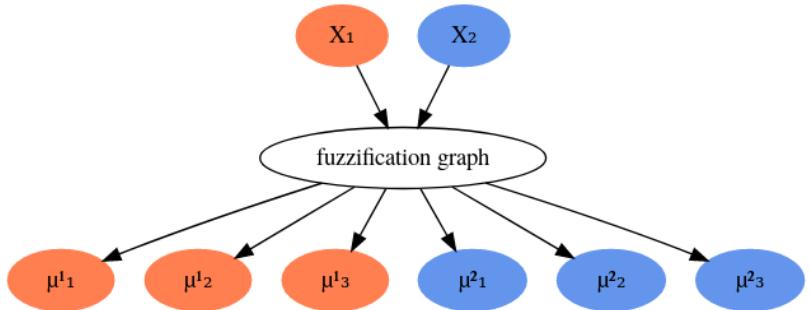
Al estar definidas de esta manera, logramos (i) que la suma de todas las pertenencias en cada punto del eje X es 1 y (ii) que cada pequeña variación del gradiente de una de las variables tiene el potencial de provocar una variación en el resto de variables.

Por último, un controlador difuso 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 difusos que contiene la variable lingüística  $l_i$  (Figura ??)

### 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 difusos de salida (un valor difuso por cada conjunto difuso de

Figura 81: Siendo el número de conjuntos difusos 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 difusos de  $X_1$  y  $X_2$  respectivamente



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 difusas que llegan al bloque). La dimensión adicional se corresponde a la variable lingüística de salida.

Dicho de otro modo, cada posible conjunto difuso de salida (es decir, para 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 82 se muestra un ejemplo con las variables obtenidas en el ejemplo de la Figura 81 tras aplicarles la  $t$ -norma.

Al estar definida la  $t$ -conorma y la acumulación con el mismo operador (i.e. el máximo), 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 difusos como conjuntos difusos tiene la salida.

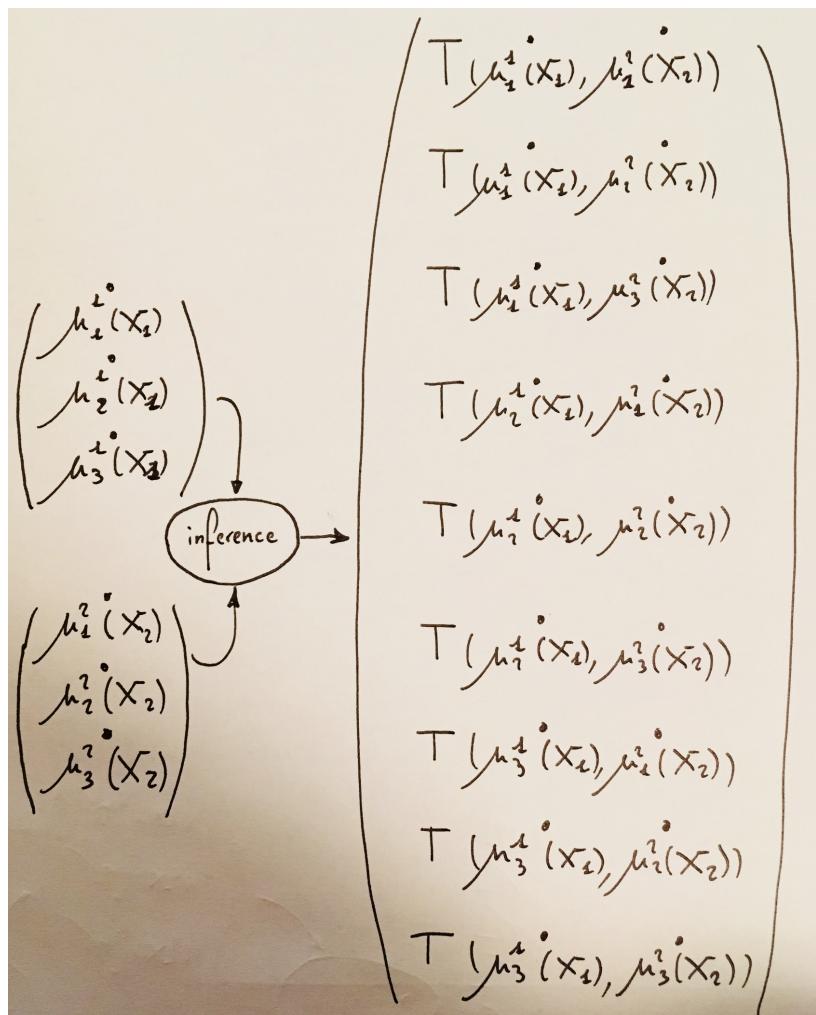


Figura 82: El producto cartesiano de las variables difusas de entrada genera todas las posibles combinaciones entre los conjuntos difusos de las variables lingüísticas. Aplicándose la  $t$ -norma a cada una de estas combinaciones tenemos todas las posibles reglas que se pueden definir en este [Sistema de Control Difuso](#).

### 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 difusos de salida y devuelve un vector de rango  $(m)$  con los valores en el dominio de la variable lingüística de salida.

### Ejemplo de ajuste de un controlador difuso

Para probar su funcionamiento, se realizará el ajuste de un controlador difuso a partir de un conjunto de entrenamiento con unos valores determinados. Estos valores se habrán obtenido a su vez de un [Sistema de Control Difuso](#) de ejemplo para comprobar lo correcto de la implementación.

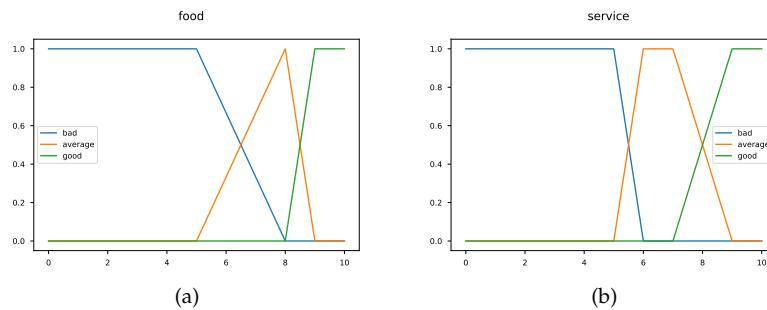


Figura 83: Particiones difusas del controlador de ejemplo. Estas particiones junto con las reglas definirán la superficie de la función que modela nuestro controlador de ejemplo.

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 83. Las reglas que determinan su comportamiento son las siguientes:

$$\begin{aligned}
 &\text{service IS good} \rightarrow \text{tip IS high} \\
 &\text{food IS good} \rightarrow \text{tip IS high} \\
 &\text{service IS good} \wedge \text{food IS average} \rightarrow \text{tip IS low} \\
 &\text{service IS average} \wedge \text{food IS good} \rightarrow \text{tip IS high} \\
 &\text{service IS bad} \rightarrow \text{tip IS low} \\
 &\text{food IS bad} \rightarrow \text{tip IS low}
 \end{aligned} \tag{21}$$

El controlador es una función con dos variables que describen la superficie mostrada en la Figura 84.

Recogeremos una muestra aleatoria uniforme de 2500 de entre los 62500 que conforman la representación de la superficie del controlador de ejemplo. Posteriormente se crea un controlador difuso y se itera el algoritmo de descenso del gradiente ADAM [[Kingma and Ba, 2014](#)].

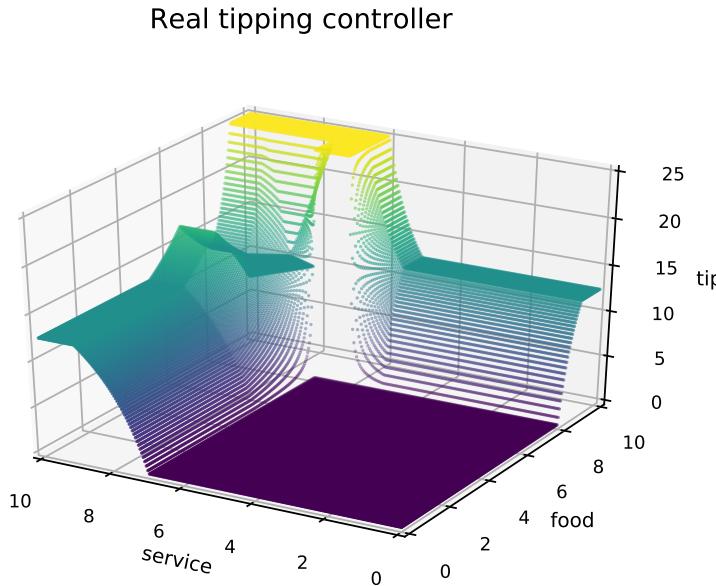
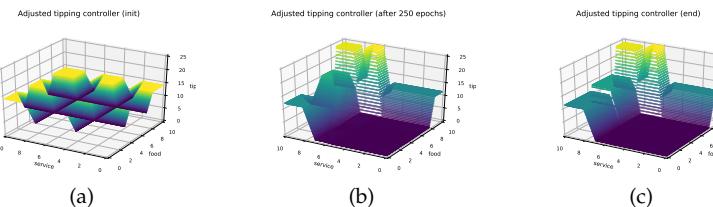


Figura 84: Las reglas y las particiones definen esta función. De ella obtendremos un conjunto de puntos aleatorio para comprobar si nuestra representación de **Sistema de Control Difuso** como grafo computacional es capaz de ajustarse a ésta con la técnica del descenso del gradiente.

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. La evolución del error durante este entrenamiento se puede ver en la Figura ??.

Durante el entrenamiento, las superficies evolucionan hacia una representación aproximada de lo que es el controlador original. En la figura 85 podemos observar cómo evoluciona la superficie que representa el controlador ajustado.

Figura 85: Particiones difusas del controlador 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 controladores difusos 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 ANN <sup>77</sup>.

<sup>77</sup> Aunque no se han extraído en este ejemplo, el proceso de extracción de particiones difusas y reglas se muestra en el capítulo **Comportamiento de cambio de carril**.

# *Visión general de ROS*

ROS es un **framework** para el desarrollo de aplicaciones relacionadas con automática y robótica. Abstira una serie de servicios normalmente provistos por el **Sistema Operativo (OS, Operative System)** 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>78</sup>. Entre sus múltiples colaboradores destacan la Universidad de Stanford<sup>79</sup> como desarrolladores originales, el instituto de investigación Willow Garage como desarrolladores principales desde el año 2008 y la **Open Source Robotics Foundation (OSRF)**<sup>80</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 **Application Programming Interfaces (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.
- La paquetería **TODO!** buscar otro nombre mejor porque esto es un chupo, compuesta 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 cono-

<sup>78</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>79</sup> El nombre original del proyecto fue *switchyard*, pero desde que tomó el relevo Willow Garage su nombre pasó a ser ROS.

<sup>80</sup> La OSRF fue una iniciativa lanzada desde el instituto de investigación Willow Garage en el año 2012.

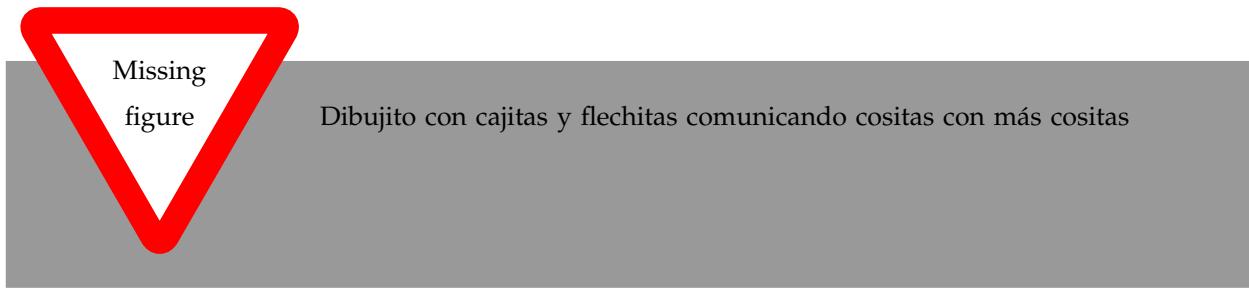


Figura 86: Descripción de los componentes de un sistema desarrollado para [ROS](#).

cer para poder comprender de qué manera un sistema está construido. Estos son:

- **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 [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.

# *Sistemas desarrollados*

Este capítulo describe todos los sistemas y el software desarrollados e implementados para realizar la tesis. Éstos son tanto los encargados de la captura de datos de los conductores, los que trabajan directamente con el simulador para integrar los controladores generados y los desarrollos para la generación de Software.

## *Outrun*

Biblioteca para la incorporación de modelos de conductor personalizados en SUMO. Hace uso de [TraCI](#).

## *Modelos de comportamiento*

*Entrenamiento de controladores difusos mediante Computación Evolutiva (EC, Evolutionary Computation)*

## *Otro software desarrollado*

## *ScanBUS*

ScanBUS es un software para la identificación de paquetes enviados por dispositivos a través del Bus CAN del vehículo.

## *Sistema para la captura de datos multidispositivo*

Para la obtención de los datos de conducción se ha desarrollado un sistema que permite la conexión a múltiples dispositivos desde diferentes interfaces. Las razones para su desarrollo son las siguientes

- Sincronización automática de datos de dispositivo en intervalos configurables de tiempo. El sistema permite la configuración de la frecuencia de captura sincronizando los datos recibidos a esa frecuencia.

- Diseño extensible a otros dispositivos. Es software está diseñado para facilitar en la medida de los posible la introducción de nuevos dispositivos usando, para ello, las interfaces apropiadas.
- Hardware compacto. El sistema está integrado en un ordenador de tipo Raspberry PI, aunque es factible su integración en otros sistemas siempre y cuando funcionen con un sistema GNU/Linux e incluyan el hardware necesario para las capturas.

### *Módulos de ROS*

Hablar que el paquete anterior quedó discontinuado y empezamos a desarrollar paquetes ara ROS, en el cual incluimos un desarrollo de CAN y un launcher.

### *Nagel-Schreckenberg*

```
$ python3 main.py \
    --highway_len 250 \
    --iterations 100 \
    --num_cars 50 \
    --max_speed 5 \
    --p 250 \
    --output output.pdf
```





## Bibliografía

- [par, 2010] (2010). Directive 2010/40/eu of the 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 text with eea relevance. *Official Journal of the European Union*, 50:207.
- [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. *The 5th International Conference on Autonomous Agents.*, (June):286–291.
- [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.
- [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., 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.
- [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. and Swamy, M. (2006). Neural networks in a softcomputing framework.
- [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".
- [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.
- [Gu et al., 2015] Gu, M., Billot, R., Faouzi, N.-e. E., Lyon, D., and Hassas, S. (2015). Multi-Agent Dynamic Coupling for Cooperative Vehicles Modeling. 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, 2003] Hatipkarasulu, Y. (2003). A Variable Response Time Lag Module for Car Following Models Using Fuzzy Set Theory. (225).
- [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.
- [Imprialou et al., 2016] Imprialou, 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.
- [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*, number 16-4359.
- [Jin, 2006] Jin, W.-L. (2006). A kinematic wave theory of lane-changing vehicular traffic.
- [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. In *IEEE on Systems, Man, and Cybernetics*, volume 42, pages 1440–1449.
- [Kikuchi and Chakroborty, 1992] Kikuchi, S. and Chakroborty, P. (1992). Car-following model based on fuzzy inference system. 1(1365).
- [Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [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, 1993] Margolus, N. (1993). CAM-8: a computer architecture based on cellular automata \*.
- [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.
- [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.
- [Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). Perceptrons.
- [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.
- [Ng, 2004] Ng, A. Y. (2004). Feature selection, l 1 vs. l 2 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.
- [Osogami et al., 2012] Osogami, T., Imamichi, T., Mizuta, H., Morimura, T., Raymond, R., Suzumura, T., Takahashi, R., and Idé, T. (2012). IBM Mega Traffic Simulator.
- [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 and Cajal, 1904] Ramón, S. and 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.
- [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.
- [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.
- [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*.
- [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.
- [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*.
- [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 ANDREWTRASK et al., ] Trask ANDREWTRASK, A., Gilmore DAVIDGILMORE, D., and Russell MATTHEWRUSSELL, M. Modeling Order in Neural Word Embeddings at Scale.
- [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. *Proceedings of the 11th communications and networking simulation symposium on - CNS '08*, (August 2016):155.
- [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. Technical report.
- [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.
- [y Cajal, 1888] y Cajal, S. R. (1888). *Estructura de los centros nerviosos de las aves*.
- [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. pages 782–791. Springer Berlin Heidelberg.



## *Índice alfabético*

- approaching, 85
- back propagation, 69
- back-propagation, 93
- car-following, 73, 80, 85–89,  
92–94
- courtesy yielding function, 89
- critical gap, 87
- critical-gap, 89
- emergency, 85
- fast-forward, 92
- free flow, 85
- free-flow, 88, 91
- gap acceptance, 87
- gap acceptance, 85, 87
- gap-acceptance, 89
- lane selection, 85
- lane-changing, 88
- lane-selection, 87
- merging, 85, 87
- neuro-fuzzy, 93, 94
- recurrente, 92
- umbral perceptual, 87



## 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{blazaid20167,  
    author = {Alberto Díaz Álvarez},  
    abstract = {XXX},  
    pages = {XXX},  
    title = {Modelado de comportamiento de conductores con técnicas de Inteligencia Computacional},  
    url = {XXX},  
    year = {5 de abril 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).