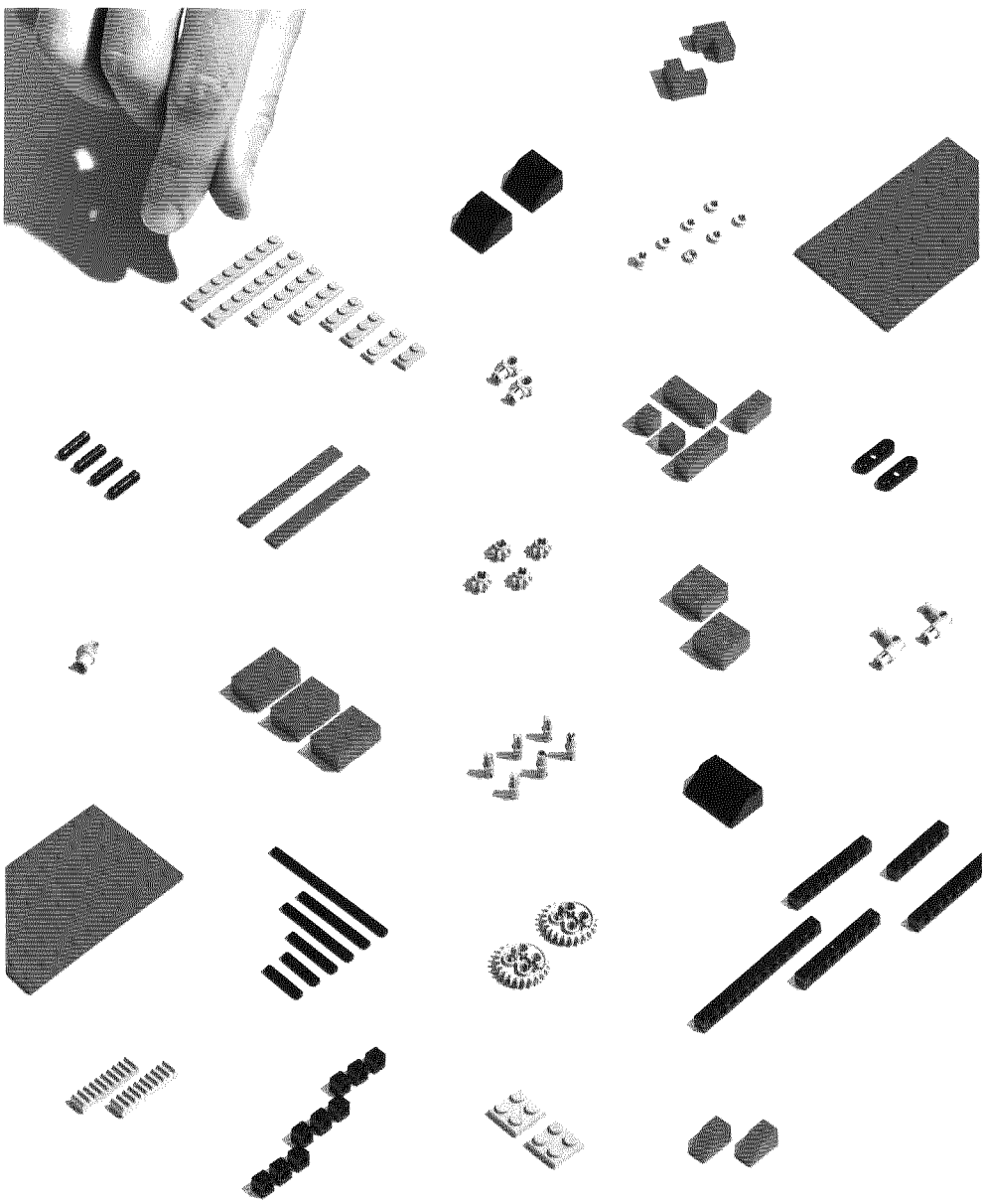
Extra

Mentre in Italia siamo ancora alle prese con l'alfabetizzazione informatica, c'è già chi parla di fine della programmazione. Per insegnare ai computer come pensare, la nuova frontiera sono le reti neurali e il *machine learning*

DI VINCENZO LATRONICO, FOTOGRAFIE DI MATTIA BALSAMINI PER IL

Oltre il codice



Negli ultimi anni la programmazione si sta imponendo come elemento cruciale non solo della formazione tecnica, ma del bagaglio culturale di ogni cittadino. Anche in Italia: la **Buona Scuola**, col progetto "Programma il futuro", vuole portarla in tutte le classi. La Open Design School, con cui Matera ha ottenuto la nomina a Capitale Europea della Cultura per il 2019, l'ha messa al centro delle sue attività. Ogni mese spunta una nuova startup che promette di rivoluzionarne i metodi di apprendimento (l'ultima, **Mosaicode**, è apparsa sulla mia *timeline* solo oggi).

Dalle pagine delle riviste, giovanissimi miliardari vegetariani la propugnano come chiave della creatività e del successo. Eppure la crescita della capacità di calcolo ha di recente reso possibile un approccio all'informatica radicalmente diverso: il *machine learning*, che idealmente potrebbe rendere obsoleta la necessità di codificare passo dopo passo i comportamenti di una macchina. I sistemi basati sul *machine learning* si stanno dimostrando tanto duttili e potenti che *Wired US* ha annunciato a caratteri cubitali, sull'ultima copertina, che «la programmazione è finita». Ma se non li programmiamo, come spieghiamo ai computer cosa fare? Semplice: glielo insegniamo. Un programma è una serie di istruzioni. Queste possono essere espresse in un linguaggio di programmazione di alto livello come **Swift** o **JavaScript** («Disegna un quadrato di 500 pixel di lato». «Unisci queste due stringhe di testo». «Stabilisci una connessione con questa porta di questo indirizzo IP») o nei termini più arcani del linguaggio-macchina in cui i linguaggi di programmazione vengono risolti per essere impartiti, come ordini, al computer. Il

principio è lo stesso: «Prima fai A, poi fai B». «Poi, se è successo C, fai D». «Altrimenti, fai di nuovo B». Questo approccio – il **pensiero computazionale** – è ciò che fa sì che la programmazione sia uno strumento intellettuale valido anche al di là della sua utilità immediata.

Programmare significa imparare a scomporre i problemi in passi elementari da eseguire in un ordine preciso. È anche il modo in cui vengono insegnate moltissime competenze ai bambini. Le addizioni in colonna, ad esempio: «Metti i due numeri uno sopra l'altro». «Somma le unità». «Segna il risultato parziale». «Segna il riporto». Se il risultato del bambino non corrisponde a quello noto all'adulto, ripercorriamo tutti i passaggi per capire cosa è andato storto.

Naturalmente, questo prevede che i problemi possano essere scomposti. Ad esempio: un programma per eliminare i fruscii da una registrazione audio dovrà prima identificare certi pattern nell'onda sonora (i fruscii), e poi compensarli modificando quell'onda o alterando il volume di altre frequenze. Per poterlo scrivere, dobbiamo essere in grado di spiegare a un computer come riconoscere un fruscio, e cioè di definirlo matematicamente in base alle sue proprietà acustiche.

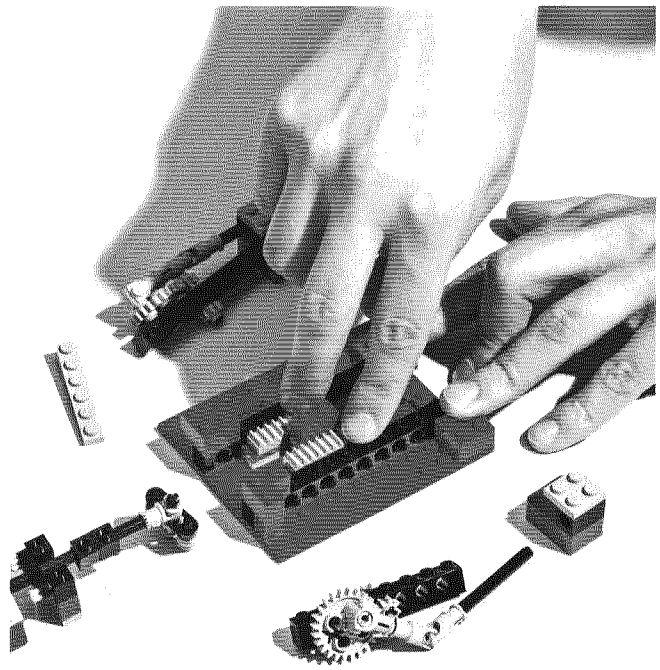
Con i fruscii è facile (più o meno). In modo simile si può scrivere un programma che riconosca i quadrati in un'immagine, o le parole scritte male in un testo. Ma se vogliamo che riconosca le foto di **Maria Callas**? Dovremmo definire oggettivamente le caratteristiche del suo volto – forme, gradienti, colori – con qualunque espressione, da qualunque angolazione e da qualunque distanza. Un compito del genere sarebbe quasi impossibile, oltre che titanico. Ma soprattutto, l'approccio ha qualcosa di controintuitivo: milioni di persone sanno riconoscere sempre una foto di Maria Callas, eppure non hanno idea di come definire le proprietà del suo volto. Ma allora come fanno?

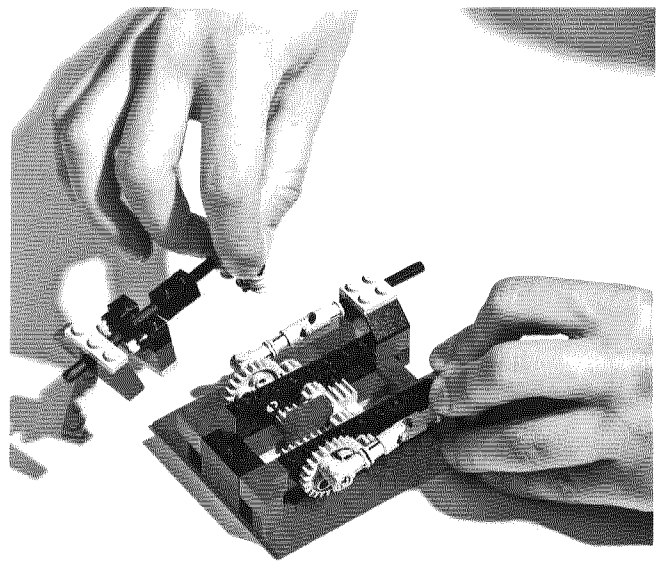
Già: come insegnamo a un bambino a riconoscere una foto di Maria Callas? Be', gliene facciamo vedere alcune, dicendogli: «Questa è lei». «Questa è lei». E poi gli mostriamo delle foto di altre donne, dicendo: «Questa no». «Questa no». Dopo alcuni esempi lo lasciamo fare da solo, correggendo quando sbaglia. A un certo punto – dopo una certa quantità di foto identificate correttamente – diremo che il bambino sa riconoscere Maria Callas.

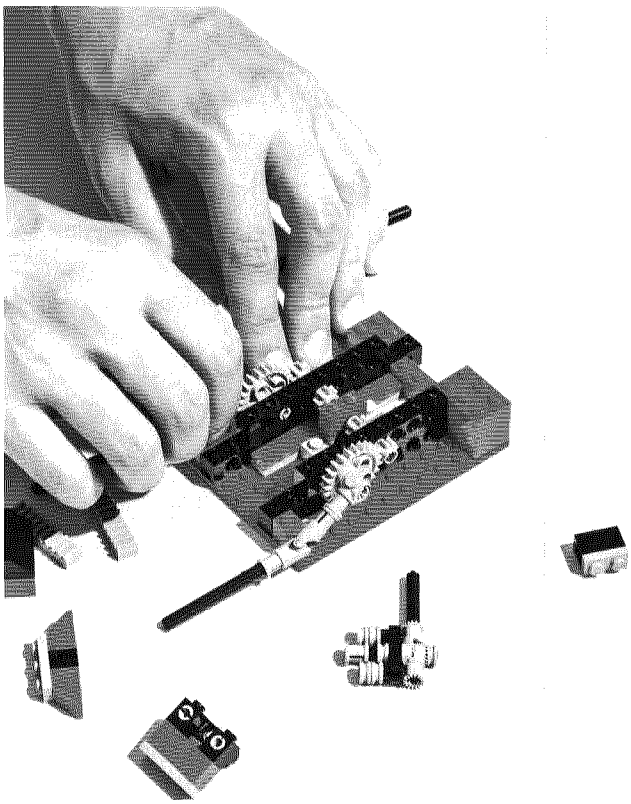
Il *machine learning* si ispira allo stesso principio. Si basa su un'architettura informatica detta **rete neurale** – una concatenazione di centri di calcolo che ricevono un input e lo trasformano in un output. Il primo input è dato dall'utente; attraversa una serie di passaggi nei nodi della rete (i "neuroni") e infine l'ultimo output viene restituito. Per programmare una rete neurale basta fornirle una grande quantità di input (foto di Maria Callas / altre foto) appaiati agli output giusti (Sì / No). Quello che succede all'input in ogni singolo passaggio (un volto scontornato, una parola eliminata) non è codificato come una serie di istruzioni: il sistema si calibra da sé, aggiustandosi in modo da produrre il risultato corretto a partire dall'input dato. Dopo un numero sufficiente di esempi, si potrà provare a vedere con un caso nuovo, se la risposta è giusta. Poi un altro. Poi un altro.

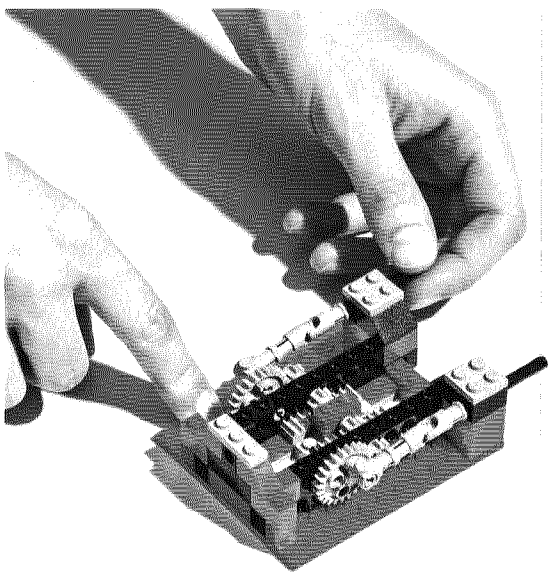
Insomma: le reti neurali non si programmano, si allenano. Idealmente, una volta impostata l'architettura di base, ciò che serve per ottenere i risultati desiderati è intuito pedagogico e pazienza, ma non è necessario (né sensato) influire con delle istruzioni di codice sull'output finale. Basta dirgli da dove partire e dove arrivare. Non serve saper programmare: il computer si programma da sé. Il primo grande successo pubblico del *machine learning* è stato solo nel marzo 2016, quando il software **AlphaGo** ha sconfitto Lee Sedol, campione di gioco del go, l'ultimo in cui gli umani conservavano la propria superiorità sulle macchine.

È interessante il paragone con Deep Blue, il computer che sconfisse il lo scacchista **Garry Kasparov** nel 1997. Deep Blue era stato programmato in modo tradizionale. Al suo turno, il software considerava tutte le possibili mosse, e le possibili risposte dell'avversario, e le possibili contro-risposte: e così via per vari livelli – si chiama "ragionamento ad albero". Poi identificava quali posizioni finali ("rami") erano desiderabili, e quali erano le strategie più probabili dell'avversario, e in base a ciò sceglieva cosa fare. La valutazione dei rami e delle strategie seguivano delle regole: dei programmatori e degli scacchisti avevano codificato degli









Programmare è un gioco da ragazzi. Nelle foto in queste pagine, abbiamo ricostruito una porta logica con i mattoncini Lego (trovate tutte le istruzioni all'indirizzo *www.randomwraith.com/logic.html*). Un gate logico è un circuito digitale in grado di realizzare una particolare operazione matematica attraverso controlli su segnali elettrici. L'operazione illustrata in queste pagine è la congiunzione (AND). È un'operazione con due variabili come input, e una come output (1 e 1 come input danno output 1; 1 e 0, 0; 0 e 1, 0; 0 e 0, 0). Un mattoncino dopo l'altro è possibile costruire tutte le operazioni alla base del linguaggio di un computer

algoritmi appositi nel software di Deep Blue (insieme a un motore di ricerca interno su un database di partite già giocate). "Allenare" Deep Blue significava guardarlo giocare e, se perdeva, capire come modificare gli algoritmi per evitare che facesse una scelta sbagliata. Negli scacchi pensare due mosse avanti significa valutare centosessantamila rami. Nel go significa valutarne **centosessanta miliardi**. Questa è una ragione che rende più complicato programmare un software per il go, ma neanche la principale. Per valutare due rami – due possibili esiti della partita – è necessario saper definire quale sia il migliore: ma neppure i campioni, spesso, sanno definire delle regole per capire se una posizione di gioco è meglio dell'altra, così come non sapremmo definire delle regole per capire se un volto è di Maria Callas. Il gioco non segue uno sviluppo lineare, come la dama cinese: fino a poco prima della fine la valutazione va a occhio, a istinto. I professionisti spesso dicono che una posizione è "più bella". È una situazione ideale per applicare il *machine learning*. E infatti AlphaGo è stato allenato con i dati di milioni di partite già giocate. Poi è stato messo a giocare contro il campione europeo, assoldato da Google. Poi è stato messo a giocare contro se stesso. Pian piano, ha imparato.

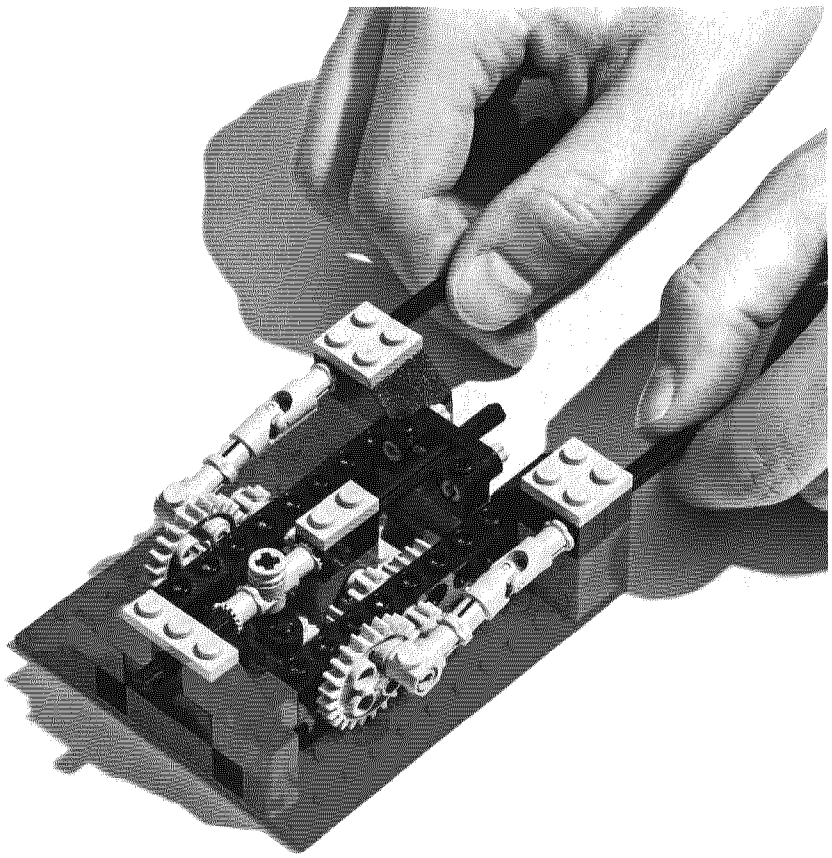
Torniamo al bambino a cui abbiamo insegnato a riconoscere Maria Callas: come facciamo a sapere che cosa ha imparato? Non possiamo. È una delle intuizioni più vertiginose di **Ludwig Wittgenstein**: noi sappiamo solo che ha identificato correttamente una certa quantità di foto. Potranno anche essere un milione, ma non abbiamo modo di sapere che non si sbaglierà alla successiva. E se si sbagliasse – se dicesse «È Maria Callas!» di fronte a una foto di **Renata Tebaldi** – non avremmo modo di dimostrargli il suo errore. Potremmo dirgli: «Guarda, il mento e il profilo dello zigomo. Sono diversi». E lui direbbe: «Anche in queste due foto della Callas – nei *Puritani* a 20 anni, nella *Tosca* a 40 – il mento e lo zigomo erano diversi». E noi diremmo: «Sì, ma alcune differenze contano, e altre no». E lui direbbe: «Quali?». E noi diremmo: «Non le sappiamo definire, ma ci sono».

Questo problema vale anche per il *machine learning*. Se ci limitiamo a fornire al computer il punto di partenza e il risultato, lasciando che trovi da sé la strada per arrivarci, poi non potremo intervenire per correggere una svolta sbagliata: perché non avremo idea di che strada sia. Si può "aprire la testa" di una rete neurale per guardarci dentro (e cioè: si può verificare, a un neurone intermedio, che cosa accade alle informazioni in transito), ma non c'è modo di capire il senso di ciò che si vede. Il computer c'è arrivato da sé. È questo che è successo quando dei ricercatori hanno trovato, in una rete neurale di Google, delle **figure multicolori**, allucinatorie. Sono girate su internet col titolo suggestivo di "sogni di un computer", ma in realtà erano i passaggi intermedi del processo che la rete aveva sviluppato per fare ciò che i ricercatori le chiedevano, e cioè riconoscere gli elementi di un'immagine. Che processo? Boh.

Lo stesso è capitato ad AlphaGo, che ha sviluppato da sé un modo per selezionare, fra i mi-

Extra

Le reti neurali potrebbero essere le componenti fondamentali di nuove architetture software. Combinando diversi strati di reti neurali, che prese singolarmente fanno cose semplici, si possono creare sistemi con comportamenti estremamente complessi. Funzionano come i mattoncini Lego: da soli dicono poco, combinati possono diventare (quasi) qualsiasi cosa. Allo stesso modo, le reti neurali combinate tra loro possono diventare software in grado di risolvere problemi complessi.
(Si ringrazia per la collaborazione l'ingegnere informatico Edoardo Pedrotti)



liardi di rami di una partita, i più promettenti (i più "belli"). Il momento di svolta della sfida è stato quando, alla trentasettesima mossa della seconda partita, AlphaGo ha piazzato una pedina in una posizione sorprendente, in una zona ancora "calma" della plancia. Sedol si è alzato ed è uscito dalla stanza per la costernazione, tornando solo alcuni minuti dopo.

A posteriori, quella si sarebbe rivelata la svolta determinante per la vittoria del computer, ma nessuno, anche adesso, a mesi di distanza, sa spiegare il ragionamento con cui ci si poteva arrivare. Certo: non c'è un ragionamento. Quella mossa non è stata decisa in base a delle regole, ma in base a un'esperienza che non si articola in istruzioni: una specie di intuito. «Non era una mossa umana», avrebbe detto poi l'allenatore della macchina.

Col crescere della capacità di calcolo, le applicazioni del *machine learning* sono sempre più pervasive ed efficaci. È un sistema perfetto per quei compiti che ci paiono irriducibili a una serie di istruzioni nella forma **se-allora**, o riducibili solo a patto di ridondanza e copiosità. Il riconoscimento facciale, la traduzione, la guida, le previsioni del tempo, le diagnosi mediche: tutti problemi in cui ci è chiaro quali sono gli input, ma non sapremmo definire con precisione in che modo vanno a determinare il risultato. Questo apre la porta a un'infinità di scenari fanta-apocalittici: chi incolpare se una rete neurale prescrive la cura sbagliata? Quale autorità può dire se un algoritmo ha imparato a guidare bene, visto che nessuno può ricostruirne i processi interni? (Il problema si è già posto nella realtà, quando Google ha preso a categorizzare foto di uomini neri come "gorilla"). Ma per ora sono problemi lontani; come lontana è l'ora della «fine della programmazione» proclamata da *Wired*.

Anche quando saranno molto più diffuse di oggi, l'architettura delle reti neurali, prima che si possano allenare, sarà comunque programmata in modo "tradizionale", così come tutte le strutture di contorno – le interfacce, le app, i database. Forse alla lunga ci sarà una ridondanza di programmatori: ma d'altronde, i programmatori stessi hanno spiegato ai librai, ai tassisti, ai difensori del copyright che all'onda della storia non si può resistere. ■