Are AIs Programs?

Mael Pégny

ANR/DFG Project Hypo 3 Beyond Logic
CNRS, Paris 1, IHPST

ANR “What is a program?” Launch Meeting.
Thursday February 8th 2018
Acknowledgments

Joint work with Issam Ibnouhsein (Quantmetry) on ML, algorithmic transparency and regulation.

All scientific illustrations (and many interesting remarks!) are courtesy of Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015
Problem

The question “what is a program?” is (implicitly) at the core of some recent evolutions in AI.

It has become common to say that some AIs are “not programs”, or that “AIs are not programmed, they are trained.”, “the machine essentially programs itself”, it is not “hand-coded”.

Core of this presentation: what is the implicit conception of programming behind this assertion, and why is it important?
The Importance of AI

AI, especially ML: one of the hottest topics in CS at the moment (scientific and industrial viewpoints). Recent progress in important tasks:

- Image classification
- Speech recognition
- Natural Language Processing
The Singularity of AI

Progress due to some methodological breakthroughs such as deep learning with neural networks.

Widespread feeling that we don’t understand why those methods outperform more conventional programming methods: inscrutable machine learning, blackboxness.

Assertion that AIs are not programs: attempt at grasping the singularity of those methods.
Remarks on our project

Most of the people in this project, yours truly included, are specialized in formal methods and theoretical foundations of CS: only one AI specialist (Selmer Bringsjord).

Risk of “logical bias”: giving priority to formal methods and foundational perspectives, and the objects that can be treated with those methods.

While we keep on building on our glorious logical tradition, let us not forget that the questions “what is a program?”, ”what is programming?” can be tackled from a more applied perspective.
Terminology issues

Terminology problem: "AI" is just a catchname for a set of models, most of them coming from (a part of) ML.

Those models might be opposed to some other models in AI just as well as to so-called "conventional" models.

Hard to keep track of the exact set of intended models through various discussions of inscrutable AI.
"AIs are not programs": sweeping proposition, as it opposes AI, and in particular ML, to all other branches of (conventional) programming.

Collective work needed to grasp the full scope of that proposition.

Applied and foundational perspectives are not exclusive from one another. “AIs are not programs” as a strong foundational statement: *AIs are irreducible to conventional programming methods, and they will consistently outperform them on some tasks.*

Questions the ability to represent the computing process by a “decision process”, “rule-based programming”.
Outline

1. Generic features and problems of ML
   - Building a ML model
   - The elusive singularity of ML

2. Working out an example: Deep Learning
   - Neural Networks
   - Gradient descent
   - Backpropagation

3. ’What is a program?’
   - Blackboxness and Problem-specific Understanding
   - Interpretable AI
Construction of a ML model

The conception of a ML model can be analyzed in five steps:

- Mathematical formalization of the problem
- Selection of an adapted data set, train/test split
- Selection of a model class (regressions, trees, neural networks, etc...)
- Instanciation of the model by learning on the data
- Performances tests and metaoptimization.
Parameters and hyperparameters optimization.

Distinction between **parameters** (variables modified by interactions with the data) and **hyperparameters**, which can only be modified by an independant metaoptimization.

No general methodology for metaoptimization process: sophisticated heuristics or brute-force search for optimal performance.
The singularity of ML methods: attempting a characterization

"AIs are not programs" is part of a set of propositions attempting to grasp the epistemological singularity of ML methods:

1. **Unpredictability.** The results of the execution of an instanciated machine are unpredictable.

2. **Blackboxness.** The developer of an AI does not understand the inner workings of the model.

3. **Specification.** It is impossible to give a formal specification for the computation: ML algorithms are specification-less computations.

4. **Automodification.** An AI is able to learn, i.e. to modify itself in interaction with the data.

5. **Programming.** You don’t program an AI, you train it.
Analysis of the opposition: an insufficient characterization I

All those properties are interdependent:

- Absence of specification is part of blackboxness.
- Blackboxness and automodification cause unpredictability (not the only causes).

The fifth property is a form of synthesis of the first four. It translates the experience that the machine is brought to perfection by an automatic interaction with the data:

- The machine is not improved by adding more instructions coded by a human.
- The result of the optimization, ie the behavior of the instanciated machine, is not easily understood in terms of chains of elementary decisions.
Analysis of the opposition: an insufficient characterization of ML

Playing dumb, this list of properties fails as a conceptual characterization of ML:

- **Unpredictability.** Sequential, deterministic programs written in a commercial language can have unpredictable behavior.

- **Automodification.** Self-modifying code was already an essential idea in Turing’s work. Also, something as simple as automatic updating could count as automodification.

- **Absence of formal specification.** A formal specification is not a given in practice, and proofs of programs are hard to obtain.

- **Programming.** Not applicable to other forms of programming by fiat, but its exact meaning is fuzzy, and dependent on the other properties.
Analysis of the opposition: an insufficient characterization

Unfair criticism, which ignores the originality of recent ML innovations:

- ML models are used not only when formal specification is difficult, but a mind-boggling question: ML programming is used when a complete parametrization of the problem is impossible (Doshi-Velez, 2017)

- Parameters optimization and hyperparameters metaoptimization are an innovative method, impossible to implement without a large amount of computational power.

- Instanciated models make unpredictable decisions on intuitive questions, and the reasons why they take the decisions they take often remain mysterious.
Our last remarks should NOT be construed as meaning “nothing new under the sun.”

Our current conceptual characterization of the originality of ML algorithms is insufficient because of a deep lack of theoretical understanding: need more knowledge to have a more rigorous characterization.

Are ML algorithms *fundamentally irreducible* to other programming methods, or is this a transitory state of the art?
Biological analogy

Let us try to design a computer model analog to our brain. Rosenblatt’s perceptron: fires if $\sum_j w_j x_j + b \geq 0$.

Crude model of how a neuron would fire based on the electrical signals it gets from other neurons. Also, a universal model of computation.
On the Importance of Being Deep

Adding layers allow to make “more and more abstract decisions”.

18/39
On the Importance of Being Smooth

Key property to enable the implementation of learning algorithms: a small change in weights and biases should only cause a small change in the output.

\[ \Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b \]  

(1)

If you make a mistake, we can produce a small change that will lead you "closer to the right answer."
Adding an activation function

A sudden change in value could cause a better answer for one instance, but radically alter the global behavior of the system. That’s why we want linearity.

To get linearity, we want to smooth out the behavior of the neuron by adding an activation function $f(wx + b)$.

The exact function does not matter as long as we obtain the desired behaviour.
The sigmoid function

Typically, switch to sigmoid neuron model. Smoothed version of the step function as activation function, real valued inputs and outputs:
Definition of a metrics

This should also be a smooth function: *cost function* or *loss function*. For instance, quadratic error function or MSE (with $a$ our output and $y(x)$ our target result):

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

**PROBLEM TRANSLATION**: our problem is now to minimize this function.

We can dive into the deep culture of optimization technics.
Looking for the Bottom of the Valley
Optimizing multivariate functions

If $C(v)$ is our multivariate cost function, and $v$ our position in the relevant space, we have a approximate equation (for a two variables ex.):

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2$$

(3)

Or in a denser notation: $\Delta C = \nabla C \cdot \Delta v$

Finding the optimum of a multivariate function with potentially billions of variables is an analytical nightmare.
Iterative optimization: gradient descent

Starting from a random point, we want to go down.
If we want this to be negative, we just have to compute:

$$\Delta v = -\eta \nabla C$$

(4)

with $\eta$ a small, positive hyperparameter, known as the learning rate, such that $\Delta C = -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$

Since $\|\nabla C\|^2$ is always positive, $\Delta C$ is always negative.
We thus have created an update rule for $v$:

$v \rightarrow v' = v - \eta \nabla C$
Computational strategy

All we have to do is to iterate the computation of the gradient $\nabla C$: the gradient will give us small steps in the direction that decreases $C$ the most.

Convergence towards a global minimum is not guaranteed!

Since $C$ is defined as an average over individual training examples, we have to compute every $\nabla C_x$, and average over them.
The necessity of computational power

Overall, the method is mathematically simple but computationally expansive: increase in computational power (GPU) was key in recent successes.

Gradient descent and backpropagation technics were already present in the 80s and 90s. But learning was very slow, to the point of being useless.

Since 2006, new technics allowed deepeer and larger networks to be used (5 to 10 hidden layers is common now).
Assumptions over the cost function

Our algorithm to compute the gradient, the backpropagation algorithm, adds two more assumptions on the cost function:

- **Definition as an average over the cost of individual examples:** backpropagation computes derivatives for a single example.
- **Function of the outputs of the neural network.** Expected result $y(x)$ is just a fixed parameter, since we are computing for one example.
What does it mean to understand the semantics of the program?

*an informal concept along the lines of ’what the programmer intends when they are writing the program’, an informal concept which could, in principle, be elucidated by asking the programmer what, for example, particular variable represents, or what a particular command does to a particular variable.*

(Graham White, “Hardware, Software, Humans: Truth, Fiction and Abstraction”, History and Philosophy of Logic 36(3), 278-301. 2015)

Notice that the semantics is elucidated by a passage from the general specification to the specific actions of a given piece of code. DL (current) blackboxness compromises this passage from the general to the particular.
Procedural thinking vs ML

Programming = analyzing your problem into subproblems that will be amenable to automatic resolution. Programming as **procedural thinking**.

Procedural thinking goes hand in hand with compositional semantical analysis: one tries to maintain a clear vision of the meaning of every subroutine, and how the combination of the subroutines meets the specification.

ML methods are not procedural thinking: DL programming is not based on a procedural analysis of the relevant tasks (visual object classification, speech recognition, NLP...).
“What is a program?”, “what is programming?” → “what kind of epistemological outputs does it bring me?”

A good program gives me an understanding of my initial problem, and an improved program, an improved understanding: programming gives me a **problem-specific** understanding (computational thinking).
A neural network does not solve the original problem, but its translation into an optimization problem.

DL is an efficient multivariate optimization technics: it optimizes a multivariate cost function with an efficient gradient computation.

It just turns out that our original problems can be translated into that format.
Problem-specific considerations: **input-output representation**, hypotheses on cost function definition (smooth, average, function of outputs).

Except for those elementary hypotheses, our method does not need to know anything about the intuitive meaning of the problem.

DL blackboxness: solving an intuitive problem without saying anything about the problem.
From intuitive problem to optimization and back

If by ”semantics” I mean an answer to the question ”how does this modification and weights and bias contribute to minimizing the cost function?” , then I do have a semantical understanding of the problem.

The inscrutability of ML semantics stems from translating back from the technical semantics of optimization (compute gradient, modify this parameter...) to the intuitive semantics of the original problem: how does this parameter modification contribute to solving our initial problem?
Understanding metaoptimization

On top of the problem of interpretations raised by gradient descent methods, hyperparameters optimization is also a fairly inscrutable part of ML technics. Brute force vs understanding (computational thinking).

But ML cannot be reduced to simple brute-force. Learning is not completely automatic because of metaoptimization:

- Heuristic strategies for metaoptimization (industrial added value).
- Decision that a given solution is optimal enough: **global optimality uncertain, balance between metrics.**
Interpretable AI

Blackbox understanding: fooling you to better understand you


- Low prediction scores for images that are looked hardly different from a correctly recognizable image.
- High prediction scores for unrecognizable images.

Neural networks seem to select only a couple features from the features used by human mind. It recognizes a fruit by its color, and a remote control by the rows of button, disregarding the global form of the object. It can thus be easily fooled by an image containing only that feature.
Interpretable AI

How do I get to know you?

What is program? \(\rightarrow\) what does it do? \(\rightarrow\) how do I understand what you do?

Intense research activity on *interpretable* or *explainable AI* (DARPA *Explainable AI*, Transalgo) \(\rightarrow\) developping technics to understand the semantics of ML algorithms:

- Bayesian approaches
- Local approximation by a decision tree.
- ...
Conclusion I

Ongoing, even though implicit, questioning on the nature of programs in ML.

It shows the merits of staying in touch with applied issues to discuss a fundamental problem.

**Core issues for our project:** Are ML algorithms a radically new form of programs? How does it affect our representation of the nature of programs?
ML combine optimization technics, heuristic metaoptimization and computational power to reach high performances for complex problems.

It does so without giving us a problem-specific, procedural analysis of the intuitive task, like a conventional decision procedure would.

**Area of research**: study the various technics used to make AI interpretable.