Peut-on définir ce qu'est un algorithme ?

Simone Martini

Dipartimento di Informatica – Scienza e Ingegneria

Un concept prémathématique

Une séquence de règles où soit chaque règle, soit la façon de les composer, sont « effectives »

Cette notion intuitive est la « pierre de touche »

de toutes définitions formelles

Comme les concepts « centrales » de toutes les sciences:

On peut les formaliser, mais la formalisation va toujours perdre (ou ajouter) quelque nuance par rapport au sens intuitif

Turing, 1936

Le sujet n'est pas la notion d'algorithme,

mais *une* définition de la notion d'*effective*, de *calculable mécaniquement*

effective = il y a un algorithme pour lui

Pour la même fonction effective il y a plusieurs algorithmes, pas tous exprimables dans un formalisme fixé

L'équivalence entre les différents systèmes pour la calculabilité est

« à codage près »

Equivalence entre formalismes pour la calculabilité

L'équivalence entre les différents systèmes pour la calculabilité est « à codage près »

Soit S un système pour la calculabilité (lambda-calculus, systèmes de Post, Python, ect.)

1. On a dans S une façon de « encoder » les entiers: [*n*]
2. Pour chaque fonction Turing-calculable entre les entiers *f*, il y a un terme F de S, qui « calcule » *f*

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

Aucune garantie sur la préservation des algorithmes !

Le « or parallèle »

$$por(x,y) = \begin{cases} 0 & \text{si } x \text{ termine ou } y \text{ termine} \\ ne\ termine\ pas & \text{autrement} \end{cases}$$

Calculable : évalue « en parallèle » x and y

Le « or parallèle » en lambda-calcul

$$por(x,y) = \begin{cases} 0 & \text{si } x \text{ termine ou } y \text{ termine} \\ ne\ termine\ pas & \text{autrement} \end{cases}$$

Calculable : évalue « en parallèle » x and y

G. Berry : **il n'y a aucune lambda-terme P** qui, pour chaque paire de lambda-termes M,N,

P N M a une forme normale si et seulement si

un terme entre M et N a une forme normale

Pourtant, le lambda-calcul est un formalisme Turing-complète !

Algorithmes et niveaux d'abstractions

Un algorithme est bien défini seulment « à niveau d'abstraction près»
[Gurevich]

La palindromicité d'une séquence: rêver, radar, kayak

Algorithmes et niveaux d'abstractions

Un algorithme est bien défini seulment « à niveau d'abstraction près»
[Gurevich]

La palindromicité d'une séquence: rêver, radar, kayak

KAYAK

KAYAK

KAYAK

KAYAK

KAYAK

Opération élémentaire: « prendre un élément quelconque dans une sequence »

Complexité: longueur('KAYAK')/2

Algorithmes et niveaux d'abstractions

La palindromicité d'une séquence: rêver, radar, kayak

Opération élémentaire: « prendre un élément quelconque dans une sequence »
Complexité: longueur('KAYAK')/2

La machine de Turing ne peut pas
« prendre un élément quelconque dans une sequence »!
Il faut scanner toute la séquence, en avant et en arrière chaque fois!

RESSASSER

Complexité: longueur('RESSASSER')^2

Algorithmes et programmes

Si on pousse sur ces observations, on risque d'identifier
algorithmes et programmes

Pour l'informaticien.ne ils ont deux sens bien différentes:

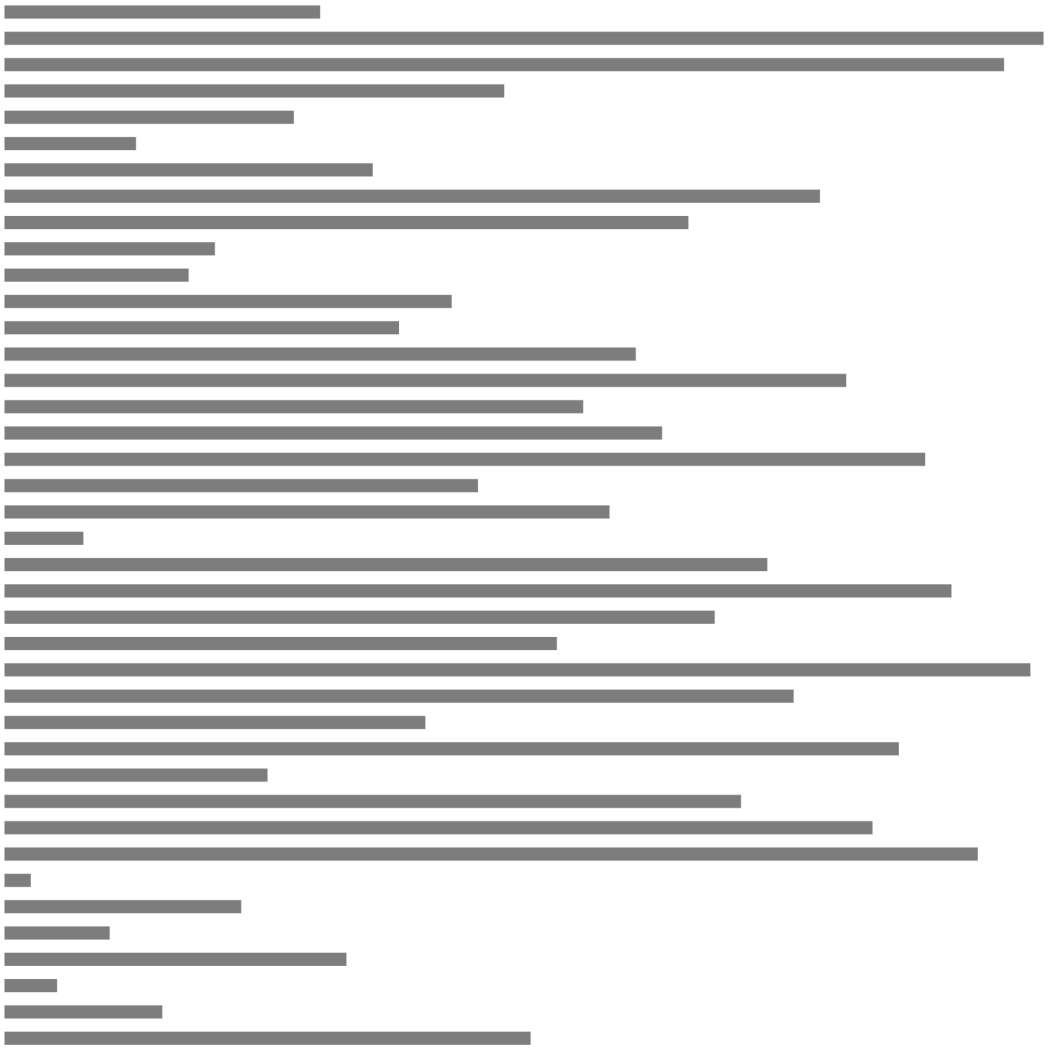
- une description générique d'un processus ;
- sa « traduction », sa « codification » dans un langage de programmation

Pourtant,

deux langages n'ont jamais le même ensemble d'opérations sur les données…

Algorithmes et programmes : *Quicksort*

Quicksort [Hoare, 1961]



Algorithmes et programmes: *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])
```

Algorithmes et programmes: *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;
}
```

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

Algorithmes et programmes: *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;
}
```

Est-ce qu'ils sont deux codages pour le *même* algorithme ?

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

programmes/algorithmes d'apprentissage neuronale

Deux acteurs :

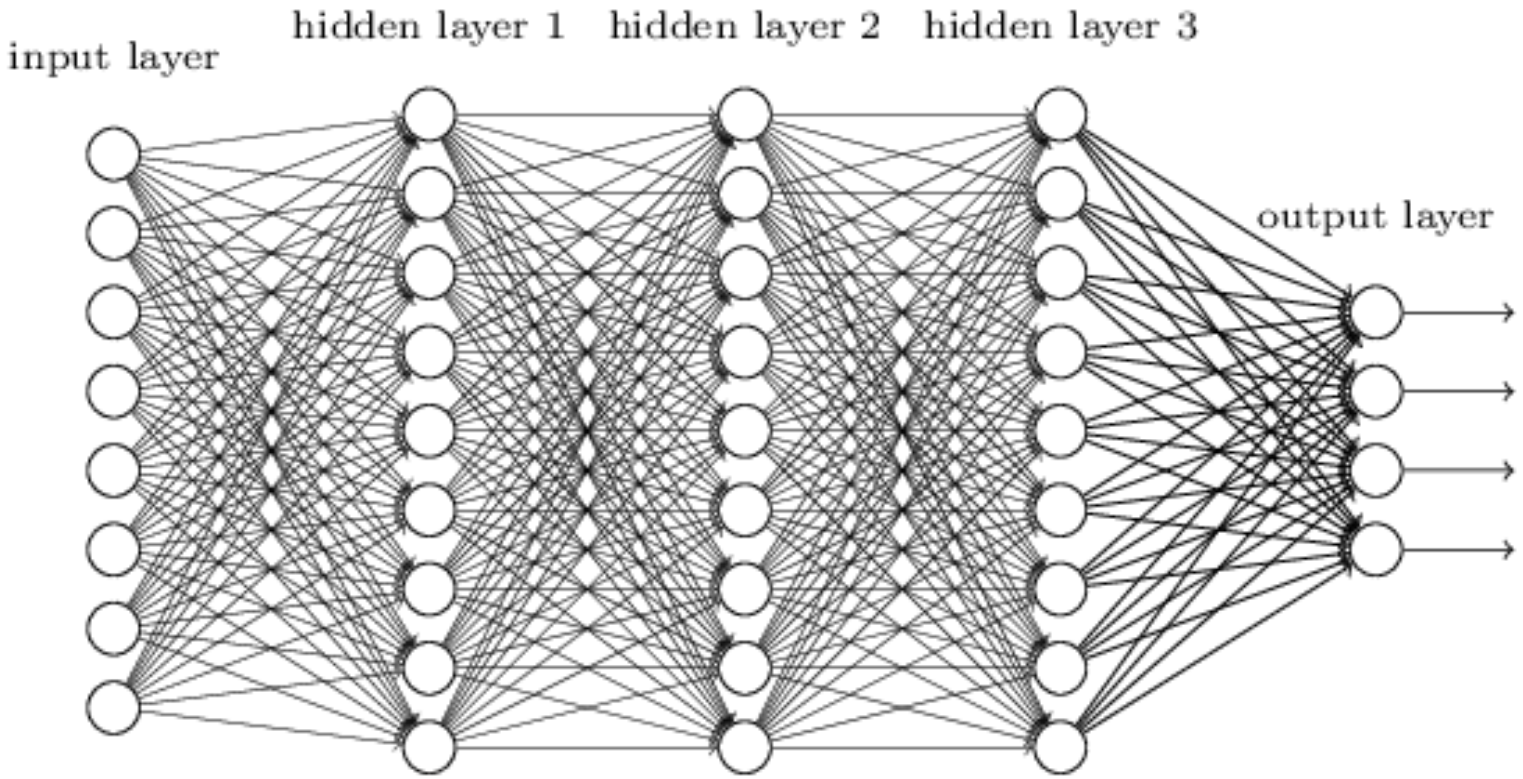
- l'algorithme d'apprentissage (générique)
- le résultat de l'apprentissage (le comportement du réseau après l'apprentissage.)

L'apprentissage spécialise (par entraînement)
un réseau générique dans une fonction particulière

programmes/algorithmes d'apprentissage neuronale

Deux acteurs :

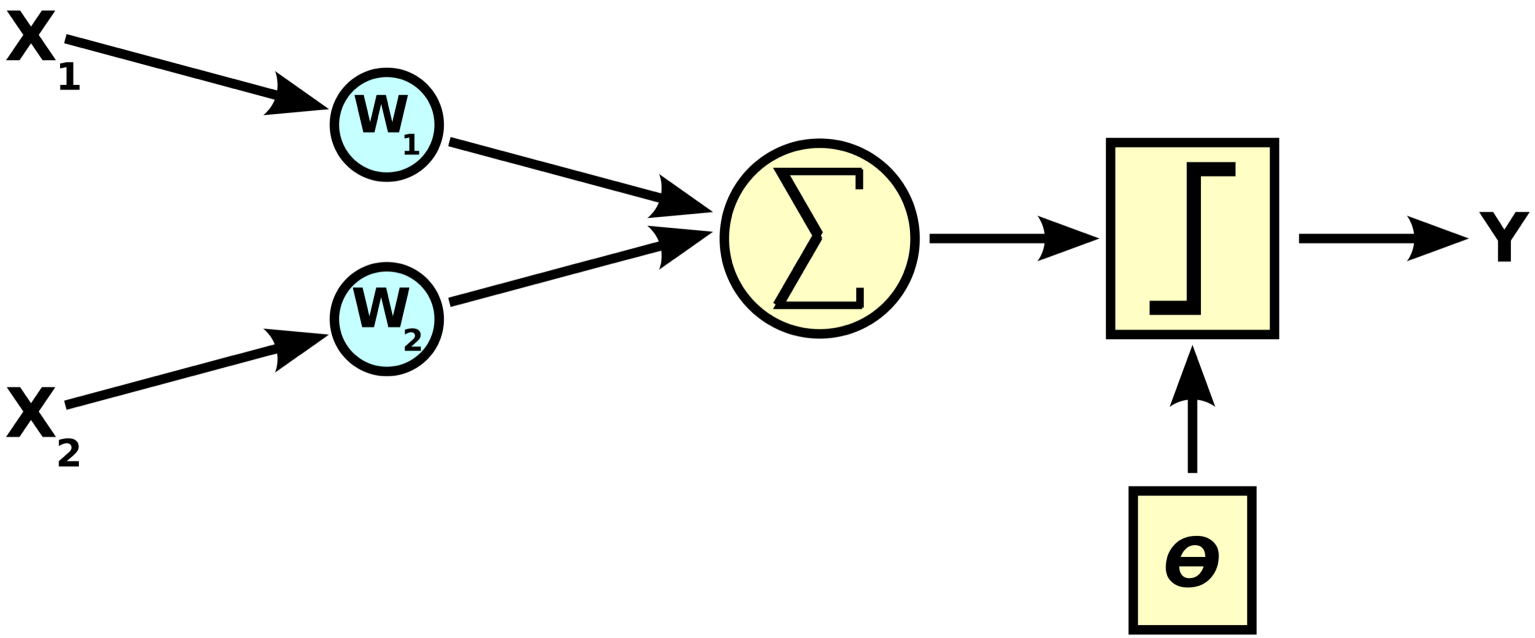
- l'algorithme d'apprentissage (générique)
- le résultat de l'apprentissage (le comportement du réseau après l'apprentissage.)



programmes/algorithmes d'apprentissage neuronale

Deux acteurs :

- l'algorithme d'apprentissage (générique)
- le résultat de l'apprentissage (le comportement du réseau après l'apprentissage.)



programmes/algorithmes d'apprentissage neuronale

L'apprentissage spécialise (par entraînement)
un réseau générique pour une fonction particulière

Tout le résultat de l'apprentissage est contenu dans le *poids* du reseau

Algorithme « classique » : on voit les choix que l'algorithme fait

« Algorithme neuronale » : tout est « opaque », caché dans les poids

Une question de *accountability…*

algorithmes qui ne terminent pas

Algorithmes classiques : terminaison, « le résultat »

Un système d'exploitation : une boucle infinie qui *fait quelque chose* par interaction avec les acteurs de la computation (ressources, processus, environnement, etc.)

Son sens n'est pas dans la transformation des données d'entrée en un résultat calculé, mais dans l'interaction, qui est fonction des données, du temps, des agents humaines…

algorithmes probabilistes

Les algorithmes dans notre vie quotidienne

- *connaissance* : il faut que l'algorithme soit public !

- *responsabilité* : qui est le responsable des décisions ?

- *compréhension* : quelles competences sont-necessaires ?

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

www.unibo.it