In Encyclopedia of Machine Learning, New York, 2010. Springer.

Neuroevolution

Risto Miikkulainen
The University of Texas at Austin

1 Definition

Neuroevolution is a method for modifying neural network weights, topologies, or ensembles in order to learn a specific task. Evolutionary computation is used to search for network parameters that maximize a fitness function that measures performance in the task. Compared to other neural network learning methods, neuroevolution is highly general, allowing learning without explicit targets, with nondifferentiable activation functions, and with recurrent networks. It can also be combined with standard neural network learning to e.g. model biological adaptation. Neuroevolution can also be seen as a policy search method for reinforcement-learning problems, where it is well suited to continuous domains and to domains where the state is only partially observable.

2 Synonyms

Evolving neural networks, genetic neural networks

3 Motivation and Background

The primary motivation for neuroevolution is to be able to train neural networks in sequential decision tasks with sparse reinforcement information. Most neural network learning is concerned with supervised tasks, where the desired behavior is described in terms of a corpus of input-output examples. However, many learning tasks in the real world do not lend themselves to the supervised learning approach. For example, in game playing, vehicle control, and robotics, the optimal actions at each point in time are not always known; only after performing several actions it is possible to get information about how well they worked, such as winning or losing the game. Neuroevolution makes it possible to find a neural network that optimizes behavior given only such sparse information about how well the networks are doing, without direct information about what exactly they should be doing.

The main benefit of neuroevolution compared to other reinforcement learning (RL) methods in such tasks is that it allows representing continuous state and action spaces and disambiguating hidden states naturally. Network activations are continuous, and the network generalizes well between continuous values, largely avoiding the state explosion problem that plagues many reinforcement-learning approaches. Recurrent networks can encode memories of past states and actions, making it possible to learn in partially observable Markov decision process (POMDP) environments that are difficult for many RL approaches.

Compared to other neural network learning methods, neuroevolution is highly general. As long as the performance of the networks can be evaluated over time, and the behavior of the network can be modified

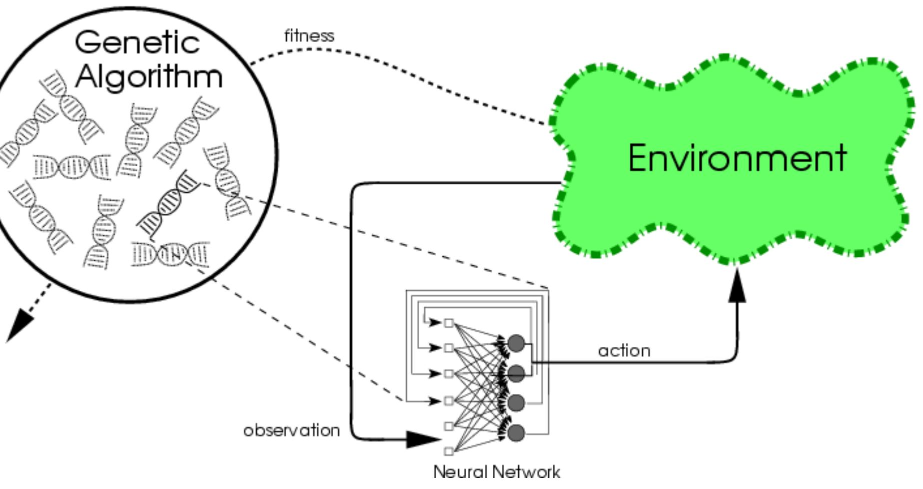

Figure 1: **Evolving Neural Networks.** population of genetic neural networks encodings (genotypes) is first created. At each iteration of evolution (generation), each genotype is decoded into a neural network (phenotype), which is evaluated in the task, resulting in a fitness value for the genotype. Crossover and mutation among the genotypes with the highest fitness is then used to generate the next generation.

through evolution, it can be applied to a wide range of network architectures, including those with non-differentiable activation functions and recurrent and higher-order connections. While most neural learning algorithms focus on modifying the weights only, neuroevolution can be used to optimize other aspects of the networks as well, including activation functions and network topologies.

Third, neuroevolution allows combining evolution over a population of solutions with lifetime learning in individual solutions: the evolved networks can each learn further through e.g. backpropagation or Hebbian learning. The approach is therefore well suited to understanding biological adaptation, and for building artificial life systems.

4 Structure of the Learning System

4.1 Basic methods

In neuroevolution, a population of genetic encodings of neural networks is evolved in order to find a network that solves the given task. Most neuroevolution methods follow the usual generate-and-test loop of evolutionary algorithms (figure 1). Each encoding in the population (a genotype) is chosen in turn and decoded into the corresponding neural network (a phenotype). This network is then employed in the task, and its performance over time measured, obtaining a fitness value for the corresponding genotype. After all members of the population have been evaluated in this manner, genetic operators are used to create the next generation of the population. Those encodings with the highest fitness are mutated and crossed over with each other, and the resulting offspring replaces the genotypes with the lowest fitness in the population. The process therefore constitutes an intelligent parallel search towards better genotypes, and continues until a network with a sufficiently high fitness is found.

Several methods exist for evolving neural networks depending on how the networks are encoded. The most straightforward encoding, sometimes called conventional neuroevolution (CNE), is formed by concatenating the numerical values for the network weights (either binary or floating point) [5, 16, 21]. This encoding allows evolution to optimize the weights of a fixed neural network architecture, an approach that is easy to implement and is practical in many domains.

In more challenging domains, the CNE approach suffers from three problems: The method may cause the population to converge before a solution is found, making further progress difficult (i.e. premature convergence); similar networks, such as those where the order of nodes is different, may have different encodings, and much effort is wasted in trying to optimize them in parallel (i.e. competing conventions); a large number of parameters need to be optimized at once, which is difficult through evolution.

More sophisticated encodings have been devised to alleviate these problems. One approach is to run the evolution at the level of solution components instead of full solutions. That is, instead of a population of complete neural networks, a population of network fragments, neurons, or connection weights is evolved [7, 13, 15]. Each individual is evaluated as part of a full network, and its fitness reflects how well it cooperates with other individuals in forming a full network. Specifications for how to combine the components into a full network can be evolved separately, or the combination can be based on designated roles for subpopulations. In this manner, the complex problem of finding a solution network is broken into several smaller subproblems; evolution is forced to maintain diverse solutions, and competing conventions and the number of parameters is drastically reduced.

Another approach is to evolve the network topology, in addition to the weights. The idea is that topology can have a large effect on function, and evolving appropriate topologies can achieve good performance faster than evolving weights only [2, 5, 18, 21]. Since topologies are explicitly specified, competing conventions are largely avoided. It is also possible to start evolution with simple solutions and gradually make them more complex, a process that takes place in biology and is a powerful approach in machine learning in general. Speciation according to the topology can be used to avoid premature convergence, and to protect novel topological solutions until their weights have been sufficiently optimized.

All of the above methods map the genetic encoding directly to the corresponding neural network, i.e. each part of the encoding corresponds to a part of the network, and vice versa. Indirect encoding, in contrast, specifies a process through which the network is constructed, such as cell division or generation through a grammar [5, 8, 17, 21]. Such an encoding can be highly compact, and also take advantage of modular solutions. The same structures can be repeated with minor modifications, as they often are in biology. It is, however, difficult to optimize solutions produced by indirect encoding, and realizing its full potential is still future work.

The fifth approach is to evolve an ensemble of neural networks to solve the task together, instead of a single network [11]. This approach takes advantage of the diversity in the population: different networks learn different parts or aspects of the training data, and together the whole ensemble can perform better than a single network. Diversity can be created through speciation and negative correlation, encouraging useful specializations to emerge. The approach can be used to design ensembles for classification problems, but it can also be extended to control tasks.

4.2 Extensions

The basic mechanisms of neuroevolution can be augmented in several ways, making the process more efficient and extending it to various applications. One of the most basic ones is incremental evolution, or shaping: Evolution is began on a simple task, and once that is mastered, the solutions are evolved further on a more challenging task, and through a series of such transfer steps, eventually on the actual goal task itself [7]. Shaping can be done by changing the environment, such as increasing the speed of the opponents, or by changing the fitness function, e.g. by rewarding gradually more complex behaviors. It is often possible to solve challenging tasks by approaching them incrementally even when they cannot be solved directly.

Many extensions to evolutionary computation methods apply particularly well to neuroevolution. For instance, intelligent mutation techniques such as those employed in evolutionary strategies are effective because the weights often have suitable correlations [9]. Networks can also be evolved through coevolution [4, 18]. A coevolutionary arms race can be established e.g. based on complexification of network topology: as the network becomes gradually more complex, evolution is likely to elaborate on existing behaviors instead of replacing them.

On the other hand, several extensions utilize the special properties of the neural network phenotype. For instance, neuron activation functions, initial states, and learning rules can be evolved to fit the task [5, 16, 21]. Most significantly, evolution can be combined with other neural network learning methods [5]. In such approaches, evolution usually provides the initial network, which then adapts further during its evaluation in the task. The adaptation can take place through Hebbian learning, thereby strengthening those existing behaviors that are invoked often during evaluation. Alternatively, supervised learning such as backpropagation can be used, provided targets are available. Even if the optimal behaviors are not known, such training can be useful: networks can be trained to imitate the most successful individuals in the population, or part of the network can be trained in a related task such as predicting the next inputs, or evaluating the utility of actions based on values obtained through Q-learning. The weight changes may be encoded back into the genotype, implementing Lamarckian evolution; alternatively, they may affect selection through the Baldwin effect, i.e. networks that learn well will be selected for reproduction even if the weight changes themselves are not inherited [1, 3, 8].

There are also several ways to bias and direct the learning system using human knowledge. For instance, human-coded rules can be encoded in partial network structures, and incorporated into the evolving networks as structural mutations. Such knowledge can be used to implement initial behaviors in the population, or it can serve as advice during evolution [12]. In cases where rule-based knowledge is not available, it may still be possible to obtain examples of human behavior. Such examples can then be incorporated into evolution, either as components of fitness, or by explicitly training the evolved solutions towards human behavior through e.g. backpropagation [3]. Similarly, knowledge about the task and its components can be utilized in designing effective shaping strategies. In this manner, human expertise can be used to bootstrap and guide evolution in difficult tasks, as well as direct it towards the desired kinds of solutions.

5 Applications

Neuroevolution methods are powerful especially in continuous domains of reinforcement learning, and those that have partially observable states. For instance in the benchmark task of balancing the inverted pendulum without velocity information (making the problem partially observable), the advanced methods have been shown to find solutions two orders of magnitude faster than value-function based reinforcement-learning methods (measured by number of evaluations; [7]). They can also solve harder versions of the problem, such as balancing two poles simultaneously.

The method is powerful enough to make many real-world applications of reinforcement learning possible. The most obvious area is adaptive, nonlinear control of physical devices. For instance, neural network controllers have been evolved to drive mobile robots, automobiles, and even rockets [6, 14, 19]. The control approach have been extended to optimize systems such as chemical processes, manufacturing systems, and computer systems. A crucial limitation with current approaches is that the controllers usually need to be developed in simulation and transferred to the real system. Evolution is strongest as an off-line learning method where it is free to explore potential solutions in parallel.

Evolution of neural networks is a natural tool for problems in artificial life. Because networks implement behaviors, it is possible to design neuroevolution experiments on how behaviors such as foraging, pursuit and evasion, hunting and herding, collaboration, and even communication may emerge in response to environmental pressure [20]. It is possible to analyze the evolved circuits and understand how they map to function, leading to insights into biological networks [10]. The evolutionary behavior approach is also useful for constructing characters in artificial environments, such as games and simulators. Non-player characters in current video games are usually scripted and limited; neuroevolution can be used to evolve complex behaviors for them, and even adapt them in real time [12].

6 Programs and Data

Software for e.g. the NEAT method for evolving network weights and topologies, and the ESP method for evolving neurons to form networks is available at http://nn.cs.utexas.edu/keyword?neuroevolution.

The TEEM software for evolving neural networks for robotics experiments is available at http://teem.epfl.ch.

The OpenNERO software for evolving intelligent multiagent behavior in simulated environments is at http://nn.cs.utexas.edu/?opennero.

7 See also

Evolutionary Computation; Reinforcement Learning.

References

[1] D. Ackley and M. Littman, Interactions between learning and evolution, in: *Artificial Life II*, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, eds., 487–509, Reading, MA: Addison-Wesley (1992).

[2] P. J. Angeline, G. M. Saunders, and J. B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, *IEEE Transactions on Neural Networks*, 5:54–65 (1994).

[3] B. D. Bryant and R. Miikkulainen, Acquiring visibly intelligent behavior with example-guided neuroevolution, in: *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA (2007).

[4] K. Chellapilla and D. B. Fogel, Evolution, neural networks, games, and intelligence, *Proceedings of the IEEE*, 87:1471–1496 (1999).

[5] D. Floreano, P. Dürr, and C. Mattiussi, Neuroevolution: From architectures to learning, *Evolutionary Intelligence*, 1:47–62 (2008).

[6] F. Gomez and R. Miikkulainen, Active guidance for a finless rocket using neuroevolution, in: *Proceedings of the Genetic and Evolutionary Computation Conference*, 2084–2095, Morgan Kaufmann, San Francisco (2003).

[7] F. Gomez, J. Schmidhuber, and R. Miikkulainen, Accelerated neural evolution through cooperatively coevolved synapses, *Journal of Machine Learning Research*, 9:937–965 (2008).

[8] F. Gruau and D. Whitley, Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect, *Evolutionary Computation*, 1:213–233 (1993).

[9] C. Igel, Neuroevolution for reinforcement learning using evolution strategies, in: *Proceedings of the 2003 Congress on Evolutionary Computation*, R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, eds., 2588–2595, IEEE Press, Piscataway, NJ (2003).

[10] A. Keinan, B. Sandbank, C. C. Hilgetag, I. Meilijson, and E. Ruppin, Axiomatic scalable neurocontroller analysis via the Shapley value, *Artificial Life*, 12:333–352 (2006).

[11] Y. Liu, X. Yao, and T. Higuchi, Evolutionary ensembles with negative correlation learning, *IEEE Transactions on Evolutionary Computation*, 4:380–387 (2000).

[12] R. Miikkulainen, B. D. Bryant, R. Cornelius, I. V. Karpov, K. O. Stanley, and C. H. Yong, Computational intelligence in games, in: *Computational Intelligence: Principles and Practice*, G. Y. Yen and D. B. Fogel, eds., IEEE Computational Intelligence Society, Piscataway, NJ (2006).

[13] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, Evolutionary algorithms for reinforcement learning, *Journal of Artificial Intelligence Research*, 11:199–229 (1999).

[14] S. Nolfi and D. Floreano, *Evolutionary Robotics*, MIT Press, Cambridge (2000).

[15] M. A. Potter and K. A. D. Jong, Cooperative coevolution: An architecture for evolving coadapted subcomponents, *Evolutionary Computation*, 8:1–29 (2000).

[16] J. D. Schaffer, D. Whitley, and L. J. Eshelman, Combinations of genetic algorithms and neural networks: A survey of the state of the art, in: *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, D. Whitley and J. Schaffer, eds., 1–37, IEEE Computer Society Press, Los Alamitos, CA (1992).

[17] K. O. Stanley and R. Miikkulainen, A taxonomy for artificial embryogeny, *Artificial Life*, 9(2):93–130 (2003).

[18] K. O. Stanley and R. Miikkulainen, Competitive coevolution through evolutionary complexification, *Journal of Artificial Intelligence Research*, 21:63–100 (2004).

[19] J. Togelius and S. M. Lucas, Evolving robust and specialized car racing skills, in: *IEEE Congress on Evolutionary Computation*, 1187–1194, IEEE, Piscataway, NJ (2006).

[20] G. M. Werner and M. G. Dyer, Evolution of communication in artificial organisms, in: *Proceedings of the Workshop on Artificial Life (ALIFE '90)*, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, eds., 659–687, Reading, MA: Addison-Wesley (1992).

[21] X. Yao, Evolving artificial neural networks, *Proceedings of the IEEE*, 87(9):1423–1447 (1999).