Chapter 3
Where is My Mind?

It was like claiming that the first monkey that climbed a tree was making progress towards landing on the moon.

Dreyfus, 'A History of First Step Fallacies' [73]

The research field of AI is concerned with devising theories, methods, and workflows for producing software artifacts which behave as intelligent subjects. Evidently, intelligence, as the property of an agent, is not of necessity inherited from the methods used to construct it: that a car has been assembled by robots does not make it a robot.

Unfortunately, even this obvious distinction can sometimes be erased in some prominent published work. To wit: the statement, "an agent that performs sufficiently well on a sufficiently wide range of tasks is classified as intelligent" was recently published by DeepMind [273] to give context to a paper claiming to have developed "the first deep RL agent that outperforms the standard human benchmark on all 57 Atari games" [14]. This invites the inference that the range of the tasks (57 games) that have been achieved warrants calling the advertised agent 'intelligent'. However, careful reading of the paper reveals that the authors have in fact developed 57 different agents. Granted, this was achieved using the same development method and system architecture, but 57 agents were nonetheless trained, rather than the claimed single agent. Here is a prime example of distilled confusion: a property (applicability to 57 tasks) of one construction method (instantiating the Agent57 system architecture) has just been 'magically' transferred to some 57 artifacts produced by the method.

This only fuels what Marcus terms the "epidemic of AI misinformation" [211] from which the world has been suffering for some years [165]. As a minimum, this damages public understanding (impinging on business expectations/validation), trust (in scientific deontology), and education at large. Indeed, in common with others [75] we are of the opinion that such 'misinformation at scale' steers both governance and research the wrong way: it gives credence even among some seasoned researchers—

J. Swan et al., *The Road to General Intelligence*,
Studies in Computational Intelligence 1049,
https://doi.org/10.1007/978-3-031-08020-3_3

or worse, the next generation of researchers—to claims that machine learning is the (only) root of general intelligence, a myth we debunk in the next two chapters. But first, to give the matter a proper grounding, we must return to the roots of ML in order to objectively assess its domain of application, appreciate its evident achievements, and delineate the boundaries of its potential.

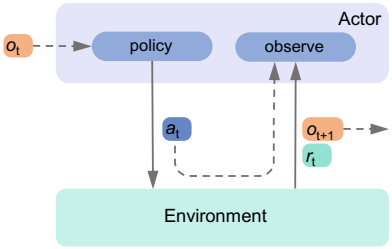
3.1 A Sanity Check

With the demise of GOFAI came a renewed effort to explore connectionist learning paradigms. Over time, the field shifted from the symbolic languages of GOFAI to 'end-to-end' feature learning. This has loosened the constraints on knowledge representation, namely moving away from an expression language with discrete symbols to something more representationally amorphous. Learning the parameters of a function approximator then becomes a fitting and regularization problem, as conceptualized by the bias-variance trade-off.

Inspired by trial-and-error learning in animals, reinforcement learning (RL) developed from work in optimal control, which is concerned with minimizing some metric of a dynamical system over time. The techniques developed for solving these problems became known as dynamic programming, and also produced the formalism of the Markov Decision Process (MDP), which is now essential to RL [331]. It provides the abstractions of states $s \in S$, actions $a \in A$, and reward function $r\colon S \times A \to \mathbb{R}$. The evolution of states is assumed to progress in discrete steps according to the transition function $P\colon S \times A \to \Delta(S)$, with a scalar reward $R_t = r(s_t, a_t)$ produced at each timestep t. These ingredients can be modified to accommodate additional complexity such as partial observability, continuous spaces, and multiple agents.

The purpose of RL is to optimize a value function based on sampled rewards which are stationary and specified a priori, in order to produce an agent (called a controller) consisting essentially of a policy that maps states to actions. It does so by shaping the policy encoded in a neural network[1] generally either via gradient descent over its weights or by 'neuro-evolution', i.e., optimizing the fitness of a policy within a population thereof. Training is unsupervised and uses for its ground truth a corpus of trials and errors logged from an number of sessions of interaction (episodes) with a simulated world—the number of episodes grows with the complexity of the task/environment distribution and is generally enormous (in other words, the sample efficiency is inordinately low). Note that, although it is possible in principle to perform trials and errors in the real world instead of in a simulator, this is rarely the case in practice, for fear of wear and tear of equipment and safety risks (on sample inefficiency and safety concerns, see Sect. 5.2).

[1] In early days, policies were encoded as mere lookup tables (as in Q-learning [208]), but this could not scale with the increasing dimensionality of the problems, hence the need to compress said policies via deep neural networks, hence 'deep RL'.




```
1: while True do
2:    t ← 0; e_t ← False
3:    Reset environment to get o_0
4:    Observe o_0
5:    while not e_t do
6:       a_t ← π(o_t)
7:       Take action a_t ; obtain step η_t = (r_t, o_{t+1}, e_{t+1})
8:       Observe (a_t, η_t)
9:       Update the policy π
10:      t ← t + 1
11:   end while
12: end while
```

Fig. 3.1 State-of-the-art high-level RL training procedure (Hoffman et al. [143]). The fact that, in practice, the procedure is implemented in sophisticated ways does not change its fundamentals

As illustrated in Fig. 3.1, the training process is synchronously coupled with the environment. The procedure it implements consists of successive sessions of interaction, each composed of a number of cycles: a cycle starts by putting the simulator on pause, invoking a subprocedure for guessing the next action based on the previous world state, and feeding the action to the simulator which responds immediately with a reward—the collected rewards are ultimately used to adjust the policy. The simulator is then resumed, it updates its state and a new cycle starts. At some point, another subprocedure decides to stop the cycle and a new session is initiated by resetting the simulator. When the controller performs well enough over selected samples of the training distribution, the procedure halts—note that the test distribution is required to be the same as the training one.

This is the basic procedure. Depending on the needs, the procedure is generally augmented with various support systems such as short-term memory, episodic memory, improved guessing subprocedures (such as intrinsic motivation heuristics and meta-learning, i.e., in-training adjustment of the guessing subprocedure), world models (model-based RL), hierarchies of policies (hierarchical RL), and so on. Regardless of its sophistication, the procedure is never learned nor performed by the controller itself: it is instead designed and performed (using a dedicated tool chain) by *human workers*. Once deployed in production, the controller matches, repeatedly, the same learned policy against sensory inputs and directs the resulting actions to its actuators ('inferencing' in ML parlance). In the main, such 'inferencing' merely amounts to applying the neural network encoding the policy to an input vector thus producing an output vector, i.e., a single mathematical operation. Regardless of how this operation is implemented, the controller is purely reactive (one could say 'blind') and does not make any situated deliberation worthy of the name, let alone taking any initiative. As Hervé Bourlard (a prominent ML researcher, head of the Swiss Idiap Research Institute) recognized recently in Forbes, "Artificial intelligence has no intelligence" [44].

In technical terms, a typical RL policy is a 'curried planner', where it is curried[2] with respect to a predefined goal/environment coupling. This stands obviously at the antipodes of what one expects from a system that learns to respond to arbitrary orders of its user in a world over which system designers have little or no control. Such a discrepancy is not contingent: it bears witness to the orthogonality of the respective ambitions of RL and general intelligence.

The purpose of RL is essentially to automate the production of software dedicated to handling a specific task in a specific environment, both defined a priori. From an engineering perspective, RL amounts to a compilation procedure. Granted, it compiles interaction logs instead of, say, C++ code, but compilation it does nonetheless ('compression' being the preferred nomenclature in ML). Assuming the resources required to build a fast[3] simulator are available (domain knowledge, funds, etc.), if one can guarantee that the world and the task will forever remain as they were during training,[4] then RL constitutes a valid cost-efficient alternative to standard engineering procedures for some classes of problems.

3.2 Real-World Machine Learning

ML has been successfully applied to automate tasks of increasing apparent complexity such as playing video games, albeit demonstrating very little with regards to real-world applications where much higher levels of complexity are the norm.

In contrast, the much less publicized application of ML to industrial automation is a far better and more rigorous exemplar of the value of ML in generating fast and accurate controllers for complex machinery in unforgiving environments. Examples abound: improving cooling in data centers, optimizing network routing for high-performance computing, optimizing chip layout, predictive maintenance, robot manipulation, digital twinning, etc. In this significant business-relevant and constrained domain, ML enjoys steady progress towards broader applicability—again, the fundamental principles are not challenged, nor do they need to be for the time being. In particular, motivated by industrial requirements, it is of critical importance to keep policy-defined behaviors within prescribed operational envelopes and this is an area of ongoing research [48, 100, 231, 270, 275, 346].

To re-emphasize: according to the expectations of the ML method, models ('policies' in case of RL) are produced in the lab according to client-provided specifications, then these models are frozen to constitute the core of deployed controllers. When the environment or task changes or whenever there is a need for improvement, new models are trained in the manufacturer's lab to replace the old ones, following

[2] https://en.wikipedia.org/wiki/Currying.

[3] Because the RL procedure is synchronous and highly sample-inefficient, simulators must be fast enough to keep the training time under manageable limits.

[4] The task is fixed and the deployment environment is drawn from the same distribution as the one used for training.

the standard procedure of software update. This is unsurprising: like any other software engineering method, ML addresses predefined controlled conditions—a claim of 'intelligence' is neither made or required; client-side engineers know what to expect.

Note that, in compliance with the continuous integration/continuous delivery paradigm (CI/CD) that pervades modern software engineering (for better or worse), it is possible to accelerate the rate of the updates. In principle, this is achieved by expanding the training distributions via the parallel aggregation of multiple data sources (as long as they are deemed, by humans in the loop, to pertain to one same task/environment), then training new policies in the background and updating the deployed controllers asynchronously.

Beyond direct application in standalone deployments, ML-generated controllers are also used as components of larger systems. For example, some of the building blocks of the Multi-level Darwinist Brain architecture (MDB) [20] consist of policies operating on world models. Reminiscent of hierarchical RL, the selection of a policy among the repertoire is itself a policy, subjected to a satisfaction model (intrinsic motivation). World models and the satisfaction model are initially given by the designers, with some guarantees that they are general enough to support a determined set of tasks and environmental conditions. Learning of policies and models is performed offline via neuro-evolution, subject to hand-crafted value functions. In that sense, an MDB controller still depends critically on the foresight of its designers, which limits its potential for autonomy. Yet it can be altered after deployment (incurring some inevitable downtime) in an incremental fashion at a cost arguably lower than that of re-engineering from scratch. This line of work culminates for example in the DREAM architecture [68]. Whereas MDB representations are tuned, ad hoc, to individual control policies, DREAM introduces a generic slow refactoring loop (called 're-representation') that continually extracts common denominators from existing representations in order to form more abstract ones to eventually facilitate transfer learning, in support of a user-defined training curriculum.

There is no contention about the value of ML as long as claims remain aligned with established principles and proven capabilities. As we have seen, whereas an agent is learned, the deployed agent does not itself learn or act autonomously. This is fully consistent with both the principles of ML and its goals and makes perfect sense from an engineering/economic point of view. What is obviously more debatable is the claim that a process designed for manufacturing purely reactive software will eventually produce thinking machines. The next two chapters will question, in depth, the plausibility of such a prophecy.