UNIVERSIDAD
COMPLUTENSE
MADRID

LECCIÓN INAUGURAL
Curso Académico 2019/2020

Las múltiples caras del algoritmo

Ricardo Peña Marí
Catedrático de Lenguajes y Sistemas Informáticos
Facultad de Informática

Madrid, 2019

Las múltiples caras del algoritmo

Ricardo Peña Marí
Catedrático de Lenguajes y Sistemas Informáticos
Facultad de Informática

Corrección, edición, diseño y maquetación
Departamento de Estudios e Imagen Corporativa. UCM

Impresión: Grafo Industrias Gráficas

Señor Rector Magnífico; ilustres autoridades; queridos compañeros; queridos estudiantes y personal de la Universidad Complutense; señoras, señores.

Este año le corresponde a la Facultad de Informática impartir la Lección Inaugural del curso que comienza, y quiero que mis primeras palabras sean de agradecimiento al decano de mi facultad, que me propuso como conferenciante, así como a mis compañeros, que ratificaron su decisión en la Junta de Facultad, lo que supone no solo su confianza en mí, sino también su afecto. A todos, muchas gracias. También agradezco a varios compañeros de mi departamento, incluido el propio decano, que me hayan hecho llegar sus aportaciones y comentarios, que sin duda han servido para enriquecer y mejorar esta lección que hoy les voy a dictar.

Es un honor y al mismo tiempo una gran responsabilidad estar hoy aquí ante tan variada y docta audiencia, y les aseguro que ha supuesto para mí un verdadero reto tratar de exponer una lección que pueda ser atractiva e interesante para un público tan experto en tantas y tan diferentes áreas de conocimiento.

1. El concepto originario de algoritmo

Los algoritmos han acompañado a la humanidad desde el comienzo de la historia. Por ejemplo, es famoso el algoritmo de Euclides para calcular el máximo común divisor de dos números mediante restas sucesivas. De tres siglos antes, es decir del siglo VI a. C., es el teorema atribuido a Pitágoras, que permite calcular un lado del triángulo rectángulo conocidos los otros dos lados. Y de 3.000 años antes son las tablillas de arcilla que los babilonios nos dejaron con numerosos cálculos numéricos, tales como la fórmula del interés compuesto o la división de dos números en base 60. De manera premonitoria, muchas de esas tablillas terminaban con la inscripción "y este es el procedimiento".

Los algoritmos existen desde mucho antes de que aparecieran máquinas capaces de ejecutarlos de forma automática. Pero a partir de un momento en la historia, empezaron a surgir artefactos mecánicos para realizar algunos de ellos. La máquina de sumar de Blaise Pascal y la de multiplicar de Gottfried Leibniz, ambas del siglo XVII, fueron instrumentos mecánicos dedicados a un solo algoritmo, respectivamente el de la suma y el de la multiplicación. Sin embargo, el paso relevante no era disponer de máquinas especializadas en determinados algoritmos, sino crear otras que pudieran ser instruidas para ejecutar cualquiera de ellos.

En esa dirección, el primer avance se produjo en 1800 con el telar programable de Joseph Marie Jacquard. Las instrucciones para realizar el dibujo en el tejido se grababan en una secuencia de cartones perforados que determinaban los hilos de la urdimbre que se levantaban en cada paso de la lanzadera, formando así el dibujo. Cambiando las bobinas de hilo, el mismo dibujo podía reproducirse en distintos colores, de la misma manera que hoy podemos ejecutar un mismo programa con distintos datos de entrada, obteniendo en cada ejecución un resultado distinto. Los cartones podían almacenarse, al igual que hoy almacenamos nuestros programas en el disco del ordenador, formando así bibliotecas de dibujos que podían reutilizarse en cualquier momento posterior. Eran, pues, verdaderos algoritmos, especializados en este caso en producir tejidos.

El telar de Jacquard inspiró al matemático británico Charles Babbage en el diseño de su Máquina Analítica, en la cual trabajó desde 1833 hasta su muerte en 1871. La máquina nunca llegó a funcionar debido fundamentalmente a las limitaciones tecnológicas de la época. Era totalmente mecánica, y la precisión

necesaria para sus piezas estaba más allá de lo que era posible conseguir en aquel periodo histórico. Pero sus diseños eran correctos y, de hecho, inspiraron a los ingenieros de los años 40 del siglo XX en la construcción de los primeros computadores electromecánicos. Tenía una unidad aritmética de cálculo y una memoria donde se almacenaban los valores numéricos necesarios para el cómputo. Podía encadenar un cómputo con el siguiente, haciendo que los resultados del primero fueran entradas para el segundo. Y así, paso a paso, hubiera podido ejecutar, en caso de haber sido construida, un algoritmo complejo. Pero lo más revolucionario era que podía ser programada mediante una secuencia de instrucciones grabadas en unas tablillas perforadas muy similares a las del telar de Jacquard. Por tanto, era una máquina programable de propósito general. Solo hacía falta "escribir" —o mejor, perforar— programas diferentes para hacer que ejecutara cálculos diferentes.

A pesar de estos importantísimos precursores, los informáticos consideramos que el padre fundador de la ciencia informática es el matemático británico Alan Turing. No voy a relatarles aquí su vida y su obra, que seguramente ustedes conocen de sobra gracias a los numerosos actos conmemorativos que tuvieron lugar en 2012, al cumplirse 100 años de su nacimiento. También, gracias a las dos o tres películas, más o menos afortunadas, que han glosado su vida y su dramática muerte. Pero sí quiero detenerme en un par de conceptos que él aportó y que son relevantes para el resto de lo que voy a relatarles en esta lección.

Corría el año 1936, tan dramático para los españoles y preámbulo de la gran guerra mundial que asolaría el mundo unos pocos años después. Turing investigaba en Cambridge [13] sobre el llamado problema de decisión, que entre otros que también afectaban a sus fundamentos, tenía convulsionada a la comunidad matemática desde principios del siglo XX. El problema estaba formulado en los siguientes términos: "Dada una fórmula arbitraria de la lógica de primer orden con suficientes elementos de la aritmética, ¿existe un procedimiento efectivo para decidir si la fórmula es válida?" Aunque tal concepto no estaba formalizado matemáticamente, por procedimiento efectivo se entendía una secuencia finita de pasos, en cada uno de los cuales se aplica mecánicamente alguna regla, elegida entre un conjunto finito de ellas. En el caso del problema de decisión, se debía partir de unos axiomas aceptados por todos y de unas reglas de deducción precisas, de tal manera que, siguiendo dichas reglas, se pudiera llegar a una conclusión afirmativa o negativa, cualquiera que fuese la fórmula de entrada. Muchos matemáticos, entre ellos el gran David Hilbert, que fue quien propuso este problema como uno de los 23 retos matemáticos del siglo XX, creían que

tal procedimiento existía y que tan solo era necesario acertar con los axiomas y con los pasos de deducción apropiados.

Turing, en cambio, concentró sus esfuerzos en formalizar la idea un tanto vaga de procedimiento efectivo. Imaginó para ello a un calculador humano realizando un procedimiento efectivo a la vez complejo y rutinario, e inspirándose en esta imagen, diseñó una máquina conceptual [13] que imitara su comportamiento:

- El calculador humano necesitaría cantidades ingentes de papel. Así, dotó a su máquina de una memoria inagotable bajo la forma de una cinta imaginaria dividida en infinitas casillas, donde se podían grabar o leer símbolos, a razón de un símbolo por casilla.

- Una cabeza lectora/grabadora estaba situada sobre una de las casillas. Representaba el punto de atención de la persona durante su cómputo.

- El calculador, al igual que su máquina, utilizaría un alfabeto finito de símbolos.

- El cómputo realizado por el humano tendría un número finito de etapas o fases. En caso contrario, nunca terminaría. Consecuentemente, su máquina pasaría por un número finito de estados.

- Cada paso de cómputo de su máquina era extremadamente elemental: en función del estado de la máquina y del símbolo bajo la cabeza lectora, la acción a realizar consistía, bien en escribir un símbolo en la casilla situada bajo la cabeza, bien en desplazar la cabeza un lugar a la izquierda o un lugar a la derecha, bien en cambiar de estado, o bien en cualquier combinación de las tres acciones descritas. Turing formalizó el conjunto de acciones posibles como una función matemática, a la que llamó función de transición.

Encantado con su nuevo juguete, Turing escribió muchas máquinas, es decir muchas funciones de transición, dedicadas a realizar cálculos tales como las operaciones aritméticas habituales, las raíces cuadrada y cúbica de un número, y otros semejantes, hasta quedar convencido de que sus máquinas eran capaces de realizar cualquiera de los algoritmos conocidos. Entonces, identificó el concepto de algoritmo, no bien formalizado hasta ese momento, con todo aquel cálculo que pudiera ser realizado por alguna de sus máquinas. A continuación, se interesó por la potencia de las mismas, dotándolas de varias cintas y varias cabezas lectoras, y de otros aditamentos, llegando a la conclusión de que el

conjunto de algoritmos que eran capaces de ejecutar no cambiaba por el hecho de ser más sofisticadas. De hecho, cualquiera de las nuevas máquinas podía ser emulada por una de las anteriores. En este camino, descubrió que podía definir una máquina capaz de emular a todas las demás: la llamó la Máquina Universal. La máquina universal recibía en una cinta la descripción de una máquina de Turing convencional. Gracias a la finitud del alfabeto y del conjunto de estados, toda máquina podía ser codificada de forma finita en la cinta. En otra cinta recibía los datos tal como los esperaba la máquina a simular. A partir de aquí, realizaba el mismo cómputo que hubiera realizado la máquina codificada. Cambiando la máquina codificada y los datos de entrada, su máquina universal podía ejecutar, pues, cualquier algoritmo. En términos modernos, una máquina universal de Turing sería equivalente a un intérprete de un lenguaje de programación.

En la máquina universal de Turing está el concepto revolucionario de computador con programa almacenado en memoria, que al término de la Segunda Guerra Mundial dio lugar a la creación de los primeros computadores electrónicos, uno de ellos construido por el propio Turing.

Hago un breve inciso para hacer notar la pequeñísima distancia que puede haber en ocasiones entre la llamada investigación fundamental y la llamada investigación aplicada. Preocupado por un problema de fundamentos de las matemáticas, y tal vez sin proponérselo, Turing nos regaló la teoría sobre la que se asientan nuestros computadores actuales y, a estas alturas de la historia, podríamos decir que se asientan las bases de la propia civilización del siglo XXI, absolutamente dependiente de estas omnipresentes máquinas.

Pero Turing no se olvidó del problema de decisión, origen de toda esta investigación. Demostró que había problemas bien definidos que sus máquinas no podían resolver. Vio que era fácil definir máquinas que nunca terminaban su cómputo para algunas entradas y que ciclaban eternamente. Se planteó estudiar si esta propiedad indeseada podía ser detectada por otra máquina, y descubrió que no. Se trata del llamado problema de parada, que puede ser enunciado del modo siguiente: no existe ninguna máquina tal que, dada otra máquina arbitraria y unos datos de entrada para ella, decida si dicha máquina parará su cómputo al ejecutarse a partir de dicha entrada. A un problema, con respuesta sí o no, que no admita una solución algorítmica se le da el nombre de indecidible. El problema de parada es el primer problema de la historia que se demostró indecidible. Este problema no es en absoluto irrelevante. Nuestros ordenadores actuales se quedan bloqueados con frecuencia a causa de bu-

cles que no terminan. Bien nos gustaría disponer de un algoritmo que pudiera detectar dicha potencial no terminación, inspeccionando tan solo el texto de un programa. Turing nos enseñó que tal algoritmo no existe. A partir de este resultado le resultó fácil demostrar que no existe ningún algoritmo para decidir en general la validez de una fórmula lógica. Es decir, resolvió en negativo el famoso problema de decisión planteado por Hilbert. Esta investigación se realizó en paralelo con un resultado equivalente que Alonzo Church había demostrado unos meses antes, utilizando el formalismo del cálculo lambda. Cuando el artículo de Church llegó a manos de Turing, lo primero que hizo fue demostrar la equivalencia de ambos formalismos y añadir un pequeño apéndice a su trabajo con dicha demostración.

En realidad, hay infinitos problemas para los que no existen algoritmos. Se puede probar esta afirmación mediante un simple razonamiento sobre sus respectivas cardinalidades: hay tantos algoritmos como máquinas de Turing, y estas forman un conjunto infinito, pero un infinito numerable, como el de los números naturales. En cambio, se pueden definir tantos problemas como funciones matemáticas, y estas constituyen un conjunto también infinito, pero un infinito no numerable, como el de los números reales. Sabemos que ambos infinitos no se pueden poner en correspondencia uno a uno, por lo que concluimos que hay necesariamente menos algoritmos que problemas.

A la vez que Turing, e inmediatamente antes y después, otros investigadores propusieron modelos de cómputo distintos de las máquinas de Turing, o si se quiere, otras formas de expresar algoritmos. Cito entre ellos los sistemas de correspondencia de Emil Post [11], la teoría de funciones recursivas de Stephen Kleene [1] y el cálculo lambda de Alonzo Church [2]. Posteriormente, se han ideado otros muchos modelos, algunos de los cuales han dado lugar a diferentes familias de lenguajes de programación, tales como la programación lógica, o la programación paralela. En todos los casos, se ha podido demostrar que el conjunto de algoritmos expresables por los distintos modelos es consistentemente el mismo. Es decir, los problemas indecidibles lo siguen siendo en todos los modelos conocidos, y lo mismo sucede con los decidibles.

En este punto es apropiado hacer una breve referencia a la computación cuántica, sugerida por primera vez por el físico Richard Feynmann en 1982 [7]. Se trata de un nuevo modelo de cómputo, asociado a un nuevo tipo de computadores, que es todavía objeto de intensa investigación y que por el momento es más rico en promesas que en realidades. Se basa en el principio de superposición

de estados de la mecánica cuántica y los pocos algoritmos desarrollados hasta ahora sacan provecho de ello para realizar una suerte de computación paralela de alto rendimiento. Su potencial ventaja estriba en reducir la complejidad de algunos algoritmos que son muy costosos en los computadores convencionales, como es el caso de la descomposición en factores primos de grandes números. Ello podría implicar tener que desarrollar un nuevo tipo de criptografía que no base su seguridad en el inmenso coste de tales algoritmos. Pero no parece que la computación cuántica vaya a convertir en decidibles los problemas que hoy son indecidibles.

Esta consistencia en la potencia expresiva de los diferentes modelos formales de cómputo ha llevado a los investigadores a conjeturar que lo que es calculable por humanos por procedimientos finitos y rutinarios —es decir los que corresponden a la idea intuitiva de procedimiento efectivo o algorítmico— coincide exactamente con lo pueden calcular los modelos formales de cómputo. Esta conjetura se conoce con el nombre de Tesis de Church-Turing y aunque es indemostrable, porque lo que es intuitivamente calculable no es un concepto bien definido, pone unos límites muy precisos a lo que se puede esperar de los métodos algorítmicos.

La aparición de los ordenadores comerciales a partir de los años 50 del siglo pasado ha promovido la invención de innumerables algoritmos en todas las áreas de actividad. Después de casi 70 años de convivir con estas máquinas y sus algoritmos, no hay aspecto de la vida cotidiana, del trabajo en las administraciones, en las empresas, en los hospitales, en los medios de transporte, o en las instituciones científicas, que no haya sido profundamente modificado por los procesos de informatización. Por ejemplo, un automóvil actual de gama media puede incluir hasta 100 procesadores, que controlan desde la mezcla de combustible, hasta los frenos, la presión de los neumáticos, el sistema eléctrico, el consumo y los diferentes paneles de instrumentos.

Nuestro teléfono móvil es un receptáculo repleto de algoritmos. Por ejemplo, cuando encendemos el navegador para movernos con seguridad durante un viaje, o en una ciudad desconocida, se activan unos cuantos algoritmos. Uno de ellos se dedica a calcular con mucha precisión nuestra ubicación en la superficie del globo terráqueo. Asistido por los satélites del sistema GPS o del sistema Galileo, por unas buenas comunicaciones y por unos relojes atómicos extremadamente precisos, nuestro teléfono calcula la distancia a tres o cuatro de dichos satélites. Mediante un algoritmo de triangulación espacial y conocida la posición de los satélites, calcula la propia posición.

Otro algoritmo descarga los mapas apropiados a dicha ubicación desde un servidor de internet. Un tercero realiza un cálculo de caminos mínimos sobre el grafo que representa el mapa y traza una ruta óptima desde la ubicación propia hasta la ubicación introducida como destino. Y un cuarto muestra los resultados sobre la pantalla y la va actualizando a medida que progresamos por nuestra ruta.

Si al cabo del rato golpeamos levemente con el dedo el icono de un fichero MP3 del mismo teléfono móvil, podemos escuchar placenteramente nuestra melodía preferida a través de los altavoces del coche. El contenido de dicho fichero ha sido creado por un algoritmo de compresión que, a partir de las muestras de amplitud de la señal de audio, ha realizado los siguientes cálculos: ha llevado a cabo una transformación, conocida como transformada rápida de Fourier, que transforma las muestras de amplitud en muestras de la misma señal en el dominio de las frecuencias, en virtud de un teorema de Joseph Fourier en el siglo XIX que prueba que toda función periódica puede ser expresada como la suma de funciones sinusoidales de distintas frecuencias. Una vez en el dominio de las frecuencias, muchas de ellas son filtradas por ser consideradas redundantes en virtud de un modelo del oído humano propuesto en la tesis del ingeniero alemán Karlheinz Brandenburg en 1989. Un algoritmo final desarrollado en 1952 por David Huffmann, doctorando estadounidense del MIT, comprime los bits de la señal de forma óptima sin perder precisión. El software de nuestro teléfono realiza las inversas de todas estas operaciones: descomprime el fichero, convierte las muestras de frecuencias en muestras de amplitud mediante la transformada inversa de Fourier y las envía a los altavoces.

Basten de momento estos dos ejemplos para dar cuenta de hasta qué punto los algoritmos se han introducido en nuestra vida y la condicionan cada vez más. En general para bien, aunque deberemos aprender a vivir con las nuevas tecnologías de forma que no perdamos algunos valores humanos esenciales. El número de nuevas opciones que nos ofrecen es tan elevado que corremos el riesgo de perdernos entre todas ellas. Debemos prestar atención a ciertas adicciones recientes, como la de desear estar constantemente conectados e informados. O como la de tener decenas o cientos de “amigos” que nos inundan con informaciones no siempre relevantes. O la pérdida de espacios para la reflexión o para la lectura sosegada de un libro. O la sustitución de la conversación personal por la comunicación estereotipada de las redes sociales y otros fenómenos semejantes que empiezan a preocupar a sociólogos, psicólogos y educadores. También empiezan a suscitarse problemas éticos

en relación con las decisiones que toman por nosotros los algoritmos. Por ejemplo, ¿con qué criterio debería actuar el software de un coche autónomo ante un inminente accidente, si cualquier decisión que tomase podría implicar la pérdida de vidas humanas?

2. Algoritmos que aprenden

Dejemos simplemente apuntados estos nuevos problemas y prosigamos con otras caras del algoritmo. Los seres humanos tenemos habilidades que es difícil convertir en algoritmos que puedan ejecutarse en una máquina. Por ejemplo, nuestro cerebro puede reconocer las letras del alfabeto, aunque estén escritas a mano y con diferentes caligrafías, pero no sabemos explicarle a un computador cómo hacerlo. Podemos leer un texto en voz alta sin grandes dificultades, o podemos convertir en texto escrito un discurso dictado por otra persona. Podemos reconocer las caras de nuestros familiares y amigos con un simple vistazo y podemos conducir un vehículo por una carretera llena de curvas y atender a la vez a las señalizaciones. Lo hacemos gracias al entrenamiento que recibimos. Pero si tuviéramos que formalizar estas actividades con la precisión suficiente para que una máquina hiciera lo mismo, no sabríamos por dónde empezar.

Nos adentramos entonces en otra rama de la Ingeniería Informática que conocemos como Inteligencia Artificial. Al propio Alan Turing se le considera el padre de esta disciplina. Estaba convencido de que el cerebro humano era simplemente una máquina muy sofisticada y de que algún día podríamos construir máquinas que emularan algunas de sus capacidades. Concibió la idea de que tal vez sería más productivo construir algoritmos sencillos, pero dotados de la capacidad de aprender, que algoritmos de entrada muy elaborados. Nos dejó incluso un trabajo, escrito en el año 1948 [14], donde define un tipo de "máquinas desorganizadas" con reglas para establecer conexiones variables producto del aprendizaje. Este trabajo es uno de los orígenes de lo que hoy conocemos como redes neuronales, cuyos resultados son cada vez más espectaculares y de las que hablaré a continuación.

En un algoritmo convencional, se leen los datos de entrada, el algoritmo los procesa y finalmente se producen los resultados de salida. Esta acción se puede repetir cuantas veces se desee con distintos datos de entrada. En uno de aprendizaje automático estas piezas se barajan en otro orden: se suministran los datos de entrada y la salida esperada para dicha entrada. Se repite esta acción

muchas veces. El resultado de este proceso es el algoritmo. Es decir, se entrena al algoritmo de un modo muy similar a como se entrena a un niño a reconocer las letras. Por ejemplo, se le muestra muchas veces la A junto con otras letras que no son la A, hasta que el algoritmo "aprende" a distinguir la letra A en medio de otras letras.

El núcleo de estos algoritmos lo forma una red de neuronas artificiales, es decir, de unas neuronas simuladas por medio de un programa convencional de ordenador. Una neurona biológica tiene unas conexiones de entrada, llamadas dendritas, y una de salida llamada axón. La neurona recibe estímulos de otras neuronas a través de las dendritas, y cuando estos estímulos alcanzan un cierto umbral, produce a su vez una señal a través del axón que, merced a unas conexiones llamadas sinapsis, sirve de estímulo a las siguientes neuronas. Los neurobiólogos han llegado a la conclusión de que nuestros recuerdos son simplemente circuitos neuronales que han sido reforzados por el aprendizaje. Nuestro conocimiento, por ejemplo, de que Granada se rindió a los Reyes Católicos en 1492 lo formarían algunas sinapsis por las que circulan unos pocos iones más de potasio de los habituales.

A semejanza de las biológicas, una neurona artificial es una función matemática de unas entradas en una salida. Cada entrada recibe un valor numérico. La función calcula la suma del valor de las entradas, y si esta alcanza un cierto umbral, produce un valor en la salida. Se asemejaría, en palabras de Pedro Domingos, autor del libro The Master Algorithm [5], a un pequeño parlamento en el que, si los votos son suficientes, se produce una decisión. Sin embargo, no todos los votos valen lo mismo. Al igual que nos fiamos más de los consejos de un amigo que de los de un desconocido, la neurona da más peso a ciertas entradas, cuyos consejos han conducido más veces a un acierto, y quita peso a las entradas que han conducido más veces a un fracaso. El resultado del aprendizaje son justamente estos pesos. Aprender consistiría entonces en encontrar los pesos que hacen que la neurona se dispare en todos los ejemplares positivos y no lo haga en ninguno de los negativos.

Las redes neuronales artificiales tuvieron unos primeros años de desarrollo entre 1945 y 1965, produciendo modelos de éxito tales como el perceptrón de Frank Rosenblatt [12], capaz de reconocer caracteres y otras formas geométricas. Entre 1965 y 1975 fueron abandonadas parcialmente, debido a diversas limitaciones y a los cuantiosos recursos computacionales que requerían y que eran más bien escasos en esos años. Posteriormente, volvieron a resurgir gracias a un algoritmo, denominado de propagación hacia atrás que permite ensamblar varias capas

de redes y propagar los errores y aciertos detectados en la salida a las capas interiores, permitiendo así ajustar los pesos de estas.

En los años 90 se consiguen redes formadas por varias subredes, cada una de las cuales sufre un proceso de aprendizaje por separado. Se empieza a hablar del *deep learning* o aprendizaje profundo. Actualmente existe un ecosistema complejo de variantes de redes neuronales, cada una de ellas especializadas en determinadas tareas. Por ejemplo, las redes convolucionales se especializan en reconocimiento de imágenes, las recurrentes, en reconocimiento del habla y en traducción de textos, las llamadas Máquinas Restringidas de Boltzmann, en entrenamiento no supervisado, las redes que aprenden por recompensa y castigo, se utilizan en juegos como el ajedrez, y así sucesivamente.

Un caso de éxito, que fue noticia en los periódicos, lo constituyó la red neuronal profunda Google Network, desarrollada por Google en 2012, capaz de reconocer caras de gatos en imágenes de vídeos. Quizás fuera la red más voluminosa desarrollada hasta entonces. Tenía nueve capas y 1.000 millones de conexiones entre sus neuronas. Otro caso que saltó a la prensa en 2018, y en el que también estaba implicado Google, fue el del programa AlphaZero, entrenado a jugar al ajedrez a partir tan solo de las reglas del juego. Tras jugar unas cuantas horas contra sí mismo, ajustó los pesos de sus jugadas de tal manera que fue capaz de batir al programa Stockfish, el mejor programa de ajedrez hasta ese momento. De 1.000 partidas, tan solo perdió 6, ganó 155 y empató el resto. La misma técnica se ha aplicado con éxito a otro juego aún más complejo que el ajedrez como es el Go.

Más sorprendente sea quizás otro algoritmo desarrollado en 2017, por una graduada en Matemáticas actualmente haciendo su tesis doctoral en Montreal, capaz de generar caras de gatos indistinguibles de las de los gatos reales. Para ello se usaron dos redes neuronales, una que generaba caras a partir de abstracciones obtenidas de procesar muchas caras de gatos reales y otra que trataba de discriminar si la cara recibida era real o había sido automáticamente generada. Realimentando los resultados de la red discriminadora a la red generadora, esta última fue afinando sus caras automáticas hasta hacerlas indistinguibles de las reales. Con esta técnica, ya se generan actualmente caras humanas. Virtuales sí, pero completamente creíbles.

Los casos de éxito de los sistemas informáticos entrenados de este modo se cuentan por centenares. Nuestros teléfonos móviles vuelven a ser un buen ejemplo de ello. Podemos “hablarle” al teléfono y él convierte nuestras palabras en

texto escrito con muy pocos errores. O podemos pedirle que nos hable y realiza el proceso inverso, leyendo textos para nosotros. Incluso podemos elegir entre distintas voces masculinas y femeninas para que nuestro navegador nos haga las indicaciones verbales oportunas durante la ruta. También sabe cómo distinguir el correo basura del correo relevante para nosotros, o se permite sugerir correcciones gramaticales a nuestros textos, o artículos de consumo que podrían ser de nuestro interés.

En medicina se han entrenado redes para detectar tempranamente la enfermedad de Alzheimer, o el cáncer de pulmón, a partir de imágenes radiológicas. En astronomía, se las ha entrenado para reconocer objetos lejanos de distintos tipos en imágenes del firmamento. Gracias a ellas, nuestro mapa del cielo es mucho más rico de lo que hubiéramos conseguido con un proceso realizado solo por humanos. El último reto, aún en vías de consecución, es la conducción autónoma de vehículos, al que las compañías de automóviles y algunas grandes tecnológicas, como Google y Apple, están dedicando actualmente cuantiosos recursos.

3. Imitando a la naturaleza

Pero la imitación del modo de aprender del cerebro humano no ha sido la única fuente de inspiración para idear nuevas formas de algoritmos. La naturaleza es rica en ejemplos de cómo los seres vivos resuelven sus problemas y se adaptan con eficacia al medio ambiente. Hay varias familias de algoritmos, agrupadas bajo el nombre genérico de algoritmos bioinspirados, que imitan ciertos comportamientos de éxito de los seres vivos. Una de ellas son los llamados algoritmos evolutivos y alguna de sus variantes como los algoritmos genéticos [9]. Aquí lo que se imita es el proceso de evolución y selección natural de las poblaciones de individuos, cuyos fundamentos están en la obra del gran Charles Darwin [3]. Según su teoría, plenamente corroborada científicamente, las dos fuerzas conductoras de la evolución son la selección natural y la reproducción imperfecta. Las poblaciones, en ausencia de limitaciones, tienden a reproducirse de forma exponencial. Cuando los recursos son insuficientes para sostenerlas, solo los individuos mejor adaptados sobreviven y tienen posibilidades de reproducirse. La reproducción da lugar, por un lado al cruce entre características de distintos individuos, produciendo otros individuos diferentes, y por otro a mutaciones aleatorias debido a que el proceso de copia de los genes no es perfecto. El propio entorno también puede producir mutaciones adicionales. Eso hace que las poblaciones no sean idénticas de

una generación a otra y de ese modo puedan adaptarse a los cambios del entorno. Los algoritmos evolutivos imitan todas estas características:

- Comienzan con una población inicial, con frecuencia aleatoria.
- Poseen una medida de la cercanía de cada individuo a la solución perseguida y con ella eliminan a los individuos menos aptos, es decir, imitan a la selección natural.
- Tienen mecanismos de cruce entre individuos y de mutación aleatoria.

Se han aplicado a problemas de optimización y de búsqueda, costosos de tratar con algoritmos convencionales. El más famoso de todos ellos es el conocido problema del viajante de comercio, que consiste en encontrar una ruta óptima que visite un conjunto de ciudades una sola vez y vuelva al punto de partida. Este problema pertenece a los problemas llamados NP-completos, y me da pie para detenerme brevemente en otro de los problemas informáticos famosos, y por cierto todavía sin resolver: la pregunta de si P=NP o P≠NP.

El público no informado tiende a pensar que los computadores pueden resolver todos los problemas, y que los que no pueden resolver hoy, podrán hacerlo mañana, porque su potencia de cálculo crece continuamente. Los informáticos sabemos, como hemos dicho, que una infinidad de problemas de cómputo no tendrán solución nunca. Pero también sabemos que, aunque otros problemas tienen algoritmos que los resuelven, estos emplean tanto tiempo de cálculo, que a efectos prácticos es como si dichos problemas fueran irresolubles. Llamamos intratables a los problemas que solo pueden resolverse con algoritmos de coste exponencial o mayor. Para hacerse una idea de lo que significa ese coste, un algoritmo exponencial en el tamaño de los datos, que ejecutara mil millones de operaciones por segundo, necesitaría un día para resolver un problema de tamaño 46 y diez mil veces la edad del universo para resolver uno de tamaño 100. En el caso del viajante de comercio, el tamaño de los datos es el número de ciudades.

A los problemas que resuelven los computadores en un tiempo razonable se les llama polinomiales, y todos ellos se agrupan en la clase P, nombrada así porque su tiempo de cómputo está descrito por un polinomio en el tamaño de los datos. Hay otra clase de problemas a la que llamamos NP, cuya definición está hecha de tal modo que incluye todos los problemas de la clase P, pero también otros muchos que se comportan de un modo intrigante. Para estos problemas intrigantes de la clase NP, los mejores algoritmos que se conocen tienen un coste exponencial y nadie ha encontrado un algoritmo polinomial para ninguno de ellos. Tampoco nadie ha demostrado que no existan tales algoritmos, es decir, que sean problemas intratables. Están, por decirlo así, en una especie de limbo

informático: no se sabe si son polinomiales o si son intratables. La teoría desarrollada en estos años ha llegado sin embargo a alguna conclusión: ha definido una subclase de la clase NP, la subclase de los problemas NP-completos, en la cual se agrupan los problemas más costosos de la clase NP, de tal forma que, si para uno cualquiera de dichos problemas se encontrara un algoritmo polinomial, entonces todos ellos se resolverían en tiempo polinomial y además la clase NP colapsaría a P, es decir tendríamos la igualdad P=NP. Más aún, si se demostrara que uno solo de los problemas NP-completos fuera intratable, entonces todos ellos lo serían y tendríamos la desigualdad P≠NP. Ninguna de estas dos cosas se ha podido probar hasta ahora.

Concluimos que para problemas como el del viajante existen pocas esperanzas de encontrar algoritmos mejores que los exponenciales. Otros problemas igual de costosos son el diseño de horarios —como bien saben nuestros vicedecanos de ordenación académica—, la planificación óptima de tareas y los problemas de empaquetamiento óptimo. Por ejemplo, algo tan aparentemente sencillo como llenar de forma óptima el maletero de un coche, o la bodega de un barco, se vuelve un problema inmanejable cuando el número de objetos supera unas pocas decenas.

Los algoritmos evolutivos representan una esperanza de resolver estos problemas, quizás de forma no óptima, pero con costes soportables. Un algoritmo evolutivo representa las posibles soluciones parciales o totales a estos problemas como individuos. Genera inicialmente una población lo suficientemente variada como para que algunos de los individuos puedan evolucionar hacia el óptimo. Mediante los procesos de selección natural, elimina las soluciones menos prometedores y, mediante los procedimientos de cruce y mutación, mezcla las características de los individuos, generando con gran probabilidad algunos aún más cercanos al óptimo. Repitiendo el proceso numerosas veces, se alcanzan soluciones suficientemente buenas con un coste razonable.

Otras familias son los algoritmos inspirados en colonias de hormigas y los que imitan los enjambres de insectos, o de pájaros, o los bancos de peces. En estas colectividades, los individuos poseen cierta inteligencia, pero también la posee el enjambre como un todo. En el caso de las hormigas [6], cuando buscan alimento por rutas en principio aleatorias, van dejando un rastro de feromonas que otras hormigas pueden olfatear. Si una ruta se convierte en prometedora de alimento, la proporción de feromonas sube a medida que la ruta es transitada por más hormigas. Las rutas poco interesantes pierden al poco tiempo sus feromonas por evaporación. De esta forma, las rutas van modificándose y convergiendo hacia una ruta óptima desde el hormiguero a la comida.

Los algoritmos basados en este comportamiento han sido usados para producir soluciones bastante cercanas al óptimo en el problema del viajante, pero también para buscar rutas óptimas de los mensajes en redes de comunicación y para la gestión de redes de transporte urbano. En estos últimos problemas tienen cierta ventaja sobre los algoritmos evolutivos, en tanto que permiten que el grafo pueda cambiar su estructura de manera dinámica. El algoritmo de colonia de hormigas puede ejecutarse continuamente y adaptarse en tiempo real a los cambios de la red.

Los algoritmos de optimización basados en enjambres [8] trabajan con una población de soluciones candidatas, que constituyen el enjambre. Los individuos se desplazan a lo largo del espacio de búsqueda conforme a unas simples reglas matemáticas que toman en consideración la mejor posición obtenida por el individuo dentro de dicho espacio y también la media de las mejores posiciones obtenidas por el resto de los individuos del enjambre. A medida que se descubren nuevas y mejores posiciones, estas pasan a orientar los movimientos de todos los individuos. El proceso se repite con el objetivo de hallar en algún momento una solución lo suficientemente satisfactoria.

4. Algoritmos que verifican algoritmos

Estas familias de algoritmos inspirados en el cerebro humano y en la naturaleza son sin duda muy útiles y mejoran muchos aspectos de nuestras vidas, pero todos ellos tienen en común que ocasionalmente pueden dar una respuesta equivocada, o dar una respuesta peor que la óptima pretendida. Por ejemplo, un correo válido puede ocasionalmente ser clasificado como spam o viceversa; un texto puede ser incorrectamente traducido; una búsqueda en internet puede dar un resultado muy alejado del deseado por el usuario; y así sucesivamente. En la medida en que seamos conscientes de esas limitaciones, tales equivocaciones ocasionales carecen generalmente de importancia. Otra cosa distinta es confiar nuestras vidas a tales algoritmos. Por ejemplo, es sabido que en 2016 un modelo Tesla de conducción autónoma en pruebas se estrelló contra el remolque pintado de blanco de un camión porque el software del automóvil interpretó que el remolque era parte del cielo brillante. Se requiere, sin duda, más investigación en esta área, antes de que podamos delegar la conducción de automóviles a un algoritmo.

Hay sistemas controlados por software que simplemente no pueden fallar porque están en juego vidas humanas, o porque un fallo ocasionaría cuantiosas pérdidas económicas. En junio de 1996, el cohete europeo Ariane V se autodestruyó a

los 37 segundos de su primer vuelo debido a un error del software de control. La razón del error fue la reutilización, sin pruebas adicionales, de una parte del software que había funcionado perfectamente en el Ariane IV. La velocidad del Ariane V era superior a la del Ariane IV, y eso causó el desbordamiento de la capacidad de una variable en el programa que controlaba la verticalidad del cohete.

Un error en el sistema informático del Ministerio de Sanidad del Reino Unido "olvidó" entre 2009 y 2018 enviar la notificación para hacer el test final de un programa de prevención del cáncer de mama a más de 300.000 mujeres de edades comprendidas entre 70 y 79 años. Estudiando la incidencia de esta enfermedad, se estima que unos 270 fallecimientos podrían haberse evitado si el sistema hubiera funcionado correctamente.

Muy recientemente, dos aviones del modelo Boeing 737 Max han caído a tierra al poco de despegar y ocasionado centenares de víctimas, debido a un diseño incorrecto del sistema de pilotaje automático de la nave. El avión tiene el centro de gravedad más retrasado que su modelo predecesor y fue dotado de un software automático que fuerza al morro a inclinarse hacia tierra cuando detecta que el avión sube demasiado vertical. El sensor que detectaba dicha verticalidad envió información errónea y el software entendió que debía inclinar el morro hacia abajo. Aquí se pueden contabilizar al menos tres errores: el primero, no duplicar el sensor, dado que era un elemento crítico; el segundo, no informar a los pilotos, que no supieron nunca lo que estaba ocurriendo; y el tercero, no permitir desconectar el software de forma manual.

Cada vez utilizamos más sistemas como los descritos, que son críticos en seguridad. Por desgracia, también cada poco tiempo los periódicos dan cuenta de fallos debidos a errores en el software. Hoy podría ocurrir que una famosa red social haya estado inoperativa por varias horas, o que un buen número de ordenadores haya sido infectado por un virus malicioso. Mañana, que hayan quedado expuestos datos personales, o las tarjetas de crédito, de miles de personas, o que los paneles informativos de un aeropuerto se hayan quedado en blanco. Se hace cada vez más imperioso poner los medios para impedir sucesos como estos. Necesitamos métodos que garanticen que los programas producidos no fallarán en ningún caso. Y si dependen de un hardware crítico, este ha de estar duplicado, o triplicado, y el software ha de estar preparado para recuperarse de los fallos ocasionales del hardware.

Ante estas imperiosas necesidades de seguridad, inspirarse en la naturaleza es de poca ayuda y debemos volver la vista a donde empezó todo: en las ma-

temáticas. Dentro de la Ingeniería Informática, el área se denomina Métodos Formales e incluye un conjunto de materias donde las matemáticas juegan un papel primordial. En los últimos 50 años, la investigación en este área ha creado multitud de teorías y herramientas que pueden asistir en la tarea de comprobar que el software cumple estrictamente la función para la que fue creado.

Para verificar que los programas cumplen las propiedades deseadas [4], primeramente se han de formalizar las mismas, lo que exige el uso de lenguajes formales de especificación, lenguajes en los que la lógica juega un papel esencial. La tarea de verificación consiste entonces en demostrar matemáticamente que las propiedades descritas son satisfechas por el texto del programa. Los informáticos sabemos desde los comienzos de nuestra profesión que las pruebas de ejecución, el llamado *testing*, tan solo es útil para detectar la presencia de errores en el programa, pero es completamente inútil para demostrar su ausencia. La razón es que, mediante *testing*, tan solo se prueba el programa en una infinitésima parte de sus ejecuciones posibles, y por tanto, pasar con éxito unas pocas pruebas no nos informa de los errores que todavía quedan latentes en el programa. Las demostraciones matemáticas dan total garantía, pero lamentablemente exigen mucho esfuerzo humano y una alta cualificación por parte de los informáticos.

La existencia de herramientas alivian gran parte de esta tarea. Por ejemplo, existen plataformas de desarrollo [10], en las que, a la vez que se escribe el programa, la plataforma va verificando todas aquellas propiedades que puede demostrar automáticamente, o cuando no puede hacerlo (recordemos la indecidibilidad del problema de decisión), reclama al programador que introduzca evidencias de que ciertas propiedades se están cumpliendo. De este modo, el desarrollo consiste en un diálogo provechoso entre el programador y la herramienta, cuyo producto final es un programa con garantías.

Con herramientas de este tipo se han desarrollado algunos sistemas críticos. Es famosa a este respecto la línea 14 del metro de París, donde trenes sin conductor, con frecuencia de paso de dos minutos, transportan hasta 40.000 viajeros en la hora punta. Desde 1998 en que fue puesta en servicio no ha tenido ningún accidente. Otro caso famoso son los servicios web de la empresa Amazon. Esta empresa tiene centenares de servidores en todo el mundo, con bases de datos distribuidas y a la vez replicadas en varios de ellos, y soporta un tráfico de varios millones de transacciones por segundo. Se trata de un sistema concurrente de enormes dimensiones, que además ha de estar preparado para hacer frente a posibles caídas de los servidores, o de las redes de datos que los conectan, y que son más frecuentes de lo que imaginamos. Cualquier error en la sincronización de

estos programas podría ocasionar pérdidas de datos, o inconsistencias entre las distintas copias de los mismos, causando en cualquier caso enormes pérdidas económicas a la compañía. El equipo que desarrolló este software hizo un uso intensivo de los métodos formales y de algunas de sus herramientas.

5. Conclusión

Y termino dando algunos datos de cómo en la Facultad de Informática de nuestra Universidad estamos haciendo frente a los numerosos desafíos aquí descritos. En los distintos títulos de grado y de máster dedicamos un amplio espacio al estudio de los algoritmos convencionales. También ofrecemos en los grados asignaturas optativas sobre aprendizaje automático y algoritmos evolutivos. En los niveles de máster se dedican asignaturas al tratamiento y almacenamiento de grandes volúmenes de datos, así como al procesado de los mismos con técnicas de clasificación basadas en redes neuronales. Finalmente, y habiendo empezado el curso pasado, ofrecemos un máster interuniversitario con las universidades Autónoma y Politécnica de Madrid dedicado a los Métodos Formales en Ingeniería Informática, donde impartimos asignaturas sobre aprendizaje automático, algoritmos bioinspirados, computación cuántica, modelización de sistemas concurrentes y distribuidos fiables y plataformas de verificación dotadas de demostradores automáticos. Pueden estar ustedes seguros de que en la Universidad Complutense respondemos, al nivel que nos corresponde como universidad puntera española, a los retos planteados por el algoritmo.

La ingeniería informática es una ciencia todavía joven, con apenas 70 años de historia. Compárese por ejemplo con las matemáticas, la física, o la química, cuyos orígenes se remontan a centenares, y en el caso de las matemáticas, a miles de años atrás. Aún con esa juventud, sus artefactos y algoritmos, unidos a la red global que hace posible que miles de ordenadores se comuniquen entre sí, están transformando nuestro mundo a pasos cada vez más acelerados.

No sabemos lo que nos deparará el futuro y apenas somos capaces de lidiar con lo que ya nos trae el presente. No nos queda otro remedio que adaptarnos. El algoritmo seguirá evolucionando y seguirá cambiando nuestras vidas. Se trata de un proceso imparable. Y los informáticos tenemos la responsabilidad de gobernar esta evolución, para que el algoritmo nos depare los máximos beneficios y a la vez podamos disminuir al mínimo sus posibles perjuicios.

Muchas gracias.

Referencias

[1] S. C. Kleene. General recursive functions of natural numbers. Mathematical Annals, (112):727-728, 1936.

[2] A. Church. An Unsolvable Problem of Elementary Number Theory. American Journal of Mathematics, 58(2):345-363, abril 1936.

[3] C. Darwin. On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life. John Murray, 1859.

[4] E. W. Dijkstra. A Discipline of Programming. Prentice-Hall, 1976.

[5] P. Domingos. The Master Algorithm: How the quest for the ultimate learning machine will remake our world. Basic Books, New York, 2015.

[6] T. Dorigo, M. Stützle. Ant Colony Optimization. MIT Press, Cambridge, Massachusetts, 2004.

[7] R. P. Feynmann. Simulating physics with computers. International Journal of Theoretical Physics, 21(6/7):467-488, 1982.

[8] R. Kennedy, J. Eberhart. Particle Swarm Optimization. Actas de la IEEE International Conference on Neural Networks, págs. 1942-48, 1995.

[9] J. R. Kopza. Genetic Programming: On the Programming of Computers by Natural Selection. MIT Press, Cambridge, Massachusetts, 1992.

[10] K. Rustan, M. Leino. Dafny: An Automatic Program Verifier for Functional Correctness. En Edmund M. Clarke y Andrei Voronkov, editores, LPAR-16, volumen 6355 de LNCS, págs. 348-370. Springer, 2010.

[11] E. L. Post. A variant of a recursively unsolvable problem. American Mathematical Society, 52:262-269, 1946.

[12] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, 65(6):386-408, 1958.

[13] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. London Mathematical Society, 42(3):230-265, noviembre 1936.

[14] A. M. Turing. Intelligent Machinery, págs. 3-23. Edinburgh University Press, Edimburgo, 1969.