ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Can we define what an *algorithm* is?

Simone Martini

Dipartimento di Informatica – Scienza e Ingegneria

A *pre*-mathematical concept

A sequence of rules, where

each rule

the way they are composed

are "effective"

The intuitive notion is the ”touchstone" of any formal definition

Its formalisation will always loose (or add)
some *nuance* in relation to the intuitive sense

Turing, 1936

It does *not* deal with the notion of algorithm

It defines the notion of *effective*, i.e. *mechanically computable*

effective = there is an algorithm that can compute it

But for a single effective function, there are *many* distinct algorithms

Many other systems for computable functions:

- Post systems, lambda-calculus, Gödel's general recursive functions
- All programming languages: e.g. C, Java, Python

They are all equivalent to each other: the Church-Turing thesis

Equivalence between formalisms for computability

The equivalence holds "modulo coding" (*à codage près*)

Let S be a system for computability (lambda-calculus, Post systems, Python, etd.)

1. In S we may "code" the integers: [*n*]
2. For every (Turing-) computable functions over the integers *f*, there is a term F in S which "computes" *f*

$$F[n] \rightarrow [f(n)]$$

No guarantees on the preservation of algorithms!

The “parallel or”

$$por(x,y) = \begin{cases} 0 & \text{if } x \text{ terminates or } y \text{ terminates} \\ it\ does\ not\ terminate & \text{otherwise} \end{cases}$$

Computable : evaluate "in parallel" x and y

The “parallel or” in the lambda-calculus

$$por(x,y) = \begin{cases} 0 & \text{if } x \text{ terminates or } y \text{ terminates} \\ it\ does\ not\ terminate & \text{otherwise} \end{cases}$$

Computable : evaluate "in parallel" x and y

G. Berry : **there is no lambda-term P** which, for any pair of lambda terms M,N:

P N M has a normal form if and only if

(at least) one of the two terms M and N has a normal form

And yet, the lambda calculus is Turing-complete!

The “parallel or” in the lambda-calculus

$$por(x,y) = \begin{cases} 0 & \text{if } x \text{ terminates or } y \text{ terminates} \\ it\ does\ not\ terminate & \text{otherwise} \end{cases}$$

Equivalence is "modulo coding":

- Each lambda term M is coded by another lambda term [M]
- We have a term Eval which encode the procedure of reduction
- There is a term[OR]

such that

Eval [OR] [M] [N]

has a normal form if and only if one of M and N has a normal form

Algorithms and abstraction levels

An algorithm is only well defined "modulo one level of abstraction" [Gurevich]

Palindromicity of a sequence: rêver, radar, kayak

Algorithms and abstraction levels

An algorithm is only well defined "modulo one level of abstraction" [Gurevich]

Palindromicity of a sequence: rêver, radar, kayak

KAYAK

KAYAK

KAYAK

KAYAK

KAYAK

Elementary operation: “select a generic element from a sequence”

Complexity: length(’KAYAK’)/2

Algorithms and abstraction levels

Palindromicity of a sequence: rêver, radar, kayak

Elementary operation: “select a generic element from a sequence”

Complexity: length(’KAYAK’)/2

A Turing machine cannot ““select a generic element from a sequence” !

Back and forth every time!

RESSASSER

Complexity: length(’RESSASSER’)^2

Algorithms and programs

At the limit, one risks identifying algorithms and programs

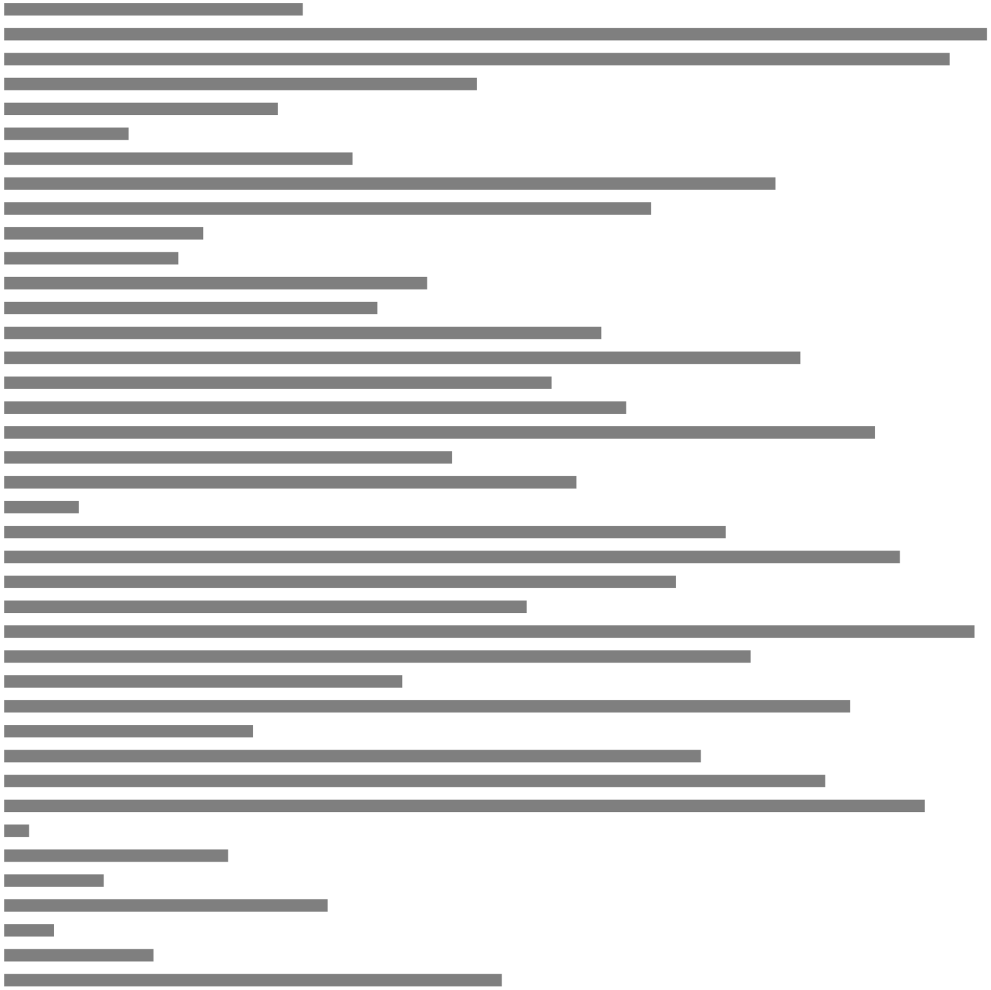
For computer science, they are different concepts:

- the generic description of a process;
- its “translation”, its ”coding", in a programming language

And yet,
two languages never have the same set of elementary operations

Algorithms and programs: *Quicksort*

Quicksort [Hoare, 1961]



Algorithms and programs: *Quicksort*

Python:

```
def QuickSort(L):
  if L==[]: return L
  pivot = L[0]
  return QuickSort([x for x in L[1:] if x < pivot])
         + [pivot] +
         QuickSort([x for x in L[1:] if x >= pivot])
```

Algorithms and programs: *Quicksort*

JAVA:

```java
public static void quickSort(int[] arr, int start, int end){
        int partition = partition(arr, start, end);
        if(partition-1>start) {
            quickSort(arr, start, partition - 1);
        }
        if(partition+1<end) {
            quickSort(arr, partition + 1, end);
        }
}
public static int partition(int[] arr, int start, int end){
        int pivot = arr[end];

        for(int i=start; i<end; i++){
            if(arr[i]<pivot){
                int temp= arr[start];
                arr[start]=arr[i];
                arr[i]=temp;
                start++;
            }
        }
        int temp = arr[start];
        arr[start] = pivot;
        arr[end] = temp;
        return start;
}
```

Python:

```python
def QuickSort(L):
  if L==[]: return L
  pivot = L[0]
  return QuickSort([x for x in L[1:] if x < pivot])
         + [pivot] +
         QuickSort([x for x in L[1:] if x >= pivot])
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Algorithms and programs: *Quicksort*

JAVA:

```
public static void quickSort(int[] arr, int start, int end){
        int partition = partition(arr, start, end);
        if(partition-1>start) {
            quickSort(arr, start, partition - 1);
        }
        if(partition+1<end) {
            quickSort(arr, partition + 1, end);
        }
}
public static int partition(int[] arr, int start, int end){
        int pivot = arr[end];

        for(int i=start; i<end; i++){
            if(arr[i]<pivot){
                int temp= arr[start];
                arr[start]=arr[i];
                arr[i]=temp;
                start++;
            }
        }
        int temp = arr[start];
        arr[start] = pivot;
        arr[end] = temp;
        return start;
}
```

Are they really two different encodings of the *same* algorithm?

Python:

```
def QuickSort(L):
  if L==[]: return L
  pivot = L[0]
  return QuickSort([x for x in L[1:] if x < pivot])
         + [pivot] +
         QuickSort([x for x in L[1:] if x >= pivot])
```

programs/algorithms of learning on neural networks

Two actors:

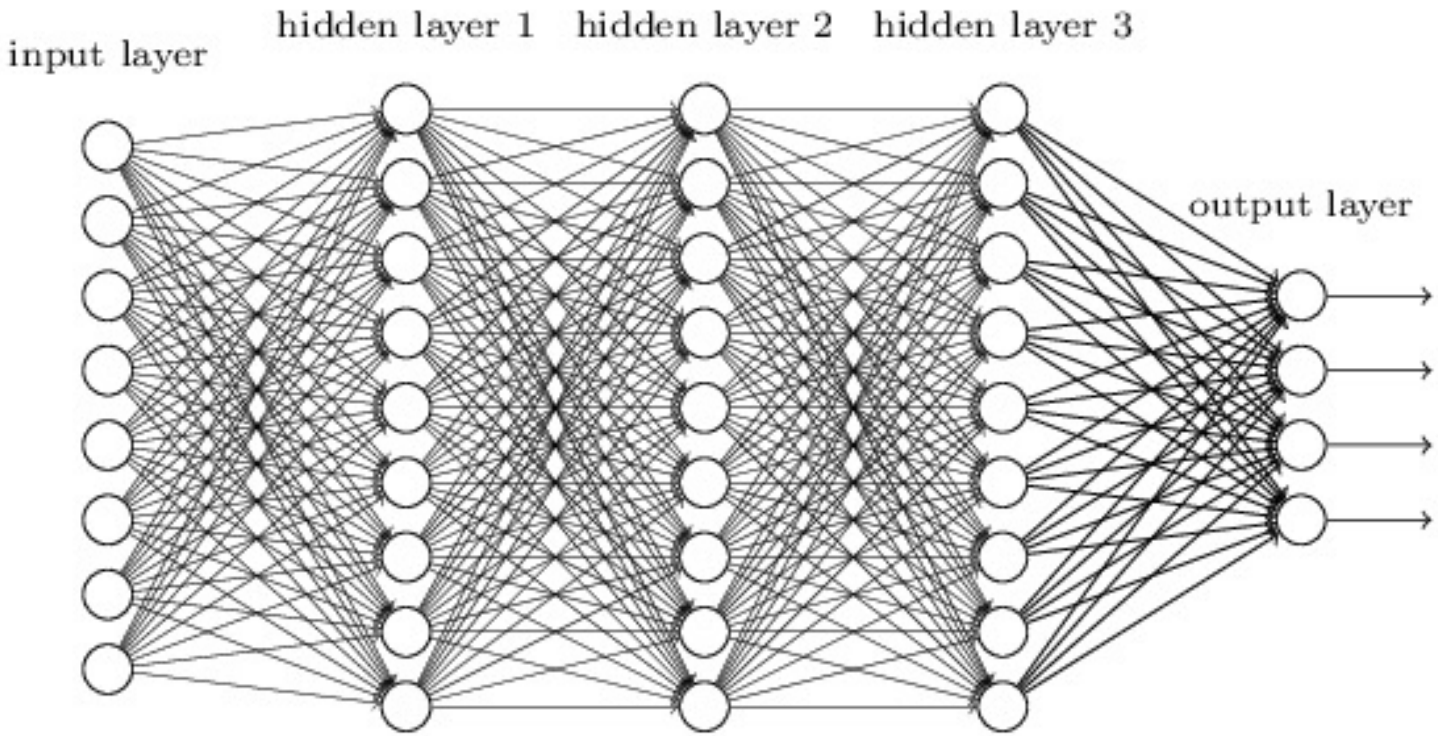
- the learning algorithm (generic)
- the learning outcome
  (the behaviour of the network, after learning)

Learning specialises (through training)
a generic network into a specific function

programs/algorithms of learning on neural networks

Two actors:

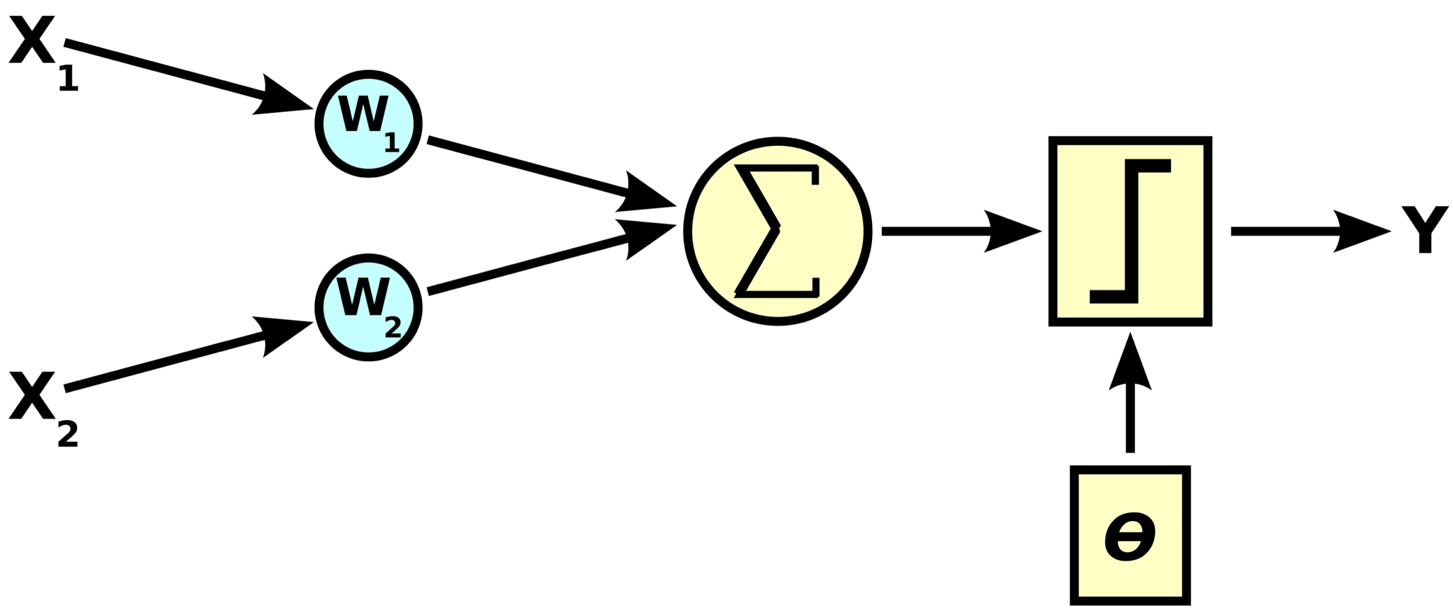
- the learning algorithm (generic)
- the learning outcome
  (the behaviour of the network, after learning)



programs/algorithms of learning on neural networks

Two actors:

- the learning algorithm (generic)
- the learning outcome
  (the behaviour of the network, after learning)



programs/algorithms of learning on neural networks

Learning specialises (through training)
a generic network into a specific function

The whole learning outcome is contained in the network *weights*

"classic" algorithm: the choices made by the algorithm are explicit
"neural" algorithm: everything is "opaque", hidden in the weights

An issue of *accountability*…

algorithms that do not terminate

Classical algorithms: termination, “the outcome”, the “result”

An operating system: an infinite loop which *does things* through interaction with the actors of computation (resources, processes, environment, people, ecc.)

Not transformation of input data into a result,
but interaction, which is a function of data, time, human actors…

the message of this lecture

An *intuitive* concept: normative, touchstone

a sequence of effective steps

There is not *a single* formalisation

of the algorithm concept

of the same algorithm

There are *various* formalisations

at different abstraction levels

each one accounts for some aspects, and loses others

the message of this lecture

the plurality of formal approaches

even, *cum grano salis*, their disagreeement

is good

how is good the pluralism of ideas in a healthy democratic society

algorithms in our daily life

- *knowledge*: the algorithm must be public!

- *responsability*: who is responsible for decisions?

- *understanding*: what skills are needed?

E.g., Maël Pégny, Issam Ibnouhsein.
Quelle transparence pour les algorithmes d’apprentissage machine?
Rev. d'Intelligence Artif. 32(4): 447-478 (2018).

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Simone Martini

Dipartimento di
Informatica – Scienza e Ingegneria

simone.martini@unibo.it