Chapter 4
Evolutionary Function Approximation

The methods presented in Chapter 3 allow the representation-learning capacity of evolutionary algorithms like NEAT to be harnessed in both off-line and on-line scenarios. However, that capacity is still limited in scope to policy search methods. Hence, Sutton and Barto’s criticism (that policy search methods, unlike temporal difference methods, do not exploit the specific structure of the reinforcement learning problem) still applies. To address this problem, we need methods that can optimize representations, not just for policies, but value function approximators trained with temporal difference methods.

At present, temporal difference methods typically require a human designer to manually design an appropriate representation for the function approximator. Poor design choices can result in estimates that diverge from the optimal value function (13) and agents that perform poorly. Even for methods with guaranteed convergence (14, 76), achieving high performance in practice requires finding an appropriate representation for the function approximator. As Lagoudakis and Parr observe:

The crucial factor for a successful approximate algorithm is the choice of the parametric approximation architecture(s) and the choice of the projection (parameter adjustment) method (76, p. 1111).

Nonetheless, representational choices are typically made manually, based only on the designer’s intuition.

This chapter introduces evolutionary function approximation (161), a new approach to TD function approximation which harnesses the representation-learning power of evolutionary methods. This approach synthesizes evolutionary and TD methods into a single method that automatically selects function approximator representations that enable efficient individual learning. When evolutionary methods are applied to reinforcement learning problems, they typically evolve a population of action selectors, each of which remains fixed during its fitness evaluation. The central insight behind evolutionary function approximation is that, if evolution is directed to evolve value functions instead, then those value functions can be updated, using TD methods, during each fitness evaluation. In this way, the system can evolve function approximators that are better able to learn via TD. This
biologically intuitive combination has been applied to computational systems in the past (61; 2; 25; 50; 56; 106) but never, to our knowledge, to aid the discovery of good temporal difference function approximators.

This approach requires only 1) an evolutionary algorithm capable of optimizing representations from a class of functions and 2) a TD method that uses elements of that class for function approximation. This book focuses on performing evolutionary function approximation with neural networks. There are several reasons for this choice. First, they have great experimental value. Nonlinear function approximators are often the most challenging to use; hence, success for evolutionary function approximation with neural networks is good reason to hope for success with linear methods too. Second, neural networks have great potential for function approximation, since they can represent value functions linear methods cannot (given the same basis functions). Finally, employing neural networks is feasible because they have previously succeeded as TD function approximators (38; 151) and sophisticated methods for optimizing their representations (57; 137) already exist.

In addition to automating the search for effective representations, evolutionary function approximation can enable synergistic effects between evolution and learning. How these effects occur depends on which of two possible approaches is employed. The first possibility is a Lamarckian approach, in which the changes made by TD during a given generation are written back into the original genomes, which are then used to breed a new population. The second possibility is a Darwinian implementation, in which the changes made by TD are discarded and the new population is bred from the original genomes, as they were at birth.

It has long since been determined that biological systems are Darwinian, not Lamarckian. However, it remains unclear which approach is better computationally, despite substantial research (110; 168; 171). The potential advantage of Lamarckian evolution is obvious: it prevents each generation from having to repeat the same learning. However, Darwinian evolution can be advantageous because it enables each generation to reproduce the genomes that led to success in the previous generation, rather than relying on altered versions that may not thrive under continued alteration. Furthermore, in a Darwinian system, the learning conducted by previous generations can be indirectly recorded in a population’s genomes via a phenomenon called the Baldwin Effect (15), which has been demonstrated in evolutionary computation (61; 2; 25; 9). The Baldwin Effect occurs in two stages. In the first stage, the learning performed by individuals during their lifetimes speeds evolution, because each individual does not have to be exactly right at birth; it need only be in the right neighborhood and learning can adjust it accordingly. In the second stage, those behaviors that were previously learned during individuals’ lifetimes become known at birth. This stage occurs because individuals that possess adaptive behaviors at birth have higher overall fitness and are favored by evolution.

Hence, synergistic effects between evolution and learning are possible regardless of which implementation is used. In Section 4.2.4, we compare the two approaches empirically. The following section details NEAT+Q, the implementation of evolutionary function approximation used in our experiments.