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

Madrid 16 de febrero de 2018

Índice general

Índice de figuras

Índice de cuadros

Inteligencia Computacional

Todo elemento dentro de un entorno se ve influenciado por multitud de variables que determinan en mayor o menor grado su comportamiento. Sin embargo, 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.

La definición que esta tesis da de la Inteligencia Computacional (IC) es la de la rama de la Inteligencia Artificial (IA) que engloba a todas aquellas técnicas que tratan de *aprender* [1] soluciones a problemas a través de la observación y el análisis de información, ya sea presente en conjuntos de datos, en el entorno o ambos.

[1] Veremos más detalles sobre el concepto de aprendizaje en este contexto y sus implicaciones en el área de la Inteligencia Artificial (IA) más adelante

Sin embargo, no existe una definición globalmente aceptada ya que, dependiendo del autor es considerada desde un sinónimo de la IA hasta un campo completamente diferenciado.

Por ello, la primera sección del capítulo ofrecerá una visión histórica de la aparición del concepto para justificar el por qué de la definición. El resto del capítulo introducirá las nociones de agente y aprendizaje y ofrecerá una explicación en detalle de las técnicas sobre las que se apoya el trabajo teórico de esta tesis: Las Red Neuronal Artificial y la .

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 o los autómatas mecánicos de los filósofos modernos (siglos XVII al XIX) donde los cuerpos vivos son como un reloj son sólo dos ejemplos de este hecho.

Podemos aventurarnos a decir que a principios del siglo XX se comienza a gestar el área de la IA con los trabajos relacionados con los principios del **conexionismo** [2].

[2] 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 conexiónes (e.g. [y Cajal, 1888] y [Ramón and Cajal, 1904])

Hacia mediados del siglo XX, estas ideas del conexionismo saltaron al campo de la computación cuando Warren S. McCulloch y Walter Pitts publicaron su trabajo *"A logical calculus of the ideas immanent in nervous activity"* [McCulloch and Pitts, 1943] [3] donde se describe el

[3] 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

primer modelo artificial de una neurona.

El trabajo suscitó tanta expectación que se comenzó a especular sobre la posibilidad de emular 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 [4] 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 [5].

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, aunque quizá 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 conociminto del entorno en el que los modelos se movían.

[4] 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.

[5] 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 1), donde se demuestra la invalidez argumentando que la máquina ha aprendido a realizar acciones sin *entender* lo que hace y por qué lo hace. Sin embargo, ¿qué garantías tenemos de que el humano sí es capaz? Si los ordenadores operan sobre símbolos sin comprender el verdadero contenido de éstos, ¿hasta qué punto los humanos lo hacen de forma diferente?.

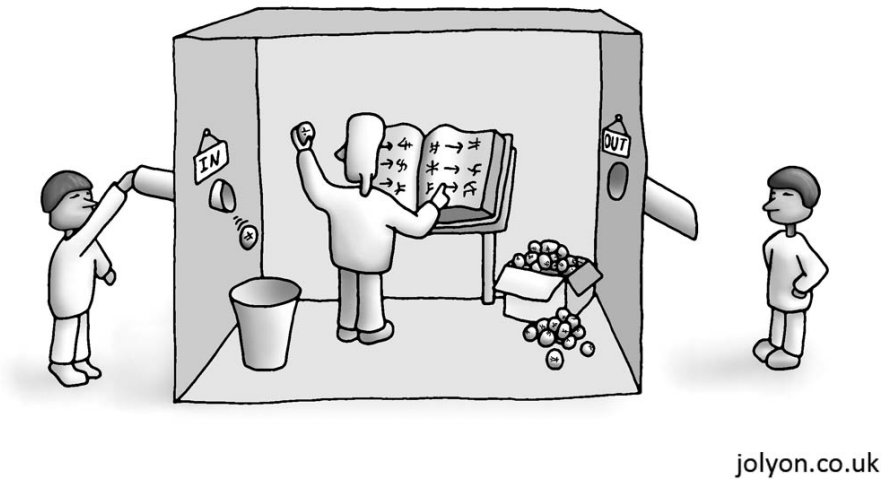


Figura 1: La *Habitación China* de John Searle es un experimento mental por el que se trata de demostrar la invalidez del Test de Turing. Partimeos de un Test de Turing donde la máquina ha aprendido a hablar chino. Reemplazamos la máquina por un humano sin idea de chino pero con un manual de correspondencias 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.

Dado que los estudios 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 varapalo para las investigaciones. En el se expusieron las limitaciones de los modelos de Red Neuronal Artificial (RNA) 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* [6].

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 IA ([Russell et al., 2003]). A finales de la década, sin embargo, empezaron a resurgir de nuevo los enfoques conexionistas, debido en gran parte a la aparición de nuevas técnicas de entrenamiento en perceptrones multicapa y por el concepto de activación no lineal en neuronas [Rumelhart et al., 1985, Cybenko, 1989]). En este momento los sistemas expertos empezaron a perder interés frente al nuevo avance del conexionismo [7]. Esta época se suele identificar como el segundo *AI Winter*, ya que tanto la investigación como las inversiones en el área se vieron disminuídas. 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 (AGs) 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 IA 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 IA 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 IA (e.g Machine Learning (ML), Recommender Systems (RS), o Natural Language Programming (NLP)). Es evidente que la IA 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 IA. El resumen se puede observar en la figura 2.

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 IA parece lógico mantenerla como un subconjunto y no como un nuevo campo del conocimiento humano. Sin embargo, algunos autores abogan por que la Inteligencia Computacional (IC) es un campo diferenciado de la IA.

[6] 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, 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.

[7] Esto no debió de sentar bien a los autores prolíficos en Sistemas Expertos. Mientras que el enfoque en estos sistemas es el clásico en la computación, donde los problemas (en este caso el conocimiento experto) son resueltos mediante operaciones sobre un lenguaje de símbolos, el enfoque del conexionismo postula que la mente, el comportamiento inteligente, emerge de modelos a más bajo nivel. Por ello, algunas voces se alzaran contra lo que se consideraba el *enfoque incorrecto* de la IA. Después de todo, los modelos desarrollados en los métodos clásicos son fáciles de interpretar mientas que los del enfoque conexionista no son del todo deducibles, más aún si estos problemas son de naturaleza estocástica.

Podemos definir la Inteligencia Computacional como la "*rama de la IA que aporta soluciones a* ***tareas específicas*** *de forma* ***inteligente*** *a partir del aprendizaje mediante el uso de* ***datos experimentales***". A diferencia de la aproximación clásica de la IA, 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 erlación entre sus multiplas variables, a la falta de información o a la imposibilidad de traducirlos a lenguaje binario.

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

Figura 2: 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 (3rd Ed.)*, [Russell et al., 2003].

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* [8]. 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:

[8] http://cis.ieee.org/

"Neural networks, genetic algorithms, fuzzy systems, evolutionary programming, and artificial life are the building blocks of CI."

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 consituyentes 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 IA hace uso de métodos incluídos en el concepto Soft Computing.

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.

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

El rol del aprendizaje en la Inteligencia Computacional

El cambio más notorio entre los dos puntos de vista de la IA 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 entrenan 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 valore 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 pardigma, el algoritmo ajusta el modelo de acuerdo a políticas de recompensa o penaliación 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/acelerar 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). A continuación se resumen los más importantes:

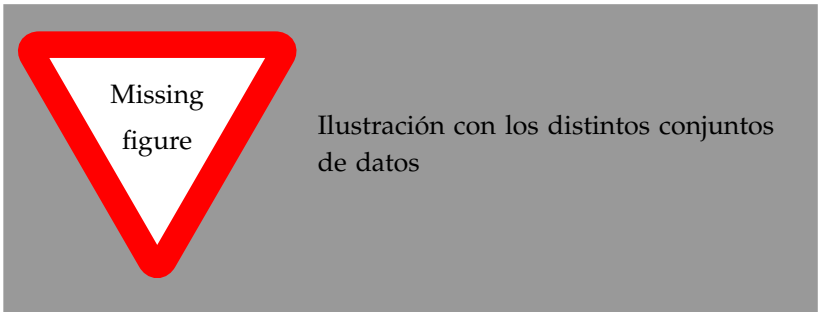


Figura 3: Los diferentes tipos de conjuntos de datos existentes.

Conjunto de datos o dataset Se refiere al conjunto total de ejemplos o datos del que disponemos. Se presupone que el dataset mantiene una distribución de datos similar al entorno real sobre el que pretendemos trabajar, lo cual puede no ser posible por multitud de factores.

Conjunto de entrenamiento El conjunto de entrenamiento (*training set* en inglés) se refiere al conjunto de datos que se usará para entrenar al modelo. Se corresponde un un subconjunto del dataset, y es necesario que mantenga una distribución de datos lo más similar posible al dataset original para poder garantizar que el modelo entrenado represente a todos los sus datos.

Epoch Un 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 porciones se las denomina *batch* o *mini-batch*.

Conjunto de validación Es el subconjunto de datos del dataset principal destinado a probar la efectividad del modelo. Al comienzo del entrenamiento, se reserva este conjunto y a lo largo del entrenamiento se valida contra él para comprobar la efectividad del modelo entrenado.

Para una correcta validación es muy importante que, como el conjunto de entrenamiento, éste mantenga una distribución de ejemplos lo más similar posible al conjunto de datos original para evitar sesgos.

Conjunto de test Durante el proceso de entrenamiento, es una práctica habitual separar el conjunto de entrenamiento en dos partes, una más grande que se usará para el entrenamiento en el actual *epoch* y otra más pequeña que se usará para validar el modelo **durante el proceso de entrenamiento** [9].

Existen autores que prescinden de este conjunto y trabajan directamente con conjuntos de entrenamiento y validación.

[9] Para evitar sesgos, un buen conjunto de test debe ir cambiando constantemente entre epochs. Debido a esto, el modelo acabará aprendiendo también el conjunto de test y por tanto sus resultados no nos sirven para validar la efectividad del modelo entrenado.

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 trabar como de los hiperparámetros [10] del modelo. Estos son la sobreespecialización y la subespecialización [11].

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

[10] En general, hablaremos de *parámetros* cuando nos referimos a valores que son inherentes al modelo (e.g. en una RNA, 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 RNA, el factor de aprendizaje).

[11] En la literatura también se habla de *high variance* o *over-fitting* y de *high bias* o *under-fitting* para referirse a la sobreespecialización y subespecialización respectivamente.

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, 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 antigüos conceptos y que incluso pueden llegar a abrir en elfuturo 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 historicos 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 4). El verdadero impacto de esta década ha sido el del uso de las GPU de las tarjetas gráfiacs como plataforma donde distribuir el cómputo.
- **Algoritmos más eficientes**. Nuevos elementos como las funciones de activación ReLU, la representación de las redes como grafos computacionales para facilitar su distribución o innovaciones en los algoritmos de entrenamiento son algunas de las mejoras en este aspecto que redunda, 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 [12] (normalmente más de dos o tres). Este tipo de capas, históricamente ha sido más difícil de entrenar,

[12] De aquí surge el nombre de **shallow network** o red superficial, en contraposición a **deep network** o red profunda.

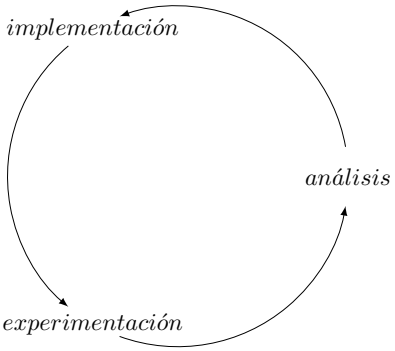


Figura 4: Ciclo de aplicación de soluciones basadas en Inteligencia Computacional. Los sistemas inteligentes se conciben, se implementan y se prueba. En la fase de experimentación se determina cómo de bien o de mal lo está haciendo y cuales son las decisiones a tomar para el nuevo ciclo. Cuandto más 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 ténicas de entrenamiento es tan importante.

debido entre otras cosas a las técnicas de entrenamiento (donde los 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 5 introduce, con una ilustración un tanto informal, las capacidades del deep learning frente a las de los modelos entrenados antes de esta época.

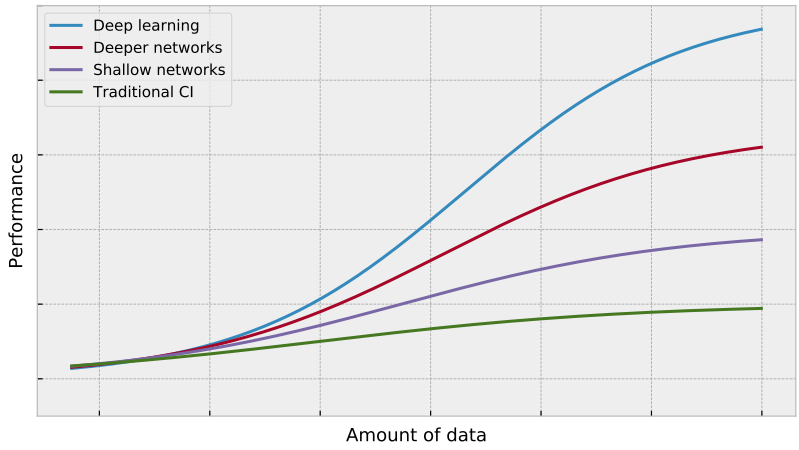


Figura 5: La enorme cantidad de datos junto con la capacidad computacional y la mejora de las ténicas de entrenamiento hacen posible que en la actualidad, con las técnicas asociadas al contexto del deep learning, los modelos entrenados sean más eficientes inteligentes. Imagen adaptada de la charla *How 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 [13].

Redes Neuronales Artificiales

Son herramientas que tratan de replicar las funciones cerebrales de un ser vivo de una manera muy fundamental, esto es, desde sus componentes más básicos, las neuronas. Para ello se basan en estudios de neurobiología y de ciencia cognitiva moderna del cerebro humano [14].

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

[13] Dentro de un contexto de aplicación específica, creemos que aún estamos muy lejos de crear vida inteligente.

[14] 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 6) hace que, tecnológicamente hablando, nos quedemos con la sensación de estar aún a años luz de aproximarnos a la complejidad de un cerebro humano.

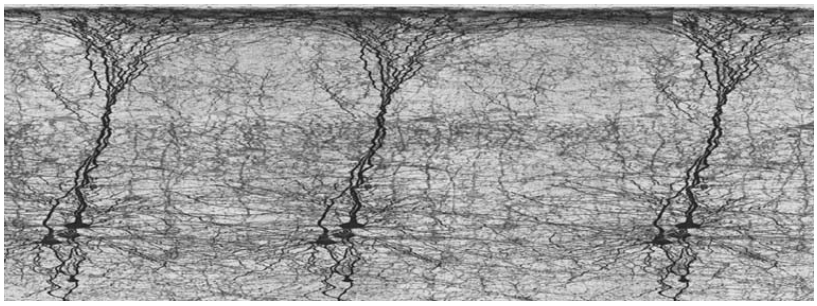


Figura 6: Sección del neocórtex humano, región asociada a las capacidades cognitivas y que supone alrededor de un 76 % del volumen total del cerebro humano. Está distribuído en 6 capas y miles de columnas que las atraviesan, cada una con alrededor de 10,000 neuronas y un diámetro de $0{,}5mm$. Como dato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

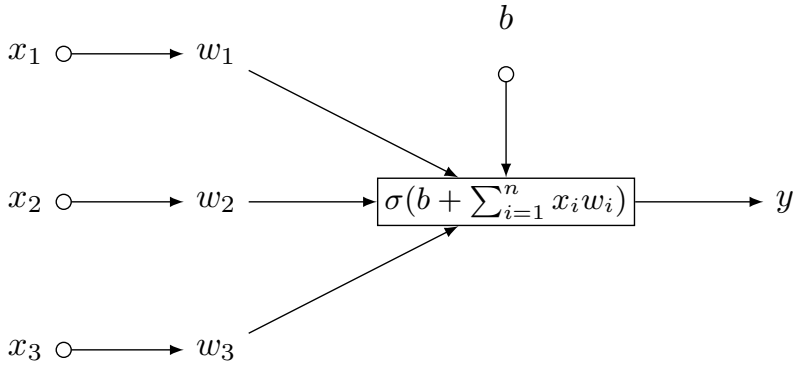


Figura 7: Variación de la representación del modelo de neurona artificial propuesto por McCulloch y Pitts. En éste, cada una de las entradas x_i es incrementada o inhibida aplicando el producto con su peso asociado w_i. La activación vendrá determinada por la aplicación de una función (denominada "de 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.

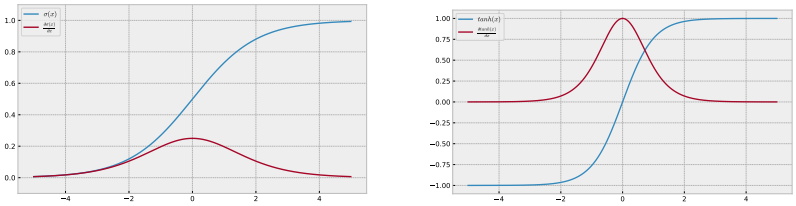
Funciones de activación

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 [15].

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 8 se muestra una representación gráfica de éstas funciones junto con su derivada.

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

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



(a) (b)

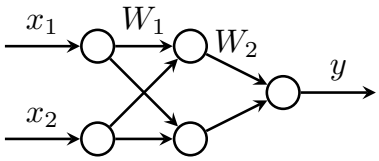
Figura 8: Las funciones de activación sigmoidal y tangente hiperbólicas. Ambas se han usado como funciones de activación no lineales gracias a que sus derivadas son continuas a lo largo de todo el dominio y además son fácilmente computables.

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, latangente 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 tengente 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

[15] 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:

$$\mathbf{y^1} = f(W_1 * \mathbf{x} + \mathbf{b_1})$$
$$y = \mathbf{y^2} = f(W_2 * \mathbf{y^1} + \mathbf{b_2})$$

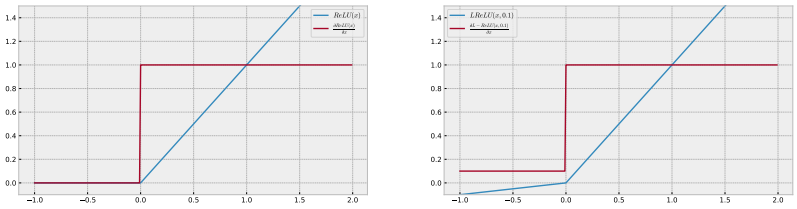
Por tanto:

$$\begin{aligned} y &= f(W_2 * f(W_1 * \mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}) \\ &= (W_2 * W_1) * \mathbf{x} + (W_2 \cdot \mathbf{b_1} + \mathbf{b_2}) \\ &= W' * \mathbf{x} + \mathbf{b'} \end{aligned}$$

Siendo $\mathbf{y}$ el vector columna de salida de la capa l, $\mathbf{b_1}$ el vector de bosbias de la capa l y $\mathbf{x}$ el vector columna de las entradas.

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

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.



(a) (b)

Figura 9: 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 tremendamente simple comparado con el resto de funciones (no deja de ser un máximo). En la figura 6

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

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

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 9, 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 ralentiazr 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 9. Su uso no está muy extendido, pero existen casos en los que su uso está justificado.

Estructura y clasificación de redes neuronales

Las RNA 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 10). 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

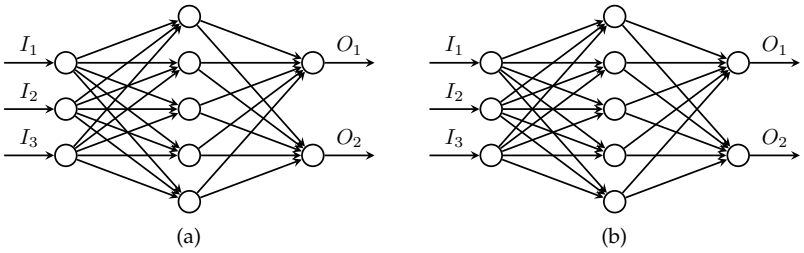


Figura 10: Diferencias entre los grafos que representan una red neuronal de tipo (a) *feed-forward* y una (b) recurrente. Las redes recurrentes presentan ciclos entre sus nodos que permiten la retroalimentación interna entre las neuronas. Suelen ser más útiles a la hora de modelar eventos en el tiempo, aunque su entrenamiento es más complejo.**TODO!**CAMBIAR LA FIGURA DE LA RECURRENTE

una forma más fiel las bases biológicas de las RNA, 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 Red neuronal convolucional (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 ténicas surgidas a partir del *deep learning*

Perceptrones multicapa En un perceptrón multicapa, las neuronas se encuentran agrupads 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 últimacapas de la red se las denomina respectivamente capa de entrada y de salida, mientras que las capas intermedias son denominadas capas ocultas. Hemos visto algunasilustraciones 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 tambien 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:

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

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* [16]) de la red calculando como aumenta o disminuye esta con peqñeas 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 descensodel 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:

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

Nótese que $\circ$ denota al producto de Hadamard (elemento a elemento). También es interesante fijarse en que $W^l \cdot \mathbf{s}^{l-1} + b^l$ 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:

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

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

$$\delta\mathbf{b}^l = \delta\mathbf{e} \tag{7c}$$

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

$$W^l = W^l + \alpha\delta W^l \tag{8a}$$

$$\mathbf{b}^l = \mathbf{b}^l + \alpha\delta\mathbf{b}^l \tag{8b}$$

[16] Existen dos términos asociados al error en RNAs: 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.

Esta es la forma original del algoritmo de back propagation. Al valor α 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, elriesgo 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 As its name suggests, a CNN uses convolutions, i.e. mathematical operations that filter regions of a structure (generally in the form of a matrix or a cube) for pattern identification and/or structure transformations. A CNN has also an arrangement of layers, but they work differently. Firstly, this architecture is separated into two groups of layers or phases that can be called pattern identification phase and prediction phase. The pattern recognition phase works with layers dealing with pattern recognitions and input subsampling while the prediction phase works with an MLP using the output of the pattern recognition phase as input. Secondly, the inputs are arranged as -dimensional matrices as opposed to the MLP that are always presented as vectors; this fact allows us working more easily with properties and patterns arranged in an n-dimensional space like images (2-dimension matrix) or videos (3-dimension matrix). The layer types in that phase are as follows: 1. Convolutional Layer. Given a -dimensional input, a convolution or filter is an n-dimensional pattern (composed by neurons) that travels along the -dimensional space to generate a new -dimensional space. A convolutional layer is a layer composed by m filter and generate a set of m new -dimensional spaces which will be the new input for another layer. 2. Local Response Normalization (LRN) layer [26]. This layer is intended to enhance the difference between the existent values in a space. They are commonly placed after a convolution layer to increase the contrast of the values in the space. In this paper, every convolution layer is followed by an LRN layer. 3. Pooling layer. Pooling is grouping together elements. A pooling layer is intended to subsample a space to ease its

management. Given an -dimensional space, a pooling filter of dimension travels the space in steps of size generating one value per step. When placed after convolutions, it is expected to subsample the space without losing the recognized pattern while reducing computation costs and avoiding overfitting. The most common operation used in pooling is the max operation.

Escribir aquí cómo es el proceso de inferencia

Escribir aquí cómo es el proceso de aprendizaje

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. Las 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 ejempo, 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.

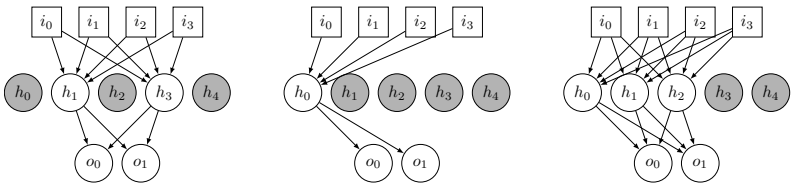


Figura 11: Tres epochs del proceso de entrenamiento de un perceptrón multicapa donde las neuronas de la capa oculta tienen una tasa de dropout del 0.5 (un 50 % de probabilidad de ser desactivada durante ese epoch). Las neuronas grises son las elegidas en dicho epoch para ser eliminadas, y por tanto no afectarán ni al resultado de salida ni al reajuste de los pesos de sus conexiones entrantes.

- 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 diferntes ténicas para la regularización de parámetros, como las l_1 y l_2 (explicadas en detalle en [Ng, 2004]). En esta tésis 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 11 describe un ejemlpo de funcionamiento en tres epochs de un perceptrón multicapa con una tasa de dropout del 0,5.

Lógica Difusa

Ampliando la teoría de conjuntos

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.

Aquí dentro hay que hablar también de operaciones entre conjuntos. Como aquí hablaremos de sismteas de orden tipo 1, en los bordes igual estaría genial hablar de qué son los de tipo 2, 3, ...

Razonamiento

Sistemas de inferencia difusa

Hablar de los tipos de sistemas (Tipos Mandamni y Sugeno diferentes).

La lógica matemática (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 ambigiüedad 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.

Explicar lógica difusa y control difuso. Indicar los controladores difusos de segundo, tercer y sucesivos niveles.

Teoría de conjuntos difusos

A diferencia de los conjuntos tradicionales, los conjuntos difusos expresan el grado de pertenencia de un elemento a la categoría representada por el conjunto. La definición podría escribirse de la siguiente manera:

TODO!Creo que habría que definir antes qué es un dominio

Definición 1: *Sea 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 $\mu_F(x) \in [0,1] \forall x \in X$.*

La función de la definición 1 se denomina **función de pertenencia**, y caracteriza unívocamente a un conjunto difuso del dominio de X.

TODO!Quizá aquí habría que decir qué es una partición de nu dominio

Operaciones entre conjuntos

La unión, intersección y el complemento son operaciones básicas en la teoría de conjuntos. **TODO!** hablar aquí de tnorm, tconorm y complemento, pero someramente. No hay que enrollarse demasiado.

Razonamiento

Al igual que en la lógica tradicional, en la Lógica Difusa el razonamiento o inferencia es la manera de extraer conclusiones a partir de premisas en función de un conjunto de reglas.

[17]

Estas reglas se expresan como implicaciones, definidas típicamente en lógica difusa como $A \rightarrow B \equiv A \wedge B$.

[17] **La implicación** en lógica se representa como $A \rightarrow B$, donde A es cualquier operación de premisas y B la conclusión que arrojan.

En lógica tradicional, el valor de verdad de una implicación es equivalente al de la expresión $\not A \vee B$. Sin embargo, en lógicas multivaluadas (y por tanto en lógica difusa) esta equivalencia da lugar a razonamientos que se pueden considerar contraintuitivos.

En el caso concreto de la lógica difusa se han propuesto muchas cantidad de equivalencias. Sólo en los trabajos [Kiszka et al., 1985] se analizan 72 alternativas al operador $\not A \vee B$.

El operador más usado no obstante es el definido como $A \wedge B$ debido a su rendimiento (en la implicación de Mamdani la T-norma se implementa como el operador mínimo).

Las dos formas de extraer conclusiones a partir de premisas en Lógica Difusa son el *modus ponens* generalizado (del que hablaremos) y el *modus tollens* generalizado, modificaciones sobre los procesos de inferencia *modus ponens* y *modus tollens* [18], dos formas similares de razonamiento (figura 12). Nosotros centraremos nuestro discurso en la primera.

[18] En realidad se llaman *modus ponendo ponens* ("la forma que al afirmar, afirma") y *modus tollendo tollens* ("la forma que al negar, niega").

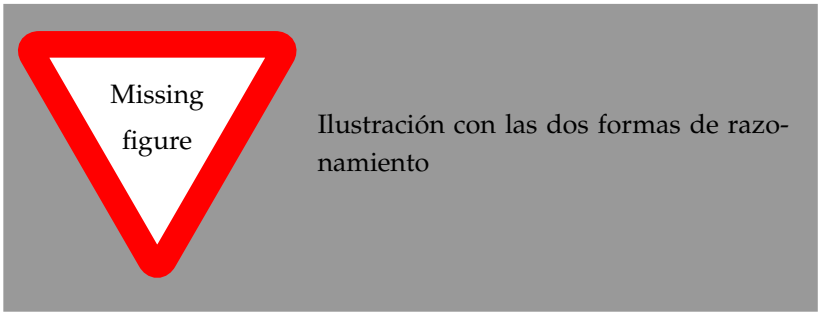


Figura 12: Formas de razonamiento en lógica tradicional: *modus ponendo ponens* y *modus tollendo tollens*.

EL MODUS PONENS GENERALIZADO es una generalización del modus ponens de la lógica tradicional donde, en lugar de expresas las reglas de forma absoluta, se expresan de forma aproximada. En la figura 13 se ilustra las diferencias fundamentales entre ambos modos.

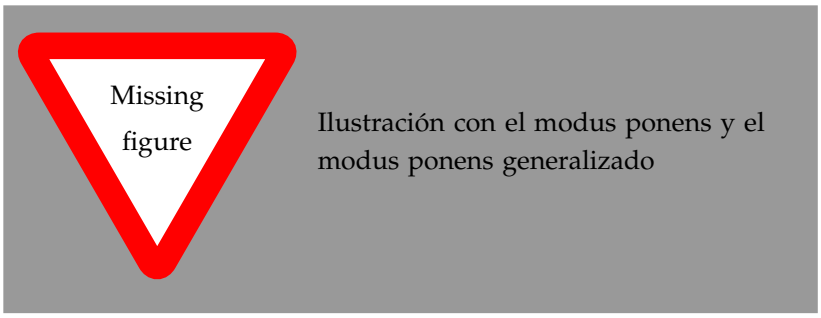


Figura 13: Proceso de razonamiento según el *modus ponens* tradicional frente al *modus ponens* generalizado. En el primero, si la premisa A es cierta, entonces la conclusión B será cierta. En el segundo, dado que la premisa A no es del todo cierta (es A'), entonces la conclusión B será cierta sólo en parte (B').

Para determinar qué grado le asignamos a un consecuente a partir de las premisas parciales y las reglas que dirigen el razonamiento se utiliza un método denominado **regla composicional de inferencia**.

Una regla $A' \rightarrow B'$ se puede representar como una implicación caracterizada por una función $I(\mu_A(x), \mu_B(y))$ ([Ful,]).

TODO!Explicar mejor porque es terrorífico.

...

Tengo que hablar también del softmax, el por qué usarlo. También de la entropía cruzada y de por qué usamos este loss sobre el resto.

He visto una cosa curiosa que es el "negative sampling". Podría hablar de ello y usarlo porque haría el entrenamiento mucho más rápido. Por lo que me ha parecido ver, se usa en lugar de la capa softmax y de lo que va es transformar la capa softmax (que es coñazo

de calcular) en una capa de k+1 clasificadores binarios, donde hay 1 correcto y k incorrectos. Se usa en clasificación.

En [Ma, 2004] hay un capítulo de razonamiento que parece que está guay. Revisarlo un poco a fondo a ver si merece la pena tirar or ahí.

Sistema de Inferencia Difusa

Los Sistema de Inferencia Difusas (o Sistema de Control Difusos) 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.

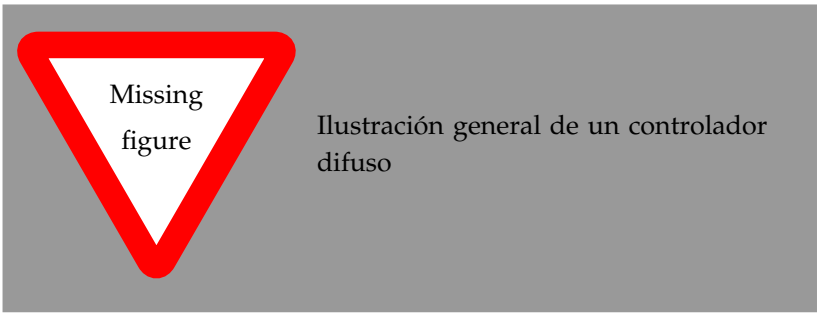


Figura 14: Diagrama del esquema general de un Sistema de Inferencia Difusa.

Habitualmente son descritos como un componente dividido en tres bloques conceptuales:

- **Fuzzificación**. Traducir los valores de entrada en crudo del dominio sobre el que está definida cada variable lingüística a sus respectivos grados de pertenencia a conjuntos difusos a través de sus funciones de pertenencia. **TODO!** Ojo, algunos controladores toman como valores de entrada conjuntos difusos según [Ma, 2004]. Habrá que buscar sobre ello.

- **Inferencia**. Realiza todo el proceso de razonamiento difuso a partir del conjunto de reglas qeu dan significado a este controlador difuso.

- **Defuzzificación**. Traduce los conjunto difuso resultado del proceso de inferencia a valores del los dominios sobre los que están definidos dichos conjuntos difusos. **TODO!** En un sugeno, la salida es una función directamente así que se podría especificar que en un tipo Sugeno,se puede ver como que la salida son sólo singletones, manteniendo la generalización del proceso de funcionamiento de un Sistema de Inferencia Difusa.

Esta división se ilustra en la figura 14.

Hay varios tipos diferentes de Sistema de Inferencia Difusa, aunque tienden a seguir el esquema básico de un controlador difuso típico (figura 14).

Sistemas de tipo Mandamni Son la primera aproximación de Sistema de Inferencia Difusa propuestos

Sistemas de tipo Takagi-Sugeno ...

Hablar someramente de los tres tipos clásicos que se usan, e indicar que al final los más usados son el Mandamni y el Sugeno. Añadir también quizá una tabla comparativa enter los tres o al menos entre los dos principales:

El consecuente de un Sistema de Inferencia Difusa de tipo Mandamni siempre es un conjunto difuso. Por tanto, el proceso de sacar un valor crisp es costoso. Lo bueno, se mantiene significado semántico de las salidas. El consecuente en un Sugeno es un valor, y se puede decir que no necesita proceso de defuzzificación. Si embargo, la respuesta pierde significado semántico si la suma de la fuerza de salida no es 1 (no entiendo qué quiero decir con esto).

Grafos computacionales

TODO!Desarrollar un poquito el tema de los grafos computacionales indicando que si bien no se tratan de una técnica dentro de la inteligencia computacional, sí que se usan para representar operaciones.

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 inteligencia computacional, sino que es simplemente una forma de representar operaciones sobre datos.

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 XXX ilustra un ejemplo de un grafo computacional.

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 XXX 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 XXX de derecha a izquierda).

TODO!Poner aquí un ejemplo de grafo computacional, por ejemplo una recta o algo así. Algo que tenga al menos dos pasos y explicar cómo se calcula la derivada y por qué esto viene genial. Luego, en el seiguiente capítulo se puede explicar el backpropagation con esto (ver http://colah.github.io/posts/2015-08-Backprop/)

El paradigma de los Agentes Inteligentes

En la figura 2 se mostraban los cuatro objetivos perseguidos por la IA. 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. El concepto de **agente** pertenece a todo el área de la computación, y no únicamente a la IA.

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 15):

- 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 la IA, y las denominaremos indistintamente *agentes*, *agentes inteligentes* o *agentes racionales* [19]. Por tanto usaremos la siguiente definición:

"Un agente es una entidad física o virtual que realiza una acción [20] 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

[19] El término preferido en esta tésis 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 que no es un inteligente cae dentro de los dominios de la filosofía.

[20] 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 imprime 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.

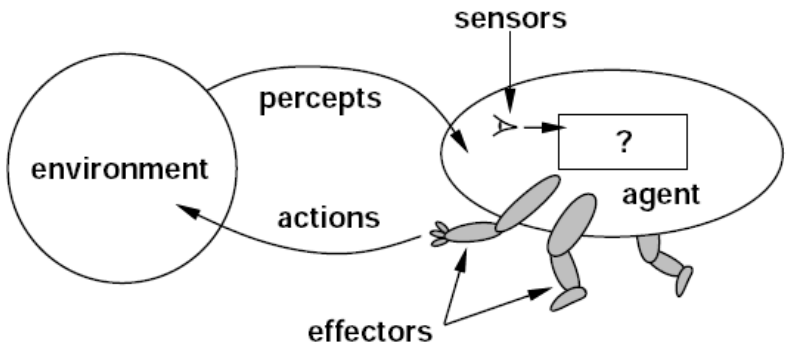


Figura 15: Esquema de un agente y sus propiedades. Aunque no existe una definición comúnmente aceptada de agente, sí que existe una serie de propiedades que los que los identifican. Es autónomo, opera realiza acciones sobre un entorno dependiendo de las percepciones que le llegan de éste y tiene la capacidad de comunicarse con el resto de elementos, incluídos otros agentes.**TODO!**CAMBIAR LA IMAGEN Y VER CÓMO SE PUEDE INCLUIR EL CONCEPTO DE COLABORACIÓN

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 actuan de la mejor manera posible sobre él (resuelven) de acuerdo a su conocimento del medio y su estado interno (en base a algoritmos como RNA, Lógica Difusa, . . .). Por ello desde mediados de los años 1990 el concepto de agente inteligente ha ganado tanta popularidad [21].

[21] Tanto es así que en algunos trabajos se define el objetivo de la IA como la implementación de la función agente, esto es, la función que realiza la correspondiencia de una percepción a una acción, para un problema dado.

Tipos de entorno

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

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

- **Observable**. Un entorno es **totalmente observable** cuando el agente es capaz de captar toda la información relevante para la toma de una decisión y no necesita mantener ningún modelo interno del entorno, **parcialmente observable** cuando la información obtenida es incompleta o tiene ruido y **no observable** cuando el agente no posee sensores.

- **Multiagente o. monoagente**. Un entorno es **multiagente** cuando requiere de múltiples agentes interactuando para llegar a una solución mientras que es **monoagente** cuando sólo requiere de uno para ello.

- **Determinista o. no determinista**. Si el estado del entorno actual depende totalmente del estado anterior, se dice que el entorno es

determinista. Si no es así, se considera **no determinista** o **estocástico** [22].

- **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 decisioń 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** [23].

[22] 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.

[23] 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.

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.

Figura 16: 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.

La figura 16 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.
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 17):

Figura 17: 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.

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 (figura18):

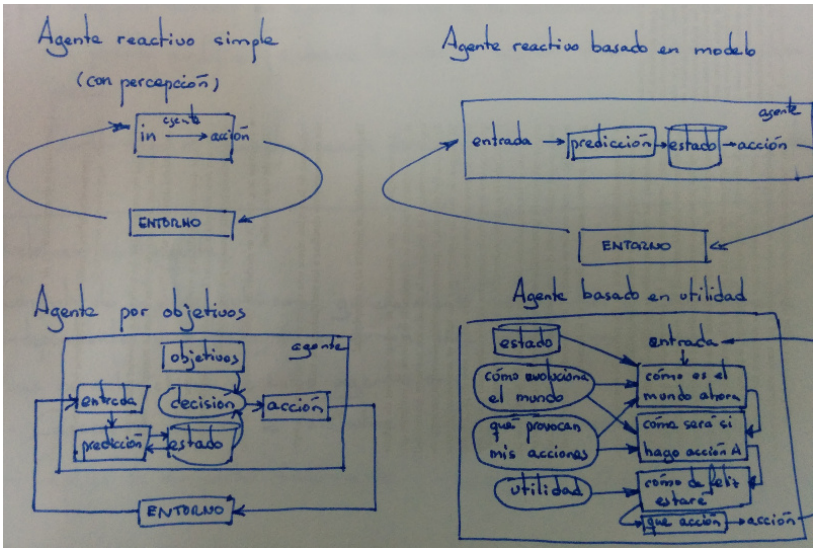


Figura 18: Distintas arquitecturas de agentes en función del comportamiento. Dependiendo de las acciones a realizar, se identifican tres tipos, los reactivos que aplican una acción sin proceso deductivo y los basados en modelo y utilidad (en algunos contextos denominados deliberativos) que basan su comportamiento en alguna forma de deducción.

- **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 tres 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 16, 17 y 18).

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 IA, 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 quizá 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 IA 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 modelosy 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 si. Esta interacción puede implementarse de diversas maneras [24] y siempre toman una o las dos formas siguientes (figura 19):

- **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.
- **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.

[24] 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.

Una ilustración de algún entorno de vida artificial donde los agentes compiten y uno de algo donde cooperen

Figura 19: 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.

Bibliografía

[Ful,]

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

[Dijkstra, 1972] Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM*, 15(10):859–866.

[Du and Swamy, 2006] Du, K. and Swamy, M. (2006). Neural networks in a softcomputing framework.

[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.

[Hebb, 1949] Hebb, D. O. (1949). The organization of behavior: A neurophysiological approach.

[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.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kiszka et al., 1985] Kiszka, J. B., Kocha\'nska, M. E., and Sliwi\'nska, D. S. (1985). The Influence of Some Parameters on the Accuracy of a Fuzzy Model. *Industrial Applications of Fuzzy Control*, pages 187–230.

[Kohonen, 1998] Kohonen, T. (1998). The self-organizing map. *Neurocomputing*, 21(1-3):1–6.

[Ma, 2004] Ma, X. (2004). Toward An Integrated Car-following and Lane- changing Model by A Neural Fuzzy Approach. pages 1–15.

[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.

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). Perceptrons.

[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.

[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.

[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.

[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.

[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.

[Shoham, 1993] Shoham, Y. (1993). Agent-oriented programming. *Artificial intelligence*, 60(1):51–92.

[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.

[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.

[Widrow and Hoff, 1960] Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. Technical report, STANFORD UNIV CA STANFORD ELECTRONICS LABS.

[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.

[y Cajal, 1888] y Cajal, S. R. (1888). *Estructura de los centros nerviosos de las aves*.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Cómo citar esta tesis

Si deseas citar esta tesis, lo primero gracias. Me alegro 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 = {16 de febrero 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.