To Appear In Encyclopedia of Machine Learning and Data Science, 3rd Edition, Dinh Phung, Claude Sammut and Geoffrey I. Webb (Eds.), New York, 2020. Springer.

Neuroevolution

Risto Miikkulainen
The University of Texas at Austin and
Cognizant Technology Solutions

1 Abstract

Neuroevolution is a method for modifying aspects of neural network design in order to learn a specific task. Evolutionary computation is used to discover designs 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, and modifying both differentiable and nondifferentiable aspects of the design, such as the architecture, weights, activation and loss functions, and learning algorithms. Neuroevolution thus serves three roles: First, it is 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. Second, it is an automatic method for discovering effective deep learning architectures. Third, especially when combined with learning in individual networks, it is a model of biological adaptation.

2 Synonyms

Evolving Neural Networks; Genetic Neural Networks; Evolutionary AutoML; Evolutionary Metalearning; Evolutionary Neural Architecture Search

3 Motivation and Background

The original motivation for neuroevolution (since the 1990s) was 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 that approach. For example, in game playing, vehicle control, and robotics, the optimal actions are not known; only after performing several actions we find out how well the actions worked, e.g. by observing whether the game was won or lost. Neuroevolution makes it possible to find a neural network that optimizes such behavior given only sparse information about how well the networks are doing, without direct information about what exactly they should be doing.

As a reinforcement learning (RL) method, neuroevolution has two advantages. First, compared to value-function methods, neural networks represent state and action spaces naturally through continuous values, largely avoiding the state explosion problem; the networks can use recurrency to encode past states and actions, thus making it possible to learn partially observable Markov decision processes (POMDP). Second, compared to other policy search methods, neuroevolution takes advantage of population-based search, making it possible to scale up to large, high-dimensional, and deceptive search spaces (Miikkulainen 2019).

The second motivation (since about 2016) is to design better deep learning architectures. Deep learning systems have become large and complex, and their structure matters: different architectures work best in different tasks. Because many components of the design interact, it is no longer possible to design them effectively by hand. The challenge is similar to that of POMDP systems: the search space is large, high-dimensional, and deceptive. Neuroevolution (in this context sometimes called Evolutionary AutoML/Metalearning/Neural Architecture Search; Elsken et al. 2019; Stanley et al. 2019), is thus an effective approach to deep learning architecture search as well.

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, reinforcement learning, 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

The basic methods are general and apply to each of the three roles of neuroevolution i.e. reinforcement learning problems, discovering effective deep learning architectures, and modeling biological adaptation. Further refinements and differences are outlined in the extensions.

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 (Fig. 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. In sequential-decision making, the fitness may be e.g. the number of games won and lost; in deep learning, it may be the accuracy of the network after training; in biological adaptation, it may be the survival rate of the individuals.

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 toward 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 parameters (i.e. weights; either binary or floating point), or in architecture optimization, the network hyperparameters (Floreano et al. 2008; Yao 1999; Schaffer et al. 1992; Stanley et al. 2019; Loshchilov and Hutter 2016). This encoding allows evolution to optimize a fixed neural network design, 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

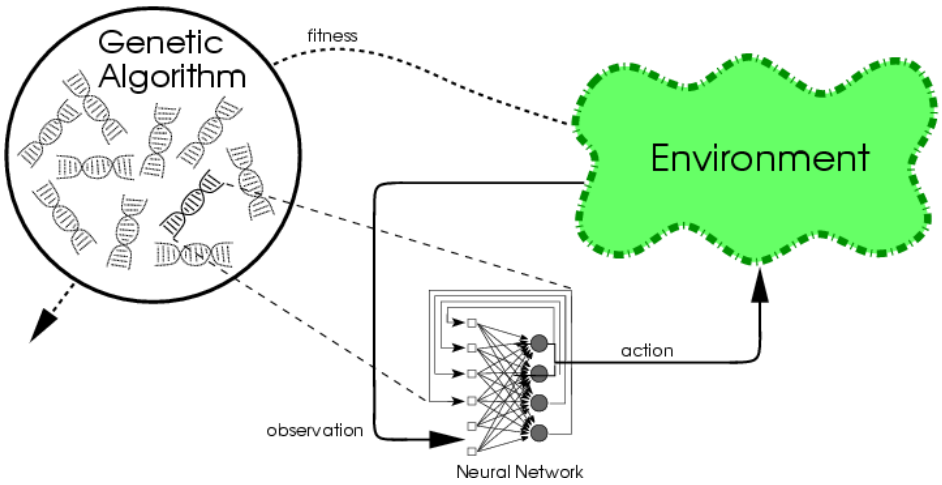

Figure 1: **Evolving Neural Networks.** A population of genetic neural network encodings (genotypes) is first created. At each iteration of evolution (generation), each genotype is decoded into a neural network (phenotype), which is evaluated in the task, resulting in a fitness value for the genotype.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

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 modules, neurons, or connection parameters is evolved (Moriarty et al. 1999; Gomez et al. 2008; Potter and Jong 2000; Miikkulainen et al. 2018; Liang et al. 2019). 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 parameters. The idea is that topology can have a large effect on function and evolving appropriate topologies can achieve good performance faster than evolving parameters only (Angeline et al. 1994; Floreano et al. 2008; Yao 1999; Stanley and Miikkulainen 2004; Stanley et al. 2019; Miikkulainen et al. 2018; Liang et al. 2019). 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 parameters 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 or through patterns generated by another neural network (Floreano et al. 2008; Yao 1999; Stanley and Miikkulainen 2003; Gruau and Whitley 1993; Stanley et al. 2009; Stanley et al. 2019; Such et al. 2017). 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.

Another approach is to evolve an ensemble of neural networks to solve the task together, instead of a

single network (Liu et al. 2000). 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 (Gomez et al. 2008). 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. First, intelligent mutation techniques such as those employed in evolutionary strategies are effective because the network parameters often have suitable correlations (Igel 2003). Second, networks can be evolved through coevolution (Stanley and Miikkulainen 2004; Chellapilla and Fogel 1999). 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. Third, behavioral diversity and novelty can be defined naturally in terms of network behavior, leading to methods that discover novel solutions (Lehman and Stanley 2010; Mouret and Doncieux 2012; Stanley and Lehman 2015).

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 through evolution, it can be applied to many different aspects of the network design. For instance, neuron activation functions, loss functions, initial states, and learning rules can be evolved to fit the task (Floreano et al. 2008; Yao 1999; Schaffer et al. 1992; Stanley et al. 2019; Bingham et al. 2020; Gonzalez and Miikkulainen 2020). It is possible to evolve modular network architectures, e.g., as a separate mutation or through minimizing wiring length, and thus discover how complex behavior arises from a combination of low-level behaviors (Clune et al. 2013; Schrum and Miikkulainen 2016).

Most significantly, evolution can be combined with other neural network learning methods (Floreano et al. 2008 Stanley et al. 2019). 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 parameter 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 parameter changes themselves are not inherited (Ackley and Littman 1992; Gruau and Whitley 1993; Bryant and Miikkulainen 2007).

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 (Miikkulainen et al. 2006). 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 toward human behavior through, e.g., backpropagation (Bryant and Miikkulainen 2007). 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 toward the desired kinds of solutions.

Interestingly, although many neuroevolution methods were developed for reinforcement learning problems, they can be extended to discovering effective deep learning architectures (Elsken et al. 2019). For instance, large architectures can be evolved using topology evolution at the high level, and component evolution in separate subpopulations at a lower level (Miikkulainen et al. 2018; Liang et al. 2019). Intelligent mutations apply not only to parameters (i.e. weights), but hyperparameters as well (Loshchilov and Hutter 2016). Activation functions and loss functions can be evolved for deep learning architectures as well (Bingham et al. 2020; Gonzalez and Miikkulainen 2020). Other techniques have been developed recently for discovering deep learning architectures, but they could also be applied to reinforcement learning problems. For instance, indirect methods, such as grammatical structures and structure embeddings, can be used to evolve topologies, and prior human knowledge by utilizing pre-existing architectures and their components to define the search space (Such et al. 2017; Miikkulainen et al. 2018; Liang et al. 2019). A smaller version of the network can be evolved first, and expanded to a larger version algorithmically (Real et al. 2019). Evolution can be used to discover architectures that learn multiple tasks at the same time, combining knowledge from all of them, and improving learning in each task (Liang et al. 2018; Meyerson and Miikkulainen 2019). Optimization of network architecture and hyperparameters can be interleaved with training (Li et al. 2019; Liang et al. 2020). Evolution can be extended beyond the architecture and learning setting to the learning algorithms themselves (Real et al. 2020).

Thus, many aspects of neural network design can be optimized through neuroevolution. One challenge for the future is to understand what makes them effective. A current hypothesis is that many of them amount to discovering and representing common structure, which leads to improved regularization. Another challenge is to determine synergies between them, i.e. co-evolve multiple aspects at the same time to take advantage of each other. While such approaches require careful design and significant computing resources, it is likely that major improvements are possible.

5 Applications

Neuroevolution methods are powerful especially in continuous reinforcement learning problems 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, Gomez et al. 2008). 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 (Valsalam et al. 2013; Togelius and Lucas 2006; Gomez and Miikkulainen 2003; Nolfi and Floreano 2000; Bongard 2011). 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, or with a surrogate model, and transferred to the real system. Neuroevolution is strongest as an off-line learning method where it is free to explore potential solutions in parallel (Francon et al. 2020).

Discovering effective deep learning architectures has applications everywhere where deep learning is used. At certain point in time, neuroevolution methods were used to create state of the art performance in several benchmarks, such as those in image recognition (Loshchilov and Hutter 2016; Real et al. 2019), language modeling (Liang et al. 2019; Rawal et al. 2020(@), multitask learning (Liang et al. 2019; Meyerson and Miikkulainen 2018), and reinforcement learning (Francon et al. 2020; Gomez and Miikkulainen 2003). In the future, such customized designs may be used to optimize aspects of design other than accuracy, such as network size or match with hardware architecture. Similarly, multitask architectures may be used to extend deep learning to domains where large training corpora are not available.

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 (Werner and Dyer 1992; Nolfi and Floreano 2000; Rajagopalan et al. 2020). It is possible to evolve the morphology and control together to create agents with natural movement (Lessin et al. 2013; Bongard 2011) and to analyze the evolved circuits and understand how they map to function, leading to insights into biological networks (Keinan et al. 2006). 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 (Miikkulainen et al. 2006; Risi and Togelius 2017).

6 Programs and Data

Software for the NEAT method for evolving network weights and topologies, and for the ESP and CoSyNE methods for evolving neurons and weights to form networks, is available at nn.cs.utexas.edu/?neuroevolution. Software for HyperNEAT indirect neuroevolution method is available at eplex.cs.ucf.edu/hyperNEATpage and for novelty search at http://eplex.cs.ucf.edu/noveltysearch/userspage. Software for Deep Reinforcement Learning via Neuroevolution is available at https://eng.uber.com/deep-neuroevolution.

ECJ (https://cs.gmu.edu/ eclab/projects/ecj), PyBrain (pybrain.org) and Sferes2 (github.com/jbmouret/sferes2) are general machine learning and evolutionary computation packages that include neuroevolution methods.

7 Cross-References

Evolutionary Computation; Reinforcement Learning

References

Ackley D, Littman M (1992) Interactions between learning and evolution. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) Artificial life II. Addison-Wesley, Reading, pp 487–509

Angeline PJ, Saunders GM, Pollack JB (1994) An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans Neural Netw 5:54–65

Bingham, G., Macke, W., and Miikkulainen, R. (2020) Evolutionary Optimization of Deep Learning Activation Functions. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2020, Cancun, Mexico)

Bongard J (2011) Morphological change in machines accelerates the evolution of robust behavior. Proc Natl Acad Sci USA 108:1234–1239

Bryant BD, Miikkulainen R (2007) Acquiring visibly intelligent behavior with example-guided neuroevolution. In: Proceedings of the twenty-second national conference on artificial intelligence. AAAI, Menlo Park

Chellapilla K, Fogel DB (1999) Evolution, neural networks, games, and intelligence. Proc IEEE 87:1471–1496

Clune J, Mouret J-B, Lipson H (2013) The evolutionary origins of modularity. Proc R Soc B Biol Sci 280(1755):20122863

Elsken T, Metzen JH, Hutter F (2019) Neural Architecture Search: A Survey. Journal of Machine Learning Research 20:1–21

Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. Evol Intell 1:47–62

Francon, O., Gonzalez, S., Hodjat, B., Meyerson, E., Miikkulainen, R., Qiu, X., and Shahrzad, H. (2020) Effective Reinforcement Learning through Evolutionary Surrogate-Assisted Prescription. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2020, Cancun, Mexico)

Gomez F, Miikkulainen R (2003) Active guidance for a finless rocket using neuroevolution. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2003, Chicago, IL)

Gomez F, Schmidhuber J, Miikkulainen R (2008) Accelerated neural evolution through cooperatively coevolved synapses. J Mach Learn Res 9:937–965

Gonzalez, S. and Miikkulainen, R. (2020) Improved Training Speed, Accuracy, and Data Utilization via Loss Function Optimization. In: Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC-2020, Glasgow, Scotland)

Gruau F, Whitley D (1993) Adding learning to the cellular development of neural networks: evolution and the Baldwin effect. Evol Comput 1:213–233

Igel C (2003) Neuroevolution for reinforcement learning using evolution strategies. In: Sarker R, Reynolds R, Abbass H, Tan KC, McKay B, Essam D, Gedeon T (eds) Proceedings of the 2003 congress on evolutionary computation. IEEE, Piscataway, pp 2588–2595

Keinan A, Sandbank B, Hilgetag CC, Meilijson I, Ruppin E (2006) Axiomatic scalable neurocontroller analysis via the Shapley value. Artif Life 12:333–352

Lehman J, Stanley KO (2010) Abandoning objectives: evolution through the search for novelty alone. Evol Comput 2011:189–223

Lessin D, Fussell D, Miikkulainen R (2013) Open-ended behavioral complexity for evolved virtual creatures. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2013, Amsterdam, the Netherlands)

Li A, Spyra O, Perel S, Dalibard V, Jaderberg M, Gu C, Budden D, Harley T, Gupta P (2019) A Generalized Framework for Population Based Training. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1791-1799

Liang, J., Meyerson, E., and Miikkulainen, R. (2018) Evolutionary Architecture Search for Deep Multitask Networks. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2018, Kyoto, Japan)

Liang, J., Meyerson, E., Fink, D., Mutch, K., and Miikkulainen, R. (2019) Evolutionary Neural AutoML for Deep Learning. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2019, Prague, Czech Republic)

Liang, J., Gonzalez, S., and Miikkulainen, R. (2020) Population-Based Training for Loss Function Optimization. arXiv:2002.04225

Liu Y, Yao X, Higuchi T (2000) Evolutionary ensembles with negative correlation learning. IEEE Trans Evol Comput 4:380–387

Loshchilov I, Hutter F (2016) CMA-ES for Hyperparameter Optimization of Deep Neural Networks. In: International Conference on Learning Representations (ICLR-2016) Workshop Track

Meyerson, E. and Miikkulainen, R. (2018) Pseudo-task Augmentation: From Deep Multitask Learning to Intratask Sharing—and Back. In: Proceedings of the 35th International Conference on Machine Learning (ICML-2018, Stockholm, Sweden)

Meyerson, E. and Miikkulainen, R. (2019) Modular Universal Reparameterization: Deep Multi-task Learning Across Diverse Domains. In: Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS-2019, Vancouver, Canada)

Miikkulainen, R (2019) Creative AI Through Evolutionary Computation. In: Banzhaf et al. Evolution in Action: Past, Present and Future. New York: Springer

Miikkulainen R, Bryant BD, Cornelius R, Karpov IV, Stanley KO, Yong CH (2006) Computational intelligence in games. In: Yen GY, Fogel DB (eds) Computational intelligence: principles and practice, Piscataway. IEEE Computational Intelligence Society

Miikkulainen, R., Meyerson, E., Liang, J., Rawal, A., Shahrzad, H., Fink, D. Francon, O., Raju, B., Navruzyan, A., Hodjat, B., and Duffy, N. (2018) Evolving Deep Neural Networks. In C. F. Morabito, C. Alippi, Y. Choe, and R. Kozma (Eds.), Artificial Intelligence in the Age of Neural Networks and Brain Computing. New York: Elsevier

Moriarty DE, Schultz AC, Grefenstette JJ (1999) Evolutionary algorithms for reinforcement learning. J Artif Intell Res 11:199–229

Mouret J-B, Doncieux S (2012) Encouraging behavioral diversity in evolutionary robotics: an empirical study. Evol Comput 20:91–133

Nolfi S, Floreano D (2000) Evolutionary robotics. MIT, Cambridge

Potter MA, Jong KAD (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol Comput 8:1–29

Rajagopalan, P., Holekamp, K. E. and Miikkulainen, R. (2020) Evolution of Complex Coordinated Behavior. In: Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC-2020, Glasgow, Scotland)

Rawal, A., Liang, J., and Miikkulainen, R. (2020) Discovering Gated Recurrent Neural Network Architectures. In H. Iab and N. Noman (editors), Deep Neural Evolution Deep Learning with Evolutionary Computation. New York: Springer

Real E, Aggarwal A, Huang Y, Le QV (2019) Regularized Evolution for Image Classifier Architecture Search. In: Proceedings of the AAAI Conference on Artificial Intelligence

Real E, Liang C, So DR, Le QV (2020) AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. arXiv:2003.03384

Risi S, Togelius J (2017) Neuroevolution in games: State of the art and open challenges. IEEE Transactions on Computational Intelligence and AI in Games 9:25–41

Schaffer JD, Whitley D, Eshelman LJ (1992) Combinations of genetic algorithms and neural networks: a survey of the state of the art. In: Whitley D, Schaffer J (eds) Proceedings of the international workshop on combinations of genetic algorithms and neural networks. IEEE Computer Society Press, Los Alamitos, pp 1–37

Schrum J, Miikkulainen R (2016) Discovering Multimodal Behavior in Ms. Pac-Man through Evolution of Modular Neural Networks. IEEE Transactions on Computational Intelligence and AI in Games 8:67–81

Stanley KO, D'Ambrosio DB, Gauci J (2009) A hypercube-based encoding for evolving large-scale neural networks. Artif Life 15(2):185–212

Stanley KO, Miikkulainen R (2003) A taxonomy for artificial embryogeny. Artif Life 9(2):93–130

Stanley KO, Miikkulainen R (2004) Competitive coevolution through evolutionary complexification. J Artif Intell Res 21:63–100

Stanley KO, Lehman J (2015) Why Greatness Cannot Be Planned: The Myth of the Objective. New York: Springer

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen R. (2019) Designing Neural Networks through Evolutionary Algorithms. Nature Machine Intelligence 1:24-35

Such FP, Madhavan V, Conti E, Lehman J, Stanley KO, Clune J (2017) Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. arXiv:1712.06567

Togelius J, Lucas SM (2006) Evolving robust and specialized car racing skills. In: IEEE congress on evolutionary computation. IEEE, Piscataway, pp 1187–1194

Valsalam V, Hiller J, MacCurdy R, Lipson H, Miikkulainen R (2013) Constructing controllers for physical multilegged robots using the enso neuroevolution approach. Evol Intell 14:303–331

Werner GM, Dyer MG (1992) Evolution of communication in artificial organisms. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) Proceedings of the workshop on artificial life (ALIFE '90). Addison-Wesley, Reading, pp 659 –687

Yao X (1999) Evolving artificial neural networks. Proc IEEE 87(9):1423–1447