On Overview of Neuroevolution and NEAT

Paul Pauls
Advisor: Michael Adam

***Abstract*—Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.**

I. Introduction

WITH the rise of Deep Learning in this decade, see especially [19], [18], [26], did the machine learning research community increasingly face the challenge of having to configure even more hyperparameters, create even more layers and run even more tests. In order to ease this design process did the concept of *Automated Machine Learning* (short form: AutoML) come into focus. One aspect of AutoML - the evolution of artificial neural networks (short form: 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 [49], while the first evolutionary algorithms were conceived in the 1950s by Alan Turing and Nils Barricelli [1].

II. Neuroevolution and Evolutionary Algorithms

To better characterize neuroevolution in this chapter is it best to first roughly categorize it, whereupon the following sections describe the categories in detail. A *neuroevolution algorithm* is a *genetic algorithm*, whose search-space (or genotypes) consist only of artificial neural networks. A *genetic algorithm* in turn is a *evolutionary algorithm* that evolves genotypes - genetically encoded representations of the actual solution (phenotype) - through mutation, recombination and selection.

A. Evolutionary and Genetic Algorithms

An evolutionary algorithm (short form: EA) is defined as 'a generic population-based and meta-heuristically optimized algorithmic solution to an applied problem' [21]. When breaking this complex definition down into simpler terms, is the first thing that should be clarified about EAs the fact that they are no algorithmic solution to the applied problem in and of themselves per se, but rather that they are meta-algorithms that create another optimized algorithm, which then solves the applied problem. An evolutionary algorithm therefore encodes a method of how to come up with the best solution to an algorithmic problem.

Evolutionary algorithms set out to finding 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 does the evolutionary algorithm assign each a *fitness score* after it is created. In the context of neuroevolution for example is this fitness score calculated by judging 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 to the problem and eventually an optimized one. This is the point at which the EAs' aspect of *meta-heuristic optimization* comes into play. 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 the high fitness scoring members, does the process constitute an optimization procedure resembling Darwinian evolution. Evolutionary algorithms therefore breed increasingly optimized algorithmic solution through evolutionary intercombination of existing algorithmic solutions - hence representing the mentioned optimization process.

The possible methods of intercombination between existing members are inspired by biological evolution, such as mutation, recombination and selection. For the sake of brevity will only the subsequent chapter II-B explain those intercombination methods in detail and how they apply to neuroevolution algorithms. The mentioned optimization process achieved through these intercombination methods is considered *meta-heuristic* because the optimization process is possible with incomplete information or limited computational capacity. Thus even when the feedback - the mentioned (possibly incomplete) information through which the EA is able to determine a sensible fitness score - of the applied problem is limited or sparse, is a fruitful traversal through the search-space still possible (e.g. via a lucky mutation).

At last can be said that the evolutionary algorithms

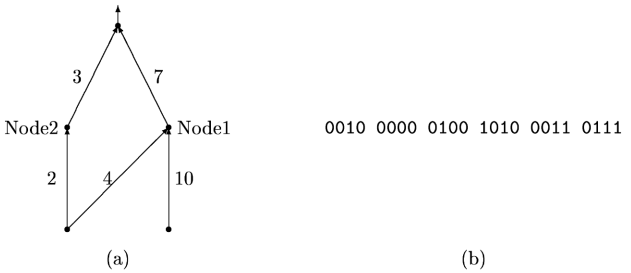

Fig. 1. Illustration of a binary encoding of an ANN. (Source: [3])

population-based methodology is considered *generic* because it does not dictate how the members of the population are encoded. Though an evolutionary algorithm needs to eventually return an algorithmic solution to the applied problem, does it not dictate that the members need to be algorithms themselves. This is were genetic algorithms come into play.

Genetic algorithms (short form: GAs) are evolutionary algorithms whose members are not algorithms per se, but genetic-like encodings which are eventually translated into algorithms by 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 encodings instead of an algorithm in memory is that intercombination methods like mutation and recombination - central aspects of evolutionary algorithms - are vastly easier and faster on relatively simple genetic encodings than on complex specified algorithms. Figure 1 illustrates a binary encoding of an ANN and showcases well how such a complex algorithm as an artificial neural network is significantly simpler represented as a genetic encoding and how such a genetic encoding can be vastly easier recombined than the algorithm itself.

B. Neuroevolution

A proper - from the ground-up - definition of Neuroevolution is now possible. Neuroevolution is the - possibly boundless - process in which by the means of a genetic algorithm its population of artificial neural networks is increasingly optimized in order to maximize the accuracy or fitness of the best ANN. Neuroevolution does so by continuously improving the members of its population through intercombination methods like mutation, recombination and selection.

The intercombinatory method of *mutation* in the context of Neuroevolution means that the genotype of a chosen member is in some way modified by adding to, changing or removing from the genotype representation. The manner and probability in which this modification takes place is completely up to the implementation specifics of the respective neuroevolution algorithm. To give an example mutation for an ANN, is it possible to imagine a binary encoded genotype as seen in figure 1, which then has some bits flipped, added or removed possibly resulting in a new node or connection in its corresponding phenotype.

The intercombinatory method of *recombination* in the context of Neuroevolution means that the genotype of two or more arbitrary (though most often high performing) members are combined, forming a new member. This is done in the hopes that those parts of the genotype that encode member-distinct features combine into the newly created genotype and encode an ANN that performs even better than both *parent*-members in isolation. 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. In this example cross two simplified genotypes consisting of three possible genes over, whereas each possible gene in the resulting children takes over a gene of the same position from either parent. Again is the manner and probability in which this modification is performed completely up to the implementation specifics of the respective neuroevolution algorithm.

Lastly is the intercombinatory method of *selection*. As previously defined does the process of neuroevolution work on populations; these however are often of fixed size in most neuroevolution algorithms. The purpose of this restriction is to force the neuroevolution process to remove low performing members from the population from which possible parents for intercombination are chosen, by only allowing a limited number of members to exist. Once all members of the population have been assigned a fitness score, is this current state of the population considered a specific *generation*. The method of *selection* then removes certain members (usually the low performing) of the population after each generation. The intercombinatory methods of mutation and recombination subsequently add new members to the population and the whole population is evaluated again, marking the start of the next generation. The method of selection is also applicable in the case of an unrestricted population size, e.g. by removing members that score too far below the current best member.

All those Details of the neuroevolution process, e.g. how the encoding scheme specifies genotypes and their translation into the phenotype ANNs, how the initial population is created, which members are chosen as parents for intercombinatory methods or simply how exactly the intercombinatory methods are performed are all left to the specific neuroevolution algorithm.

III. NeuroEvolution of Augmenting Topologies (NEAT)

Kenneth O.Stanley and Risto Miikkulainen co-published in the year 2002 an approach to neuroevolution they called *Neuroevolution of Augmenting Topologies* - or in short *'NEAT'* - in their paper "Evolving Neural Networks through Augmenting Topologies" [5]. 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 many neuroevolution frameworks 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 frameworks that support and projects that are realized with NEAT.

A. Key Aspects of NEAT and Differences to Preceding Neuroevolution

In his dissertation titled 'Efficient Evolution of Neural Networks through Complexification' [8] and published in 2004, did Stanley present three main technical challenges to the current state of neuroevolution and evolving structure incrementally. He asked 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?"

Stanley answered these questions with the following key aspects of NEAT, which differentiate it from the preceding neuroevolution algorithms and resulted in a method that "can evolve a diverse population of increasingly complex topologies separated into unique species. This approach results in powerful evolution that can solve benchmark problems faster than previous methods, and also makes entirely new applications possible."

1) Historical Markings: Historical markings were introduced to overcome the challenge of 'Competing Conventions' - visualized in illustration 2. This example illustrates the attempt to cross over [A,B,C] and [C,B,A], which can result in [C,B,C], a representation that has lost one third of the information that both of the parents had. The probability of severe genetic damage like this when looking at nontrivial crossovers (where an offspring is not simply a duplicate of a parent) with no protection against this sort of information erasement lies at 66.6% - as thoroughly calculated by Stanley in [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 (which in the context of NEATs' direct encoding effectively means whenever a new node or connection appears), a global innovation number is 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.

Keeping track of those genes allows to identify genes that represent the same structure (though possibly with different weights) and as a result allows to solve the 'Competing Conventions Problem'. The principle of historical markings does so by not allowing the same gene (with the same innovation number and therefore representing the same structure) to be present multiple times in the genotype. The principle also prevents from genes erasing on another when crossing over because the innovation number clearly shows if two genes are different and therefore should not eradicate one another.

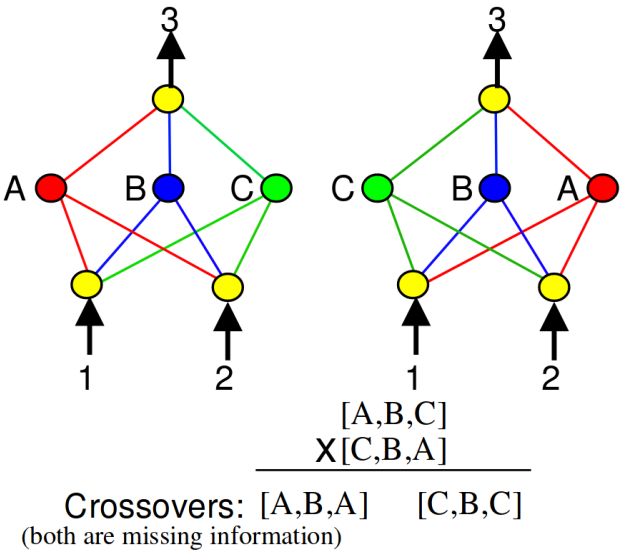

Fig. 2. Illustration of the 'Competing Conventions Problem'. (Source: [6])

Instead, due to the dynamic-length encoding employed by NEAT, is the genotype simply elongated and both genes (and therefore both features) are placed in the genotype. This does however not mean that genes in the genotype can't be removed, though this is not done in the crossover process but rather in the mutation process, which can *disable* individual genes.

This elaboration of the principle of historical markings explains how the crossover in illustration 2 would take place without severe genetic damage. Then again does figure 3 illustrate such a functional, meaning information preserving, recombination process explicitly with the usage of innovation numbers.

2) Speciation: Speciation, also known as *niching*, has been studied in GAs but has rarely been applied to NE prior to its introduction in NEAT [8, page 25]. Though it is a crucial part of NEAT and as Stanley verified empirically in his PhD thesis does the lack of speciation significantly limit NEAT's ability to add and maintain new topologies [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 set up the new member for another mutation that in turn makes it outperform all other members. Speciation therefore 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 protects innovation by using explicit fitness sharing, a method under which individuals that are sharing a niche are forced to also share the fitness score of all members in the niche. This effectively means that each members fitness is postadjusted according to the following equation [8, chap 3.3]:

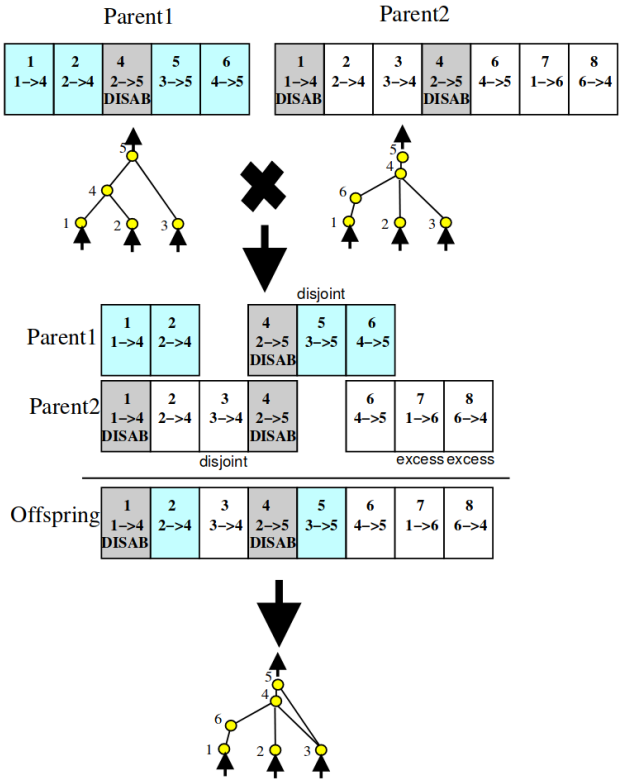

Fig. 3. Recombination of ANNs in NEAT through Innovation Numbers. (Source: [8])

If f_i is i's fitness before sharing, then after sharing it becomes f'_i according to

$$f'_i = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))} \tag{1}$$

This explicit fitness sharing therefore effectively means that new species don't have to compete with the best members, but only the best niches, which again are put into perspective through the lower performing members in the niche. This favors small and most often new niches and therefore keeps diversity in niches and hence topological innovation high. An important point to make though is that this principle of speciation obviously requires a distance metric between genotypes in order to determine if two members are sufficiently different to place them in distinct species. NEAT provides such a distance metric through the principle of historical markings.

3) Complexification: The principle of complexification is the simplest one and addresses a problem Stanley identified in most preceding neuroevolution algorithms. 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 traverse through the search space by mutation, as many genetic features were already present in the initial population. Most often did these algorithms also include 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 wasn't a necessity anymore in NEAT, as the principle of speciation protects innovation and the principle of historical markings allows for lossless and clean recombination between members. The principle of complexification therefore states to provide a minimal initial population, which is only expanded upon. Because new genes (or 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.

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 on 'Double pole balancing without velocity information' task. (Source: [6])

B. Performance of NEAT

Stanley & Miikkulainen also included thorough performance analysis in their papers introducing NEAT in 2002 and 2004 [5], [6], [8]. Their analysis showed NEAT significantly outperforming most preceding NE methods in terms of speed and required evaluations in key benchmarks like the double pole balancing task [2] and the XOR topology building task.

The simpler of the two benchmarks - the XOR topology building task - is a sanity check for TWEANN algorithms. Because XOR is not linearly separable does a neural network require hidden units to solve it and 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 - unlike the XOR topology building task - this benchmark is well suited for performance comparison. This benchmark has been used in RL and NE research for over 30 years [6, 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 lists the table showing the results of the 'double pole balancing without velocity information' experiment, in which NEAT is by far the fastest system on this challenging task, clearly even outperforming ESP (Enforced SubPopulations by Gomez & Miikkulainen [4]). This is relevant as ESP, unlike all other NE systems at the time, performs at least in the other 'double pole balancing with velocity information' experiment at the same level as NEAT [5, 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 as it is still relevant due to its high performance. One very

prominent example of a great application of NEAT is the most accurate measurement to date of the mass of the top quark, which was computed by a very large team at the Tevatron collider in 2009 [14]. Though NEAT was not only applied in physics, but also enabled pioneering work in advancing video game playing AI or medical AI. The work [17] of a single researcher - Matthew Hausknecht - on Atari video game playing utilizing HyperNEAT (a very promising Advancement of NEAT introduced in 2009) was so pioneering that it was later referenced by the DeepMind Team [24], who eventually created the famous AlphaStar AI. NEAT was also applied in a recent study published in 2018, which compared the achieved accuracy of different neuroevolution approaches in the context of detecting tumors in medical images [40]. NEAT's performance was similar to the established algorithms, even when executing a much smaller number of generation.

Each of these applications implemented their own version of NEAT, as - unlike for classical machine learning - there exists no well established framework for neuroevolution. The official NEAT software catalogue [53] lists 25 different implementations and libraries for vanilla NEAT alone, the most prominent one probably being NEAT-Python [52] by CodeReclaimers. Training an agent to play SuperMario World by utilizing this NEAT-Python library can be realized in under 100 lines of code with good results [54].

There are also currently efforts being made [57] to implement NEAT in the upcoming Tensorflow 2.0 [55] release as an addon [56] library utilizing Tensorflows newly introduced dynamic computational graphs.

IV. Conclusion

As this paper outlines - and the past decade proves - is neuroevolution a promising approach to developing topologies and weights of artifical 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 [13] and its refinement ES-HyperNEAT [15] in 2009/2011, further enhancing it with new fitness function objectives like 'Novelty Search' [16] in 2011. Furthermore did Miikkulainen & Liang bring the innovations of Deep Learning to Neuroevolution introducing CoDeepNEAT [29] in 2017, while Real & Aggarwal were even able to design a neuroevolution system that surpasses hand-designs for the first time in 2018 [50].

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.

V. Acknowledgments

I wish to thank Rezsa Farahani, from the Google Brain team, and Michael Adam, from the Technical University Munich, for their support and valuable input.

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] Yao - Evolving Artificial Neural Networks; 1999; http://avellano.fis.usal.es/~lalonso/compt_soft/articulos/yao99evolving.pdf
[4] Gomez, Miikkulainen - Solving Non-Markovian Control Tasks with Neuroevolution; 1999; http://nn.cs.utexas.edu/downloads/papers/gomez.ijcai99.pdf
[5] Stanley, Miikkulainen - Evolving Neural Networks through Augmented Topologies; 2002; http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf
[6] Stanley, Miikkulainen - Efficient Evolution of Neural Network Topologies; 2002; http://nn.cs.utexas.edu/downloads/papers/stanley.cec02.pdf
[7] Geard, Wiles - Structure and Dynamics of a Gene Network Model Incorporating Small RNAs; Dec 2003; https://ieeexplore.ieee.org/document/1299575
[8] Stanley - Efficient Evolution of Neural Networks through Complexification; Aug 2004; http://nn.cs.utexas.edu/downloads/papers/stanley.phd04.pdf
[9] Reisinger, Miikkulainen - Acquiring Evolvability through Adaptive Representations; Jul 2007; http://nn.cs.utexas.edu/downloads/papers/reisinger.gecco07.pdf
[10] Mattiussi, Duerr, et al - Center of Mass Encoding: A self-adaptive representation with adjustable redundancy for real-valued parameters; Jul 2007; https://infoscience.epfl.ch/record/101405
[11] Floreano, Duerr, et al - Neuroevolution: From Architectures to Learning; Jan 2008; http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.182.1567
[12] Mattiussi, Marbach, et al - The Age of Analog Networks; Sep 2008; https://www.aaai.org/ojs/index.php/aimagazine/article/view/2156
[13] 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
[14] 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
[15] 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
[16] Lehman, Stanley - Novelty Search and the Problem with Objectives; Oct 2011; https://www.cs.ucf.edu/eplex/papers/lehman_gptp11.pdf
[17] Hausknecht - A Neuroevolution Approach to General Atari Game Playing; 2012; https://www.cs.utexas.edu/~mhauskn/projects/atari/movies.html
[18] Ciresan, Giusti, et al - Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images; 2012; http://papers.nips.cc/paper/4741-deep-neural-networks-segment-neuronal-membranes-in-electron-microscopy-in.pdf
[19] 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
[20] Woergoetter, Porr - Scholarpedia Article on 'Reinforcement Learning'; Sep 2012; http://www.scholarpedia.org/article/Reinforcement_learning
[21] Holland - Scholarpedia Article on 'Genetic Algorithms'; Oct 2012; http://www.scholarpedia.org/article/Genetic_algorithms
[22] Fogel, Fogel, et al - Scholarpedia Article on 'Evolutionary Programming'; Oct 2013; http://www.scholarpedia.org/article/Evolutionary_programming
[23] Lehman, Miikkulainen - Scholarpedia Article on 'Neuroevolution'; Oct 2013; http://www.scholarpedia.org/article/Neuroevolution
[24] Mnih, Kavukcuoglu, et al - Playing Atari with Deep Reinforcement Learning; Dec 2013; https://arxiv.org/abs/1312.5602
[25] Pascanu, Ganguli, et al - On the Saddle Point for Non-Convex Optimization; May 2014; https://www.researchgate.net/publication/262452520
[26] Schmidhuber - Deep Learning in Neural Networks; Apr 2014; https://arxiv.org/abs/1404.7828
[27] Kim, Rigazio - Deep Clustered Convolutional Kernels; Mar 2015; https://arxiv.org/abs/1503.01824
[28] Fernando, Banarse, et al - Convolution by Evolution; Jun 2016; https://arxiv.org/abs/1606.02580
[29] Miikkulainen, Liang, et al - Evolving Deep Neural Networks; Mar 2017; https://arxiv.org/abs/1703.00548
[30] Xie, Yuille - Genetic CNN; Mar 2017; https://arxiv.org/abs/1703.01513

[31] Negrinho, Gordon - DeepArchitect: Automatically Designing and Training Deep Architectures; Apr 2017; https://arxiv.org/abs/1704.08792

[32] Real, Moore, et al - Large-scale Evolution of Image Classifiers; Jun 2017; https://arxiv.org/abs/1703.01041

[33] Stanley - Neuroevolution: A Different Kind of Deep Learning; Jul 2017; https://www.oreilly.com/ideas/neuroevolution-a-different-kind-of-deep-learning

[34] Brock, Lim, et al - SMASH: One-Shot Model Architecture Search through HyperNetworks; Aug 2017; https://arxiv.org/abs/1708.05344

[35] Salimans, Ho - Evolution Strategies as a Scalable Alternative to Reinforcement Learning; Sep 2017; https://arxiv.org/abs/1703.03864

[36] Jaderberg, Dalibard, et al - Population Based Training of Neural Networks; Nov 2017; https://arxiv.org/abs/1711.09846

[37] Zhang, Clune, et al - On the Relationship Between the OpenAI Evolution Strategy and Stochastic Gradient Descent; Dec 2017; https://arxiv.org/abs/1712.06564

[38] Stanley, Clune - Welcoming the Era of Deep Neuroevolution; Dec 2017; https://eng.uber.com/deep-neuroevolution/

[39] Liu, Simonyan, et al - Hierarchical Representation for Efficient Architecture Search; Feb 2018; https://arxiv.org/abs/1711.00436

[40] 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

[41] Such, Stanley, et al - Accelerating Deep Neuroevolution: Train Atari in Hours on a Single Personal Computer; Apr 2018; https://eng.uber.com/accelerated-neuroevolution/

[42] Such, Madhavan, et al - Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning; Apr 2018; https://arxiv.org/abs/1712.06567

[43] Zoph, Vasudevan, et al - Learning Transferable Architectures or Scalable Image Recognition; Apr 2018; https://arxiv.org/abs/1707.07012

[44] Lehman, Chen, et al - ES Is More Than Just a Traditional Finite-Difference Approximator; May 2018; https://arxiv.org/abs/1712.06568

[45] Lehman, Chen, et al - Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients; May 2018; https://arxiv.org/abs/1712.06563

[46] Zhong, Yan, et al - Practical Block-Wise Neural Network Architecture Generation; May 2018; https://arxiv.org/abs/1708.05552

[47] Rawal, Miikkulainen - From Nodes to Networks: Evolving Recurrent Neural Networks; Jun 2018; https://arxiv.org/abs/1803.04439

[48] Conti, Madhavan, et al - Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty Seeking Agents; Oct 2018; https://arxiv.org/abs/1712.06560

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

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

[51] Sun, Xue, et al - Evolving Deep Convolutional Neural Networks for Image Classification; Mar 2019; https://arxiv.org/abs/1710.10741

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

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

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

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

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

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