Neural Computing and Applications (2020) 32:519–545
https://doi.org/10.1007/s00521-019-04160-6

REVIEW ARTICLE

On the automated, evolutionary design of neural networks: past, present, and future

Alejandro Baldominos[1] · Yago Saez[1] · Pedro Isasi[1]

Received: 19 April 2018 / Accepted: 19 March 2019 / Published online: 27 March 2019

Abstract
Neuroevolution is the name given to a field of computer science that applies evolutionary computation for evolving some aspects of neural networks. After the AI Winter came to an end, neural networks reemerged to solve a great variety of problems. However, their usage requires designing their topology, a decision with a potentially high impact on performance. Whereas many works have tried to suggest rules-of-thumb for designing topologies, the truth is that there are not analytic procedures for determining the optimal one for a given problem, and trial-and-error is often used instead. Neuroevolution arose almost 3 decades ago, with some works focusing on the evolutionary design of the topology and most works describing techniques for learning connection weights. Since then, evolutionary computation has been proved to be a convenient approach for determining the topology and weights of neural networks, and neuroevolution has been applied to a great variety of fields. However, for more than 2 decades neuroevolution has mainly focused on simple artificial neural networks models, far from today's deep learning standards. This is insufficient for determining good architectures for modern networks extensively used nowadays, which involve multiple hidden layers, recurrent cells, etc. More importantly, deep and convolutional neural networks have become a de facto standard in representation learning for solving many different problems, and neuroevolution has only focused in this kind of networks in very recent years, with many works being presented in 2017 onward. In this paper, we review the field of neuroevolution during the last 3 decades. We will put the focus on very recent works on the evolution of deep and convolutional neural networks, which is a new but growing field of study. To the best of our knowledge, this is the best survey reviewing the literature in this field, and we have described the features of each work as well as their performance on well-known databases when available. This work aims to provide a complete reference of all works related to neuroevolution of convolutional neural networks up to the date. Finally, we will provide some future directions for the advancement of this research area.

Keywords Neuroevolution · Evolutionary algorithms · Deep neural networks · Convolutional neural networks

1 Introduction

In the 1980s, once the *AI Winter* had come to an end, artificial neural networks (ANNs) reemerged to become one of the most flexible tools to solve a variety of AI problems. This emergence was driven by the discovery of the backpropagation algorithm, which enabled to learn the parameters (weights) of an ANN by using gradient descent to reduce a loss function computed among the ANN output and the real label of the data. The backpropagation of errors in connected networks was first described by Linnainmaa in his Master thesis in 1970 [72], although it was Werbos who first explicitly applied it to neural networks [131]. It would be Rumelhart, Hinton and Williams in 1986 who would popularize this process years later [102], by experimentally proving the emergence of internal representations in hidden layers.

However, backpropagation had an important limitation at the time: It could only be used in the context of supervised learning. This was an important handicap, as neural

✉ Alejandro Baldominos
abaldomi@inf.uc3m.es

Yago Saez
yago.saez@uc3m.es

Pedro Isasi
isasi@ia.uc3m.es

[1] Computer Science Department, Universidad Carlos III de Madrid, Avda. de la Universidad 30, Leganes 28911, Madrid, Spain

networks proved to be a promising approach for many reinforcement learning problems. It would require more than a decade to explore the application of gradient descent to reinforcement learning by enforcing the Bellman optimality [5]. Additionally, in some problems the network performance can be measured, yet a loss function cannot be computed. As a result, it became an interesting problem to discover techniques able to find optimal weights for an ANN even when gradient descent could not be applied.

An additional issue with ANNs had to do with their topology, i.e., the way in which neurons are connected within the network. In particular, there were not analytic procedures to determine the optimal topology of an ANN for a problem at hand, and in many cases trial-and-error was used to find a topology that worked fine. Although some *rules-of-thumb* emerged to provide guidelines for designing optimal topologies, the truth is that they did not work [40]. Cybenko in 1989 [24] showed that although a single hidden layer and sigmoid activation could approximate any continuous function, the size of such layer could not be established to leverage a solution of a desired quality. Cybenko indeed suggested that the majority of problems would require an astronomically large number of units. Researchers in an old Usenet thread from 1995 even described the rules-of-thumb proposed in several books and articles for choosing a topology as "total garbage" [93].

By the end of the 1980s, researchers had started to use evolutionary algorithms (EAs) instead of backpropagation to evolve the weights of ANNs, and few years later similar approaches were used for determining the best network topologies. This research field would later be called "*neuroevolution*" (NE) and has been proved effective in many applications.

However, most early works in NE assume only one hidden layer. Nowadays, the availability of hardware resources has led to the development of ANNs with many hidden layers, known as "*deep neural networks*" (DNNs), and the recent inclusion of representation learning in ANNs has given birth to *convolutional neural networks* (CNNs), which involve layers able to automatically extract relevant features from raw data. CNNs have been applied to very diverse domains and problems in the last years, including character, image and speech recognition, natural language processing, automatic image captioning, signal processing, etc.

CNNs can involve very complex topologies, with dozens of convolutional layers comprising diverse depths and filter sizes, and with several dense or recurrent layers with a variable number of neurons and diverse activation functions. Since manual design of CNN topologies is expensive, NE shows as a promising approach to obtain optimal topologies for a given problem. A schematic illustration of this process is shown in Fig. 1: It should be noted that in this approach the resulting CNN topology is trained using backpropagation, and later evaluated over the validation dataset.

Although this paper will try to make an exhaustive analysis of the automatic and evolutionary design of neural networks, we will focus in emphasizing the application of evolutionary techniques in the specific design of CNNs, since these models have been proved very successful in solving AI problems that remained unsolved for decades, allowing their expansion in the industry like never before. Also, the complexity of CNNs makes the topology design and the assignment of the parameters the cornerstone of their success or failure, turning the CNN design and training into a very sensitive and complex process, whose automation is highly recommended. Finally, we strongly believe that this new line is starting a groundbreaking change in neuroevolution, and pioneering works in this area are worth reviewing. To the best of our knowledge, this is the first work of this kind, and in addition to describing each of the works in the field, we will also summarize their features and their performance on well-known databases when available, hoping that it can become a reference guide for readers interested in applying neuroevolution to the problems of their own.

This paper is structured as follows: Sect. 2 describes some of the most relevant concepts of evolutionary computation, and Sect. 3 analogously introduces neural networks, which are useful to understand this paper, and in particular how their topology can be evolved. Section 4 explains in further detail the motivation for optimizing CNN topologies. Section 5 briefly explains what NE is and describes some relevant milestones in the history of this field. Section 6 provides a more exhaustive survey on how NE is nowadays being applied to CNNs. Finally, in Sect. 7 we try to foresee what we believe the future of this field is, settling the basis for future works and encouraging academia to develop new research lines in this area.

2 Basic concepts on evolutionary computation

An important area of application of computer science is optimization: the selection of an optimal or set of optimal elements from a larger set of candidates according to some specific criteria. Optimization problems are pervasive and can be found in nature. In extension, many problems can be modeled as an optimization problem, coming from diverse fields, such as economics, politics, engineering, game theory, or evolutionary biology. In fact, many of these problems (specifically, those belonging to the NP-Complete class of complexity) can be restated to become

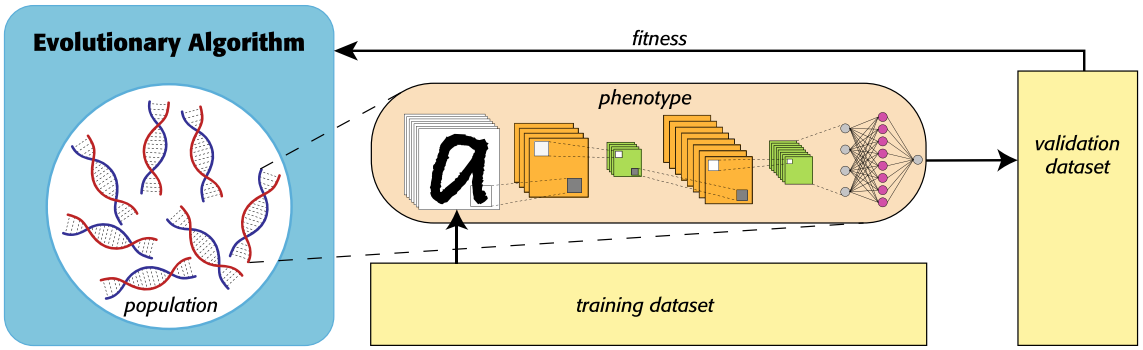


Fig. 1 Evolution of convolutional neural networks for supervised learning

equivalent among them. Unfortunately, finding an optimal solution for many of these problems becomes intractable upon a large enough search space, meaning that a solution cannot be provided by computers in a reasonable amount of time.

In the past, some authors have defined evolutionary biology as an optimization process, with John Maynard Smith being one of the greatest exponents of this school of thought [77, 92]. The field of evolutionary computation arises in this intersection between optimization problems and biology, and it tries to approach optimization problems using mechanisms dictated by Darwinian theory of natural selection and evolution.

The term "evolutionary computation" is used to refer to a discipline which encompasses a family of biologically inspired optimization algorithms known as evolutionary algorithms. Most of these algorithms adhere to the following properties [14, 121]:

- Metaheuristics: Evolutionary algorithms often perform optimization within a search space that is too large as to be computationally feasible to be tractable. These techniques do not implement knowledge specific to the problem (as in heuristic search), but instead rely on a measure of quality of each potential solution (called fitness), which may encode partial or even very little knowledge about the problem, and which serves for the purpose of driving the search process. The algorithms and procedures used for searching are general, meaning that can be reused in different problems with few to no changes, although in some cases some problem-specific knowledge can be introduced to enhance the search process. On the other hand, the main disadvantage of evolutionary algorithms is that they are stochastic in nature and are not guaranteed to find an optimal solution.
- Population based: In every state of the search process, a set of candidate solutions (known as a population) is maintained. The evolutionary algorithm bases the search procedure in operating with the population by applying operators that involve several individuals at the same time. However, the quality metric (fitness) is intrinsic to each individual.
- Biologically inspired: Evolutionary algorithms are inspired by natural principles, in particular, by those established in the Darwinian theory of evolution. For such reason, these algorithms often implement operators such as selection, recombination or mutation, that are applied over one or more individuals of the population.

Historically, the application of principles taken from natural evolution into optimization algorithms arose in the 1960s, although the rationale dates back to at least the 1950s, with Alan Turing mentioning this idea in the paper in which he introduced the imitation game [124]. In particular, he stated:

> [...] We have thus divided our problem into two parts. The child program and the education process. These two remain very closely connected. We cannot expect to find a good child machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse. There is an obvious connection between this process and evolution [...] One may hope, however, that this process will be more expeditious than evolution. The survival of the fittest is a slow method for measuring advantages.

Starting in the 1960s, some implementations of these ideas started to arise, which lead to different paradigms which ultimately settled the field of evolutionary computation. Most evolutionary algorithms today belong to one of the following classes:

- Evolutionary programming: This paradigm was presented by Fogel et al. [36] in 1966. The working mechanism of evolutionary programming consists of writing a computer program whose structure remains fixed but which relies on different numerical

parameters, which are then optimized. The main strategy for optimizing these parameters is random mutation across generations.

- Genetic algorithms: This paradigm was introduced by Holland [53] in 1975. It proposes the representation of a candidate solution to a problem in the form of a genetic encoding, which is often a binary string, even though alternative representations have been suggested that do not conform to the building blocks established by Holland. The evolution then takes place by iteratively applying genetic operators, the most common being selection, recombination and mutation, which generates offspring based on the genetic material of parents.
- Evolution strategies: This paradigm was introduced by Rechenberg [99] and Schwefel [109] in the 1970s. In this approach, a vector of real-valued numbers is evolved through selection and mutation, whose strength is often regulated by means of self-adaptation. In a first version of evolution strategies, only one parent and one child exist at the same time, and the best from both is kept for the next generation: this particular implementation is known as $(1 + 1)$-ES. Additional versions have been described with several children, leading to $(1 + \lambda)$-ES or $(1, \lambda)$-ES strategies, where in the former children can compete with the parent and in the latter the parent is disregarded. Also, several parents can be considered, and sometimes recombination is also included as an operator, leading to $(\mu/\rho+, \lambda)$-ES. It should be noted that while the classical canonical version of an evolution strategy differs from genetic algorithms in the use of real encoding instead of binary and in the lack of a recombination operator, particular innovations or implementations on these methods might reduce the gap between the two techniques, even turning them indistinguishable.
- Genetic programming: This paradigm is the most consistent with Turing's ideas. Early implementations of this approach first appeared about 30 years later, with those from Forsyth [37] and Cramer [22], although John Koza is acknowledged to be one of the key researchers that helped to establish the field [62, 63]. In genetic programming, programs have been traditionally represented as trees, which can be modified using genetic operators such as recombination or mutation.

A general execution framework for most evolutionary algorithms is shown in Fig. 2. It is remarkable that some algorithms might not adhere completely to this framework. For example, evolutionary programming or evolutionary strategies seldom use crossover, and their population might comprise only one individual. Still, the basic idea remains: Evolutionary algorithms are iterative processes (where each iteration is called a "generation") which evaluate a set of solutions and apply evolutionary operators on them expecting their quality to rise generation after generation. The most common operators, which are shown in the figure, are the following:

- Selection: This operator is intended to resemble natural selection (survival of the fittest) and is in charge of assigning the genetic material of the best solutions a higher chance to reproduce and remain in the following generations. There are different approaches for selection, although two are particularly common: tournaments and roulette. In tournaments, random individuals are chosen from the population and confronted among them, with the fittest individual winning the tournament and getting the chance to reproduce with another individual, which had also been a champion of a different tournament. In roulette, a probability distribution is assigned to the population, with the probability of one individual being proportional to its fitness; and then individuals are randomly chosen to reproduce following this probability distribution.
- Reproduction: This mechanism (also called recombination or crossover) is used to combine the genetic material of two selected solutions, expecting that at least part of their offspring is fittest that each of the parents. The process is roughly illustrated in Fig. 3a, where multi-point crossover has been used (i.e., different cutting points are established for the genomes and they are combined as shown in the figure). Other typical forms of reproduction are single-point crossover (same as displayed but only with one cutting point) or uniform crossover (where genes in the parents' genome interleave). In genetic programming, reproduction often involves exchanging two subtrees between the parents.
- Mutation: This operator enhances exploration by randomly mutating part of the individual's genome. In Fig. 3b, we show an example of bit-flipping mutation, which is common in genetic algorithms. In this case, only one bit is flipped, but it is often common to define a parameter, called "mutation rate," which establishes the percentage of the genome size to be mutated. When the genome comprises real numbers (such as in evolutionary programming or evolution strategies), the mutation can be done by changing one value to a new one following a Gaussian distribution centered in the previous value (and, interestingly, the standard deviation can evolve over generations depending on whether we want to promote exploration or exploitation). In genetic programming, mutation can be implemented by different means, from mutating one operand or one operator in one node, to mutating a whole subtree.

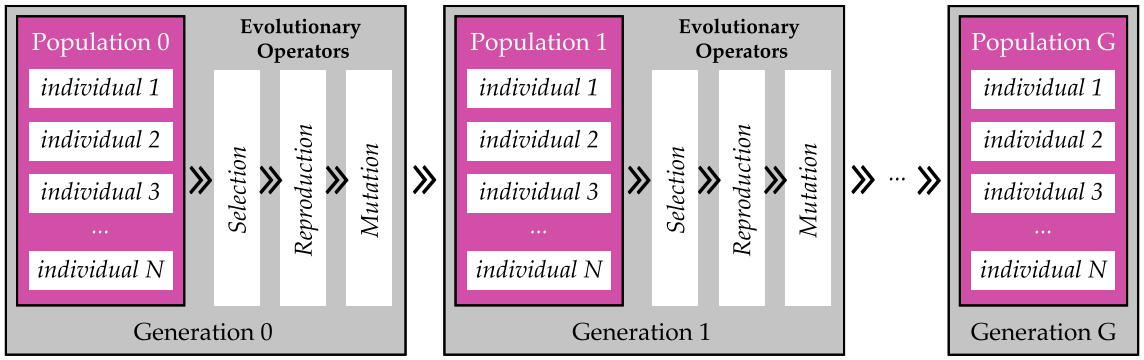
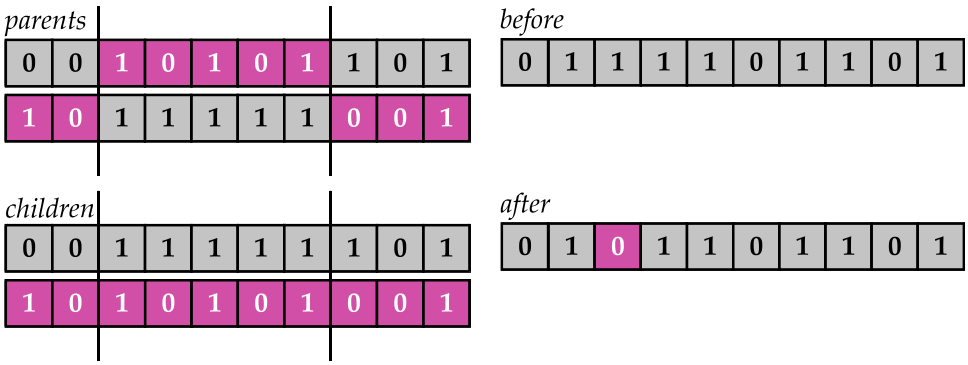


Fig. 2 General running framework for evolutionary algorithms



(a) Reproduction. **(b)** Mutation.

Fig. 3 Illustration of operators typically used in evolutionary computation

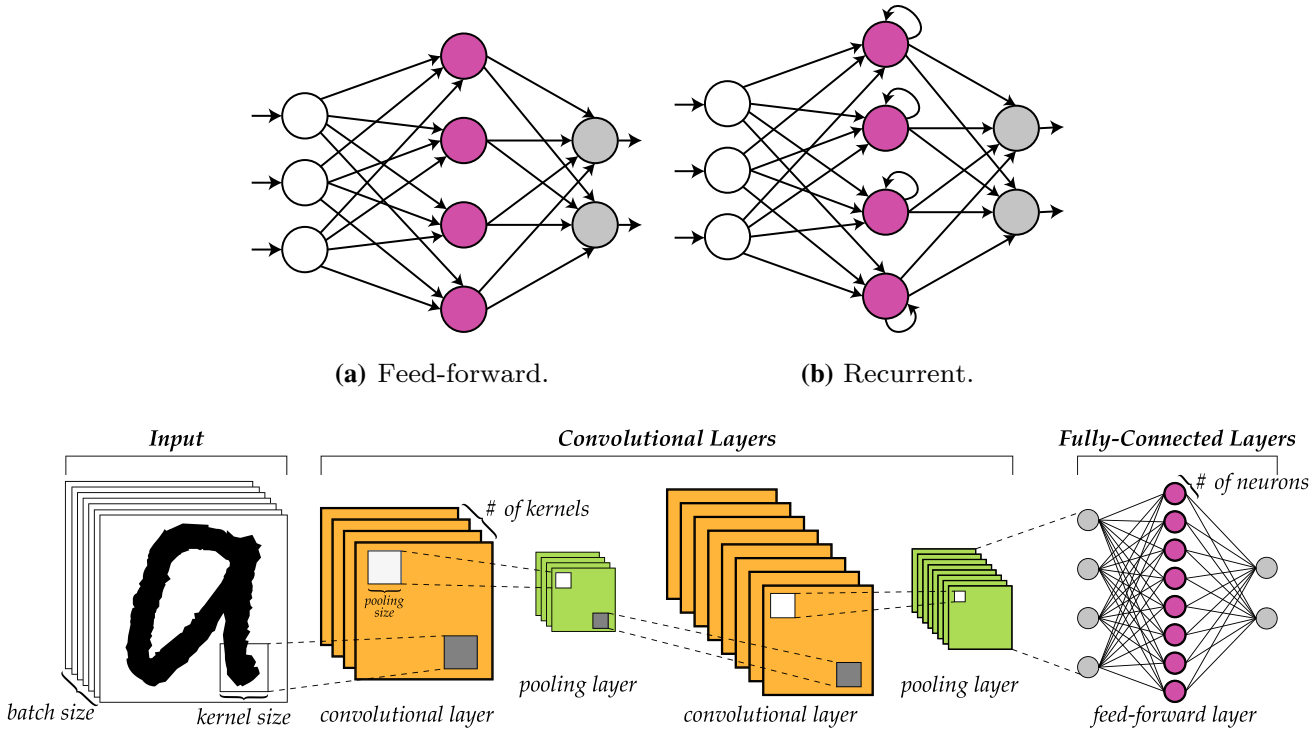
3 Basic concepts on neural networks

Neural networks are artificial intelligence models that arose in the early 1950s. It is considered that the first artificial neural network was Marvin Minsky and Dean Edmonds' Stochastic Neural Analog Reinforcement Calculator (SNARC) [84] in 1951, built in hardware using vacuum tubes. A significant achievement in AI arrived later in 1957 with Frank Rosenblatt's invention: the perceptron [101]. The perceptron was a neural network-based algorithm for learning a linear classifier to discriminate data in a binary classification problem. Its first implementation was in software for the IBM 704, yet it was later implemented in hardware known as the Mark 1 perceptron. Despite the relative simplicity of the perceptron algorithm, the project raised significant attention from the media, and the New York Times described it as "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence" [89].

In the following years, the high expectations in the perceptron lead to an increase in funding in artificial intelligence research. However, results turned out to be disappointing. One of the most desired applications of artificial intelligence was machine translation, due to the interest of the US government during the Cold War in translating Russian documents in a fast and automatic way. However, by the mid-1960s, the US National Research Council had spent more than 20 million dollars without remarkable results, concluding that human translation was cheaper and more efficient. Also, in 1969, Marvin Minsky and Seymour Papert explained some of the limitations of artificial neurons and the perceptron, such as the inability to learn a linear classifier to discriminate simple functions such as the exclusive OR (XOR), in their book *Perceptrons* [85]. This resulted in what is known as the "AI winter," which froze the research and the interest in AI during the 1970s.

However, in mid-1985 there was a significant renaissance of neural networks. One of the most relevant contributions by that time was the paper of Rumelhart et al. in Nature [102] which helped to popularize backpropagation, enabling an efficient way to automatically learn the optimal weights of a neural network by computing the derivative of the error and applying gradient descent. In the late 1980s and in the 1990s, the field of artificial neural networks had grown significantly.

By that time, the most common neural network topology was the multilayer perceptron (MLP), a feed-forward network (see Fig. 4a) where neurons in one layer are



(a) Feed-forward. **(b)** Recurrent.

(c) Convolutional.

Fig. 4 Different approaches to artificial neural networks

connected to neurons in the following layer. Because of the vanishing gradient problem, which made the gradient very small in the first layers when a large number of layers was used, these networks often came with only one hidden layer.

Still, despite the apparent simplicity of such models, the number of design decisions regarding their topology and training procedure was relatively large. Even when considering fully connected models (i.e., those in which every neuron in one layer is connected to all neurons in the following layer), the number of hidden layers and the number of neurons per layer have to be decided. With those two being the main two variables deciding the network topology, there are still a number of hyperparameters that must be optimized as well: the neurons' activation function (hyperbolic tangent and sigmoid were common at the time, though new functions have been designed throughout the years, such as the popular ReLU function), whether or not to use regularization or the learning rate.

Additionally, in the 1990s recurrent neural networks gained significant popularity, given their good performance at processing time series, with interesting applications such as speech recognition. In their most basic form, recurrent neural networks are those in which neurons can be connected to themselves or even to neurons in previous layers (see Fig. 4b). This change in the connectivity pattern can increase the number of decisions to make regarding the network topology. Additionally, with the years new recurrent models appeared that solved some of the issues of these primitive recurrent topologies, such as their inability to learn patterns properly when the temporal context goes back long into the past. One of the most remarkable contributions in this kind of novel recurrent models is long short-term memory (LSTM) cells [52], and gated recurrent unit (GRU) cells [20]. Deciding which variation of recurrent implementation to use can be considered as an additional parameter of the network design process.

Finally, in the late 1990s convolutional neural networks (CNNs) were presented by LeCun et al. [68, 69]. The idea behind CNNs is to provide some layers performing a convolutional operator over the input in order to automatically learn relevant features from data, which will later be introduced into a trainable classifier, such as a feed-forward or recurrent network. A sample topology of this kind of networks can be found in Fig. 4c. CNNs comprise one of the most important contributions to the field currently known as "deep learning."

Convolutional neural networks involve a higher complexity in their topology. Besides the setup of the trainable classifier, which is similar to what we previously described as feed-forward or recurrent networks, the design of

convolutional layers imposes several parameters to be specified.

To understand these parameters, it is useful to actually know how convolutional layers work. First, raw data will be introduced to the first layer, which will output some "*feature maps*," that will then be introduced as the input for the following layer. This process is done iteratively until the last convolutional layer has computed its feature maps. In general, it is considered that the more convolutional layers the network has, the more abstract features it will be able to extract.

Each convolutional layer will contain several kernels (also known as filters or patches), which *convolve* the input to generate a feature map as an output. The input data and the kernels will be shaped as multi-dimensional arrays (tensors). Optionally, an additional operator known as "pooling" can be placed after a convolutional layer, whose purpose is to reduce the size of the feature maps by downsampling them. A common practice to do so is max pooling, where the input is reduced by taking a subtensor of the feature map and replacing it by its maximum value.

As a result of this process, there are numerous parameters to consider when designing the convolutional layers of a CNN. The first obvious parameter is the number of convolutional layers. As stated before, the larger the number of layers, the more abstract features that can be extracted. However, many convolutional layers may not only increase the computational cost of the learning process, but also consume the dimensionality of the data. Additionally, other relevant parameters are the number of kernels in each layer, the size of the kernels (which again, can vary from layer to layer), whether pooling is performed after each layer, the pooling size, the activation function for each kernel, etc.

Moreover, the advances in deep learning research have introduced further innovations that can increase even more the number of parameters to be determined. For example, research has shown that introducing data in small batches during gradient descent can enhance convergence; and therefore, the size of such batches becomes a new hyperparameter. Also, new regularization techniques have been developed to easily fight overfitting (i.e., the model learning too well the training set and generalizing poorly over the validation and test sets), such as dropout [113]. Finally, many new learning functions have been developed which introduce enhancements over traditional gradient descent, aiming at improving convergence. In some cases, these learning functions involve additional hyperparameters that must be optimized as well.

4 Motivation

The latest edition of the Encyclopedia of Machine Learning and Data Mining [104] defines the topology of a neural network as the "*way the neurons are connected*" [81] and admits that it is "*an important factor in network functioning and learning*." However, this encyclopedic entry states that "*the most common topology in supervised learning is the fully connected, three-layer, feed-forward network*," ignoring later advances in deep neural networks and convolutional neural networks, where a larger number of layers (and thus of configurable parameters) is used.

The importance of the topology in neural networks has been addressed several times in the past. One of the first works studying this impact was presented in 1989 by Baum and Haussler [10], where they suggested theoretical upper and lower bounds on the sample size versus the network size for the sake of improving generalization in networks with one hidden layer.

Also, Lawrence et al. [67] tried to study in 1996 the correlation between network size and optimal generalization. In their work, they support the observation that larger networks can produce better training and generalization error. However, their results also show that some oversized networks suffer from overfitting: MLP with two hidden units outperformed 10 and 20 hidden units when approximating the function $y = \sin(x/3)$ in the range [0, 5]. However, when a larger range ([0, 20]) was used, the 50-units MLP was the best performer. They also conclude that committees or ensembles (a group of model working together to provide a single response) can be more beneficial when input data are noisy. Prior to that, Caruana [17] reported that large networks rarely do worse than small networks, though he only studied a limited set of problems and this statement cannot be generalized to further domains.

More recently, in 2011, Hermundstad et al. [49] tested different architectures, from 'fan' architectures (one hidden layer with many units), to 'stacked' architectures (many hidden layers with very few units each), along with intermediate architectures. However, they kept the total number of weights as steady as possible (between 37 and 41). Even if there were very few parameters and topologies were similar to some extent, authors concluded that "*different network architectures produce error landscapes with distinguishable characteristics, such as the height and width of local minima, which in turn determine performance features such as speed, accuracy, and adaptability*."

Some works have explored the impact on performance of different ANN topologies in specific domains. For instance, Choudhary et al. [21] looked for the influence of one versus two hidden layers in the scope of character

recognition and concluded that the addition of one hidden layer leads to higher accuracy of the neural network. In particular, accuracy in one test set improved from 65.38 to 88.46%, and in other test set from 80 to 84.61%. It is worth noting that this work is very limited, as researchers only compared one and two hidden layers using 10 hidden units in each one, and higher differences in accuracy could be expected when adding more neurons or further layers.

As for convolutional neural networks (CNNs), recent works have explored the performance of diverse well-known architectures. A good benchmark is annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [103], as many different topologies have been tested over the years. For example, a 2016 work from Mishkin et al. [86] studies the influence of different parameters in CNN topologies on the ILSVRC problem. This work is interesting as it explores how the accuracy is affected by the network width, batch size, or activation functions. By the end of the paper, authors provide some recommendations for a good topology, based on the knowledge they acquired from evaluating the performance of several CNNs, yet these are only valid for the ImageNet domain.

An even more recent work by Canziani et al. [16] reviews the accuracy values reported in the literature for very relevant models in the ImageNet domain, and studies the performance of these models in several dimensions, including accuracy, power consumption, speed or memory utilization. It is remarkable that the accuracy ranges from 54 to 80%, and it gets more impressive if we consider that these models have all been published in a 4-year period: AlexNet was introduced in 2012 by Krizhevsky et al. [65], resulting in the first approach of convolutional neural networks to ILSVRC, whereas Inception-v4 was presented in 2016 [120] by Szegedy et al. from Google. From this study, we can also conclude that larger networks, or networks with more operations do not imply a higher accuracy in all cases. The best performing network, Inception-v4, has an average size and number of operations when compared to its competitors.

Also, in the domain of human activity recognition, Hammerla et al. [43] have evaluated different deep learning models and conclude that convolutional neural networks show the most characteristic behavior with respect to the topology when compared to other models (non-convolutional deep neural networks and LSTM networks), as a fraction of model configurations do not work at all. They also state that functional setups show little variance in their performance, though they report a 7% difference in F1 score between peak and median performance, which we consider to be a remarkable variance.

Consequently, the influence of neural network design decisions can be fundamental for their performance and can be even greater when recurrent neural networks (RNNs) come into play, as they enable many new applications when time series are available. The effectiveness of RNNs is well demonstrated in Andrej Karpathy's blog [56], by applying them to train character-level language models with impressive outcomes and describing the internal behavior of the network. An interesting conclusion is drawn in the recent work by Lipton and Berkowitz [73], who after reviewing 3 decades of research in recurrent neural networks, have concluded [emphasis added]:

> While LSTMs and BRNNs (bidirectional recurrent neural networks) have set records in accuracy on many tasks in recent years, it is noteworthy that advances come from novel architectures rather than from fundamentally novel algorithms. Therefore, ***automating exploration of the space of possible models***, for example via genetic algorithms or a Markov chain Monte Carlo approach, ***could be promising***. Neural networks offer a wide range of transferable and combinable techniques. New activation functions, training procedures, initialization procedures, etc. are generally transferable across networks and tasks, often conferring similar benefits. As the number of such techniques grows, the practicality of testing all combinations diminishes. It seems reasonable to infer that as a community, neural network researchers are exploring the space of model architectures and configurations much as a genetic algorithm might, mixing and matching techniques, with a fitness function in the form of evaluation metrics on major datasets of interest.

In summary, the architecture of a neural network is an important factor affecting its performance, and its impact is especially noticeable in CNNs, where very complex topologies that can even comprise recurrent layers are considered. Additionally, recent developments in hardware devices (such as improved GPUs—graphic processor units—or specific architectures for tensor processing) are enabling the community to rapidly iterate over different topologies and models, since both forward propagation and backward propagation are significantly accelerated when using these devices.

5 Past

5.1 Background

As previously mentioned, the discovery of backpropagation and its popularization by Rumelhart et al. [102] meant the awakening of artificial neural networks within the AI scene. Backpropagation was used for determining the best

weights for a network in order to minimize a loss function between the computed output and the real output.

Few years later, due to the absence of an analytic procedure to compute the best topology of ANNs, there were an increasing interest on developing techniques for determining, or at least estimating, optimal neural network topologies. Some of the earliest approaches have tried to estimate the optimal topology for solving a certain problem using constructive or destructive algorithms. In the former, the process starts with a minimal network and new nodes and connections are added during the training phase, until performance stops improving (or starts degrading). As for destructive algorithms, the idea is similar yet starting with an oversized network from which nodes and connections are removed. Examples of these simple algorithms can be found in the works by Fahlman and Lebiere [33], Frean [38], Mozer and Smolensky [88], Sietsma and Dow [111] or Hirose et al. [51]. However, one of the most cited works regarding destructive algorithms is that by LeCun et al. published in 1990 [70].

Another early contribution was published by Wang et al. [130] in 1994, though the article had been submitted to the journal 2 years before, in 1992. In this work, authors constrained their research to neural networks with only two hidden layers and proposed an algorithm for determining the optimal number of hidden units by evolving the network topology during training using an algebraic approximation. Though they validated their approach by simulation over five noise-free and noise-corrupted datasets, they did not use more than 10 neurons per hidden layer, and thus their work explores a very small search space.

However, early after the need for automatically determining the best topologies arose, a relevant new field entered the scene: "neuroevolution" (NE). The Encyclopedia of Machine Learning [104] defines NE as follows [80]:

> 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 non-differentiable activation functions, and with recurrent networks.

5.2 Early steps

The concept of NE arose in the late 1980s,[1] with some authors motivating research in this area, such as Miihlenbein and Kindermann [79]. Most early works in NE are concerned with evolving the weights of ANNs. A straightforward approach is to encode the weights of the network in the genotype, either as a binary string or a list of real numbers. Some of these early works are those by Montana and Davis [87] or Whitley and Hanson [133] in 1989.

By that time, the application of NE for learning the weights of ANNs was specially interesting for those cases where backpropagation was not a good choice, e.g., multilayer or recurrent networks (RNNs). Whitley et al. [132] considered the evolution of multilayer feed-forward networks in 1991. Regarding RNNs, some early works are those by Torreele in 1991 [123] or de Garis in 1992 [27], who stated NE was "*more flexible and powerful than the traditional neural network paradigms*." While most works involved genetic algorithms (GAs), Scholz [108] suggested a modified version of evolutionary strategies (ES) in 1991.

Besides evolving the weights of a neural network, evolutionary computation was used in the beginning for a diversity of tasks related to the improvement of neural networks. For example, Harp et al. used genetic algorithms to find good values for the learning rate and decay in 1989 [46], finding that resulting values are higher than expected in some problems. Additionally, Belew et al. also used genetic algorithms to find a suitable initialization of the network for backpropagation in 1991 [11]. In 1990, Chalmers [18] used genetic algorithms in order to decide the best learning algorithm while keeping the topology fixed, although their results on feed-forward networks cannot be assessed since there are no results reported on benchmark datasets.

Some early works were also concerned with designing ANN topologies. For example, in 1989 Miller et al. [83] suggested using GAs for evolving the network structure. There is an explosion of this area in the early 1990s: Harp et al. [47] described NeuroGENESYS, a GA for learning the network structure and some additional learning parameters, yet using backpropagation for learning the weights, and a similar approach was described by Schaffer et al. [105]. Also that year, Kitano [60] suggested an alternative encoding based on graph generation grammars using GAs, arguing that the process is more efficient because it is able to generate more connectivity patterns while reducing the chromosome length; and in 1991 Schiffmann et al. [107] compared fixed and evolved network topologies with an application to handwritten digits recognition.

The first extensive survey in this area had been provided by Schaffer et al. [106] already in 1992, in an international workshop on combinations of genetic algorithms and neural networks. It is remarkable that, as early as in 1992, there was so much research interest in this field. By that

[1] However, the term "neuroevolution" would be coined years later.

time, more authors were already working in the evolutionary design of the neural network structure; e.g., Hancock [44] explored the performance of different recombination operators when evolving the network structure, Elias [32] described the use of a genetic algorithm to evolve the connection patterns of a neural network implemented over analog hardware, Dasgupta and McGregor [25] used structured genetic algorithms for the evolution of both the weights and topology of application-specific neural networks, Karunanithi et al. [57] described the use of genetic cascade learning to improve the network topology by adding one hidden unit at a time, and Lindgren et al. [71] evolved the topology and weights of a neural network for regular language inference.

It is remarkable that only 6 years after backpropagation was introduced, there was a large community of researchers addressing its issues and applying EAs to determine the weights and topology of ANNs. A summary of these works was published in 1995 by Balakrishnan and Honavar [6], according to a taxonomy based on the genotype representation, the network topology, the variables of evolution and the application domain.

An additional review was provided in 1993 by Yao [135], where the author distinguished three main types of works: evolution of weights, of architectures, and of learning rules. Yao's work described the need for automatic design of neural networks architectures:

> It is well known that EANN architecture has significant impact on EANN information processing abilities. Unfortunately, EANN architecture still has to be designed by experienced experts through trial-and-error. There is no systematic way to design an optimal (near optimal) architecture for a particular task.

Yao also agreed with Miller et al. [83] in that GA-based approaches are suitable for finding optimal solutions within the search space, i.e., the set of candidate solutions composed of all possible ANN architectures, because of the next characteristics of the search space:

- It is potentially infinite, since the number of possible nodes and connections is unbounded.
- It is non-differentiable, as changes in the number of nodes or connections are discrete but can have a continuous effect on the network performance.
- It is complex and noisy, because the mapping between a network and its performance is indirect and stochastic due to the randomness of initial weights.
- It is deceptive, since similar network architectures can lead to very different performances.
- It is multimodal, since very different architectures can have similar performance.

After addressing the advantages of using evolutionary computation for determining optimal or near-optimal topologies for artificial neural networks, Yao established a taxonomy of works on neuroevolution based on the encoding: direct or indirect.

In direct encoding, a binary chromosome specifies whether a connection between two nodes exists or not. To use this schema, we need some previous knowledge about the problem in order to determine a maximal topology, i.e., determining an upper bound to the number of layers and hidden units, which can be as large as desired. Once the maximal topology comprising N nodes is established, each neuron is numbered and a binary matrix $C = (c_{ij})_{N\times N}$ is created. A '1' in the position c_{ij} indicates that there is a connection from neuron i to neuron j, where as a '0' indicates that such connection does not exist. Additional constraints can be imposed, for example to guarantee a feed-forward neuron (only enabling connections from nodes in one layer to nodes in the following one). These additional constraints can be observed in the genotype by removing genes c_{ij} so that j is in not in the following layer than i. While this encoding is very natural, it entails two handicaps: First, certain assumptions must be made about the topology of the network ahead, in order to determine the maximal topology. Second, unless certain constraints are imposed (which requires still more knowledge and prior decisions on the topology), the size of the chromosome grows quadratically with the number of nodes in the network, posing a $\mathcal{O}(n^2)$ complexity.

In indirect encoding, some important features of the neural network topology are considered in the chromosome, instead of the full connectivity pattern. It leads to a more compact encoding when compared with the direct one. Yao describes three main approaches to this encoding that had been explored as of 1993: connectivity parameters (specifying a set of parameters that characterize the topology of a neural network, such as the number of layers, the number of nodes in each layer, etc.), developmental rules (recursive equations or production rules, such as those found in generative grammars, that can be used to build a topology) and, to a lesser extent, fractal representations of connectivity (inspired by some of the processes of biological development, see Merrill and Port [78]).

Yao also remarks that some works also encode learning parameters, such as the learning rate. Some have been mentioned in the previous section; e.g., the work by Harp et al. [46] in 1989.

In 1994, Gruau [42] presented his doctoral dissertation, where he suggested representing ANNs via cellular encoding, using grammar trees to describe the network's structure which would be optimized using GAs.

Also that year, Angeline et al. [2] presented GNARL, which stands for GeNeralized Acquisition of Recurrent Links, an evolutionary programming-based approach with direct encoding for evolving the structure and weights of a recurrent neural network. In evolutionary programming, the recombination operator is not used, and only mutation is performed to obtain new individuals. In GNARL, the number of input and output neurons are defined by the problem, and the number of hidden units varies from 0 to a user-defined maximum $h_{\max}$. Neurons are not previously assigned to layers, so there can be any recurrent connectivity pattern. In fact, individual nodes or groups of nodes can remain disconnected from the inputs and outputs neurons, thus being ignored when constructing the neural network.

In 1997, Vonk et al. [127] published a book describing the application of NE in automatic generation of ANN topologies. They studied GAs for optimizing the weights of the network, and GP and GAs with grammar encoding for the topology. As for GP, they had previously presented GPNN [128]. They tested their approach only in two simple problems (XOR and one-bit adder), but admitted that the system did not scale out well for real-world problems.

5.3 Consolidation of neuroevolution

One the most relevant early approaches of neuroevolution arose when Yao and Liu [137] proposed EPNet in 1997, a system based on evolutionary programming for evolving artificial neural networks using direct encoding. In EPNet, both architecture and weights are evolved simultaneously, and authors noted that they put their focus on evolving the behavior of neural networks, keeping a behavioral link between parents and offspring during the evolutionary process. This is one of the motivations for which they chose evolutionary programming over genetic algorithms. Because they used direct encoding, the user needed to specify a maximum number (N) of hidden nodes allowable in the network. The specification of the neural network requires two matrices, each of them with dimension $(m+N+n)\times(m+N+n)$, being m the number of input nodes and n the number of output nodes; and a binary vector of length N. The first matrix is a binary connectivity matrix determining the existence or not of a link between two nodes, whether the second matrix specifies the connection weights. The vector specifies whether nodes exist or not in the network. Because Yao and Liu decided to constrain the search space to feed-forward networks, some constraints can be imposed on these matrices: Only the upper triangle of the matrix will be used, and the connections between input nodes will be enforced to 0.

The main innovation of EPNet is the mutation scheme: Yao and Liu came up with a sequence of mutations that were only executed if the previous action did not improve the network performance, comprising the next stages: (1) hybrid training of the connection weights using backpropagation with a custom learning rate or simulated annealing, (2) deleting one or more hidden nodes by setting the corresponding bits in the vector to 0, (3) deleting several connections, based on their importance, by setting the corresponding bits in the connectivity matrix to 0, and (4) adding new connections with a small random weight, and new nodes using cell division [91]. The mutation scheme in EPNet gives preference to changes in the connection weights rather than to architectural modifications. Also, when the topology is mutated, removal of nodes and connections is preferred over addition, to reduce the ANN size during evolution. Finally, after evolution concludes, further training is performed using both the training and validation sets with backpropagation.

After proposing EPNet, Yao and Liu validated their proposal using different real-world problems, including: the N parity problem, the two-spiral problem, medical diagnosis problems (including breast cancer, diabetes, heart disease and thyroid data sets), the Australian credit card assessment problem and MacKey–Glass chaotic time series prediction problem. Authors concluded that EPNet led to very competitive results because of the few constraints imposed on the network architectures, thus resulting in a large search space. However, they admitted that EPNet involved many user-defined parameters.

In 1999, Yao [136] reviewed the field of NE and pointed out some future directions for the new millennium. Yao also established a general framework for NE in different levels: connection weights, topology and learning rules. Finally, he concluded that using EAs at the three levels can be computationally expensive, and suggested that it is a better idea to use these techniques only in some levels.

In 2002, Stanley and Miikkulainen [115] presented NeuroEvolution of Augmenting Topologies (NEAT), a solution which would become one of the most cited and used systems in NE. NEAT evolved both the topology and the weights of the neural network using genetic algorithms with a direct encoding. Unlike in the working scheme of evolutionary programming, genetic algorithms include a recombination operator in order to perform the crossover between two parents to generate offspring. For this reason, the encoding must be thought in order to ease recombination, and authors stated that NEAT's encoding eases lining up corresponding genes when two genomes cross over during recombination. Networks were encoded using two vectors: one for nodes and other for connections. The nodes vector includes a list of input, hidden and output nodes, which can be connected. The connections vector include

the input and output nodes of the connection, its weight, whether it is enabled or not, and a so-called *innovation number* (described later).

In NEAT, mutation could affect both weights and topology. In the former case, weights evolved following the standard mutation scheme in GAs. In the latter case, structural mutations could either add connections or add new nodes. When adding connections, a new item was added to the connections vector with a random weight between two existing, unconnected nodes. Adding nodes was slightly more complicated: A new position is added to the nodes vector and then, a random connection c is chosen. To be more specific, if we considered n_i to be the input node of connection c, n_o to be the output node of that connection, and n_m the new node created during mutation; then, connection c is disabled in the connections vector and two new connections are added: c_1 linking n_i to n_m, and c_2 linking n_m to n_o.

Regarding the aforementioned *innovation number*, it is an incremental value added to each connection when created, indicating at which stage of the evolution process the gene appeared. In order to perform crossover, two individuals were lined up based on their innovation numbers, leaving gaps when the innovation number in one parent is not present in the other (see Fig. 4 in the original NEAT paper [115]) and offspring were composed by randomly choosing from either parent at matching genes. Disjoint and excess genes (those in one parent with innovation numbers not present in the other parent) would always be included from the fittest parent.

As opposed to EPNet [137], NEAT's mutation scheme always increased the network size; therefore, the acronym stands for "*augmenting topologies*." This would allow the search to start in a space of reduced dimensionality, increasing the search space only when required and leading to minimal topologies. After evaluating its performance, the authors concluded that NEAT was powerful for evolving ANNs, being more efficient than other NE techniques.

The approach used in NEAT led to some novel techniques such as HyperNEAT by Stanley et al. in 2009 [114], where an indirect encoding was used in order to evolve *compositional pattern-producing networks* (CPPNs), enabling the efficient representation of large-scale ANNs (with over eight million connections) or ES-HyperNEAT by Risi and Stanley in 2012 [100], a work similar to the former but automatically deducing node geometry.

In 2005, Kassahun and Sommer introduced Evolutionary Acquisition of Neural Topologies (EANT) [59], a work closely related to NEAT [115] and the work by Igel [55]. In particular, they evolved the network starting from a minimal topology and the weights using CMA-ES (Covariance Matrix Adaptation Evolutionary Strategy, refer to the work by Hansen [45] for further details). However, the authors considered that their main contribution is the encoding, which allowed to be evaluated without decoding it: The genome in EANT is a linear sequence of genes which can represent different entities of the ANN, either a neuron, an input neuron, or a connection (feed-forward or recurrent) between two neurons. All genes stored the weight between the neuron they represented and the neuron to which it was connected, and connection genes (or "jumper genes") also contained one number specifying the neuron to which it was connected.

The main advantage of their work is the proposal of a single theoretical and mathematical framework called *common genetic encoding* (CGE). This framework was formally proved to be complete, as it was able to represent all possible phenotypes, and closed, since every valid genotype represents a valid phenotype [58].

In 2007, Siebel and Sommer proposed EANT2 [110], improving the efficiency of EANT. EANT2's performance was evaluated by evolving a network which must control a robotic arm equipped with a camera in order to steer the arm to a certain object, and compared against NEAT's performance, showing that EANT2 obtained a higher fitness.

NE is a highly prolific field, and many works have been published across almost 3 decades. Many significant works and contributions have been gathered in previous surveys of the state of the art; e.g., Floreano et al. in 2008 [35] or Ding et al. in 2013 [30].

In recent years, some novel open-source NE frameworks have appeared. The availability of source code allows researchers to work on their own implementations and applications. One of these frameworks is MABE, whose source code is currently maintained and receives frequent contributions [50]. The foundations of the MABE framework were presented in 2011 by Edlund et al. [31], and the framework is general enough as to support different encoding methods and optimization techniques.

6 Present: drifting toward deep learning

In the previous section, we have described almost 3 decades of research in neuroevolution. However, during most of this time neural networks were relatively small, in some cases containing only one hidden layer with few hidden units or few recurrent connections, due to the limitations in computational power.

The recent emergence of deep and convolutional neural networks has brought back the need for designing topologies that are suitable to tackle specific problems. However, DNNs can have dozens of feed-forward or recurrent layers with thousands of units, and neurons can implement

diverse activation functions. Besides, CNNs can also have several convolutional layers with thousands of filters of various sizes, apart from different pooling and padding setups. As a result, DNNs and CNNs can have hundreds of thousands or even millions of weights, and innovations must be introduced in NE for it to adapt to these new topologies.

The number of works studying the application of NE to the optimization of deep and convolutional neural networks is still very scarce due to the novelty of CNNs and the computational cost of training and testing the performance of these networks, although it has grown significantly during 2017 and 2018. In this section we will cover the most relevant works in this new, rapidly evolving field.

6.1 The origins

An early approach of neuroevolution to deep learning was proposed by Koutník et al. in 2014's edition of GECCO [61]. In the abstract, authors state that their work "is the first use of deep learning in the context of evolutionary reinforcement learning," and to the best of our knowledge, it is also the first attempt to evolve convolutional neural networks published in the proceedings of a flagship conference; although earlier works had used grid search or bayesian optimization to find a small set of optimal hyperparameters (Snoek et al. [112] or Bergstra et al. [13]). However, in this work, the topology of the neural network is not encoded, but rather a fixed architecture is used comprising four convolutional layers with max pooling and finally a small recurrent network with three hidden units. The weights of the convolutional layers that eventually output feature vectors from raw inputs, and the weights of the recurrent neural network are learned separately. In the former case, 993 weights were optimized to maximize the variance of output representations, encoding them in a real-valued genome evolved using CoSyNE [41]. As for the recurrent neural network, it comprises only 33 weights, which are encoded and evolved using the same mechanism while fixing a sigmoid activation function.

Another work was published in 2015 by Verbancsics and Harguess [126], where they proposed a modification on HyperNEAT [114] to support the evolution of convolutional neural networks, by adding a new CNN substrate able to represent this kind of networks. The methods were briefly described in a preprint published in arXiv in 2013 [125]. In their approach, they use their variation of HyperNEAT to learn the weights of a feature extractor shaped as the convolutional layers of a CNN (resembling the LeNet-5 architecture [69]). The output of this feature extractor, which is a vector of features, is then introduced to a multilayer perceptron which is trained using classical backpropagation. However, their experiments using the evolved network over the MNIST dataset led to a test error rate of 7.9%, which is one order of magnitude higher than the performance of most CNN-based works.

Also by the end of 2015, Young et al. [139] introduced multi-node evolutionary neural networks for deep learning (MENNDL), a framework for optimizing the hyperparameters of a neural network using genetic algorithms, with a focus on high-performance computing. From the evolutionary perspective, their proposal turned out to be very simple, evolving only six hyperparameters: the number of filters plus the filter size for a fixed 3-layers convolutional architecture. Their proposal was tested against the CIFAR-10 dataset, but the final error rate is not reported. Same authors [138], later in 2017, published a novel work where the expressiveness of the genetic encoding was improved, allowing for a variable number of layers (using on–off bits) and evolving up to eleven parameters in convolutional layers, as well as hyperparameters in pooling and fully connected layers, including the evolution of activation functions. Nevertheless, the paper provided just a few details on the encoding, and the focus was again placed on the high-performance computing.

During 2016, three related works were published. The first was published by Loshchilov and Hutter [75] where they propose using CMA-ES (covariance matrix adaptation evolution strategy) to evolve the hyperparameters of a deep neural network. In particular, 19 hyperparameters are considered including optimizer parameters (learning rate, momentum, etc.), batch size, dropout rate, number of filters in the convolutional layers or number of units in the fully connected layer. However, the number of layers is fixed prior to the evolutionary process, and most of the hyperparameters involved are related to the optimizer rather than the topology. In fact, neither filter sizes or activation functions are evolved, and recurrent layers are not included in the search space. Authors report performance of their proposal in the MNIST and CIFAR-10 databases, with approximate error rates of 0.27% and 9.3%, respectively (values are not shown in the text, and these values are inferred from the figures provided by the authors). This seems that the first time such competitive performance is obtained by applying neuroevolution to CNNs, although it seems from the authors' description that this error rate refers to the validation set instead of the test set.

The second work was published by Fernando et al. [34], from Google DeepMind. In their proposal, they suggest the creation of a differentiable version of a compositional pattern-producing network (a type of neural network suitable to be evolved using augmenting topologies approaches, such as NEAT), which they call DPPN and evolve using genetic algorithms with three different types of mutation (add random node, add random edge, and remove random edge) and crossover. Their solution is innovative as

they use direct encoding to simultaneously evolve the topology and the initial weights of the connections, eventually generating a convolutional architecture which is embedded within a fully connected network, with filters included in each hidden unit. This network is then evolved using backpropagation. Authors report results on the MNIST database, but instead of working on a classification problem, they explore the problem of image reconstruction using the evolved network.

The third work was published by Tirumala et al. [122], who suggest the use of a coevolutionary algorithm, comparing both a competitive and a cooperative version. The ways in which solutions are encoded and the evolutionary process is carried out are not described with great level of detail, but authors only evolve the weights of a fixed architecture with 5 fully connected layers, not including convolutional layers. Authors report a test error rate over the MNIST database of 1.2% with cooperative coevolution and 3.7% with competitive coevolution.

6.2 The rise

Most of the works evolving CNN topologies started to appear during 2017, leading to a rise in the field of neuroevolution of deep learning networks. Some of these works are only available in preprint repositories, such as arXiv. Next, we will describe each of these works in further detail.

6.2.1 GeNet

Xie and Yuille [134] have worked on a GA to evolve the topology of a CNN to perform visual recognition. Authors considered a constrained case with a limited number of layers, with already predefined building blocks, such as convolution or pooling, and recognize the need to perform heuristic search in order to find the optimal topology for their needs.

GeNet's proposal is interesting, since it leads to convolutional structures which differ from widely used sequential architectures. In a nutshell, it could be seen as if each convolutional layer (called a "stage" in GeNet) had a whole acyclic graph of convolutional operators. Nodes in each stage are connected and numbered, and connections are only allowed from one node to another with a higher number in order to avoid cycles. The input to the stage is always introduced to the first node, which applies a convolution operation over it. In all other nodes, tensors coming from the different input nodes are aggregated via element-wise summation, then convolution is performed, and finally batch normalization and ReLU are followed. Convolution operators within the same stage will share the same width, height, and number of filters, and stages are separated by pooling.

GeNet encodes the network structure as a fixed-length binary string by dividing the network in S stages, where the s-th stage ($s \in \{1, 2, \ldots, S\}$) contains K_s nodes, denoted as $v_{s,k_s}, k_s \in \{1, 2, \ldots, K_s\}$. In the chromosome, each stage is represented with $1 + 2 + \cdots + (K_s - 1) = \frac{1}{2}K_s(K_s - 1)$ bits, where each bit represents the existence or not of a link between two nodes of the stage. Thus, the first bit encodes the existence of a link from $v_{s,1}$ to $v_{s,2}$, the next two bits encode the existence of links from nodes $v_{s,1}$ and $v_{s,2}$ to $v_{s,3}$, respectively, and so on and so forth until the last $K_s - 1$ bits encode the existence of links from $v_{s,1}, \ldots, v_{s,K_s-1}$ to v_{s,K_s}.

While this approach is innovative because convolutional operators can be performed in a non-sequential fashion, it has the drawback that many parameters must be defined prior to the evolutionary process, namely the number of sequences (S), the number of nodes per each sequence (K_s), the number of filters and their width and height for each sequence, and the size of the pooling operator. Thus, the search space is constrained, and many degrees-of-freedom could be added to the genetic search in the space of possible CNNs. Also, GeNet only evolves the structure of the convolutional layers, not evolving the fully connected or recurrent part of the CNN. Weights are learned using backpropagation.

Xie and Yuille evaluated the performance of GeNet using the SHVN, CIFAR-10, and CIFAR-100 datasets, obtaining a test error rate of 1.97%, 7.10%, and 29.03%, respectively. There results are not particularly competitive with the state of the art, yet authors state that networks evolved by GeNet are less deep than outperforming CNNs. Interestingly, authors conclude that "*generated structures, most of which have been less studied before, often perform better than the standard manually designed ones*," suggesting that evolution of CNN topologies is a promising area which is yet to be explored.

6.2.2 CoDeepNEAT

Miikkulainen et al. [82] have presented CoDeepNEAT, first posing an interesting thought: *"Human engineers can optimize a handful of configuration parameters through experimentation, but DNNs have complex topologies and hundreds of hyperparameters. Moreover, such design choices matter; often success depends on finding the right architecture for the problem. Much of the recent work in deep learning has indeed focused on proposing different hand-designed architectures on new problems."*

CoDeepNEAT follows the same working principles than NEAT, yet adapted to work with CNNs. In this case, nodes

in the chromosome no longer represent neurons, but whole layers in the CNN. Each node contains a table of real-valued and binary hyperparameters that are subject to mutation. These hyperparameters specify the layer type (convolutional, fully connected or recurrent) and its properties. Some of these properties are: number of convolutional filters, dropout rate, kernel size, number of neurons, activation function, etc. Also, connections no longer have weights, but just indicate how layers are interconnected. Finally, the chromosome also contains a set of global hyperparameters that do not belong to any specific layer, such as learning rate, momentum, etc. As for the fitness function, the genome is converted into a CNN whose weights are learned using a training dataset, and a performance metric is computed.

One interesting novelty in CoDeepNEAT is that layers in the CNN may not adhere to a sequential structure, allowing arbitrary connectivity between layers. Therefore, the ability of merging layers must be introduced when one layer has more than one input layers. As in GeNet, this can be done using element-wise summation, though it can require downsampling to the minimum layer size.

While the approach described so far is called "DeepNEAT," authors proposed a coevolutionary variant. First, they notice that some successful CNN architectures are composed of "modules" that are repeated several times, where these modules can have complicated internal structures (as it happened in GeNet stages). Therefore, these modules would be implementations of small neural networks, which can then be aggregated to form complex CNN topologies. The authors then establish the concept of a "blueprint," which is a graph describing how different modules are interconnected to form the final network. In this coevolutionary variant, which they call "CoDeepNEAT," both modules and blueprints are evolved simultaneously in order to create modular networks. This variant is called "CoDeepNEAT." Also, authors implemented the possibility to include LSTM units within the network.

Finally, authors tested CoDeepNEAT's performance using the CIFAR-10 domain. After data augmentation, they achieved a test error rate of 7.3%, which is not particularly competitive within the state of the art, though authors claim that the resulting topology converges much faster than better architectures. It is remarkable that CoDeepNEAT allows learning very complex networks involving convolutional, feed-forward and recurrent or LSTM layers, strongly relying on the mutation of these parameters.

6.2.3 EXACT

Desell [29] introduced EXACT (Evolutionary eXploration of Augmenting Convolutional Topologies) in a poster session in GECCO 2017. It is remarkable that most of the work describes how the algorithm is supported by a largely distributed architecture using volunteer computing.

To perform the evolutionary process, the author based his work on NEAT. Desell relied on the realization that the structure of the convolutional layers in a CNN can be evolved by solely determining the filter sizes and how they are connected, and in consequence he designed specific mutation and crossover operators. Regarding mutations, one or several of the following mutation operations are performed at each generation, depending on some user-defined hyperparameters:

- Disabling a random edge in the genome. If this led to unreachability of some output node, then the mutated genome is discarded and a new attempt at mutation is performed.
- Enabling a random edge in the genome.
- Splitting a random edge by creating a new node (just as it was done in NEAT) in the middle, with a filter size which is the mean of the filter sizes of the source and target nodes. Also, a depth value is included which is also the mean of both nodes.
- Adding an edge between two random nodes, given the condition that the edge goes from a node n_i to a node n_o where n_o has larger depth than n_i.
- Changing the filter size in a random node (by increasing or decreasing each dimension in either one or two units).
- Changing the filter size in a random node only in one dimension (similar to the previous one, but acting only in one of the filter dimensions).

As for crossover, edges will be added from the parents to the child with different probabilities depending on whether the parent is the fittest or the less fit of the couple; then, non-selected edges are also added, yet they are disabled in the child. Finally, nodes are added to the child, and when both parents share a node with the same innovation number, then the parameters will be chosen from the fittest parent.

EXACT allows obtaining complex structures involving many convolutional filters, although it does not evolve neither pooling operators, activation functions, fully connected or recurrent layers and other hyperparameters such as the learning rate. After training the network using backpropagation, Desell reported an error rate on MNIST of 1.68%, which unfortunately is significantly higher than most CNN-based approaches.

6.2.4 Large-scale evolution of image classifiers

Another work, also with the focus put in scalability was published by Real et al. [98], from the Google Brain team, in 2017. Authors use a genetic algorithm relying on a

tournament as the selection operator, which is deployed atop a massively parallel infrastructure. The topologies are encoded by means of a graph similarly as done in GeNet or DeepNEAT, with nodes this time being translated into convolutional layers. In the case that one node receives several inputs of different sizes, then authors will chose one such inputs as the primary, and then convert all other inputs to match the primary's size, either by padding (if their were smaller) or truncation (if they were larger). Besides this structure, the learning rate is also stored in the genome to be evolved.

Given a parent, Real et al. rely on mutations to generate the offspring. The set of allowed mutations include mechanisms for altering both the topology of the model or its training. When it comes to the topology, mutation allows for inserting a convolution node, removing a convolution node, altering the stride, the number of channels of the filter size of a convolution node, and inserting or removing a connection between random layers. Batch normalization and ReLU activation can be optionally applied after convolution, depending on the chromosome. Regarding the different aspects of the training process, mutation enables to alter the learning rate, reset the weights of the model or keep training with the previously learned weights. Authors insist on that these mutations were chosen because they resemble the design and refinement process that a human expert would follow. Reported error rates are 5.4% on CIFAR-10 and 23% on CIFAR-100. Interestingly, authors have also explored building an ensemble to test the models on the CIFAR-10 dataset, reducing the error down to 4.5%. They realized that very good solutions are found soon during evolution and are then slightly improved over the course of generations. This could mean that we could obtain good, yet not state-of-the-art models after a short time.

More recently, in 2018, Real et al. [97] have introduced the concept of a regularized evolutionary algorithm, where the oldest model is removed from the population in every generation. When compared against reinforcement learning and random search, authors conclude that neuroevolution and reinforcement learning perform similarly well, although the former is faster, and both considerably surpass random search. It is worth mentioning that authors report running large-scale experiments in 450 GPUs over a week, and dedicated evolution experiments in 900 TPUv2 during five days.

6.2.5 DEvol

Joe Davison, from Microsoft, presented in 2017 an open-source project called DEvol [26], for automated CNN design via genetic programming. We have not been able to find a publication describing this project.

In his proposal, the genome connects several nodes sequentially, each node representing a layer. Hyperparameters for each layer are also evolved, including the number of filters, the dropout rate, the activation functions, etc. From the code documentation, it can be inferred that DEvol supports a variable number of convolutional and dense layers. When tested over the MNIST dataset, they have achieved a test error rate of 0.6%, which is fairly good yet not competitive with the state of the art.

6.2.6 Genetic programming for CNN design

Also in 2017, Suganuma et al. [117] published a work using Cartesian GP to optimize CNN architectures, which was a best paper candidate in GECCO 2017. In their proposal, authors represent the network as an acyclic graph, with nodes defined on a 2-dimensional grid with N_r rows and N_c columns for a total of $N_r \times N_c$ nodes. These nodes will become operators, which can be of different types as we will define later. The number of rows defines the level of parallelism, meaning that nodes in the same column are executed in parallel, whereas rows conform the sequential aspect of the network, meaning that nodes in one column are executed after the previous column. In fact, Suganuma et al. allow connections to "jump over" some layers, forcing that input connections of a node in column N_{c_i} can come from nodes in columns from N_{c_i-l} to N_{c_i-1}, where l is the "levels-back" parameter (the maximum number of layers in the past that can be considered for the input of the current layer).

Within each cell in the grid, there are several integers that encode the type and connections of the node associated with that cell. The meaning of these integers will vary depending on the type of the node. Authors have considered the following types of operators (which they call "blocks"): convolutional, residual (which combines convolution and tensor summation), max pooling, average pooling, concatenation and summation. These operators allow to resemble complex non-sequential network types such as GoogleNet or residual networks [48]. The output of the last node (which will be of an aggregation operator, such as summation or concatenation) will then be passed to a softmax classifier. Nodes can be set inactive in the genome, meaning that even if the grid size is established prior to begin the evolution, the number of nodes can vary. The evolutionary process is based on a modified $(1 + \lambda)$ evolution strategy, relying only on genome mutations.

This approach is flexible regarding the possible architectures that can be expressed. However, their approach only evolves convolutional layers, not observing fully connected or recurrent layers, nor hyperparameters

optimization. Authors have reported a test error rate on the CIFAR-10 database of 5.98% with data augmentation.

6.2.7 Hyperparameter optimization using evolution strategies

By the end of 2017's summer, Bochinski et al. [15] published a work describing a system based on evolutionary computation to optimize the hyperparameters of CNNs, which they call IEA-CNN. In their proposal, authors evolve a good number of hyperparameters, including the number of filters and the filter size in convolutional layers and the number of neurons in fully connected layers. Pooling is not included as an option, and other training hyperparameters are manually established by the authors.

An interesting design decision involves sorting the evolved layers by descending complexity; i.e., layers are first evolved and then sorted to form the network, so that first layers will be those with more filters or more neurons, reducing the search space factorially on the number of layers. For the evolutionary algorithm, authors implement a $(\mu + \lambda)$ evolution strategy, therefore encoding the hyperparameters in a vector of real numbers.

Besides, authors also suggest how to extend this framework to allow for the joint optimization of CNN committees, by using a fitness function that takes the global classification error of the population, and naming this alternative CEA-CNN.

Finally, authors reported a test error rate in the MNIST database without data augmentation of 0.34% using an individual model and 0.24% using a committee of 34 CNNs, this last result being extremely competitive among the state of the art.

6.2.8 EvoCNN

In October 2017, Sun et al. [118] published a preprint paper describing the use of a GA for evolving deep CNNs. A remarkable feature of this proposal is that the encoding supports variable-length chromosomes, therefore providing a natural encoding for different numbers of layers.

The hyperparameters evolved by EvoCNN include the number of filters and their size, the stride size and the convolution type for convolutional layers, the filter and stride size and pooling type in pooling layers, and the number of neurons in fully connected layers. Besides, weights are also evolved, but instead of evolving all weights, the mean value and standard deviation are evolved instead for each layer, and weights are later randomly assigned following a Gaussian distribution.

Because the chromosome can be of different lengths, a specific crossover operator was implemented in order to allow recombination, which involves aligning parent chromosomes of different lengths by separating the list of convolutional layers, pooling layers and fully connected layers. Also, a specific environmental selection operator is introduced to promote diversity in the population, which is combined with elitism (conserving the best individuals across generations) to help improve the fitness. Also, authors suggest using an efficient fitness computation which reduces the amount of training epochs for each individual, using the mean classification error and standard deviation over the different validation batches in the last epoch as the fitness value. Standard deviation is only used in case of a tie when sorting individuals by mean error.

Authors apply EvoCNN to a variety of image recognition domains, including many MNIST variants described by Larochelle et al. [66]. In basic MNIST, they report a test error rate of 1.18%, which is large compared to the state of the art.

6.2.9 Grammatical evolution of CNNs

More recently, in 2018, Baldominos et al. [7] proposed a system which was able to evolve many different aspects of the convolutional neural network, including convolutional, fully connected and recurrent layers, activation functions and other learning hyperparameters. They tested two different approaches: one based on genetic algorithms and another using grammatical evolution, a particularization of a genetic algorithm where a generative grammar is used to map the chromosome into a CNN topology.

For the GA, authors propose a binary encoding with a fixed-size chromosome, enabling a variable number of layers by using activation bits for each layer. In the case of convolutional layers, the number of filters, their size, and the activation functions are evolved, as well as the existence of pooling layers and their size. In fully connected layers, the hyperparameters subject to optimization are the layer type (feed-forward or recurrent), the number of units, the activation function, and whether regularization or dropout is used. In the case of the grammatical evolution, authors use roughly an equivalent search space, just exploiting the improved encoding provided by this technique.

One of the innovations introduced in this work is the use of a niching scheme in the evolutionary algorithms to enhance diversity in the population, with authors claiming that this added diversity improves the performance of the process.

This work observes many different aspects of optimization, although it is constrained to sequential networks, unlike other works described before, which could allow more complex structures. Authors have tested their proposal with MNIST obtaining a test error rate of 0.37% without data augmentation nor preprocessing. In a more

recent work published in late 2018 [9], authors proposed an improvement of the previous system by evolving committees of CNNs in order to improve the performance, reducing it to 0.25%. Also, the same system with a different encoding was used to tackle a human activity recognition problem [8].

6.3 The settlement

So far we have seen some relevant works which have started the application of neuroevolution to deep and convolutional neural networks. As of 2018, the field is settling and the number of works is starting to grow significantly. In this section, we will briefly describe the most relevant of these works.

6.3.1 Hierarchical representations

Liu et al. [74] from Carnegie Mellon University and DeepMind have published in ICLR 2018 a work representing neural networks by means of the computation graph, with a single input and a single output. Therefore, the architecture can be defined as a tuple $(G, \mathbf{o})$, where $\mathbf{o} = \{o_1, o_2, \ldots\}$ is the set of available operations and G is the graph identified by its adjacency matrix, where $G_{ij} = k$ means that the k-th operation (o_k) will be executed in between nodes i and j.

Then, authors use a genetic algorithm for evolving individuals using mutation, which allows adding, removing, or editing edges in the graph. The set of operations proposed in this work allows the creation of convolutional and pooling layers, although they do not evolve fully connected or recurrent.

6.3.2 Lamarckian evolution

In 2018, Prellberg and Kramer [94] presented a new neuroevolution work in PPSN 2018. In both cases, a $(1 + 1)$ evolution strategy is used to evolve the convolutional layers in a CNN. In their approach, the evolutionary algorithm relies only on a mutation operator, which is intended to improve one individual one generation at a time. The mutation can add a new building block (which is equivalent to a convolutional layer) with a random number of filters, a random filter size, and a stride of one, or modify some of the hyperparameters of such building block by adding or removing filters or changing the filter size or the stride. Mutation can also delete a building block.

In this work, a mechanism is introduced to support weights inheritance, so that once a network is trained, its child can reuse the weights, except for the new layers and filters resulting from mutation, where the parameters are randomly initialized using Glorot initialization [39].

This work is closely related to one previously published by Kramer [64] in the same year. In the work published by Kramer alone, fully connected layers and activation functions are evolved, and parameters are learned using gradient descent. However, in the work of Prellberg and Kramer, authors simplify the previous one by fixing fully connected layers at the end of the network as well as activation functions.

In both works, a niching scheme and mutation rate control are used for supporting the evolutionary procedure. Kramer reported a maximum MNIST accuracy of 99.1%, and the work by Prellberg and Kramer attained an accuracy of 89.3% in CIFAR-10 and 66.1% in CIFAR-100 without data augmentation. These results are not state of the art, but according to authors they are not intended to be, since they focus on showing the advantages of weights inheritance.

Also in June 2018, Prellberg and Kramer presented a different approach [95] where a GPU-optimized evolutionary algorithm evolved the weights of a CNN (a total of 92,000 parameters), despite that they only obtain a 98% accuracy on the MNIST dataset.

6.3.3 DENSER

Another work has been presented by Assunção et al. [3], called DENSER (deep evolutionary network structured representation). In this work the evolutionary algorithm optimizes the topology of the network and the activation functions. Authors claim that it can be used also to evolve learning hyperparameters and, as a novelty, the hyperparameters of the data augmentation stage (a procedure widely used in computer vision to synthetically enlarge the dataset by applying transformations to the original images). However, it is not clear how this is done given the described genetic representation.

An interesting approach of their work is that the representation of candidate solutions is done at two different levels. In the first level, the topology of the neural network is encoded in a chromosome and evolved using a genetic algorithm. The genome can be interpreted as 3 tuples where the first element is an initial non-terminal symbol in a context-free generative grammar, and the second and third elements refer to the minimum and maximum number of times that the grammar will be used to generate a string with such an initial symbol. In the second level, grammatical evolution is used to determine the hyperparameters of each layer based on the previously mentioned grammar. This representation allows for a variable number of layers of different kinds (convolutional, pooling, and fully connected) and the number of supported hyperparameters is large, including the number of filters, the filter size,

activation functions, the use of batch normalization, or the number of units in fully connected layers.

In the evolutionary process, DENSER relies on both crossover and mutation. Different mutations are supported: In the case of the GA level, mutations can add, replicate, or remove layers, whereas in the grammatical level mutations allow to generate a different string or to mutate a numeric value of the individual.

Assunção et al. have used the CIFAR-10 dataset in order to evolve the network topology and then have transferred such topology to other computer vision datasets. In particular, they report test error rates of 0.35% in MNIST without data augmentation, 4.74% in CIFAR-10, and 21.25% in CIFAR-100.

6.3.4 DECNN

In late 2018, Wang et al [129] proposed the application of a hybrid differential evolution algorithm to the optimization of convolutional neural networks, an approach they called "DECNN." They use an interesting encoding strategy based on internet protocol (IP) addresses. In this encoding, the hyperparameters of different layer types (convolutional, pooling, and fully connected) are first encoded in a binary string. Authors use 12 bits for convolutional (number of filters, filter size and stride size), 5 bits for pooling (kernel size, stride and pooling type) and 11 bits for fully connected (number of neurons). These binary strings will then constitute a "subnet" (making the analogy with networking), which is its search space. Authors state that the fact that pooling layers are represented by only 5 bits makes them less prone to be chosen, and therefore, they add 6 more bits as a placeholder, so its final size is also 11 bits. Finally, authors add a prefix bit string exclusive to each layer type, finally encoding each subnet as a 2-byte (16-bits) IP address.

Then, differential evolution is started by initializing a random population, which involves the creation of individuals of different length and is followed by applying standard mutation and crossover, with the only particularity being that in crossover parents are trimmed to the shortest length in order to generate offspring. Additionally, authors have included a second crossover operator with the purpose of taking two parents of the same size and generating two children of different lengths, an effect that is achieved by choosing a different cutting point in each of the parents' chromosomes.

Authors test their proposal in different MNIST variations, achieving a test error rate of 1.08% in basic MNIST, which is rather high when compared to the state of the art.

6.3.5 AE-CNN

Sun et al. [119] have recently published a preprint where they propose the evolution of CNNs using a genetic algorithm with two building blocks: ResNet block (standing for residual network) and DenseNet. The ResNet block is formed by three convolutional layers and a skip connection that effectively combines the input to the block with the output of the last layer using tensor summation, whereas the DenseNet block is formed of four convolutional layers, forcing that the input to each layer is the combination of all previous outputs (e.g., the input to the fourth layer is the combination of the feature maps produced by the first, second and third layers).

The evolutionary algorithm used, which they call "AE-CNN" (automatically evolving CNNs), is mostly a canonical implementation of a genetic algorithm, yet the best individuals for both the parents population and the offspring population are selected for the next generation to take place. The GA encoding is a chromosome of so-called units, which can be either ResNet unit, DenseNet unit, or Pooling unit. The former two can contain a variable number of ResNet blocks and DenseNet blocks, respectively, while the latter consist of a single pooling layer. Each unit is encoded using an integer that determines its type from among these three as well as parameters for each unit (the number of blocks and the input and output size in the case of ResNet and DenseNet, and the pooling type in the case of pooling units). Chromosomes can have variable length, and authors propose the use of single-point crossover, choosing different cutting points in each of the parents. Mutations can be of three different kinds: add a new unit to the chromosome, remove a unit, or modify the parameters of a unit.

The advantage of AE-CNN is that a short encoding allows for very deep neural architectures. This is because a single gene in the chromosome (a unit) can lead to the generation of several blocks which at the same time are formed by three or four convolutional layers. On the contrary, this does not really allow for fine-tuning of the CNN, and simple architectures which work very well for many problems do not have the chance to be generated. Also, although complex structures can be evolved, the number of hyperparameters considered is small, affecting mostly the input and output sizes of the convolutional layers.

Sun et al. test the performance of AE-CNN over the CIFAR-10 and CIFAR-100 databases, attaining test error rates of 4.7% and 22.4%, respectively, which are competitive with the state of the art.

6.3.6 Evolutionary gradient descent for DNNs

Cui et al. [23] from IBM Research presented a paper in NIPS 2018 proposing the use of evolutionary computation to complement stochastic gradient descent (SGD) during the training phase of a deep neural network. In particular, they alternate SGD and the evolutionary algorithm at each generation. By using SGD, each individual is optimized independently, and then, they interact as part of the evolutionary stage.

In the process, which they call evolutionary stochastic gradient descent (ESGC) the initial population is first created. In every generation, SGD is first performed by applying an optimization to each individual in the population. If the fitness of an individual degrades after this phase, then the parameters will be rolled back to the previous generation. In the evolutionary step, individuals are combined using crossover and mutation to generate offspring, like in a regular genetic algorithm. Finally, the best individuals from both the parents and the offspring are chosen to be part of the following generation.

This proposal is not evolving any aspect of the topology; instead, it is focused on evolving only weights, which are encoded in a real-valued vector. Authors test their system on different domains, including speech recognition, image recognition, and language modeling. In the case of CIFAR-10, they report a test error rate of 8.34%.

6.3.7 Neuroevolution of CNNs for reinforcement learning

All of the works we have seen so far have been ultimately tested in supervised learning scenarios, commonly using well-known classification datasets for benchmarking, such as MNIST or CIFAR. However, in 2018 Such et al. [116] from Uber AI Labs published a preprint describing the application of evolution of CNNs for solving a different problem: reinforcement learning.

In their work, authors purposely design a very simple implementation of a genetic algorithm to test its performance. They focus on optimizing weights instead of the topology, but because storing all parameters in a real-valued vector is impractical in very deep networks because of the extremely large size of the search space, authors propose a compressed representation whose size increases linearly with the number of generations (in the order of thousands) and it is independent from the number of parameters (which is often in the order of millions), at the cost of requiring additional computation to generate the model from the chromosome.

Authors have tested their approach on different reinforcement learning problems: learning to play Atari games directly from pixels, solving an Atari-scale maze, and a control problem involving a humanoid robot learning to walk. According to their results, the GA is not particularly competitive with other techniques, leading to worse solutions or similar solutions requiring higher training time, although it performs very well in some of the Atari games.

6.4 Summary

In this section, we have reviewed the field of neuroevolution applied to deep and convolutional neural networks. This is a very new field which dates back to 2014 and started to rise in 2017, and is currently settling, with a considerable number of works testing different proposals. The importance of this field can be observed given the amount of works which are started to be published in top-tier conferences such as GECCO, ICLR, and NIPS and in journals, and an increasing number of papers can be found in preprint repositories.

The importance of the field is such that in recent months even a dedicated chapter in a Springer book [54] was published about this topic, although it only surveys three works from among all that were considered in this work.

Because every work introduces some novelties and places the focus on different aspects to be optimized, we have included a comparison of all reviewed works in Table 1 including whether they support a variable number of layers (VL), whether they evolve convolutional layers (C), fully connected layers (FC), recurrent layers or LSTM cells (R), activation functions (AF), learning hyperparameters (H), whether they support the creation of ensembles (E), or whether weights are evolved along with the topology (W). We also report the performance of these works over well-known image recognition databases when available, since these datasets are widely used as a benchmark for comparing CNN models.

Also, a graph is depicted in Fig. 5 showing how some of the neuroevolution works described in this paper depend on each other. This graph only shows arrows for those works that extend or strongly relies on others, not showing a dependency when some works only share some similarities in their methods. NEAT occupies a key position in this graph, since there it has strongly influenced many works, including some aiming at the evolution of CNNs. Also, it can be seen how many of the recent works are decoupled. This can be due to the fact that neuroevolution of CNNs is very recent, and the community is still exploring different approaches and techniques before starting to exploit some of the already available.

It can be seen how most works allow evolving the design of the convolution operators and support a variable number of layers. However, the approach used varies significantly from work to work. On the other hand, the number of works allowing the evolution of fully connected layers, recurrent layers, activation functions, and

Table 1 Comparison of different DNN and CNN neuroevolution techniques, including their supported features and error rates over well-known benchmarking datasets

Work	Method	Ref	VL	C	FC	R	AF	L	E	W	M (%)	C10 (%)	C100 (%)
The origins													
Koutník et al. [61]	ES	Section 6.1								•			
Verbancsics and Harguess [126]	GA	Section 6.1								•	7.9		
Young et al. [138, 139]	GA	Section 6.1	•	•	•		•						
Loshchilov and Hutter [75]	ES	Section 6.1		•				•			0.27	9.3	
Fernando et al. [34]	GA	Section 6.1		•	•								
Tirumala et al. [122]	GA	Section 6.1								•	1.2		
The rise													
GeNet [134]	GA	Section 6.2.1		•								7.1	29.03
CoDeepNEAT [82]	GA	Section 6.2.2	•	•	•	•	•	•				7.3	
EXACT [29]	GA	Section 6.2.3	•	•							1.68		
Real et al. [98]	GA	Section 6.2.4	•	•				•	•			4.5	23
DEvol [26]	GP	Section 6.2.5	•	•	•		•				0.6		
Suganuma et al. [117]	ES	Section 6.2.6	•	•								5.98	
CEA-CNN [15]	ES	Section 6.2.7	•	•	•				•		0.24		
EvoCNN [118]	GA	Section 6.2.8	•	•	•					•	1.18		
Baldominos et al. [7, 9]	GA	Section 6.2.9	•	•	•	•	•	•	•		0.25		
The settlement													
Liu et al. [74]	GA	Section 6.3.1	•	•							3.6		
Prellberg and Kramer [64, 94]	ES	Section 6.3.2	•	•	•		•				0.9	10.7	33.9
DENSER [3]	GA	Section 6.3.3	•	•	•		•				0.35	4.74	21.25
DECNN [129]	DE	Section 6.3.4	•	•	•						1.08		
AE-CNN [119]	GA	Section 6.3.5	•	•								4.7	22.4
Cui et al. [23]	GA	Section 6.3.6								•		8.34	
Such et al. [116]	GA	Section 6.3.7								•			

Ref, the reference of the section where the work is described; *VL*, variable number of layers; *C*, convolutional layers; *FC*, fully connected layers; *R*, recurrent layers; *AF*, activation functions; *L*, learning hyperparameters; *E*, ensembles or committees; *W* weights; *M*, MNIST; *C10*, CIFAR-10; *C100*, CIFAR-100

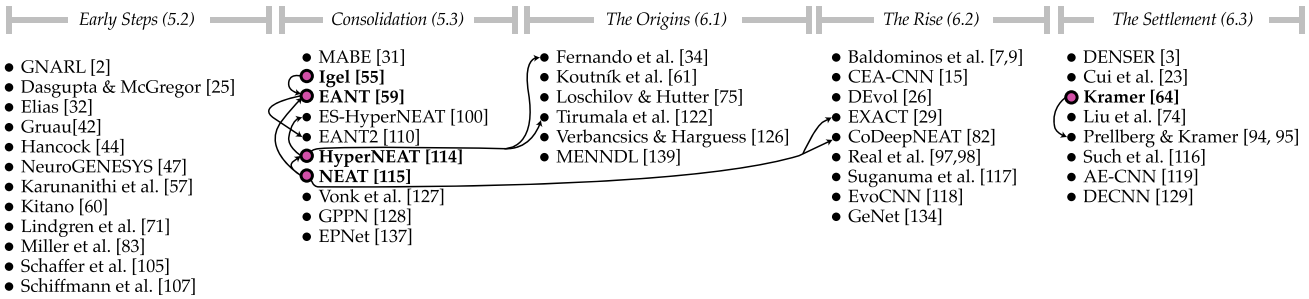


Fig. 5 Graph showing how some of the neuroevolution works explored in this survey depend on previous work. Over each group of works, the section of the paper where those works are described can be found

optimization hyperparameters is very limited. Only two works, CoDeepNEAT and the work by Baldominos et al., are shown to be very complete and flexible approaches for neuroevolution of CNNs.

7 Future

Neuroevolution has been widely used for almost 3 decades to automatically obtain neural network topologies and models using evolutionary algorithms. This field has been proved successful in many different domains and

applications and has received significant attention of the scientific community.

Nowadays, remarkable improvements in hardware architectures have popularized the use of more complex models, involving deep and convolutional topologies spanning several layers with a large number of hyperparameters. Unlike traditional feed-forward networks with only one hidden layer, CNNs can involve very complex topologies with many convolutional, feed-forward or recurrent layers, and supporting many different setups in each layer. Previous works have shown that the network topology can have a significant impact in its performance and, under the absence of analytic procedures for determining optimal topologies, neuroevolution shows as a promising approach to tackle the automatic design of CNN topologies.

Advances in GPU hardware and the development of specific deep learning primitives (e.g., NVIDIA's cuDNN [19]) and frameworks (e.g., Theano [12] or TensorFlow [1]) have made CNNs available even for personal use. However, one major weakness in the use of NE for the design of CNNs is the enormous resource consumption during the optimization process. NE requires the evaluation of thousands, or even hundreds of thousands, of different individuals, each of which involves a complete training process of a network with a complex topology and its evaluation.

Only in very recent years has been neuroevolution of CNNs shown to be a feasible technique, and thus, it is a promising yet mostly unexplored field. Whereas some neuroevolution approaches for traditional ANNs have reached a significant degree of maturity (e.g., EPNet or NEAT), the number of works where the evolution of CNNs is tackled is still small, yet growing. In this paper, we have summarized some of the most relevant milestones in the history of neuroevolution and have extensively focused on reviewing very recent works that have put the focus on the evolution of deep and convolutional neural networks. A timeline showing all these works and important landmarks at a glance is shown in Fig. 6.

The good news is that current trends in technology include manufacturing specific chipsets and product lines for deep learning (e.g., Google Tensor Processing Units—TPU—or NVIDIA Tesla), release of this hardware under an infrastructure-as-a-service model (e.g., Google Cloud TPUs [28] or more recently Amazon EC2 P3 instances [4]), or construction of large AI supercomputers (e.g., NVIDIA's DGX SATURNV, comprising 125 servers with a total of 1000 powerful GPUs optimized for deep learning [90]).

These innovations should constitute the fuel that empowers this new promising research area. As hardware resources get more affordable and accessible, the feasibility of training and testing thousands of CNNs increases and researchers can focus on exploring novel NE techniques or extending NE to unexplored domains.

For those cases where NE is used to attain the highest performance in terms of accuracy or other quality metric, then the evolution of ensembles must be considered. So far, very few works explore NE of ensembles or committees of ANNs. Still, in most cases, ensembles perform better than any of the models involved in them, turning this research line into a one which worth to be explored. The main advantage of evolutionary computation regarding ensembles is that population-based techniques lead to a very natural source of models to build a committee. Specific techniques from the field of quality diversity [96] can be considered and embodied into NE, leading to another promising research line that involves the study of the effects of increasing genetic diversity in the ensemble performance.

On the other hand, some works may want to focus not in attaining the highest performance, but rather in obtaining simpler or more efficient models, for example, in terms of time or energy expenditure. Decreasing time can help to obtain populations whose individuals are faster to train and validate, reducing the cost of fitness computations. As for energy expenditure, its reduction can be especially interesting when resulting models are going to be embedded in portable devices, such as smartphones or wearables.

In this case, multi-objective evolutionary algorithms are an interesting approach to optimize the different objectives, taking advantage of the benefits of NE described before. This line has barely been explored [76] and future work can settle the foundations for obtaining high-performance models which are also energy efficient or which involve minimal topologies.

It seems natural that the evolution of machine learning leads to learning more and more generic tasks. At the moment, most models learned are tailored specifically for solving particular problems and are difficult to apply to different domains, even if they are very similar. One possibility involves the generation of more versatile models, by means of modular learning, where learning is not conducted as a whole, but by small independent modules. Each module is generated independently to solve a certain task that is common to a very diverse set of machine learning problems. Therefore, these larger problems can only be solved by cooperation of several of these modules. As these modules are learned autonomously, they can be successfully reused to be part of the solution in other different domains. In this case, the learning system must be able to identify the subtasks, define specific modules for each of them, identify possible candidates in a set of previously learned modules, and link the modules with each other so that the whole resulting model is coherent and effective.

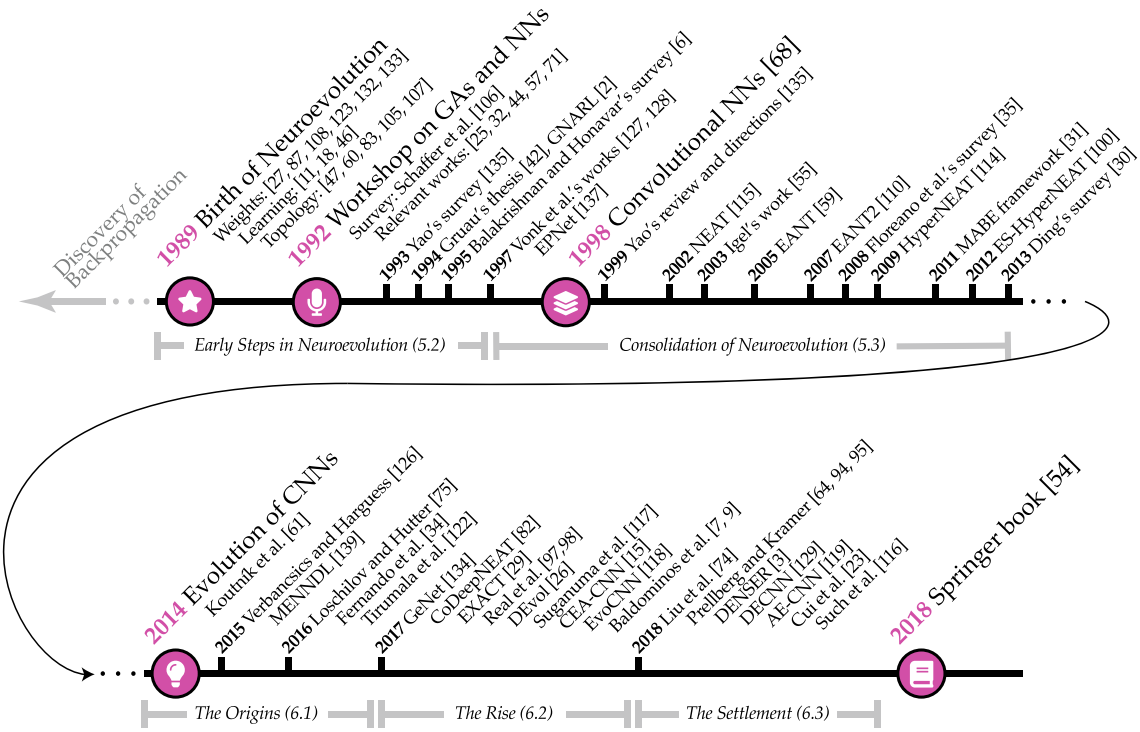


Fig. 6 Timeline summarizing the chronology of all the neuroevolution works reviewed in this paper, with their references. Some relevant milestones in the history of the field are shown as well. Under the timeline, the section of the paper where the works are described is shown, to serve as a reference

More specifically, a collection of simple neural modules could be generated, which are trained (at least partially) beforehand that could be used as building blocks for the rapid and efficient generation of complex systems of neural networks. It seems evident that NE would have a fundamental role in the development of these complex models, being able to mold the modules and assemble them quickly, efficiently and in a natural way in order to achieve a successful result for this larger, more difficult task.

Another field to explore, closely related to the idea of reusing previously learned models, is transfer learning. With a very large set of possible problems to solve, which is continuously growing, it remains as an interesting field of study to explore how a topology optimized for solving one problem can be reused (with or without modifications) to solve a different yet similar problem. To illustrate this with an example, it seems reasonable to think that a topology learned for recognizing handwritten characters can be used, at least to some extent, for successfully recognizing characters from digitized books. In a broader sense, many different problems hold some common aspects which can enable researchers to transfer some of the expertise acquired in one problem to others. In this case, NE can provide the means for learning more general topologies, which can be transferred to other problems or domains, as well as for determining the best way to transfer the knowledge in each scenario.

In summary, NE of CNNs remains a mostly unexplored and promising field which has gained significant attention in the last year, and current advances in technology enable researchers to develop new works within this research line. Much work is still to be done, but given the large applicability of CNNs and their success, the automatic evolutionary design of their topologies is a very promising area which must be tackled as of today.

Acknowledgements This research is partially supported by the Spanish Ministry of Education, Culture and Sports under FPU fellowship with grant number FPU13/03917.

References

1. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, Zheng X (2016) TensorFlow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation, pp 265–283
2. Angeline PJ, Saunders GM, Pollack JB (1994) An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans Neural Netw 5(1):54–65

3. Assunção F, Lourenço N, Machado P, Ribeiro B (2018) DENSER: deep evolutionary network structured representation. Genet Program Evol Mach **(in press)**
4. AWS: Amazon EC2 P3 Instances (2017). https://aws.amazon.com/es/ec2/instance-types/p3/. Last queried on 14 Nov 2017
5. Baird L (1999) Reinforcement learning through gradient descent. Ph.D. thesis, School of Computer Science, Carnegie Mellon University
6. Balakrishnan K, Honavar V (1995) Evolutionary design of neural architectures—a preliminary taxonomy and guide to literature. Technical report, Iowa State University. Paper 26
7. Baldominos A, Saez Y, Isasi P (2018) Evolutionary convolutional neural networks: an application to handwriting recognition. Neurocomputing 283:38–52
8. Baldominos A, Saez Y, Isasi P (2018) Evolutionary design of convolutional neural networks for human activity recognition in sensor-rich environments. Sensors 18(4):1288
9. Baldominos A, Saez Y, Isasi P (2018) Model selection in committees of evolved convolutional neural networks using genetic algorithms. In: Intelligent data engineering and automated learning—IDEAL 2018. Lecture Notes in Computer Science, vol 11314. Springer, pp 364–373
10. Baum EB, Haussler D (1989) What size net gives valid generalization? Neural Comput 1(1):151–160
11. Belew RK, McInerney K, Schraudolph NN (1991) Evolving networks: using the genetic algorithm with connectionist learning. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) Artificial life II. Addison-Wesley, MA, pp 511–547
12. Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, Turian J, Warde-Farley D, Bengio Y (2010) Theano: a CPU and GPU math compiler in Python. In: 9th Python in science conference
13. Bergstra J, Yamins D, Cox D (2013) Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. J Mach Learn Res 28(1):115–123
14. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 35(3):268–308
15. Bochinski E, Senst T, Sikora T (2017) Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In: 2017 IEEE international conference on image processing, pp 3924–3928
16. Canziani A, Paszke A, Culurciello E (2017) An analysis of deep neural network models for practical applications. arXiv:1605.07678
17. Caruana R (1993) Generalization vs. net size. NIPS Tutorial. Denver, CO
18. Chalmers DJ (1990) The evolution of learning: an experiment in genetic connectionism. In: 1990 Connectionist Models Summer School, pp 81–90
19. Chetlur S, Woolley C, Vandermersch P, Cohen J, Tran J, Catanzaro B, Shelhamer E (2014) cuDNN: efficient primitives for deep learning. arXiv:1410.0759
20. Cho K, Van Merriënboer B, Bahdanau D, Bengio Y (2014) On the properties of neural machine translation: encoder–decoder approaches. arXiv:1409.1259
21. Choudhary A, Rishi R, Dhaka VS, Ahlawat S (2010) Influence of introducing an additional hidden layer on the character recognition capability of a BP neural network having one hidden layer. Int J Eng Technol 2(1):24–28
22. Cramer NL (1985) A representation for the adaptive generation of simple sequential programs. In: 1st international conference on genetic algorithms and their applications, pp 183–187
23. Cui X, Zhang W, Tüske Z, Picheny M (2018) Evolutionary stochastic gradient descent for optimization of deep neural networks. In: Advances in neural information processing systems 31. NIPS Proceedings
24. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2:303–314
25. Dasgupta D, McGregor DR (1992) Designing application-specific neural networks using the structured genetic algorithm. In: International workshop on combinations of genetic algorithms and neural networks, pp 87–96
26. Davison J (2017) DEvol: Automated deep neural network design via genetic programming. https://github.com/joeddav/devol. Last visited on 01 July 2017
27. de Garis H (1992) Steerable GenNETS: the genetic programming of steerable behavior in GenNETS. In: Towards a practice of autonomous systems, pp 272–281
28. Dean J, Hölzle U (2017) Build and train machine learning models on our new Google Cloud TPUs. https://www.blog.google/topics/google-cloud/google-cloud-offer-tpus-machine-learning/. Published on 17 May 2017
29. Desell T (2017) Large scale evolution of convolutional neural networks using volunteer computing. In: 2017 genetic and evolutionary computation conference companion, pp 127–128
30. Ding S, Li H, Su C, Yu J, Jin F (2013) Evolutionary artificial neural networks: a review. Artif Intell Rev 39(3):251–260
31. Edlund JA, Chaumont N, Hintze A, Koch C, Tononi G, Adami C (2011) Integrated information increases with fitness in the evolution of animats. PLOS Comput Biol 7(10):e1002236
32. Elias JG (1992) Genetic generation of connection patterns for a dynamic artificial neural network. In: International workshop on combinations of genetic algorithms and neural networks, pp 38–54
33. Fahlman SE, Lebiere C (1990) The cascade-correlation learning architecture. In: Touretzky DS (ed) Advances in neural information processing systems, vol 2. Morgan Kaufmann. Los Altos, CA, pp 524–532
34. Fernando C, Banarse D, Reynolds M, Besse F, Pfau D, Jaderberg M, Lanctot M, Wierstra D (2016) Convolution by evolution: differentiable pattern producing networks. In: 2016 genetic and evolutionary computation conference, pp 109–116
35. Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. Evol Intell 1(1):1–47
36. Foley LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. Wiley, Hoboken
37. Forsyth R (1981) BEAGLE a Darwinian approach to pattern recognition. Kybernetes 10(3):159–166
38. Frean M (1990) The upstart algorithm: a method for constructing and training feedforward neural networks. Neural Comput 2(2):198–209
39. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feed forward neural networks. In: 13th international conference on artificial intelligence and statistics, vol 9. JMLR Proceedings, pp 249–256
40. Gnana Sheela K, Deepa SN (2013) Review on methods to fix number of hidden neurons in neural networks. Math Probl Eng 2013:425740
41. Gomez F, Schmidhuber J, Miikkulainen R (2008) Accelerated neural evolution through cooperatively coevolved synapses. J Mach Learn Res 9:937–965
42. Gruau F (1994) Neural network synthesis using cellular encoding and the genetic algorithm. Ph.D. thesis, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon
43. Hammerla NY, Halloran S, Plötz T (2016) Deep, convolutional, and recurrent models for human activity recognition using wearables. In: 25th international conference on artificial intelligence, pp 1533–1540

44. Hancock PJB (1992) Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In: International Workshop on combinations of genetic algorithms and neural networks, pp 108–122
45. Hansen N (2006) The CMA evolution strategy: a comparing review. In: Towards a new evolutionary computation. Springer, pp 75–102
46. Harp SA, Samad T, Guha A (1989) Towards the genetic synthesis of neural networks. In: 3rd international conference on genetic algorithms, pp 360–369
47. Harp SA, Samad T, Guha A (1990) Designing application-specific neural networks using the genetic algorithm. In: Advances NIPS 2. Morgan Kaufmann, pp 447–454
48. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition. IEEE
49. Hermundstad AM, Brown KS, Bassett DS, Carlson JM (2011) Learning, memory, and the role of neural network architecture. PLOS Comput Biol 7(6):e1002063
50. Hintzelab. MABE: Modular Agent Based Evolution Framework (2017). https://github.com/Hintzelab/MABE. Last visited on 27 June 2017
51. Hirose Y, Yamashita K, Hijiya S (1991) Back-propagation algorithm which varies the number of hidden units. Neural Netw 4(1):61–66
52. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
53. Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press, Ann Arbor
54. Iba H (2018) Evolutionary approach to deep learning. In: Evolutionary approach to machine learning and deep neural networks. Springer, pp 77–104
55. Igel C (2003) Neuroevolution for reinforcement learning using evolution strategies. In: 2003 IEEE congress on evolutionary computation, pp 2588–2595
56. Karpathy A (2015) The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Published on 21 May 2015
57. Karunanithi N, Das R, Whitley D (1992) Genetic cascade learning for neural networks. In: International workshop on combinations of genetic algorithms and neural networks, pp 134–145
58. Kassahun Y, Edgington M, Metzen JH, Sommer G, Kirchner F (2007) Common genetic encoding for both direct and indirect encodings of networks. In: 9th annual conference on genetic and evolutionary computation, pp 1029–1036
59. Kassahun Y, Sommer G (2005) Efficient reinforcement learning through evolutionary acquisition of neural topologies. In: 13th European symposium on artificial neural networks, pp 259–266
60. Kitano H (1990) Designing neural networks using genetic algorithms with graph generation system. Complex Syst 4:461–476
61. Koutník J, Schmidhuber J, Gomez F (2014) Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In: 2014 annual conference on genetic and evolutionary computation, pp 541–548
62. Koza JR (1989) Hierarchical genetic algorithms operating on populations of computer programs. In: 11th international joint conference on artificial intelligence, pp 7768–774
63. Koza JR, Rice JP (1992) Genetic programming: the movie. MIT Press, Cambridge
64. Kramer O (2018) Evolution of convolutional highway networks. In: Sim K, Kaufmann P (eds) EvoApplications 2018: applications of evolutionary computation, vol 10784. Lecture Notes in Computer Science. Springer, Berlin, pp 395–404
65. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Advances NIPS 25. NIPS Proceedings, pp 1097–1105
66. Larochelle H, Erhan D, Courville A, Bergstra J, Bengio Y (2007) An empirical evaluation of deep architectures on problems with many factors of variation. In: 24th international conference on machine learning, pp 473–480
67. Lawrence S, Giles CL, Tsoi AC (1996) What size neural network gives optimal generalization?. Technical report, Institute for Advanced Computer Studies, University of Maryland, Convergence properties of backpropagation
68. LeCun Y, Bengio Y (1998) Convolutional networks for images, speech, and time series. In: Arbib MA (ed) The handbook of brain theory and neural network. MIT Press, MA, USA, pp 255–258
69. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
70. LeCun Y, Denker JS, Solla SA (1990) Optimal brain damage. In: Advances NIPS 2. Morgan Kaufmann, pp 598–605
71. Lindgren K, Nilsson A, Nordahl MG, Rade I (1992) Regular language inference using evolving neural networks. In: International workshop on combinations of genetic algorithms and neural networks, pp 75–86
72. Linnainmaa S (1976) Taylor expansion of the accumulated rounding error. BIT Numer Math 16(2):146–160
73. Lipton ZC, Berkowitz J (2015) A critical review of recurrent neural networks for sequence learning. arXiv:1506.00019
74. Liu H, Simonyan K, Vinyals O, Fernando C, Kavukcuoglu K (2018) Hierarchical representations for efficient architecture search. In: 6th international conference on learning representations
75. Loshchilov I, Hutter F (2016) CMA-ES for hyperparameter optimization of deep neural networks. In: 2016 international conference on learning representations workshop track
76. Lu Z, Whalen I, Boddeti V, Dhebar Y, Deb K, Goodman E, Banzhaf W (2018) NSGA-NET: a multi-objective genetic algorithm for neural architecture search. arXiv:1810.03522
77. Maynard Smith J (1978) Optimization theory in evolution. Ann Rev Ecol Syst 9:31–56
78. Merrill JWL, Port RF (1991) Fractally configured neural networks. Neural Netw 4(1):53–60
79. Miihlenbein H, Kindermann J (1989) The dynamics of evolution and learning—towards genetic neural networks. In: Pfeifer R, Schreter Z, Fogelman-Soulié F, Steels L (eds) Connectionism in perspective. Elsevier, pp 173–197
80. Miikkulainen R (2017) Neuroevolution. In: Sammut C, Webb GI (eds) Encyclopedia of machine learning and data mining. Springer, pp 899–904
81. Miikkulainen R (2017) Topology of a neural network. In: Sammut C, Webb GI (eds) Encyclopedia of machine learning and data mining. Springer, Boston, MA, pp 1281–1281
82. Miikkulainen R, Liang J, Meyerson E, Rawal A, Fink D, Francon O, Raju B, Shahrzad H, Navruzyan A, Duffy N, Hodjat B (2017) Evolving deep neural networks. arXiv:1703.00548
83. Miller GF, Todd P, Hedge SU (1989) Designing neural networks using genetic algorithms. In: 3rd international conference on genetic algorithms, pp 379–384
84. Minsky ML (1954) Theory of neural-analog reinforcement systems and its application to the brain-model problem. Ph.D. thesis, Princeton University
85. Minsky ML, Papert SA (1969) Perceptrons: an introduction to computational geometry. MIT Press, Cambridge

86. Mishkin D, Sergievskiy N, Matas J (2016) Systematic evaluation of CNN advances on the ImageNet. arXiv:1606.02228
87. Montana DJ, Davis L (1989) Training feedforward neural networks using genetic algorithms. In: 11th joint international conference on artificial intelligence, pp 762–767
88. Mozer MC, Smolensky P (1989) Skeletonization: a technique for trimming the fat from a network via relevance assessment. In: Advances NIPS 1. Morgan Kaufmann, pp 107–115
89. New York Times (1958). New Navy device learns by doing; psychologist shows embryo of computer designed to read and grow wiser. http://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html
90. NVIDIA: The world's most efficient supercomputer for AI and deep learning (2017). http://images.nvidia.com/content/pdf/infographic/dgx-saturnv-infographic.pdf. Last visited on 15 July 2017
91. Odri SV, Petrovacki DP, Krstonosic GA (1993) Evolutional development of a multilevel neural network. Neural Netw 6(4):583–595
92. Parker GA, Maynard Smith J (1990) Optimality theory in evolutionary biology. Nature 348:27–33
93. Prechelt L (1995) Neural Net FAQ . https://www.cs.cmu.edu/Groups/AI/util/html/faqs/ai/neural/faq.html. Last modified on 23 Feb 1995
94. Prellberg J, Kramer O (2018) Lamarckian evolution of convolutional neural networks. arXiv:1806.08099
95. Prellberg J, Kramer O (2018) Limited evaluation evolutionary optimization of large neural networks. arXiv:1806.09819
96. Pugh J, Soros L, Stanley K (2016) Quality diversity: a new frontier for evolutionary computation. Front Robot Artif Intell 3:40
97. Real E, Aggarwal A, Huang Y, Le QV (2018) Regularized evolution for image classifier architecture search. arXiv:1802.01548
98. Real E, Moore S, Selle A, Saxena S, Leon-Suematsu Y, Tan J, Le QV, Kurakin A (2017) Large-scale evolution of image classifiers. In: Proceedings of the 34th international conference on machine learning, vol 70. JMLR Proceedings
99. Rechenberg I (1971) Evolutionsstrategie – optimierung technischer systeme nach prinzipien der biologischen evolution. Ph.D. thesis, Technische Universität Berlin
100. Risi S, Stanley KO (2012) An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. Artif Life 18(4):331–363
101. Rosenblatt F (1957) The perceptron–a perceiving and recognizing automaton. Technical report, Cornell Aeronautical Laboratory
102. Rumelhart D, Hinton G, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536
103. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC (2015) ImageNet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252
104. Sammut C, Webb GI (eds) (2017) Encyclopedia of machine learning and data mining. Springer, Berlin
105. Schaffer JD, Caruana RA, Eshelman LJ (1990) Using genetic search to exploit the emergent behavior of neural networks. Phys D Nonlinear Phenom 42(1–3):244–248
106. Schaffer JD, Whitley D, Eshelman LJ (1992) Combinations of genetic algorithms and neural networks: a survey of the state of the art. In: International workshop on combinations of genetic algorithms and neural networks, pp 1–37
107. Schiffmann W, Joost M, Werner R (1991) Performance evaluation of evolutionarily created neural network topologies. In: Schwefel HP, Männer R (eds) Parallel Problem Solving from Nature. PPSN 1990. Lecture Notes in Computer Science, vol 496. Springer, pp 274–283
108. Scholz M (1991) A learning strategy for neural networks based on a modified evolutionary strategy. In: Schwefel HP, Männer R (eds) Parallel Problem Solving from Nature. PPSN 1990. Lecture Notes in Computer Science, vol 496. Springer, pp 314–318
109. Schwefel HP (1974) Evolutionsstrategie und numerische optimierung. Ph.D. thesis, Technische Universität Berlin
110. Siebel NT, Sommer G (2007) Evolutionary reinforcement learning of artificial neural networks. Int J Hybrid Intell Syst 4(3):171–183
111. Sietsma J, Dow RJF (1991) Creating artificial neural networks that generalize. Neural Netw 4(1):67–79
112. Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems 25. NIPS Proceedings, pp 2951–2959
113. Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
114. Stanley KO, D'Ambrosio DB, Gauci J (2009) A hypercube-based encoding for evolving large-scale neural networks. Artif Life 15(2):185–212
115. Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evolut Comput 10(2):99–127
116. Such FP, Madhavan V, Conti E, Lehman J, Stanley KO, Clune J (2018) Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv:1712.06567
117. Suganuma M, Shirakawa S, Nagao T (2017) A genetic programming approach to designing convolutional neural network architectures. In: 2017 genetic and evolutionary computation conference companion, pp 497–504
118. Sun Y, Xue B, Zhang M (2017) Evolving deep convolutional neural networks for image classification. arXiv:1710.10741
119. Sun Y, Xue B, Zhang M (2018) Automatically evolving cnn architectures based on blocks. arXiv:1810.11875
120. Szegedy C, Ioffe S, Vanhoucke V, Alemi A (2016) Inception-v4, Inception-ResNet and the impact of residual connections on learning. In: 31st AAAI conference on artificial intelligence, pp 4278–4284
121. Talbi EG (2009) Metaheuristics: from design to implementation. Wiley, Hoboken
122. Tirumala SS, Ali S, Ramesh CP (2016) Evolving deep neural networks: a new prospect. In: 12th international conference on natural computation, fuzzy systems and knowledge discovery, pp 69–74
123. Torreele J (1991) Temporal processing with recurrent networks: an evolutionary approach. In: 4th international conference on genetic algorithms, pp 555–561
124. Turing AM (1950) Computing machinery and intelligence. Mind 59:433–460
125. Verbancsics P, Harguess J (2013) Generative neuroevolution for deep learning. arXiv:1312.5355
126. Verbancsics P, Harguess J (2015) Image classification using generative neuroevolution for deep learning. In: 2015 IEEE winter conference on applications of computer vision, pp 488–493
127. Vonk E, Jain LC, Johnson RP (1997) Automatic generation of neural network architecture using evolutionary computation, advances fuzzy systems–application and theory, vol 14. World Scientific Publishing, Singapore
128. Vonk E, Jain LC, Veelenturf LPJ, Johnson RP (1995) Automatic generation of a neural network architecture using evolutionary

computation. Electronic Technology Directions to the Year 2000:144–149
129. Wang B, Sun Y, Xue B, Zhang M (2018) A hybrid DE approach to designing CNN for image classification. In: 31st Australasian joint conference on artificial intelligence
130. Wang Z, Di Massimo C, Tham MT, Morris AJ (1994) A procedure for determining the topology of multilayer feedforward neural networks. Neural Netw 7(2):291–300
131. Werbos PJ (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Committee on Applied Mathematics, Harvard University
132. Whitley D, Dominic S, Das R (1991) Genetic reinforcement learning with multi-layer neural networks. In: 4th international conference on genetic algorithms, pp 562–569
133. Whitley D, Hanson T (1989) Optimizing neural networks using faster, more accurate genetic search. In: 3rd international conference genetic algorithms, pp 391–396
134. Xie L, Yuille A (2017) Genetic CNN. In: Proceedings of the 2017 IEEE international conference on computer vision
135. Yao X (1993) A review of evolutionary artificial neural networks. Int J Intell Syst 8(4):539–567
136. Yao X (1999) Evolving artificial neural networks. Proc IEEE 87(9):1423–1447
137. Yao X, Liu Y (1997) A new evolutionary system for evolving artificial neural networks. IEEE Trans Neural Netw 8(3):694–713
138. Young SR, Rose DC, Johnston T, Heller WT, Karnowski TP, Potok TE, Patton RM, Perdue G, Miller J (2017) Evolving deep networks using HPC. In: Machine learning on HPC environments workshop, pp 3924–3928
139. Young SR, Rose DC, Karnowsky TP, Lim SH, Patton RM (2015) Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: Workshop on machine learning in high-performance computing environments