Neuroevolution of Augmenting Topologies

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

THIS shall be my introduction. And this shall be my citation [1].

II. Neuroevolution in General

"Neuroevolution is a machine learning technique that applies evolutionary algorithms to construct artificial neural networks, taking inspiration from the evolution of biological nervous systems in nature." - Scholarpedia

The architecture of a network, i.e. how the neurons are connected to each other, plays a very important role in whether or not an ANN can be trained to successfully learn a task. Over the years, human experts carefully designed complex architectures such as the VGGNet, AlexNet, GoogleNet, ResNet and many more to achieve and often surpass human-level performance on many different tasks. However, there is no single architecture that is sufficient for all tasks, and therefore manually designing well performing ANNs is a very challenging, time consuming and complex task that requires knowledge and experience. A typical ten layer network, for example, can have $10\hat{1}0$ candidate networks. This makes it impractical to evaluate all combinations and select the best performing one. [5] Consequently, a lot of effort is currently put into automating the process of finding good ANN architectures. Solving this requires answering many questions, such as i) how to design the components of the architecture, ii) how to put them together, and iii) how to set the hyperparameters. There are two different paradigms that caught people's attention in approaching this problems, namely a) using artificial intelligence (AI) based search methods, and b) using evolutionary techniques to generate networks. The latter is referred to as neuroevolution and, similar to machine learning, is "... yet another example that old algorithms combined with modern amounts of computing can work surprisingly well" [5]

- The network topology must be fully hand-engineered before training starts and only connection weights encapsulate learned knowledge while the network topology remains the same. - As a result it may introduce oversaturated neural units which don not take part in the training/inference process but simply consuming computing resources - special methods must be applied to avoid sticking into local optima such as L1/L2-norm, dropout regularizations, etc - exploding or diminishing learning gradient issues during back/forward propagation - implemented solutions can generalize only in the narrow scope learned from provided training samples

Neuroevolution is most commonly applied in artificial life, general game playing and evolutionary robotics.

Neuroevolution, can be applied more broadly than traditional Machine Learning because it does not require a large set of correct input-output pairs. Instead, it only requires that a) the performance can somehow be measured over time, and b) the behaviour of the networks can be modified through evolution. [5]

Neuroevolution is in competition with Gradient Descent. Around 2017 researchers at Uber stated they had found that simple structural Neuroevolution algorithms were competitive with sophisticated modern industry-standard gradient-descent deep learning algorithms, in part because Neuroevolution was found to be less likely to get stuck in local minima [5]

First methods using neuroevolution can be traced back to the 1980s and 1990s, where direct genetic algorithms were used to evolve the weights of fixed ANNs.[5]

Depending on the specific AI based solution, AutoML can be seen as neuroevolution. Google AI, for example, presented both a reinforcement learning approach as well as an evolutionary approach to find, i.e. generate, ANNs and refer to both of them as AutoML. [5]

Inspired by biological neural networks, ANNs consist of many artificial neurons that are connected to each other. In a process, mostly referred to as training or learning, ANNs are optimised to solve specific, predefined tasks such as detecting faces. Learning is usually realised by employing a form of stochastic gradient descent (SGD) in order to change the network parameters in such a way that a future prediction for the same input is closer to the desired output. [5]

- Conventional Neuroevolution: Evolve only the strength of the connection weights for a fixed network topology - TWEANNs (Topology and Weight Evolving Artificial Neural Network): Evolve both the topology of the network and its weights - Parallel or sequential Evolving: A separate distinction can be made between methods that evolve the structure of ANNs in parallel to its parameters and those that develop them separately

III. Evolutionary and Genetic Algorithms

A genetic algorithm is a search heuristic that is inspired by the theory of natural evolution. This algorithm reflects the

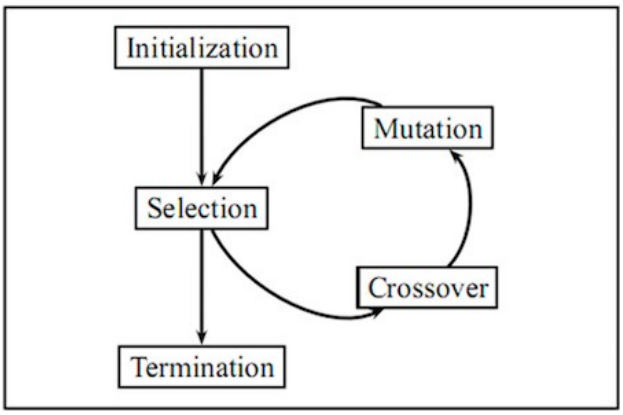

Fig. 1. Source: [4]

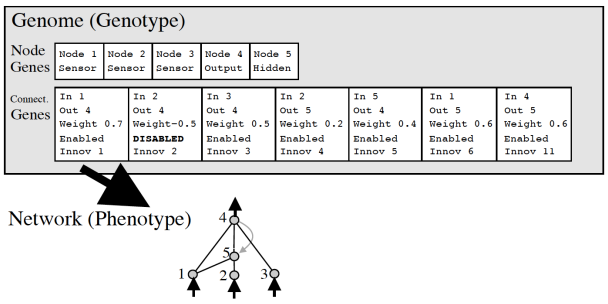

Fig. 2. Source: [2]

process of natural selection where the fittest individuals are selected for reproduction in order to produce even fitter offspring of the next generation. This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

Genetic algorithms are often used in tandem with other methods, acting as a quick way to find a somewhat optimal starting place for another algorithm to work off of. [4] An Evolutionary Algorithm contains four overall steps: initialization, selection, genetic operators, and termination. These steps each correspond, roughly, to a particular facet of natural selection. [4]

In order to begin our algorithm, we must first create an initial population of solutions. It will often be created randomly (within the constraints of the problem) or, if some prior knowledge of the task is known, roughly centered around what is believed to be ideal. It is important that the population encompasses a wide range of solutions, because it essentially represents a gene pool; ergo, if we wish to explore many different possibilities over the course of the algorithm, we should aim to have many different genes present. [4]

To determine the portion of top-performing members of a population in each generation are all members of the population evaluated according to a fitness function. A fitness function is a function that takes in the characteristics of a member, and outputs a numerical representation of how viable of a solution it is. Creating the fitness function can often be very difficult, and it is important to find a good function that accurately represents the data; it is very problem-specific. EAs can also be extended to use multiple fitness functions. This complicates the process somewhat as we can end up with a set of optimal points when using multiple fitness functions. The set of optimal solutions is called the Pareto frontier, and contains elements that are equally optimal in the sense that no solution dominates any other solution in the frontier. [4]

This step really includes two sub-steps: crossover and mutation. After selecting the top members (typically top 2, but this number can vary), these members are now used to create the next generation in the algorithm. Using the characteristics of the selected parents, new children are created that are a mixture of the parents' qualities. Now, we must introduce new genetic material into the generation. If we do not do this crucial step, we will become stuck in local extrema very quickly, and will not obtain optimal results. This step is mutation, and we do this, quite simply, by changing a small portion of the children such that they no longer perfectly mirror subsets of the parents' genes. Mutation typically occurs probabilistically, in that the chance of a child receiving a mutation as well as the severity of the mutation are governed by a probability distribution. [4]

Eventually, the algorithm must end.This occurs either when the algorithm has reached some maximum runtime, or the algorithm has reached some threshold of performance. At this point a final solution is selected and returned. [4]

The question of encoding comes from the question of how do we wish to represent individuals genetically in our algorithm. The way in which we encode our individuals lays out the path for how our algorithm will handle the key evolutionary processes: selection, mutation and crossover (also known as recombination). Any encoding will fall into one of two categories, direct or indirect. [2] Evolutionary algorithms operate on genotypes. In neuroevolution, a genotype (in Biology: genetic representation of the creature; in Neuroevolution: encoding of the neural net) is mapped to a neural network phenotype (in Biology: physical representation (think visible features) of the creature; in Neuroevolution: Actual topology and weights of the neural net) that is evaluated on some task to derive its fitness. A direct encoding will explicitly specify everything about an individual. If it represents a neural network this means that each gene will directly be linked to some node, connection, or property of the network. This can be a binary encoding of 1s and 0s, a graph encoding (linking various nodes by weighted connections), or something even more complex. The point is that there will always be a direct connection between genotype and phenotype that is very obvious and readable. [2] An indirect encoding is the exact opposite. Instead of directly specifying what a structure may look like, indirect encodings tends to specify rules or parameters of processes for creating an individual. As a result, indirect encodings are much more compact. The flip side is that setting the rules for an indirect encoding can result in a heavy bias within the search space, therefore, it is much harder to create an indirect encoding without substantial knowledge about how the encoding will be used. [2]

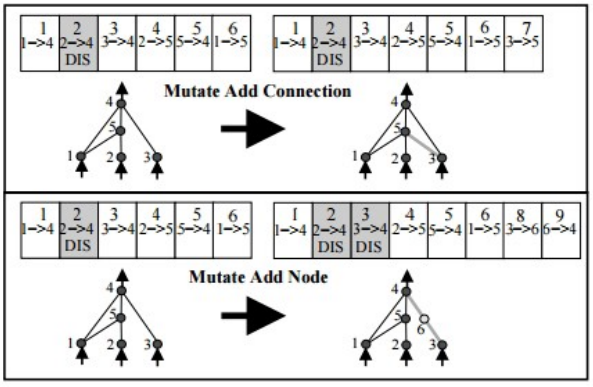

Fig. 3. Source: [2]

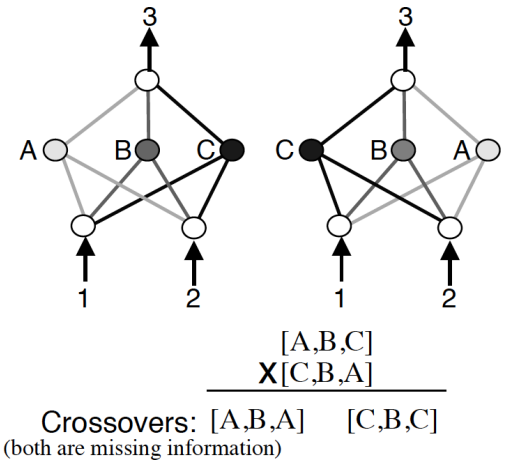

Fig. 4. Source: [2]

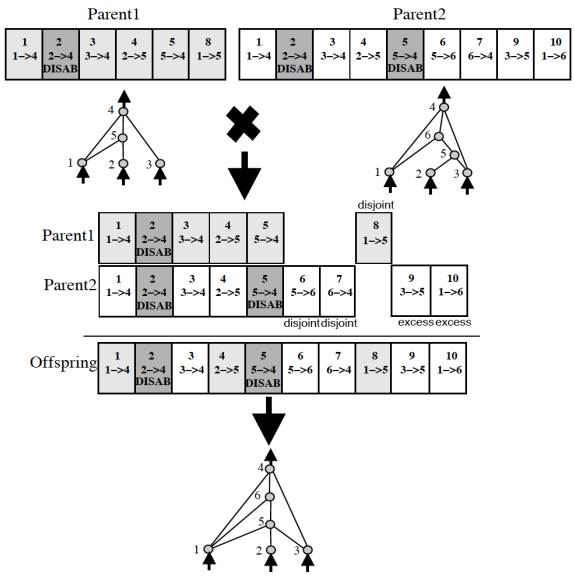

Fig. 5. Source: [2]

IV. NEAT

NEAT evolves both the parameters and architecture of ANNs and thus is an example of TWEANN (Topology and Weight Evolving Artificial Neural Network). NEAT employs a set of three different mutations: i) modify a weight, ii) add a connection between nodes, and iii) insert a node while splitting an existing connection. [5]

Novelty search is another invention that abandons the goal to maximise a fitness function. Instead, it rewards individuals that behave differently. This change often leads to much better solutions in the end because novel solutions perform badly in the beginning and need time to fully manifest their potential. [5]

In NEAT, mutation can either mutate existing connections or can add new structure to a network. If a new connection is added between a start and end node, it is randomly assigned a weight. [2]

Another big issue with evolving the topologies of neural networks is something that the NEAT paper calls “competing conventions.” The idea is that just blindly crossing over the genomes of two neural networks could result in networks that are horribly mutated and non-functional. If two networks are dependent on central nodes that both get recombined out of the network, we have an issue. [2]

More than that, genomes can be of different sizes. How do we align genomes that don’t seem to be obviously compatible? In biology, this is taken care of through an idea called homology. Homology is the alignment of chromosomes based on matching genes for a specific trait. Once that happens, crossover can happen with much less chance of error than if chromosomes were blindly mixed together. [2]

NEAT tackles this issue through the usage of historical markings (as seen above), which are used to properly line up encodings for any sort of crossover. By marking new evolutions with a historical number, when it comes time to crossover two individuals, this can be done with much less chance of creating individuals that are non-functional. Each gene can be aligned and (potentially) crossed-over. Each time a new node or new type of connection occurs, a historical marking is assigned, allowing easy alignment when it comes to breed two of our individuals. [2]

A very interesting idea put forth in NEAT was that most new evolutions are not good ones. In fact, adding a new connection or node before any optimization of weights have occurred often leads to a lower performing individual. This puts new structures at a disadvantage. How can we protect new structures and allow them to optimize before we eliminate them from the population entirely? NEAT suggests speciation. [2]

Speciation simply splits up the population into several species based on the similarity of topology and connections. If the competing convention problem still existed, this would be very hard to measure! However, since NEAT uses historical markings in its encoding, this becomes much easier to measure. A function for deciding how to speciate is given in the paper, but the important part to note is that individuals in a population only have to compete with other individuals within

that species. This allows for new structure to be created and optimized without fear that it will be eliminated before it can be truly explored. [2]

More than that, NEAT takes things one step forward through something called explicit fitness sharing. That means that individuals share how well they are doing across the species, boosting up higher performing species, though still allowing for other species to explore their structure optimization before being out evolved. [2]

Complexification: We'll start with simple and evolve complexity if and when it's needed

A large goal of the NEAT paper was to create a framework for evolving networks that allowed for minimal networks to be evolved. The authors did not want to create an algorithm that first found good networks and then had to reduce the number of nodes and connections after the fact. Instead, the idea was to build an algorithm that started with the minimal amount of nodes and connections, evolving complexity as time goes on if and only if it is found to be useful and necessary. [2]

NEAT sets up their algorithm to evolve minimal networks by starting all networks with no hidden nodes. Each individual in the initial population is simply input nodes, output nodes, and a series of connection genes between them. By itself, this may not necessarily work, but when combined with the idea of speciation, this proves to be a powerful idea in evolving minimal, yet high-performing networks. [2]

V. HyperNEAT

HyperNEAT drops the idea of direct encoding and replaces it with indirect encoding, because in order to evolve a network like the brain (with billions of neurons), direct coding can not provide the necessary speed to make this approach feasible.

A. Motivation for HyperNEAT

The creators of HyperNEAT highlight that if one were to look at the brain, they see a network with billions of nodes and trillions of connections. They see a network that uses repetition of structure, reusing a mapping of the same gene to generate the same physical structure multiple times. The human brain is constructed in a way so as to exploit physical properties of the world: symmetry (have mirrors of structures, two eyes for input for example) and locality (where nodes are in the structure influences their connections and functions). Traditional dense, feed-forward networks where all nodes in one layer are connected to all nodes in the next do obviously not follow this structure. Neither do the networks constructed by the vanilla NEAT algorithm? They tend to be disorganized, sporadic, and not exhibit any of these nice regularities. Enter in HyperNEAT! Utilizing an indirect encoding through something called connective Compositional Pattern Producing Networks (CPPNs), HyperNEAT attempts to exploit geometric properties to produce very large neural networks with these nice features that we might like to see in our evolved networks. [3]

Instead of using the NEAT algorithm to evolve neural networks directly, HyperNEAT uses NEAT to evolve CPPNs.

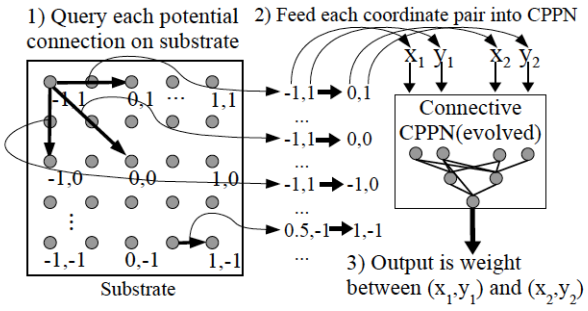

Fig. 6. Source: [3]

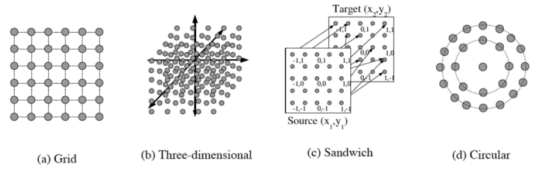

Fig. 7. Source: [3]

B. Compositional Pattern Producing Networks (CPPNs)

Things like Gaussians give rise to symmetry and trigonometric functions help with repetition of structure.

1) Substrate: In the context of HyperNEAT, a substrate is simply a geometric ordering of nodes. The simplest example could be a plane or a grid, where each discrete (x, y) point is a node. A connective CPPN will actually take two of these points and compute weight between these two nodes. We could think of that as the following equation: $CPPN(x_1, y_1, x_2, y_2) = w$ [3]

In doing this, every single node will actually have some sort of weight connection between them (even allowing for recurrent connections). Connections can be positive or negative, and a minimum weight magnitude can also be defined so that any outputs below that threshold will result in no connection. [3]

In the case where the nodes are arranged on some sort of 2 dimensional plane or grid, the CPPN is a function of four dimensions and thus we can say it is being evolved on a four dimensional hypercube. This is where HyperNEAT gets its name from. [3]

the configuration of the substrate is critical. Therefore, the structure of our nodes are tightly linked to the functionality and performance that may be seen on a particular task.

Interesting substrates are the three dimensional cube(making CPPN six-dimensional) and the sandwich configuration allowing for nodes on one half to connect to the other half. This can be seen easily as an input/output configuration.

VI. Conclusion

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.

REFERENCES

[1] Example Cite, *Source*, Apr. 2019. www.example.com
[2] https://towardsdatascience.com/neat-an-awesome-approach-to-neuroevolution-3eca5cc7930f
[3] https://towardsdatascience.com/hyperneat-powerful-indirect-neural-network-evolution-fba5c7c43b7b
[4] https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac
[5] https://www.inovex.de/blog/neuroevolution/