An Introduction to Neuroevolution

Paul Pauls, *Technical University of Munich (TUM)*
Advisor: Michael Adam, *Technical University of Munich (TUM)*

***Abstract*—Neural networks have proven effective at solving difficult problems but designing their architectures can be challenging, especially when applying deep learning techniques with countless parameters to configure. This paper shows one way to automatically create those deep learning topologies by introducing the system *Neuroevolution of Augmenting Topologies* (abbr. NEAT). First the term neuroevolution is defined from the ground up, whereupon the key aspects of the benchmark neuroevolution algorithm NEAT are illustrated, closing with an outlook of advancements building upon NEAT.**

I. Introduction

WITH the success of Deep Learning in recent years [15], [14], [18] faced the machine learning research community the challenge of having to configure even more hyperparameters, create even more layers and run even more training-epochs. In order to ease this design process came the concept of *Automated Machine Learning* (abbr. AutoML) into focus. One aspect of AutoML - the evolution of *Artificial Neural Networks* (abbr. ANNs) - is addressed in this paper and its most prominent representative - NEAT - is thoroughly discussed.

This evolution of artificial neural networks is called *Neuroevolution* and it is a form of evolutionary algorithm that generates specific ANNs through modification of its parameters, topology and rules in order to maximize the ANN's accuracy or fitness score. The neuroevolution algorithm seeks to modify the ANN in an evolutionary process similar to the Darwinian process that produced human brains and its process-summarizing maxim 'survival of the fittest'. First methods using neuroevolution can be traced back to the 1980s and 1990s [22], while the first evolutionary algorithms were conceived in the 1950s by Alan Turing and Nils Barricelli [1].

II. Neuroevolution and Evolutionary Algorithms

A *neuroevolution algorithm* is a *genetic algorithm*, whose search-space (genotypes) consists of artificial neural networks. A *genetic algorithm* in turn is an *evolutionary algorithm* that evolves genotypes - genetically encoded representations of the actual solution (phenotype) - through mutation, recombination and selection. This chapter will introduce each of these terms and revisit how exactly they apply to neuroevolution.

A. Evolutionary and Genetic Algorithms

An *Evolutionary Algorithm* (abbr. EA) is defined as "a generic population-based and meta-heuristically optimized algorithmic solution to an applied problem" [16]. This fundamentally means that evolutionary algorithms provide an algorithmic solution to an applied problem that is increasingly optimized by some component of the EA. An evolutionary algorithm therefore also encodes a method of how to come up with the best (or most often just highly optimized) solution to an algorithmic problem.

Evolutionary algorithms find this best algorithmic solution through a *population-based* method. This means that EAs manage an arbitrary variety of algorithmic solutions - all of which differ and solve the applied problem with various grades of accuracy or fitness scoring. Each of these algorithmic solutions is called a *member* of the evolutionary algorithms' population. To determine the best member of the population assigns the evolutionary algorithm each member a *fitness score*. In the context of neuroevolution for example is this fitness score calculated by evaluating the accuracy of the artificial neural network.

However, the question remains how the members of the population are created in a sensible way so that they may represent a reasonable algorithmic solution. EAs utilize the method of *meta-heuristic optimization* to achieve this. Each new member in a population is conceived by recombining and/or mutating a single or multiple existing members of the population. Presuming that the existing members, which are recombined to create the new member, are chosen to be members with the highest fitness score, does the process reflect an optimization procedure resembling Darwinian evolution. Evolutionary algorithms therefore breed a single increasingly optimized algorithmic solution through evolutionary intercombination of already existing algorithmic solutions - hence representing the mentioned optimization process.

The possible methods of intercombination between existing members are inspired by biological evolution as well. These intercombination methods - such as mutation, recombination and selection - are only addressed in the subsequent chapter II-B to clarify how they apply to neuroevolution in particular. Furthermore is the mentioned optimization process considered *meta-heuristic* because innovations are randomly created and transferred within the population through these intercombination methods, requiring no outside input whatsoever. Thus, even when the feedback from the applied problem or environment is incomplete or sparse, is a fruitful traversal through the search-space still possible (e.g. via a lucky mutation or a recombination of two fit members).

Moreover is the population-based methodology considered *generic* because it does not specify how the members of the population are encoded. Though an evolutionary algorithm needs to eventually return an algorithmic solution to the applied problem, do the members not need to be algorithms themselves.

Genetic Algorithms (abbr. GAs) are evolutionary algorithms whose members are not algorithms per se, but genetic-like encodings which are eventually translated into algorithms by

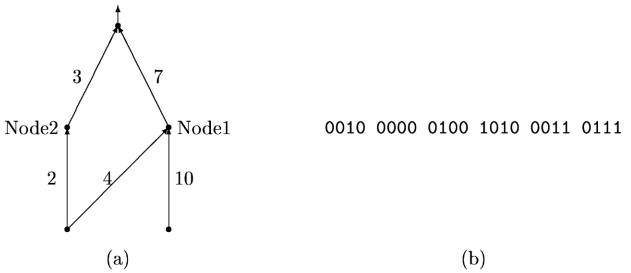

Fig. 1. Neural Network topology (a) and its corresponding Binary Encoding (b) (Source: [4])

a user-specified component of the genetic algorithm. The gene-like encoded member in a GA is called its *genotype*, whereas its corresponding translated algorithm is called its *phenotype*.

The defining advantage of representing members as a gene-like encoding instead of an algorithm in memory is that intercombination methods like mutation and recombination are vastly easier and faster on relatively simple genetic encodings than on complex specified algorithms. Figure 1 shows a neural network topology (a) and its corresponding binary encoding (b). This figure illustrates well how such a complex algorithm as an artificial neural network is significantly simpler represented as a genetic encoding and can in turn significantly simpler be recombined than the algorithm itself.

B. Neuroevolution

A proper definition of neuroevolution is now possible. Neuroevolution is the - possibly boundless - process in which by the means of a genetic algorithm the population of artificial neural networks is increasingly optimized in order to maximize the accuracy or fitness of the best ANN.

The intercombinatory method of *mutation* in the context of neuroevolution is defined as adding, modifying or removing structure to or from the ANN representation of a chosen member. This is achieved by modifying the members corresponding genotype representation in a sensible manner resulting in an effective change in its corresponding ANN phenotype. To give an example for the mutation of an ANN, take the binary encoded genotype seen in figure 1 and then flip, add or remove some bits, possibly resulting in a new node or connection in its corresponding phenotype.

The intercombinatory method of *recombination* in the context of neuroevolution is defined as crossing over the ANNs of two or more arbitrary members of the population, forming a new ANN with topological features from both *parents*. This crossover of ANNs is achieved by combining the genotype of the parent-members. The intention behind this is that the topological features of each parent combine and form a new ANN that hopefully performs even better than both parents. An example of such a recombination, though an impaired one as the figure actually represents the flawed process of the *Competing Conventions* problem, can be seen in figure 2. This figure shows two simplified ANNs (one encoded with [A,B,C], the other with [C,B,A]) crossing over and the resulting offspring having genetic damage as each is missing information from its parent. This figure however also shows how each possible gene position in the offspring is filled with a gene from either parent, creating valid though genetically impaired offspring.

The intercombinatory method of *selection* in the context of neuroevolution is defined identically as it is defined in EAs or GAs. The lowest performing members of the population are removed periodically in order to increase the overall quality of the gene pool and not waste resources on stuck members. This periodical removal of the lowest performing members is either executed after a member is performing too far below the other members or after each *generation* in the evolutionary process. Once all members of a population have been mutated/recombined, evaluated and assigned a fitness score, is this current state of the population considered a specific generation - whereupon the method of selection can then be applied.

The specifics of the neuroevolution process, e.g. how the intercombinatory methods are performed exactly, how the encoding scheme specifies genotypes and their translation into the phenotype ANNs, how the initial population is created, etc. are all left to the neuroevolution algorithm itself.

III. NeuroEvolution of Augmenting Topologies (NEAT)

Kenneth O.Stanley and Risto Miikkulainen co-published an approach to neuroevolution in the year 2002, which they called *Neuroevolution of Augmenting Topologies* (abbr. NEAT) [6]. At the time of envisionment was NEAT considered one of the most promising approaches to ANN evolution and it is still considered a viable approach and a benchmark algorithm in neuroevolution to this day. The first section in this chapter will outline what aspects of NEAT set it apart from the preceding research and enabled its leap in performance, which will be looked upon in the second section. This chapter closes with a short overview of great implementations and projects that realize or are realized with NEAT.

A. Key Aspects of NEAT and Differences to Preceding Neuroevolution

Stanley presents in his dissertation, titled 'Efficient Evolution of Neural Networks through Complexification' [8], three main technical challenges to the current state of neuroevolution and evolving structure incrementally. He states the following three questions that summarize these challenges [8, page 4]:

- "Is there a genetic representation that allows disparate topologies to cross over in a meaningful way?"
- "How can topological innovation that needs a few generations to be optimized be protected so that it does not disappear from the population prematurely?"
- "How can topologies be minimized throughout evolution without a contrived fitness function that measures complexity explicitly?"

The following key aspects of NEAT solve the stated problems and result in a powerful evolutionary method that can solve benchmark problems faster than previous methods and make entirely new applications possible.

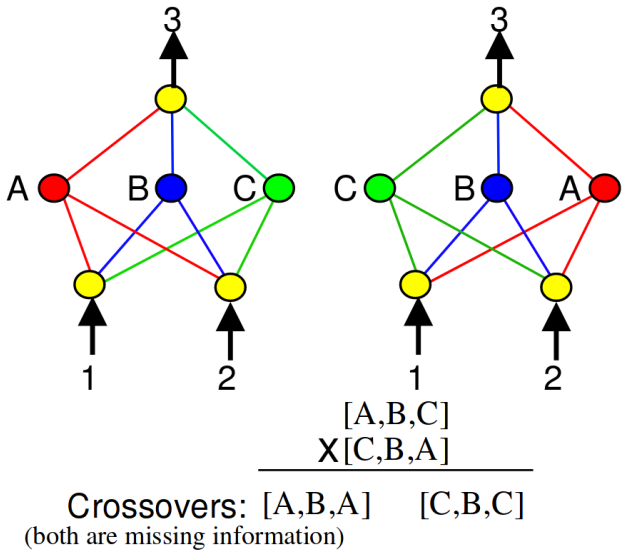

Fig. 2. Illustration of ANN crossover and the *Competing Conventions* problem. (Source: [7])

1) ***Historical Markings****:* Historical markings were introduced to overcome the previously mentioned challenge of *Competing Conventions* - visualized in figure 2. This example illustrates the attempt to cross over [A,B,C] and [C,B,A], which can result in offspring like [C,B,C], a representation that has lost one third of the information that both of the parents had. The probability of such severe genetic damage when looking at nontrivial crossovers (where an offspring is not simply a duplicate of a parent) with no protection against this sort of information erasure lies at 66.6% [8, page 19].

The approach of historical markings offers a solution to this problem by introducing a global innovation number. Whenever a new gene appears through structural mutation is a global innovation number incremented and assigned to that gene. This allows to track the historical origin of each gene and consequently which genes match up between all members of the population.

This identification of genes representing the same structure enables NEAT to solve the Competing Conventions problem. This is possible by first of all, not allowing the same gene (with the same innovation number and therefore representing the same structure) to be present multiple times in the genotype. Secondly does the principle also prevent genes from erasing one another when crossing over as it is now apparent if both genes are different and an erasure of either would therefore be an erasure of information. Instead, due to the dynamic-length encoding employed by NEAT, is the genotype simply elongated and both genes (and therefore both represented structures) are placed next to each other in the genotype. Genotypes that have become too large through this procedure can also shrink again through the method of mutation, which can *disable* and therefore remove individual genes.

A functional, meaning information preserving, crossover of 2 ANN genotypes is illustrated in figure 3. The genes of the genotypes of both parents are lined up next to each other according to their innovation number in ascending order.

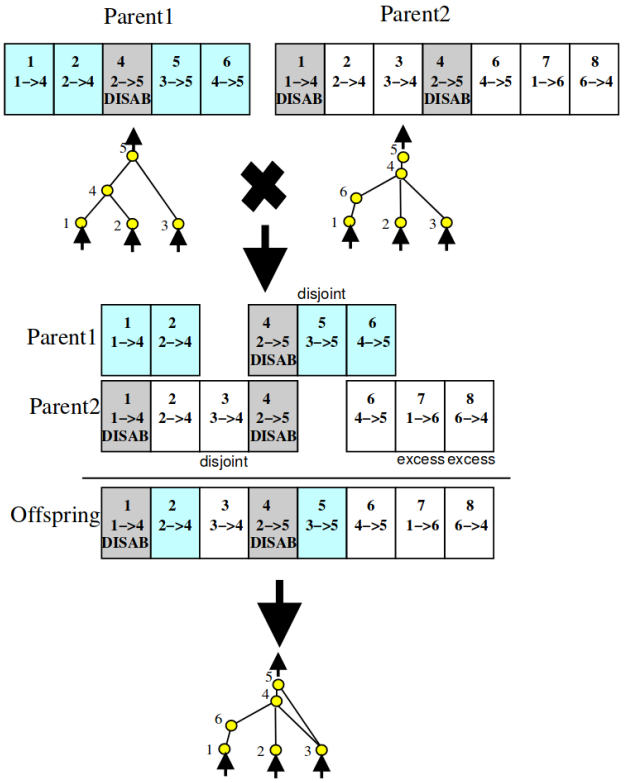

Fig. 3. ANN crossover in NEAT using *Innovation Numbers*. (Source: [8])

The crossover process then simply consists of concatenating all present genomes according to their innovation number, resulting in an elongated genome for the new member.

2) ***Speciation****:* Speciation has been studied in GAs but has rarely been applied to NE prior to its introduction in NEAT [8, page 25], even though it is crucial and the lack of speciation significantly limits its ability to add and maintain new topologies [7] [8, chap 4.3].

The underlying problem and reason for the necessity of speciation is that topological innovation - e.g. a drastically interrupting node or connection - often underperforms compared to the other more matured members of the population. Though this topological innovation may simply need time to optimize its weights or set up the new member for another mutation that in turn enables a significant performance boost. Speciation protects innovation following the philosophy that new ideas must be given time to reach their potential before they are eliminated. In the context of neuroevolution does this mean that innovative members must be given time to optimize their weights or further mutate favorably before they are eliminated.

The principle of speciation achieves this protection of innovation by using explicit fitness sharing. This method divides the population into niches according to their topological distinctness, which can be determined through the innovation numbers that make up the genotype. Each member of a niche is then forced to share the fitness score of all other members in the niche according to a specified equation [8, chap 3.3].

This equation postadjusts each members fitness towards the niches' average, which effectively means that new species don't have to compete with the best members, but only the

Method	Evaluations	Generalization	No. Nets
CE	**840,000**	300	16,384
ESP	**169,466**	289	1,000
NEAT	**33,184**	286	1,000

Fig. 4. Performance Comparison for 'Double pole balancing without velocity information' task between CE, ESP and NEAT. (Source: [7])

best niches. Each niche in turn is put into perspective through the lower performing members in the niche whose weights are likely not set optimally. This favors small and most often new niches since weight optimization isn't a factor anymore and therefore keeps diversity in niches and hence topological innovation high.

3) ***Complexification****:* Many preceding neuroevolution algorithms would start out with an often considerably large and randomly mutated initial population with the intention to provide genetic diversity in the gene pool. The intention was that this diversity would provide the means to thoroughly and quickly traverse through the search space by recombination, as many genetic features were already present in the initial population. Most often included these algorithms also a functional unit of the algorithm that was only intended to minimize the ANN through degenerative mutation. This was required as the necessity of topological innovation - its initially randomly mutated population - usually left significant clutter in the form of unnecessary genes and genetic features.

This necessity of a randomly mutated initial population to produce innovation however isn't a necessity in NEAT, as the principle of historical markings facilitates innovation through lossless and clean recombination and the principle of speciation protects innovation. The principle of complexification therefore states to provide a minimal initial population, which is only expanded upon. Because new topological features only appear and endure if they represent a beneficial new structure, does this also eliminate the resource-costly requirement of a functional unit responsible for minimization.

B. Performance of NEAT

Stanley & Miikkulainen analysis [6], [7], [8] of NEAT showed significant performance improvements in comparison to most preceding neuroevolution systems in terms of speed and required evaluations in key benchmarks like the double pole balancing task [2] and the XOR topology building task.

The XOR topology building task is a sanity check for *Topology and Weight Evolving Artificial Neural Network* algorithms. Because XOR is not linearly separable does a neural network require hidden units to solve it, which therefore makes this benchmark suitable for testing NEATs' ability to evolve structure. It is clear that NEAT solves the XOR problem without trouble and in doing so keeps the topology small [8, chap 4.1].

However, the double pole balancing task is of more importance, as this benchmark is well suited for performance comparison. This benchmark has been used in reinforcement learning and neuroevolution research for over 30 years [7, chap 3.A] and consists of two poles connected to a moving cart by a hinge, whereas the neural network must apply force to the cart to keep the poles balanced for as long as possible. Figure 4 shows a performance comparison between NEAT and the two other benchmark neuroevolution algorithms at the time - *Cellular Encoding* (abbr. CE) [3] and *Enforced SubPopulations* (abbr. ESP) [5]. The superior performance of NEAT in comparison to ESP is especially noteworthy in this task, as, unlike all other NE algorithms at the time, ESP performs at in the 'double pole balancing with velocity information' experiment at the same level as NEAT [6, chap 4.3.2]. This underlines NEATs' ability to perform especially well in feedback-sparse environments.

C. Practical Applications of NEAT

Practical applications of the NEAT system are plentiful due to it being a benchmark algorithm in the field of neuroevolution. NEAT was successfully employed in a wide field of research, some among them being physics [10], medical diagnosis [21], video game AI [13] and robotics [19].

Each of these applications however implemented their own version of NEAT, as - unlike for classical machine learning - there exists no well established framework for neuroevolution. The official NEAT software catalog [25] lists 25 different implementations and libraries for vanilla NEAT alone, the most prominent one probably being NEAT-Python [24]. Training an agent to play Super Mario World by utilizing this NEAT-Python library can be realized in under 100 lines of code with good results [26].

There are also currently efforts being made [29] to implement NEAT in the upcoming Tensorflow 2.0 [27] release as an add-on [28] library utilizing Tensorflows' newly introduced dynamic computational graphs.

IV. Conclusion

As this paper outlines is neuroevolution a promising approach to developing topologies and weights of artificial neural networks. NEAT was a significant advancement for the field of neuroevolution at the time it was introduced in the year 2002 and is still to this day an important benchmark. Even so did the research community of neuroevolution not stop innovating and created groundbreaking new neuroevolution algorithms - some being advancements of the original NEAT. Stanley, Risi, D'Ambrosio and Lehman introduced HyperNEAT [9] and its refinement ES-HyperNEAT [11] in 2009/2011, further enhancing it with new fitness function objectives like *Novelty Search* [12] in 2011. Furthermore did Miikkulainen & Liang bring the innovations of Deep Learning to neuroevolution introducing CoDeepNEAT [20] in 2017, while Real & Aggarwal were even able to design a neuroevolution system that surpasses hand-designs for the first time in 2018 [23].

Creating established neuroevolution frameworks that will efficiently use computational resources like the GPU will likely further this field of research even more, as the traditional machine learning frameworks already demonstrated.

REFERENCES

[1] Turing - Computing Machinery and Intelligence; Oct. 1950; https://academic.oup.com/mind/article/LIX/236/433/986238

[2] Anderson - Learning to control an inverted pendulum using neural networks; 1989; https://ieeexplore.ieee.org/document/24809

[3] Gruau, Bernard-lyon, et al - Neural Network Synthesis Using Cellular Encoding And The Genetic Algorithm; 1994; https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.5939

[4] Yao - Evolving Artificial Neural Networks; 1999; http://avellano.fis.usal.es/~lalonso/compt_soft/articulos/yao99evolving.pdf

[5] Gomez, Miikkulainen - Solving Non-Markovian Control Tasks with Neuroevolution; 1999; http://nn.cs.utexas.edu/downloads/papers/gomez.ijcai99.pdf

[6] Stanley, Miikkulainen - Evolving Neural Networks through Augmented Topologies; 2002; http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

[7] Stanley, Miikkulainen - Efficient Evolution of Neural Network Topologies; 2002; http://nn.cs.utexas.edu/downloads/papers/stanley.cec02.pdf

[8] Stanley - Efficient Evolution of Neural Networks through Complexification; Aug 2004; http://nn.cs.utexas.edu/downloads/papers/stanley.phd04.pdf

[9] Stanley, D'Ambrosio, et al - A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks; 2009; http://axon.cs.byu.edu/~dan/778/papers/NeuroEvolution/stanley3**.pdf

[10] Aaltonen, Adelman, et al - Measurement of the top-quark mass with dilepton events selected using neuroevolution at CDF; Apr 2009; https://www.ncbi.nlm.nih.gov/pubmed/19518620

[11] Risi, Stanley - Enhancing ES-HyperNEAT to Evolve More Complex Regular Neural Networks; Jul 2011; http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.365.4332

[12] Lehman, Stanley - Novelty Search and the Problem with Objectives; Oct 2011; https://www.cs.ucf.edu/eplex/papers/lehman_gptp11.pdf

[13] Hausknecht - A Neuroevolution Approach to General Atari Game Playing; 2012; https://www.cs.utexas.edu/~mhauskn/projects/atari/movies.html

[14] Ciresan, Giusti, et al - Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images; 2012; http://people.idsia.ch/~juergen/nips2012.pdf

[15] Krizhevsky, Sutskever, et al - ImageNet Classification with Deep Convolutional Neural Networks; 2012; https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[16] Holland - Scholarpedia Article on 'Genetic Algorithms'; Oct 2012; http://www.scholarpedia.org/article/Genetic_algorithms

[17] Mnih, Kavukcuoglu, et al - Playing Atari with Deep Reinforcement Learning; Dec 2013; https://arxiv.org/abs/1312.5602

[18] Schmidhuber - Deep Learning in Neural Networks; Apr 2014; https://arxiv.org/abs/1404.7828

[19] Cully, Clune, et al - Robots that can adapt like animals; May 2015; https://www.nature.com/articles/nature14422

[20] Miikkulainen, Liang, et al - Evolving Deep Neural Networks; Mar 2017; https://arxiv.org/abs/1703.00548

[21] Franca - Neuroevolution of Augmenting Topologies Applied to the Detection of Cancer in Medical Images; Feb 2018; http://www.bcc.ufrpe.br/sites/www.bcc.ufrpe.br/files/Luiz%20Fran%C3%A7a.pdf

[22] Frolov - Neuroevolution: A Primer on Evolving Artificial Neural Networks; Oct 2018; https://www.inovex.de/blog/neuroevolution/

[23] Real, Aggarwal, et al - Regularized Evolution for Image Classifier Architecture Search; Feb 2019; https://arxiv.org/abs/1802.01548

[24] CodeReclaimers - NEAT Python; Jun 2019; https://github.com/codereclaimers/neat-python

[25] NEAT Software Catalog; Jun 2019; http://eplex.cs.ucf.edu/neat_software/

[26] Pauls - SuperMario World NEAT Agent; Jun 2019; https://github.com/PaulPauls/SuperMarioWorld-NEAT-Agent

[27] Tensorflow 2.0 Beta; Jun 2019; https://www.tensorflow.org/beta

[28] Tensorflow addons; Jun 2019; https://github.com/tensorflow/addons

[29] Pauls - Tensorflow Neuroevolution; Jun 2019; https://github.com/PaulPauls/Tensorflow-Neuroevolution