6 Neuroevolution of Behavior

An important area of neuroevolution is to construct agents that behave intelligently in a simulated or real environments. Such behavior spans several levels: At the lowest level, the neural networks optimize control tasks, such as locomotion for robots or production in bioreactors. At gradually higher levels, they optimize behavioral strategies e.g. for navigation, game play, or cognitive domains. At the very highest level, they may implement decision strategies e.g. for business, healthcare, and society in general. This chapter reviews successes and challenges in such domains, and also discusses how human expertise can be incorporated into the discovery process.

6.1 From Control To Strategy

Neuroevolution is naturally well suited for controlling agents and discovering behavioral strategies for them, in both physical and virtual environments. However, in many domains the environment can change in unexpected ways. The behavior has to adapt, sometimes by tuning existing behaviors, sometimes by deploying distinctly different behaviors at different times, and sometimes by discovering entirely new behaviors. Neuroevolution approaches to discovering such flexible behaviors, and indeed prospects for evolving generally intelligent agents, are reviewed in this section.

6.1.1 Successes and challenges

One of the most natural applications of neuroevolution is to discover effective behavior through interaction with the environment: The network receives sensor values as input, and issues control commands to effectors as output. If the network is recurrent, it can integrate inputs over time, and thus disambiguate partially observable environments. It can understand and take advantage of physical effects such as friction and momentum, remember objects that may be currently hidden from view, and so on.

For instance, in driving a simulated race car, neuroevolution discovered that it could get through curves faster by tracing a wider trajectory. This strategy is counterintuitive because such trajectories are longer; however, they allow for higher speeds which is more effective in the end. In robot-arm control, neuroevolution discovered a way to compensate for an inoperative main motor: It couldn't turn around its main (vertical axis), so it evolved instead to turn the arm away from the target, then swing it toward the target very fast, creating enough momentum to turn the entire robot around. In controlling a simulated spacecraft, when it did not have the jets to stop its forward movement, it instead turned

it around and then stopped the turn, resulting in a hard stop. In playing the Gomoku (or 5-in-a-row) board against other programs submitted into a tournament, it discovered that it could win by making a move very far away–the other programs expanded their board size to incorporate it, and crashed because they ran out of memory. There are numerous similar examples in the literature, demonstrating creative ways of controlling simulated and real robots, sometimes compensating for problems, other times achieving goals in creative ways (Fullmer and Miikkulainen 1992; Moriarty and Miikkulainen 1996; Sit and Miikkulainen 2005; Lehman et al. 2020).

Info Box: Neuroevolution at UT Austin Connectionist Models Summer School was a series of workshops organized in the late 1980s and early 1990s to promote the burgeoning field of neural networks—or connectionism, as it was then called. The 1988 version was organized at Carnegie Mellon by Dave Touretzky, Geoff Hinton and Terry Sejnowski. Some 100 students participated, including me (Risto Miikkulainen), eager to learn how to bring about a big change in AI. It was an exuberant convergence of ideas—and one of them was neuroevolution. It wasn't actually one of the topics in lectures; it was brought up in one of the breakout sessions by Mike Rudnick, a PhD student from Oregon Graduate Institute. Genetic Algorithms had gained some popularity, and Mike thought they could be used to construct neural networks as well. I was working on connectionist natural language processing then, but the idea seemed fascinating to me and I put it aside hoping to get back to it someday.

That didn't take long–in Spring 1991, during my first year as an assistant professor at UT Austin, an undergrad named Brad Fullmer wanted to do an honors thesis, and ended up evolving neural networks for an agent that roamed a virtual world and decided which objects in it were good and which were bad—launching a research direction in my lab on virtual agents that continues to this day! Brad developed a marker-based encoding technique where junk DNA could become functional later, which I think still should be explored more. Dave Moriarty, a PhD student, picked up the topic about a year later, and developed his own approach, SANE (part of an appropriately named system called Sherlock), about evolving a population of neurons, i.e. parts of a network instead of full neural networks. Dave's solution to forming full networks was to evolve network blueprints. In parallel, Tino Gomez came up with another solution, Enforced SubPopulations, i.e. evolving neurons for each location in the network separately. At the time, the ideas were separate partly so that Dave and Tino could each make a distinct contribution in their dissertations—it wasn't until 22 years later that we realized we could bring them together to evolve deep learning architectures in CoDeepNEAT!

At that time, I was ready to write a book about neuroevolution: The idea of evolving elements for a dense structure (i.e. neurons for a fully connected network) was elegant and the applications to control and behavior compelling. But a third PhD student, Ken Stanley, at about 1999 started to make noises about how the network's topology mattered as well, and that we could optimize the topology of a sparse neural network for the task. It didn't fit the paradigm, and I told him I

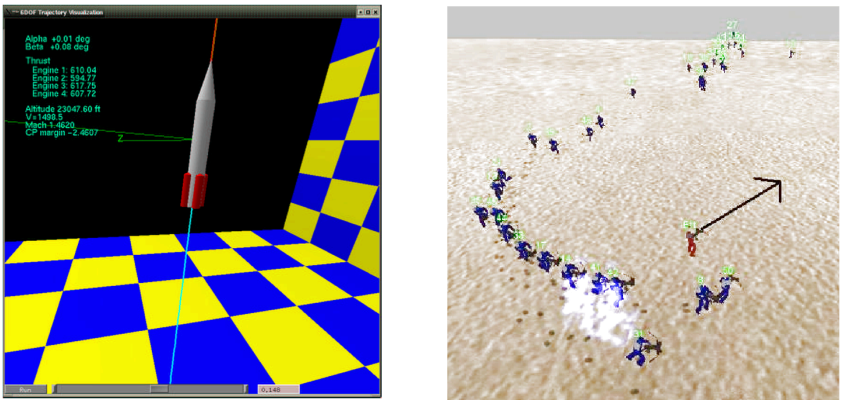
didn't think it would work—which probably only made him work on it that much harder. That idea eventually became NEAT, and one of the most enduring ideas in neuroevolution. Ken went on to build his own group at the University of Central Florida, and to develop several new ideas with students who've in turn formed their own groups in academia and industry—including a fellow named Sebastian, but that's another story.

When discussing behavior, it is often useful to separate it into two different levels. At a lower level, the challenge is to discover an effective single behavior, i.e. to devise optimal control. At a higher level, the challenge is to utilize multiple behaviors appropriately, i.e. to devise an optimal behavioral strategy. The challenges and solutions are different in the two cases.

Neuroevolution is well suited to discovering single behaviors in challenging domains, i.e. those that are dynamic, nonlinear, and noisy. For instance, in rocket control the goal is to keep the rocket flying straight, even though it is an unstable system and can easily lose stability due to atmospheric disturbances. Large rockets with multiple engines have them each on a gimble, making it possible to turn them through control algorithms, which is heavy, expensive, and difficult (indeed, rocket science). Smaller rockets instead have large fins that create enough drag at the back of the rocket to turn it into a stable system, with a cost in performance. It turns out a neurocontroller can be evolved simply to control the amount of thrust in each of the engines, and thus keep the rocket stable even without any fins at all (Figure 6.1; Gomez and Miikkulainen 2003). Such control is precise, robust, and effective, and would be difficult to design by hand.

However, by itself such control is not particularly robust. It works well within the conditions encountered during training, but it does not extend well to new conditions. Yet in the real world, such changes abound. In rocket control, the rocket parameters may vary, and weather conditions may vary; the rocket may need to fly through atmospheric disturbances. A walking robot may need to get around or over obstacles, or deal with a surface covered with water or ice. Sensors may drift or break entirely; actuators have wear and tear or may become inoperative. Coping with such variation is, of course, a major challenge for neural networks: While they interpolate well within the space of their training, they do not extrapolate well outside it.

Similar successes and challenges can be seen at higher levels of behavior as well, i.e. in discovering effective behavioral strategies. A good example is the NERO video game (Stanley, Bryant, and Miikkulainen 2005). In this game, simulated robots are engaged in a battle in a virtual world where they can sense objects, their teammates, opponents, and line of fire, and move around and shoot. The player does not control them directly, but instead has the task of training them to behave effectively in the battle. This goal means coming up with a curriculum of gradually more complex challenges, such as approaching a target, shooting accurately, avoiding fire, coordinating an attack, and coordinating a defense. The player achieves these behaviors by manipulating multiple objectives, i.e. the fitness function coefficients along several measurable dimensions of behavior. Interestingly, it is possible to design curricula that are more effective than others, in that they result in more sophisticated behavior that takes more factors into account. There also does not appear to be a single



(*a*) Rocket control (*b*) NERO video game

Figure 6.1: **Neuroevolution of effective control and behavioral strategies.** (*a*) Neuroevolution discovers a controller that can keep the rocket stable by controlling the amount of thrust to its four engines. It is accurate enough so that fins are no longer required, allowing the rocket to fly much higher with the same amount of fuel. It is, however, difficult for the controller to generalize to variations in the rocket parameters and environmental conditions. (*b*) In the NERO video game, a human player trains the agents through a curriculum of exercises to attack a target while at the same time avoiding fire from opponent agents. This is a sophisticated behavior, but a good team needs other behaviors as well, such as defending and sharpshooting, which are difficult to evolve at the same time. A challenge for neuroevolution, thus, is to discover flexible, multimodal behavior on its own, as an important step towards general intelligence. For animations of these behaviors, see https://neuroevolutionbook.com/neuroevolution-demos. Figure (*a*) from Gomez and Miikkulainen 2003; Figure (*b*) from Stanley, Bryant, and Miikkulainen 2005

strategy that always works better than others, but team A can beat B, which can beat C, which can beat A—this is precisely what makes the game interesting for a human player.

However, NERO also illustrates the limitations of the standard neuroevolution approach in discovering behavioral strategies. Throughout the evolutionary process, it elaborates on earlier behaviors and usually produces a sophisticated final behavior that subsumes all of them. However, the most successful teams in the game are composed by hand from individuals evolved separately toward different goals: sharpshooters, attackers, defenders, etc. Evolution does not spontaneously evolve agents that could deploy such very different behaviors at different times, nor a strategy for switching among them appropriately. Yet if neuroevolved agents are to be deployed in the real world, such flexible multimodal behavior is likely to be required. There are offensive and defensive modes in many games; the opponent may utilize a different strategy; the agent may be part of a team with different abilities.

Such flexibility in control and strategy is a hallmark of general intelligence. Much recent work has focused on techniques that would allow discovering and utilizing it, as will be discussed in the next three subsections.

6.1.2 Discovering robust control

Control means managing the effectors of a real or simulated agent so that it reaches its target in an effective manner. Usually, the controller observes the current state of the agent and environment through sensors (in a closed-loop or feedback control setting), and therefore can be naturally implemented in a neural network. The advantage is that such networks can deal with noise, nonlinear effects, and partial observability in a natural way. It is still challenging for them to react to changes that were not seen in training, which happens all the time in any complex environment in the real world. Therefore, several techniques have been developed to make them robust in such situations.

Perhaps the simplest way of encouraging robust control is to add noise to the outputs of the controller. Such trajectory noise means that the control does not have precisely the desired effect, but continually places the controller into situations from which it has to recover (Gomez and Miikkulainen 2004). Interestingly, trajectory noise is more effective than sensor noise in producing this effect. Apparently, adding noise to sensors may confuse the agent about what it should do, but does not similarly place it to useful training situations.

This idea can also be put to work more directly by using evolution to discover such situations automatically. For instance, if the desired actions can be specified for each situation, the controller could be trained with gradient descent. But how can the desired actions be specified? The answer is that a separate neural network can be evolved to generate them. That is, for each input situation, a teacher network generates the targets, and a controller network is trained by gradient descent to reproduce them. The teacher's fitness depends on how well the controller it trains performs in the task. How is this approach any different from evolving a network to generate good actions directly? It turns out the targets that the teacher evolves to generate do not actually correspond to optimal outputs in the task, as was demonstrated in a foraging robot domain (Nolfi and Parisi 1994a). Instead, they evolve to represent maximally effective learning experiences, i.e. those that allow learning to proceed faster and more robustly. They may be exaggerated, more varied, and more difficult situations, thereby leading to better final performance in the task.

This approach can be generalized further into a setting where problems are coevolved with solutions. For instance, a set of objective functions can be evolved for maze running, encouraging solutions that get closer to the goal, but also maximize several novel objectives. Such evolution was more effective in discovering solutions to harder mazes than fixed-fitness evolution and novelty search (Sipper, Moore, and Urbanowicz 2019). Similarly, coevolution of obstacle courses and runners results in more effective running behavior. Evolution starts with simple courses and gradually complexifies them as better runners are discovered, eventually constructing behavior that far exceeds what direct evolution could do. This system, POET (R. Wang et al. 2019b), will be described in more detail in Section 9.3. Such coevolution can also occur naturally in competitive environments, such as zebras and hyenas described in Section 7.2.2. Each species evolves to compensate for the more sophisticated strategies that the other species discovers, resulting in an arms race of more complex behaviors that would be discovered if the other species were fixed. In all these cases, neural network controllers are evolved in a task that is not fixed, but becomes more challenging as evolution progresses, automatically encouraging robust and general solutions and more complexity that can be achieved in a static setting.

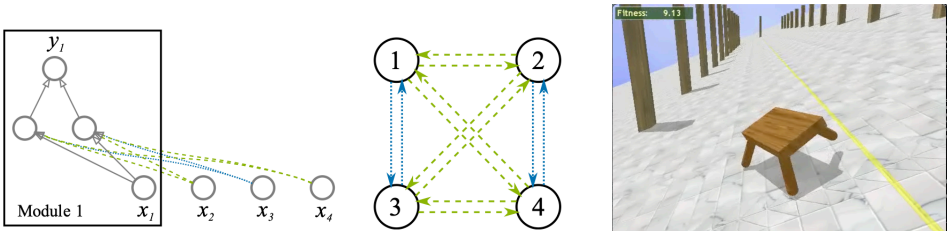
Novelty search, discussed in more detail in Section 5.3, can be seen as a related but subtly different approach. In novelty search, individual controllers are rewarded if they generate behavior that is different from that seen before during evolution. Thus, the idea is to create as much diversity as possible, and to explore the space of behaviors as completely as possible. Eventually, some individuals will be chosen as solutions because they happen to perform well in the task of interest—which is not driving novelty search directly. Importantly, the process of discovering these solutions is very different from goal-directed search. The process may include stepping stones that have little to do with the ultimate task. The solutions may thus be built on a more general and therefore robust foundation. This result was seen clearly in the bipedal walk example in Section 5.3: Whereas fitness-based evolution resulted in a rigid, slow walk that often fails, novelty search discovered a dynamic, fast walk that is remarkably robust.

In some cases we may know something about the system we are controlling, and it may be possible to take such knowledge into account in designing the network architecture that is then evolved to control it. For instance in multilegged walking, each leg should be controlled in a similar way, and there are symmetries between the left and the right side, and possibly the front and the back. These symmetries result in a number of possible gaits: For instance, four-legged animals such as horses can trot (move diagonal legs in phase), bound (move front legs in phase and back legs in phase), pace (move legs on each side in phase), and pronk (move all legs in phase). These basic gaits can then be adjusted according to the speed and terrain.

The symmetry-breaking approach can be formalized computationally in bilevel neuroevolution approach (Valsalam and Miikkulainen 2011; Valsalam et al. 2013). Each leg controller, or a module, receives the angle of the leg it controls as its input, and outputs the desired angular velocity of that leg. In addition, through intermodule connections, it receives input from all the other modules (Figure 6.2). The process starts with a population of fully symmetric individuals, where all leg controllers are identical, and they are all connected with the same intermodule connections. The connection weights are initially assigned randomly, and evolved as usual through mutation and crossover in order to find the best individuals with the current symmetry.

At the higher level, evolution then explores different symmetries. Through symmetry mutations, the initial symmetry is broken and the connections start to diverge. Some of the modules are no longer constrained to be the same, and some of the intermodule connections are no longer constrained to be the same. In this manner, evolution evaluates more symmetric solutions before evaluating less symmetric ones. This bias allows it to discover simpler and more general gaits first, and more complex ones later if they turn out necessary. Interestingly, on flat ground, highly symmetric individuals evolve that are capable of all four main gaits. Depending on how their leg positions are initialized, they may pace, trot, bound, or pronk. Also, they can dynamically switch between them. For instance, an individual may start with a bound gait, but hit a simple obstacle that prevents it from moving its legs the way it attempts–it can then switch to a trot, which moves the legs over the obstacle one at a time. Such robustness emerges automatically from the constraints of maximal symmetry among the controllers.

However, the environment may also present challenges where less symmetric solutions are required. The terrain may be cluttered with major obstacles, or slippery and inclined;



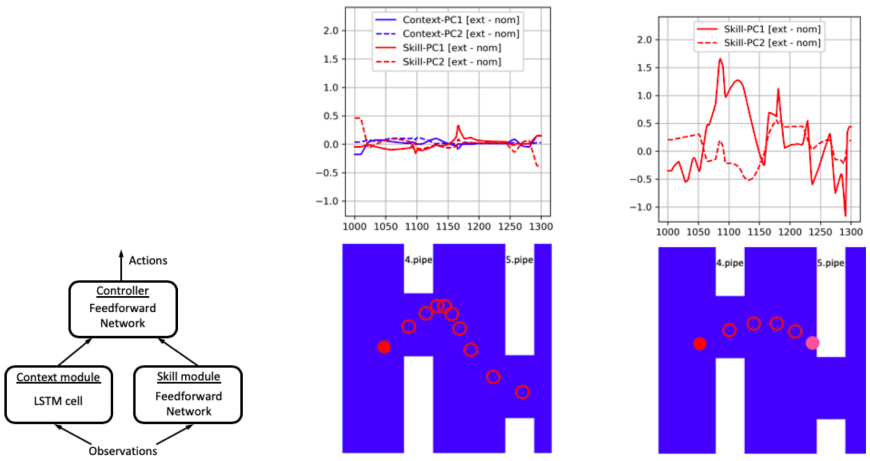
(*a*) Leg controller (*b*) Overall symmetry (*c*) Walking sideways on an incline

Figure 6.2: **Evolving symmetries for four-legged walking.** In this experiment, neuroevolution was extended to take advantage of symmetry in the four-legged robot. (*a*) Each leg has its own controller neural network, and each one receives input from the others. (*b*) Evolution starts with fully symmetric designs and breaks the symmetry as needed, i.e. allowing the weights on the different connections to diverge (as indicated by the colors). Such highly symmetric networks allow the robot to take advantage of the four main gaits on the flat ground. (*c*) A controller crossing a slippery incline requires a less symmetric solution than a straightforward walk on flat ground: It evolved to use the front downslope leg primarily to push up so that the robot could walk straight. In this manner, neuroevolution can demonstrate how principles such as symmetry help construct robust behavior. For animations of these behaviors, see https://neuroevolutionbook.com/neuroevolution-demos. (Figures (*a*) and (*b*) from Valsalam and Miikkulainen 2011)

faults may occur in the system, i.e. some legs may be damaged or inoperative and no longer move as expected. It turns out the symmetry evolution approach can discover solutions for many such cases by breaking more of the symmetry. For instance when it has to walk sideways on a slippery incline, the front downslope leg evolved a role of simply pushing the agent upwards, while the other three propelled it forward. It would be difficult to design effective gaits for such situations by hand; yet the systematic approach to understanding the symmetry of the agent and constraining evolution to take advantage of it makes it possible to discover them effectively and robustly.

Another powerful approach to dealing with variation in the environment is to model it explicitly within the controller. That is, the system consists of three neural network components: A skill network that takes actions, a context network that models the environment, and a decision network that uses the current representation of the context to modulate the actions of the skill module (Figure 6.3; Li and Miikkulainen 2018; Tutum, Abdulquddos, and Miikkulainen 2021).

This context+skill approach was first developed for opponent modeling in poker, where it resulted in a surprising ability to generalize against new opponents. When evolved to play well against only four canonical simple behaviors (always raise, always call, always fold, follow raw hand strength statistics), it was able to beat Slumbot, the best open-source poker player at the time. The skill module evolved to make reasonable actions based on the sequence in each game; the context module evolved to recognize the canonical behaviors that Slumbot used at different times; and the decision-maker evolved to adjust the actions based on the context.



(*a*) Context+skill network (*b*) Context+skill control (*c*) Skill-only control

Figure 6.3: **Modeling the environment explicitly with a context network.** In many domains, conditions can vary significantly and unexpectedly, requiring extrapolation beyond training. For instance in an extended Flappy Bird domain, the strength of the forward flap, upward flap, gravity, or drag can change. (*a*) In such settings, it can be beneficial to model the variation explicitly with a context network; the decision maker can then use the context to modulate the actions of the skill network appropriately. (*b*) The context network evolves to standardize the variation so that the decision-maker sees little of it (shown here through the first principal components of the context and skill module output over time on top, lined up with the bird's location in the bottom). It can thus perform well in a new situation, such as the decreased strength of the upward flap, or an increased drag. (*c*) Without context, the skill network outputs vary much more, making it difficult for the decision maker to generalize. In this manner, explicit understanding of the context extends the behavior robustly to variations of the domain. For animations of these behaviors, see https://neuroevolutionbook.com/neuroevolution-demos. (Figures from Tutum, Abdulqudos, and Miikkulainen 2021)

It turns out that the approach can be generalized to robust control more generally, including games such as Flappy Bird, Lunar Lander, and CARLA (simulated driving). For instance in flappy bird, it can be used to play robustly when the game conditions change. In this game, a bird flies at a constant speed through a horizontal track where it has to avoid hitting pipes that appear at constant intervals. The player takes a "flap" action to push the bird up, and gravity will pull it down constantly. Precise timing of the flap actions is required to avoid the pipes, and they have to anticipate not just the next pipe but the location of those that follow as well. In an extended version of the game, another action, a forward flap is added, causing a forward push that is constantly slowed down by drag. Different versions of the game can be generated by simply adjusting the strength of the up and forward push and the strength of gravity and drag.

It turns out that without the context module, the flappy bird controller does not generalize much at all beyond the versions seen during training, i.e. with +/-20% of variation on the four parameters. As is usual in neural networks, the controller can interpolate between

situations it has seen before, but cannot handle situations that would require extrapolation. With context, however, it can fly robustly in conditions that vary +/- 75%, i.e. in conditions that require significant extrapolation.

It is interesting to analyze how context modulation achieves such robustness. One might expect that the context network outputs change significantly in new situations, making it possible for the decision-maker to modulate the skill network's actions accordingly. However, the opposite is actually true: The outputs of the context and skill actually change very little, requiring very little new behavior from the decision-maker. In effect, the context network evolved to standardize the different situations and map them to a limited range where the actions are known. Such a principled understanding of the domain extends to a much broader range of conditions, and therefore leads to extrapolation.

The context+skill approach can also be useful in coping with environments that change. As will be discussed in Section 6.1.3, the real world is rarely constant, but instead, there are changes due to outside factors, wear and tear in the mechanics, noise and drift in the sensors, and so on. The context module can learn to anticipate such changes and modulate the skill module accordingly. For instance in the gas sensor drift domain (Warner, Devaraj, and Miikkulainen 2024), it learned the direction and magnitude of such changes over time, allowing it to classify future examples significantly more accurately than a model that was simply trained to be as general as possible.

Changes in the environment may not always be predictable over time and may exceed the generalization ability of the controller networks. In such cases, some kind of rapid online adaptation may be necessary. However, neuroevolution is usually applied as an offline method, i.e. the controllers are evolved during a training period ahead of time and then deployed in the application. Further adaptation would then require another period of offline evolution. Continuing evolution during deployment is difficult because it creates many candidates that are not viable. Indeed the exploratory power of evolution, which is its greatest strength, makes it difficult to apply it online, where every performance evaluation counts. Historically, this was the main difference between reinforcement learning, which was intended as an online lifelong learning method, and evolutionary computation, which was an offline engineering approach. This difference has blurred recently: Many reinforcement learning approaches are now offline—and similarly, there are versions of neuroevolution that can work online (Section 8.1; Agogino, Stanley, and Miikkulainen 2000; Metzen et al. 2008; Cardamone, Loiacono, and Lanzi 2009; Silva et al. 2015).

For instance, once the initial neurocontrollers have been evolved offline, they can be refined online using particle swarming (PSO; Gad 2022; Kennedy and Eberhart 2001). PSO is loosely based on the movement of swarms such as birds or insects. A population is generated around a well-performing individual, and changes made to each individual by combining its own velocity (i.e. history of changes) with that of the best individuals in the population. PSO therefore provides a way to find local optima accurately. Combining a GA and PSO thus provides for both exploration and exploitation: GA can make large changes to the solutions, discovering diverse approaches and novelty, and PSO can refine them through local search. Such combinations of global and local and search, or memetic algorithms, are useful in neuroevolution in general, including neural architecture search

(Lorenzo et al. 2017; Lorenzo and Nalepa 2018; ElSaid et al. 2023). They can also implement online adaptation: Assuming the changes in the environment are gradual, they can create alternative solutions that still perform well but also track the changing requirements.

For instance in the bioreactor control domain, micro-organisms grow by consuming a nutrient substrate which is continuously fed into the reactor. The growth process is dynamic, nonlinear, and varies unpredictably. The best production is achieved close to the maximum liquid level of the reactor; however, this level must not be exceeded, otherwise the reactor needs to be shut down. While the initial controllers constructed through neuroevolution were able to keep the reactor operational, fine-tuning through PSO improved the production significantly. When changes were introduced into the simulation, online adaptation through PSO was able to keep the operation safe, while still tracking the economic optimum closely (van Eck Conradie, Miikkulainen, and Aldrich 2002b, 2002a). In this manner, online adaptation can be used to add robustness to the control that would be difficult to achieve otherwise.

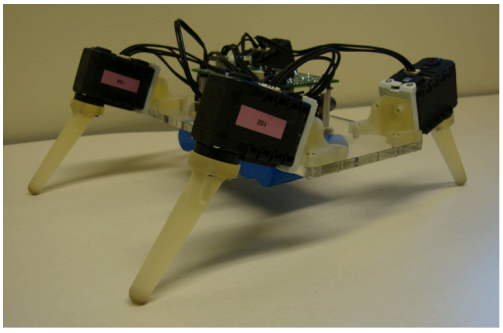
Thus, neuroevolution can naturally deal with noisy and nonlinear domains, and there are many ways to make it robust when the domain varies significantly. But are such solutions robust enough to cope with variation in the physical world? This question will be addressed next.

6.1.3 Transfer to physical robots

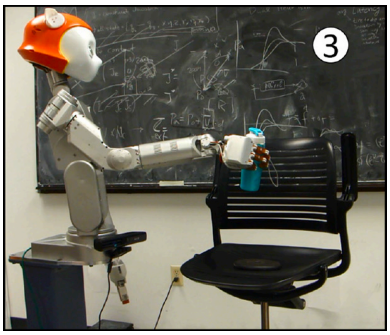
There is generally a reality gap between simulation and physical reality: Simulations are clean and deterministic, and real world is noisy, nondeterministic, includes external factors that are not part of the simulation, there's give and wear and tear in the wheels and motors, etc. As a matter of fact, the robotics community is often not very impressed even with very impressive simulation results, and justifiably so.

However, neuroevolution is in a good position to make transfer to real robots possible. By its very nature, controllers are evolved to cope with imperfections, and even take advantage of them, as was seen in the robot with an inoperative main motor in Section 6.1.1. A similar result was obtained in the four-legged walking domain (Valsalam et al. 2013). An actual physical four-legged robot was constructed with a similar structure to the simulations. Its four legs were each angled away from the center and rotated around a circle, thus each propelling it forward with a slight angle (Figure 6.4*a*). Such a gait made it possible to walk forward as well as turn at will. Most remarkably, when one of the legs became inoperative, an asymmetric gait evolved where the remaining leg on the same side traced a wider arc than the two on the other, allowing the robot to still walk straight. Thus, not only did the neuroevolution approach transfer to physical robots, it also came up with a solution to a situation that would have been very difficult to design by hand.

If transfer to the physical world is anticipated, the simulation can be extended with mechanisms that simulate the physical challenges. For instance, factors such as wind, variable friction, and uneven terrain can be programmed into the simulation. However, it is more difficult to simulate all possible imperfections that might occur, such as slippage, blocked sensors, loose connections, battery drainage, and wear and tear. One way to deal with such issues is to add noise and stochastic blockage to the simulated sensors and effectors. Both kind of noise allow simulating the world more realistically. As mentioned above, effector (or trajectory) noise also allows training the controller in more varied situations.



(*a*) Physical four-legged robot



(*b*) Dreamer robot with Mekahand

Figure 6.4: **Transfering control to physical robots.** In these two examples, the controller neural network is evolved in simulation and then used to control the corresponding physical robot. (*a*) A four-legged physical robot evolved to walk straight even with one leg inoperative. (*b*) An accurate simulator of a robotic arm was used to evolve controllers that generalize well to new situations and imprecise computation. In this manner, it is not only possible to transfer to physical robots, but also construct controllers that are robust against noise, faults, and new situations. [Figure (*a*) from Valsalam et al. 2013 and Figure (*b*) from Huang et al. 2019.] For an animation of the four-legged robot, see https://neuroevolutionbook.com/neuroevolution-demos

Recently, robotics simulators have become accurate enough to support transfer in many cases. For instance in robotic grasping, it is possible to evolve a neural network controller and transfer it into the physical robot as is (Huang et al. 2019). NEAT was used with the Graspit! simulator and transferred to the Dreamer robot's Mekahand (Figure 6.4*b*). The resulting controller was surprisingly robust, coping with sensor and effector inaccuracies as well as novel objects well. Most interestingly, it was robust against imprecise computation: When the grasping had to be completed very fast, only approximate information about the process was available, yet the controller managed to grasp the object safely in most cases.

Even though neuroevolution of behavior mostly focuses on virtual agents, much if it actually originates from robotics. The field of evolutionary robotics emerged in the 1990s and continues to this day (J. C. Bongard 2013; Doncieux et al. 2015; Vargas et al. 2014; Nolfi and Floreano 2004). The controllers and sometimes also the hardware is evolved, and often the controllers are simple neural networks. The original motivation was that robot control is difficult to design by hand, and can be more readily done through neuroevolution (Cliff, Harvey, and Husbands 1993). Simulations are often a useful tool, however, it is also possible to evolve the controllers directly on robotic hardware. For instance, recurrent discrete-time neural networks were evolved on the Khepera miniature mobile robot to develop a homing behavior (Figure 6.5*a*; Floreano and Mondada 1996a). The network developed an internal topographic map that allowed it to navigate to the battery charger with minimal energy simply in order to survive.

An interesting direction is to evolve both the controllers and hardware at the same time. Indeed, such coevolution can facilitate evolution of more complex and robust solutions

(J. Bongard 2011). For instance in evolving locomotion, the robots may start with an eel-like body plan and gradually lose it in favor of a legged design. The gaits on robots that go through such a process can be more robust than those evolved on the legged design directly. To make morphological innovations feasible, it may be useful to protect them by temporarily reducing evolutionary selection pressure (Cheney et al. 2018). Such protection is a useful general principle in discovering complexity, similar to speciation in NEAT (Section 3.4).

The most extreme demonstration of this approach is GOLEM (genetically organized life-like electromechanics;Figure 6.5*b* Lipson and Pollack 2000). Not only were the hardware designs and the neural network controllers coevolved, but the robots themselves were 3-D printed according to the evolved designs. The designs were evaluated for their locomotive ability in simulation. The best ones were then printed and evaluated in the physical world, and found to perform as expected. The evolved virtual creatures (Dan Lessin, Don Fussell, and Miikkulainen 2014; Daniel Lessin, Donald Fussell, and Miikkulainen 2013) discussed in Section 14.5 extend this approach to more complex morphologies and behaviors, all the way to fight-or-flight, albeit in simulation and with a hand-constructed syllabus. However, it is possible to imagine a future where robot bodies and brains are coevolved automatically, the results created on multimaterial 3D printers—and once the printing is finished, the robots wake up and walk off the printer on their own.

Evolutionary robotics has already been scaled up to swarms, i.e. robot teams that exhibit collective behavior (Dorigo, Theraulaz, and Trianni 2021; Trianni et al. 2014). The challenge in this area is to evolve the swarm to perform tasks that single robots could not. For instance, such robots can hook up and form a linear train that can get over obstacles and gaps that a single robot could not (Figure 6.5*c*). Many interesting issues come up in evolving neural controllers for such robots. For instance, should they all be clones of each other, or each evolved to fill a specific role in the team? Collective behavior in general is an important area of neuroevolution, discussed in depth in Chapter 7.

6.1.4 Discovering flexible strategies

The neuroevolved solutions so far have focused on control. At this level, adaptation most often means modulating or adjusting a single existing behavior: Throttle one of the engines a little more, move one leg a little faster, flap a little harder. When behavior extends from such low-level control to a high-level strategy, goal-driven coordination of multiple behaviors is required. For instance, offensive vs. defensive play in robotic soccer may require getting open vs. covering an opponent; actions required of a household robot are very different when it is vacuuming vs. emptying the dishwasher vs. folding laundry; game agents may need to gather resources, attack, and escape.

Evolving high-level strategies is challenging not only because the agent must have command of a much larger repertoire of behaviors, but it also needs to know when and how to switch between them. Proper switching is difficult for two reasons: first, in some cases it may have to be abrupt, i.e. small changes in the environment may require drastically different actions; second, sometimes the different strategies need to be interleaved or blended instead of making a clean switch.

The first challenge can be illustrated e.g. in the half-field soccer domain, where five offenders try to score on five defenders, using eight behaviors: getting open and intercepting the ball, and holding the ball, shooting at the goal, and passing it to one of the four