Extracted from:

Programming Machine Learning

From Zero to Deep Learning

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Programming Machine Learning

From Zero to Deep Learning

Paolo Perrotta
edited by Meghan Blanchette

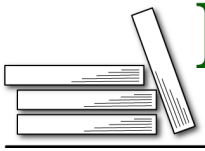
Programming Machine Learning

From Zero to Deep Learning

Paolo Perrotta

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-660-0
Book version: B1.0—March 20, 2019

Programming vs. Machine Learning

Here's an example of the difference between machine learning (or simply "ML") and regular programming. Imagine building a program that plays videogames. With traditional programming, that program might look something like this:

```
enemy = get_nearest_enemy()
if enemy.distance() < 100:
  decelerate()
  if enemy.is_shooting():
    raise_shield()
  else:
    if health() > 0.25:
      shoot()
    else:
      rotate_away_from(enemy)
else:
  # ...a lot more code
```

...and so on. Most of the code would be a big collection of `if..else` statements, mixed with imperative commands such as `shoot()`.

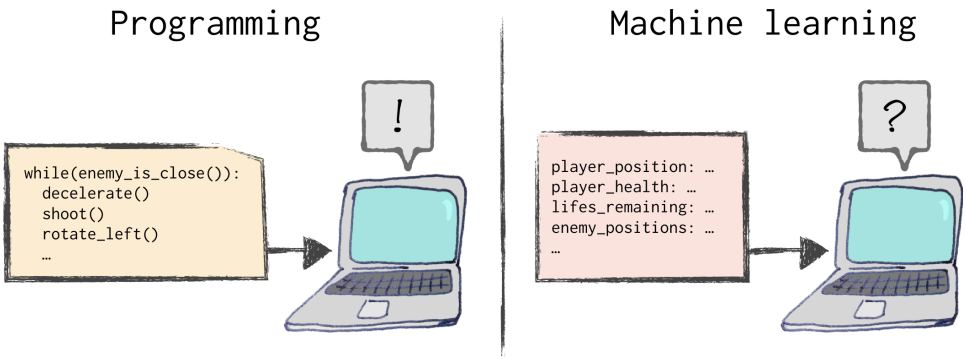
Granted, modern languages give us the means to replace those ugly nested `if`s with more pleasant constructs—polymorphism, pattern matching, or event-driven calls. The core idea of programming, however, stays the same: you tell the computer what to look for, and you tell it what to do. You must list every condition and define every action.

This approach has served us well, but it has a few flaws. First: you must be exhaustive. You can probably imagine dozens or hundreds of specific situations that you'd have to cover in that videogame-playing program. What happens if the enemy is approaching, but there is a power-up between you and the enemy, and the power-up is shielding you from enemy fire? A human player would quickly notice the situation and take advantage of it. Your program... well, it depends. If you coded for that special case, then your program will deal with it—but we know how hard it is to cover all special cases, even in structured domains like accounting. Good luck listing each and every possible special case in complex domains like playing videogames, driving a truck, or recognizing an image!

Even if you could list all those decisions, you'd have to know how to take them in the first place. That's a second limitation of programming, and a showstopper in some domains. For example, take a *computer vision* task like our original problem: identifying pneumonia in chest scans.

We don't *really* know how a human radiologist recognizes pneumonia. Yes, we have a high-level idea of it, like: "the radiologist looks for opaque areas". However, we don't know how the radiologist's brain recognizes and evaluates an opaque area. In some cases, the expert herself cannot tell you how she came to a diagnosis, except for a rather vague: "I know by experience that pneumonia doesn't look like this". Since we don't know how those decisions happen, we cannot instruct a computer to take them. That's a problem shared by all typically human tasks, such as tasting beer, or understanding a sentence.

Machine learning, on the other hand, turns traditional programming on its head: instead of giving *instructions* to the computer, ML is about giving *data* to the computer, and asking it to figure out what to do.



The idea of a computer "figuring out" anything sounds like wishful thinking, but there are actually a few different ways to make it happen. In case you're wondering, all of them still require running code. That code, however, isn't a step-by-step procedure to solve to the problem, like in traditional programming. Instead, the code in machine learning tells the computer how to crunch the data, so that the computer can solve the problem by itself.

As an example, here is one way that a computer can figure out how to play a videogame. Imagine an algorithm that learns how to play by trial and error. It starts by giving random commands: "shoot", "decelerate", "rotate", and so on. If those commands eventually lead to success, such as a higher score, the algorithm remembers this experience. If they lead to failure, such as death, the algorithm also takes note. At the same time, it also takes note of the state of the game: where are the enemies, the obstacles, and the power-ups? How much health do we have? And so on.

From then on, whenever it encounters a similar game state, the algorithm is a bit more likely to attempt the successful actions than the unsuccessful ones. After many cycles of trial and error, such a program would become a

competent player. In 2013, a system using this approach reached superhuman skills in a bunch of old Atari games.[1]

This style of ML is called *reinforcement learning*. Reinforcement learning works pretty much like dog training: you reward "good" behaviour, so that the dog does more of it.

(I also tried the same approach with my cat. So far, I failed.)

Reinforcement learning is just one way to let a computer figure out a problem. In this book, we'll focus on another style of machine learning—arguably the most popular one. Let's talk about it.

1. deepmind.com/research/publications/playing-atari-deep-reinforcement-learning/