An overview of neuroevolution techniques

Vincent Hoekstra

December 14, 2011

Preface

This paper is an obligatory part of the Business Mathematics and Informatics course program at the VU university in Amsterdam. It is written by Vincent Hoekstra, a student of the Business Mathematics and Informatics master program. The goal of this paper is to provide the reader with a complete overview of the field of neuroevolution, i.e. the field that uses evolutionary algorithms to train neural networks.

I would like to thank Guszti Eiben and Berend Weel for their support and advice during the process of writing this paper.

Contents

Chapter 1

Introduction

1.1 Goal

The human brain is responsible for all of human's inventions and societies, it is a special object capable of incredible calculations. The system that made this object possible was evolution. Over the last decades humans have made progress in both making machines that are capable of handling a lot of calculations in a short time span, and in understanding the underlying mechanisms of the human brain. Nonetheless, computation done by computers is not at all similar to how the human brain handles computations. Artificial Neural Networks are an attempt to use the same structure as that of the human brain, but now used by software and hardware.

Neural networks are networks of nodes connected in a similar way as how the neurons in the brain are connected. But just as the human brain itself, such a network is not of much use until it is properly trained. Many ways exist to train these neural networks, and this paper concentrates on one of them. Human brains were developed by biological evolution and in a similar way neural networks can be trained by using evolution.

Evolutionary computing uses the same characteristics as biological evolution to search for optimal solutions to computational problems. Neuroevolution is a method to train neural networks by using this problem-solving technique. The goal of this paper is to present an overview of all the neuroevolution techniques and to give a summary of this promising training method.

To give this overview, the paper is divided in the following sections. First, the underlying structures and computational techniques of both neural networks and evolutionary computing are explained to bring the reader up to speed. Then the paper is divided in to three parts, in these parts neuroevo-

lution is discussed in terms of the evolutionary representation, the types of neural networks and the applications for which neuroevolution is particularly suitable. Of particular interest is the relative frequency of the different techniques that are used for neuroevolution and the development of these techniques over the years. By presenting the results in this way, the reader is presented with a complete overview of the field of neuroevolution.

1.2 Neural networks

Artificial neural networks are computational structures modeled after techniques in biology, particularly the brain and the nervous system. Neural networks consist of three types of nodes: input nodes, hidden nodes and output nodes. Normally the input and output nodes are given by the task to which the neural network is assigned. For example, consider a network that is used for controlling an artificial robot in search for food. Input nodes could be assigned to the different sensors of the robot, output nodes could be assigned to actions like eating, moving in a certain direction or staying put.

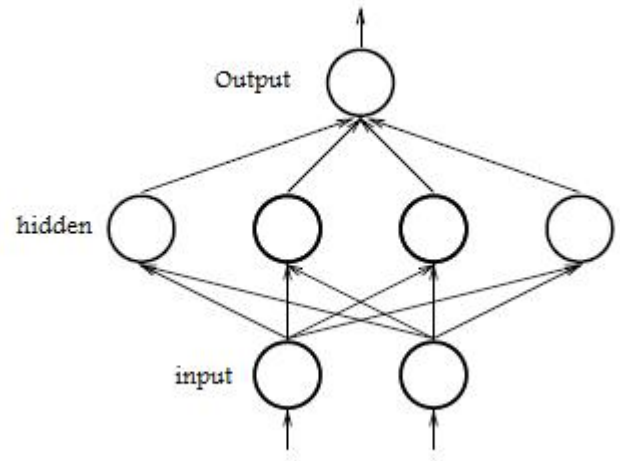

Figure 1.1: An example of an artificial neural network structure. *source: Evolving Networks Using the Genetic Algorithm with Connectionist Learning, [BMS90]*

In a simple neural network, output nodes and hidden nodes have connections linking to them from other nodes, all these links have a specific weight.

The value of hidden or output nodes can be calculated by a function that takes as input the weighted sum of all nodes linked to that specific node, this function is called the *transfer function*. By training each weight in the network, the network can provide the right output given some input values. Figure 1.1 shows an example of an artificial neural network. The network that is shown here is a feedforward network, which means a network without any connection loops.

There are multiple characteristics that describe a neural network. The opposite of a feedforward network is a recurrent network, recurrent networks can have connection loops and even connections going to input nodes or connections from output nodes. Some networks also have multiple layers of hidden nodes. If a hidden node has a connection to another hidden node, the second node is part of a higher layer within the network structure. The overall amount of hidden nodes is a measure, albeit not an exact one, for the complexity that the network can handle. More difficult tasks in general need more hidden nodes. Another characteristic of neural networks is the type of transfer function that is used by a neuron to calculate its output. Neural networks are applied in many fields, they can, among other things, be used for classification problems, optimization problems and controlling artificial agents.

1.3 Evolutionary computing

Evolutionary algorithms are a class of problem solving algorithms that work similar to biological evolution in that it improves a population of individuals instead of just one individual. Biological evolution can be depicted in the following way. There is an environment that can only support a certain number of individuals, these individuals compete for certain resources. Because the environment can not support a limitless population size, there has to be some selection which members of the species are able to survive and to reproduce. The individuals that are able to compete for the resources in the best way are favoured by this selection process, Darwin, [Dar], called this "Survival of the fittest". Combined with the fact that some specific traits that makes a individual fitter than its rivals can be inherited, this provides a system where on average each generation is fitter than the previous one.

There is an important distinction between the phenotype of an individual and its genotype. The phenotype is the form of the individual in the environment where its fitness is tested. The genotype is the encoding of the phenotype which can be inherited by offspring. For example in biology, Someone's body and mind would be considered the phenotype and someone's

chromosomes would be considered the genotype.

Evolutionary algorithms are based on this system. But in this case the population does not consist of biological species but of potential solutions to a problem. Evolutionary algorithms provide a structured search through the solution space. It is an iterative procedure where each iteration equals a generation. In each generation individual solutions are evaluated, the best solutions recombined into offspring, the offspring is subject to some mutation and at last the best solutions are select to survive to the next generation. In this process the phenotype representation is the representation that is evaluated and the genotype is the representation that is mutated and recombined into offspring. For example in neuroevolution, the phenotype is the actual neural network and the genotype is some representation of which the entire network can be constructed, for example a vector of all the weights in the network.

Due to this clear distinction between the genotype and phenotype evolutionary algorithms perform well in unsupervised learning problems and more complex problems. An important challenge in evolutionary algorithm design is to keep the population as diverse as possible to keep the population from converging prematurely.

Chapter 2

Representations

In this section the different representations of the genotype will be discussed. This choice of representation is a crucial part of any evolutionary algorithm and determines what kind of crossover and mutation can be applied. Floreano et. al.[FDM08] distinguishes three types of representations.

- Direct representations.
- Developmental representations.
- Indirect or implicit representations.

Sections 2.1-2.3 will discuss each of these representations and what sets that type of representation apart from the others. these sections will also deal with specific representations that fall into that category and how mutation and crossover work in each of these types of representation. Section 2.4 will provide the reader with a comparison and overall taxonomy of the representations.

2.1 Direct representations

Direct representations are representations where (parts of) the genotype has a 1-to-1 mapping with (parts of) the phenotype. This means that every part of the genome can be translated into a specific part of the neural network. An advantage of this representation is that it is easier to understand how the network is constructed from the genotype representation. Though, there are some negative effects as well. For example, when dealing with a bigger network, the genotype representations gets bigger with a similar amount, which means more computation time.

Most direct representations have a fixed number of neurons and the sole task of the evolutionary algorithm is finding the weights of the links between the nodes. This means that *a priori* knowledge is necessary to determine the appropriate amount of hidden nodes and therefore that these fixed-size-representations are less robust than other representations. In general a rule of thumb is, if there are more hidden nodes in a neural network, then the network can handle more complex problems (with more dimensions). That these fixed-size representations are less robust was described by Stanley in [Sta04] in the following way: when the chosen number of hidden nodes is too small, the optimal solution might not exist in that search space. But if the chosen number of nodes is too large the search space becomes too big to find the optimal solution.

The alternative is to evolve the network structure as well. This involves more complex representations and therefore a different search space. This section will describe some of these fixed size direct neuroevolution representations and alternatively, the direct topology representations.

2.1.1 Weight vector representations

Some of the earliest neuroevolution algorithms (around 1990-1995, [BMS90, Wie91] were weight vector representations of a fixed size and had a fixed topology. A number of hidden nodes would be chosen and the genotype would be a (real) vector of all the connection weights. For example, Belew et. al. [BMS90] used a bit string representation for the weights. These types of neuroevolution algorithms are often called *conventional neuroevolution (CNE)* methods because these methods were the first succesful attempts at neuroevolution. In these cases one genotype encompasses all the weights in the network and can be translated into a full neural network. Mutation was mostly done by letting every weight have a small chance of changing between generations. Recombination could be done by swapping parts of the vectors. These representations often had a big chance of premature convergence and therefore other methods were developed, for example by Moriarty and Miikkulainen, [MM96], These methods will be discussed later.

After a period of time that different techniques were preferred, weight vector representation have again gained in popularity recently. Christian Igel, [Ige03], had a very succesful attempt at creating a neuroevolution algorithm based on real weight vectors which was based on (mu/mu,lambda) Covariance Matrix Adaptation - Evolution Strategies (CMA-ES). This method proved succesful when used with small population sizes. Recombination was done by global intermediate recombination and gaussian mutation was applied. Gomez et. al, [GSM06], also developed an effective algorithm that uses real

weight vectors, but they made use of a subpopulation for every individual weight in the network, and permuting these subpopulations to create and evaluate networks. This way they ensured that the population stayed as diverse as possible. They called this technique CoSyNE.

In conclusion, weight vector representations are still applicable and have been gaining popularity in recent years. The fixed topology is still quite restricting, but recent attempts with fixed topology have been performing better or equal to other more complex topology-evolving methods. This added to the fact that weight vectors are easily comprehensible makes them a viable class of neurevolution algorithms.

2.1.2 Neuron representations

In an attempt to improve diversity in populations of conventional neuroevolutionary algorithms, Moriarty and Miikkulainen came up with an innovative idea in 1994, [MM96]. Instead of using a vector representation of an entire network, individuals in a population depicted single neurons (with their respective weights). They called this method Symbiotic, Adaptive Neuro Evolution (SANE). This method is only applicable when considering a fixed size (feedforward) topology. Every individual in the population depicts a hidden node in the form of a vector of its weights and all of the hidden nodes have a link to each output/input node. The goal of this technique was to create different species within the population. because in an optimal neural network every hidden node performs a specific task within the network. Individuals were evaluated by randomly selecting an amount of neurons equal to the size of the network and measuring the performance of this network, every node in the network would get this fitness value added to their set of fitness values. After multiple evaluations every hidden node would have been measured a couple of times and the overall fitness of a node would be the average of all of the performances of the networks that it was in. This method ensures that at some point neurons start to specialize in certain tasks. Neurons would be ranked according to average fitness and the top 25% was used for crossover. Crossover would be performed by using one-point crossover within the real valued vector of each neuron, which creates two children for every two parents. Mutation would change individual weights with a chance of 1%.

In 1999, Gomez and Miikkulainen extended the system of SANE by using sub-populations. They called this technique Enforced Sub-Populations (ESP), it used a similar algorithm as SANE but it had a sub-population for every hidden node in the network. ESP had two advantages over the original neuron representation when considering the evolutionary process. For

one, the species that would evolve gradually when executing SANE would already be in place and second, there would only be recombination within a certain species. Another major advantage was that the prerequisite of a only feedforward network is no longer necessary. Hidden nodes from different sub-populations could develop links to each other after a sufficient amount of generations. Another advantage of ESP in comparison to conventional neuroevolution is that in CNE networks with a single "bad" node could still have a high fitness, in ESP every type of node is evaluated by its own average performance, so this event is not likely to happen. In 2000, Kaikhah and Garlick [KG00], made another extension to the neuron representation technique. They extended the algorithm by allowing a variable amount of hidden nodes.

These neuron representations were one of the first deviations from the conventional neuroevolution path, and are based on a quite unique representation that is worth mentioning. However, both implicit and other direct representations have been performing better over the last decade.

2.1.3 Topology representations

in 2002, Stanley and Miikkulainen [SM02b] argued that the topology of a neural network also affects their functionality. However, traditional neuroevolution focused on choosing a fixed topology in advance and just evolving the network weights (See previous sections). The question was, if evolving the topology would be advantageous compared to evolving only the weights and assuming full connectivity between nodes. The main advantage of evolving the structure would be that no heuristic methods would have to be used to find the appropriate amount of hidden nodes and thus could lead to a more robust algorithm.

One of the first attempts of evolving the network topology next to the connection weights was done by Dasgupta and Mcgregor, [DM92], in 1992. For their evolutionary algorithm they implemented a two-level structured genetic algorithm. In this representation, the first level was a binary connection matrix to determine which nodes had conections between them. The second level was a bit string that represents the connection weights. This approach could handle both feedforward and recurrent networks and had the property that feedforward networks only used they upper triangle of the connectivity matrix. A drawback of this system was that a connection matrix was necessary that was of size n x n where n is the number of hidden nodes.

In 1994, Angeline et. al. [ASP94] argued that crossover, and therefore Genetic Algorithms, is not well-suited for evolving the network topology. Instead, they proposed a method that was based on evolutionary programming.

Evolutionary programming is a class of evolutionary algorithms where there is no parent selection, offspring is solely created by mutation. They called this technique the GNARL Algorithm (which stands for GeNeralized Acquisition of Recurrent Links). The search space for this algorithm was relatively big, because networks had only three restrictions, given a maximum number of nodes, in each network: 1. There had to be no links to an input node. 2. There had to be no links from an output node. 3. There should be at most one link from node x to node y. This allowed a lot off useless nodes in the network, for example, hidden nodes could exists with no links to it. Connectivity weights were modeled as real weights and were mutated by gaussian noise. structural mutation was done by adding nodes or links. To make sure that these changes had no radical effects on the fitness compared to the parent, new connections would be initialized with zero weight. Another mutation was the deletion of a node, where there is no way of compensating for an eventual radical change in fitness. An interesting feature of GNARL was the fact that it uses an individuals temperature to determine the severity of the mutation. The temperature *T(i)* of and individual i was calculated by:

$$T(i) = \frac{fitness(i)}{fitness_{max}}$$

This way an individual with a fitness that is far less than the optimal fitness is mutated severely, but an individual that is close to the optimal solution is only mutated slightly. This is a way of ensuring that the search converges to the optimal fitness.

One of the more succesful direct representations that can also evolve the structure of the network is called NEAT (Neuro Evolution of Augmenting Topologies) and is developed by Stanley and Miikkulainen in 2002 [SM02b, Sta04]. They argued that one of the main problems for topology representations and neuroevolution in general is that of *Competing Conventions*. Which means that multiple different genotypes decode into the same phenotype, which could have a serious negative effect on the algorithm, because two fit parents that accidentally have the same phenotype will likely produce inferior offspring. NEAT brought two major innovations:

1. To battle the competing conventions problem, NEAT would store ancestral information of each individual.

2. Neat would use speciation and would use different fitness measures for newly formed topologies and fully weighted networks.

NEAT uses *explicit fitness sharing*, which means that similar individuals share their fitness values. Furthermore, because smaller networks are not

easily evolved when an initial population has a lot of bigger networks, NEAT starts out with no hidden nodes, the simplest neural network possible. The representation that NEAT uses is specifically developed to allow meaningfull structural crossover, each genome consists of a list of connections. Each connection specifies the in-node, out-node, connection weight, whether or not the connection is enabled, and an innovation number. The innovation number is used to find corresponding genes.

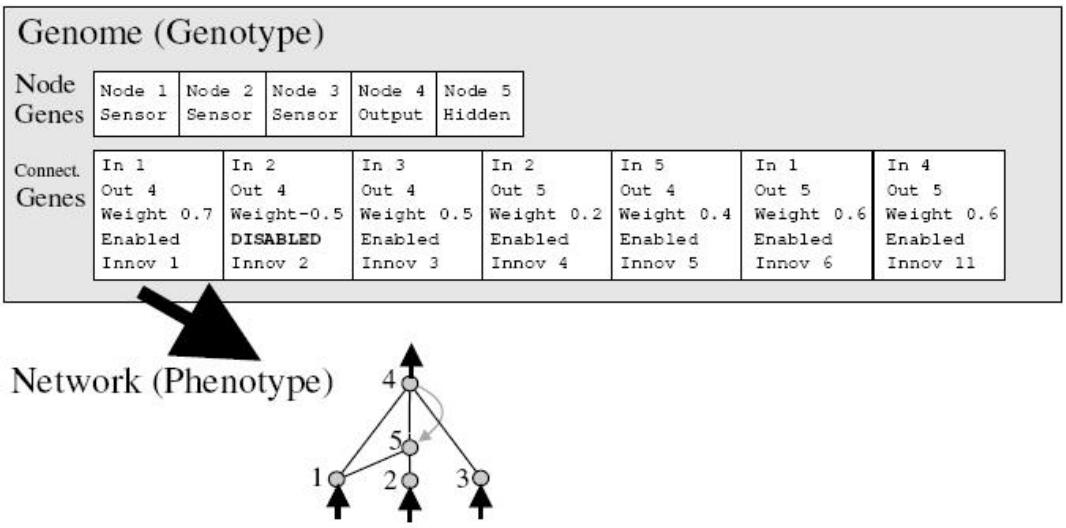

Figure 2.1: The genotype and phenotype representation of the NEAT algorithm, consisting of nodes and connections. *Source: Efficient Evolution of Neural Networks through complexification, [Sta04].*

Mutation can be done by altering the weights of a connection, or mutation can be structural which is done in two ways: adding a node or adding a connection. When adding a node, it is done by splitting one connection into two parts seperated by the new node. It is clear that, in the NEAT system, mutation is used to create bigger networks. Whenever a new connection is created, it gets a new innovation number, NEAT ensures that each connection between two of the same nodes (in every member of the population) have the same innovation number. It creates speciation on the connection level. This way, crossover can be done in a sensible way by comparing the innovation numbers of the individual connections within both genomes. This is shown in Figure 2.2.

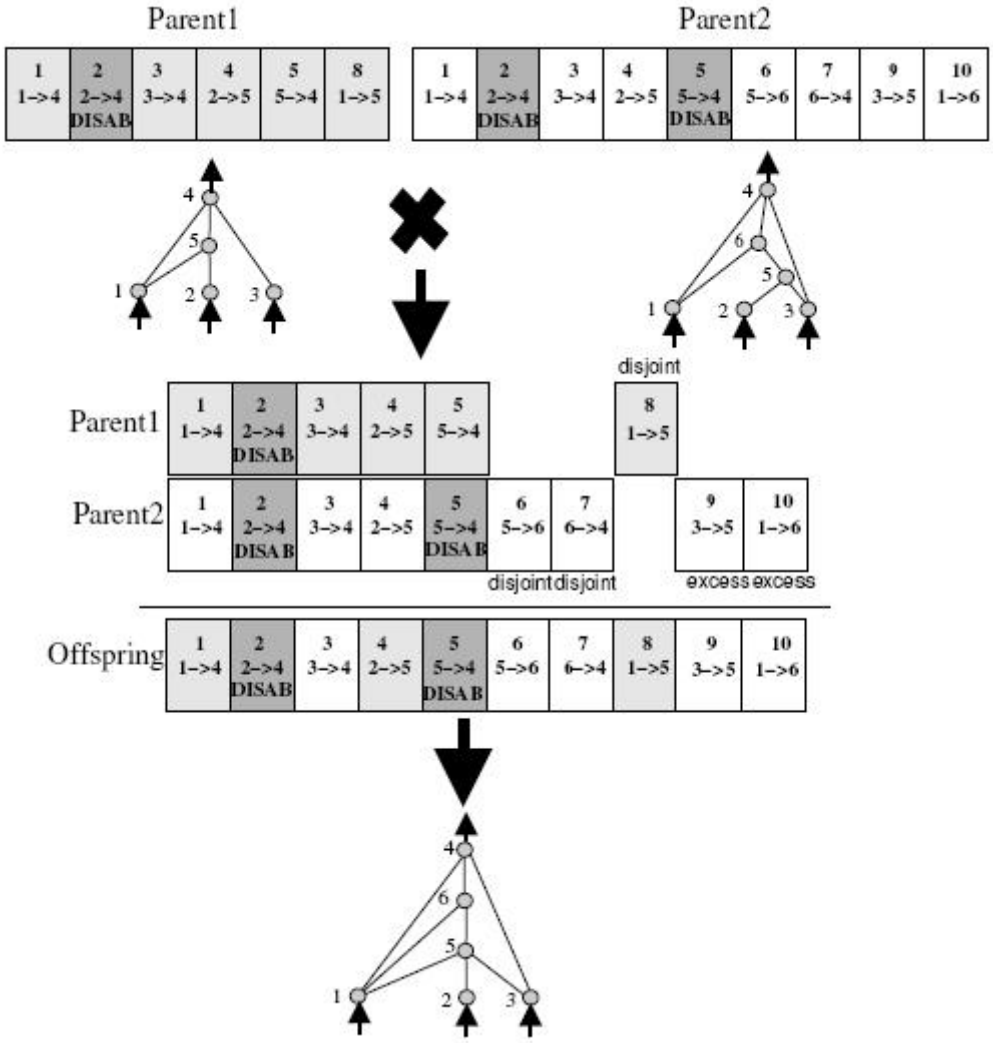

Figure 2.2: Crossover in the NEAT algorithm. Both parents look different but have a couple of similar connections, which is shown by the innovation numbers. Each connection that exists in both parents also exists in the offspring, whenever a connection is present in one of both parents it has a chance of being added to the offspring. The same is true when connections are disabled in one or both of the parents. *Source: Efficient Evolution of Neural Networks through complexification, [Sta04].*

2.2 Developmental representations

Developmental representations were created to counter one of the disadvantages of direct representations, direct representations faced difficulties when larger neural networks had to be constructed. Developmental representations tried to tackle this problem by assuming that larger neural networks are build up from smaller pieces. Gruau, [Gru95], made the assumption that the human brain has to have some modular structure to be able to do the computations that the brain is capable of, in a similar way that most computer programmes use modularity. When assuming that larger neural networks have these smaller modules for specific tasks, evolutionary algorithms could benefit from this feature. Developmental representations are representations that are specificially designed to exploit this characteristic.

Kitano, [FDM08], was one of the first to use this encoding method. He used a technique that is similar to the structured genetic algorithm, [DM92], discussed earlier. Kitano also used a connectivity matrix consisting of 1's and 0's. Instead of encoding the entire matrix, the representation was in the from of a 2 x 2 matrix of symbols and some rules which specified how to develop the entire connectivity matrix. Recursively, each symbol in a 2 x 2 matrix of symbols would develop into its own 2 x 2 matrix, up until a point is reached where each 2 x 2 matrix is a predetermined matrix consisting of 1's and 0's. Figure 2.3 shows how this process works.

Gruau, [Gru95], created another developmental representation called *Cellular Encoding*, which was inspired by cell division in biology. It is based on a language that is used to construct networks from a single cell. A cell would represent a node in the network and when the cell divided into two childcells specific rules would describe how the links between the two new cells and the already existing cells would be established. Floreano et. al., [FDM08], concluded that despite that ability for developmental representations to create large networks with compact representations, it is not clear to what extent this contributes to solving a problem. The modular architectures that are used by developmental representations can be useful in specific situations, for example controlling a multi-legged robot. However, in these cases, developmental representations still needed additional mechanisms to deal with several other problems.

2.3 Indirect representations

Indirect representations are inspired by biological genes, in particular *Gene Regulatory Networks* and DNA. biological gene networks can be seen as a se-

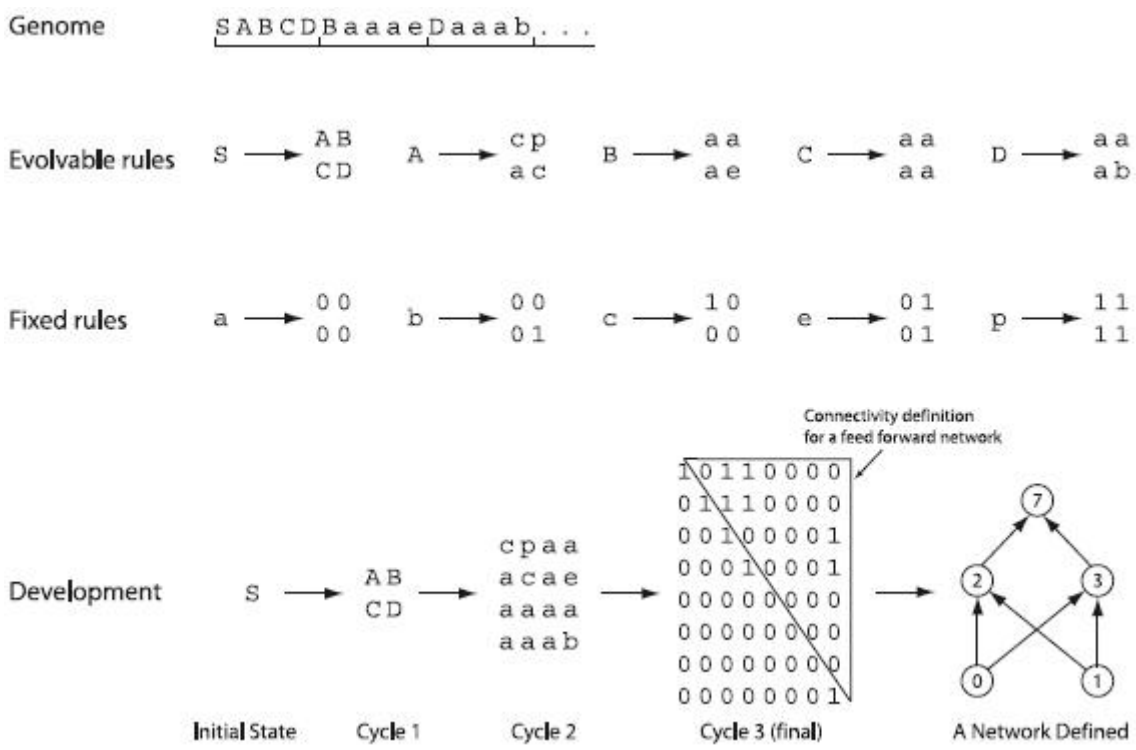

Figure 2.3: Kitano's developmental representation consisting of evolvable rules and fixed rules.*Source: Neuroevolution: From architecture to learning, [FDM08].*

quence of genetic characters, with a coding region that is specified by specific coding patterns, called *promoter* and *terminator* regions, [FDM08]. Implicit encodings are based on this system and the most succesful and well-known application is the technique called *Analog Genetic Encoding* or AGE, [DMF06].

The genome in AGE is represented by a sequence of characters from a finite alphabet, for example the ASCII uppercase alphabet. Special strings of characters are defined as *device tokens* and *terminal tokens* that respectively signal the start and end of specific coding regions in the genome. Each node in the network is encoded in the genome string in the following way: Device token - sequence of characters - terminal token - sequence of characters - terminal token. The first sequence of characters is used to find all the weights of connections going to the node and the second is used to find the weight for connections from the node. This means that, just as is the case with developmental representations, a specific set of fixed rules is necessary that are used to calculate the weights from the character sequences. This set of

rules is called an *interaction map*, a weight is calculated by combining the characters of the outgoing node with those of the ingoing node and using the interaction map to find the weight. This is explained in Figure 2.4.

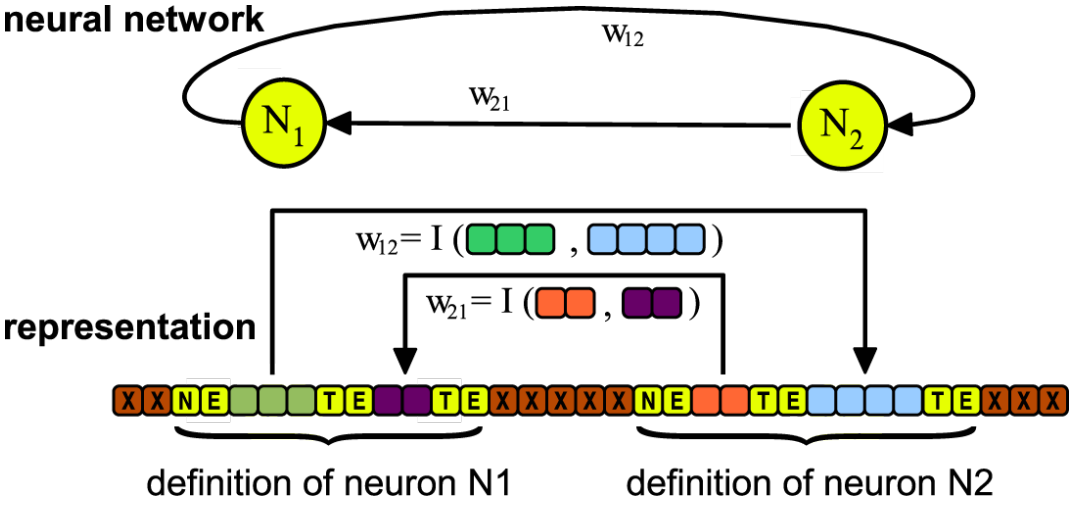

Figure 2.4: The representation and phenotype-genotype mapping of the AGE algorithm. *Source: Neuroevolution with Analog Genetic Encoding, [DMF06].*

In the genome, if a device token is not followed by enough terminal tokens that part of the genome is considered invalid. This way there is no protection needed for genetic operators like mutation and crossover and they can be applied to the token, coding, and non-coding parts of the genome. The initial population is created by generating individuals with a given number of neurons with random terminal tokens. The following mutation and crossover methods can be used in the AGE algorithm, from [DMF06].

- **Character deletion, insertion and substitution.** A character is removed, inserted or substituted in the genome.
- **Fragment deletion, transposition and duplication.** Two points of the genome are chosen and the intervening fragment is deleted, transferred or copied to another point of the genome.
- **Device insertion.** A device token is inserted in the genome (with randomly generated terminal tokens.
- **Homologous Crossover** Fragments of the genome are recombined using a randomly chosen crossover point and searching another parent which has a sequence of characters similar to that in the area around

the crossover point. Then children are created by swapping the strings on both sides of the crossover point, [Mat05].

- **Genome duplication.** The entire genome is duplicated.

Implicit representations are robust and are modeled after a succesful biological system. Its disadvantage is that it is a complex way of encoding and at this point simpler representations are able to do the same task. It also needs a set of fixed rules to construct the phenotype but does not have the advantage of developmental encodings that the size of the genome is reduced effectively. Indirect representations are relatively new and new developments might improve its popularity.

2.4 Taxonomy

Neuroevolution has been developed over the last couple of decades, it is interesting to know which types of algorithms are used more frequently and in which direction the field is heading. The first representations had a fixed topology and a fixed number of hidden nodes. These representations had some drawbacks and other representations were developed to circumvent these disadvantages. These methods ranged from direct representations, like NEAT, to developmental and indirect representations. Recently however, fixed topology networks like CMA-evolution strategies and CoSyNE are gaining popularity. This might be because these simpler representations do have a smaller search space, because they do not have to search for the right topology. Figure 2.5 shows the overall taxonomy of previously mentioned representations. Figure 2.6 shows a timeline from the first application of neuroevolution until now. NEAT is still the most popular representation to date, followed by CMA-ES. Developmental approaches have not seen much use in recent years and AGE is by far the most succesful implicit representation. AGE, CoSyNe, and CMA-ES have all had similar or better results than NEAT at several benchmark tests, the popularity of the NEAT algorithm could be ascribed to its understandability combined with its robustness.

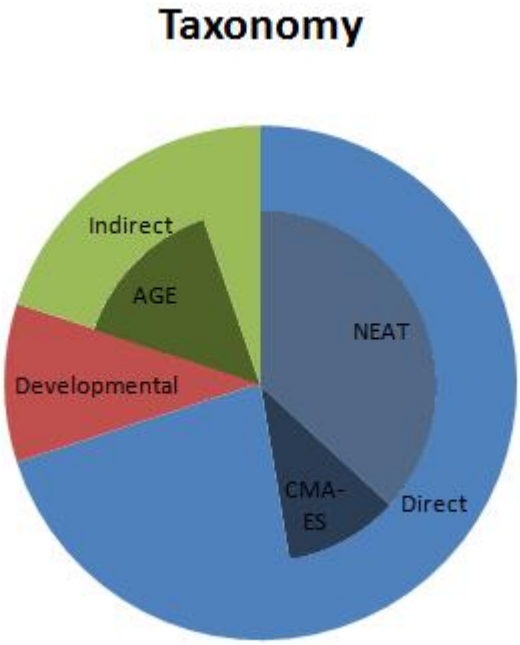

Figure 2.5: The taxonomy of neuroevolution representations.

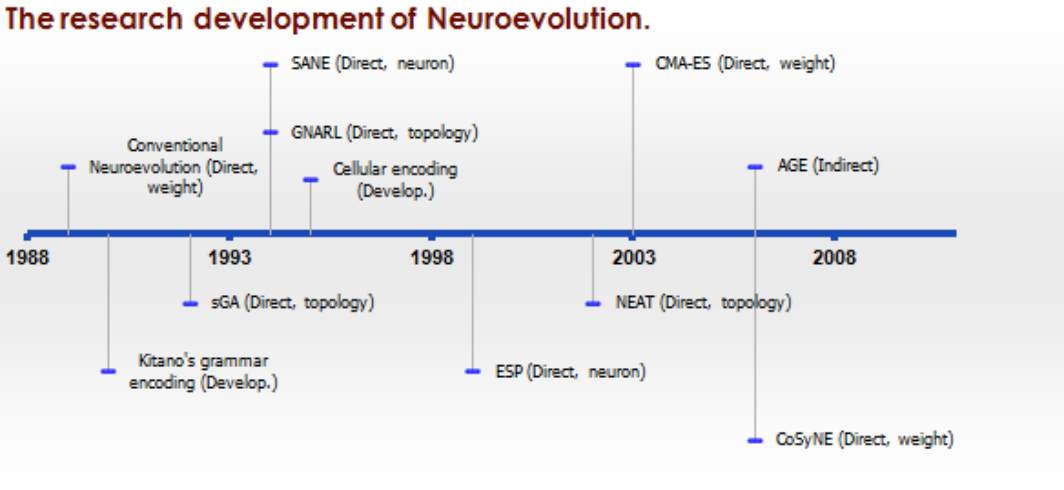

Figure 2.6: Timeline with the years that the most important neuroevolution techniques were developed. the type of representation of each technique is between the brackets.

Chapter 3

Neural Networks

There are multiple types of neural networks, the purpose of this section is to discuss which kind of networks are typically used combined with evolutionary algorithms. One of the characteristics of neural networks is the type of transfer function that is used by the neuron within the network. figure 3.1 shows a typical artificial neuron.

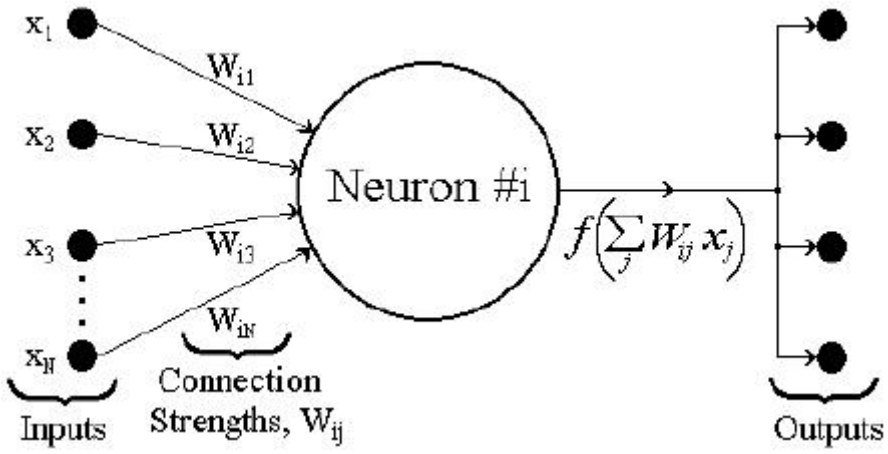

Figure 3.1: A typical artificial neuron, The input for the transfer function is the weighted sum of the input connections. *Source: Computing with Spiking Neurons, [FBoTDoES98].*

The transfer function *f(x)* can differ, using a binary threshold is an example of a discrete transfer function. A sigmoid function is an example of an analog transfer function. Both examples are shown in figure 3.2. The sigmoid function is the transfer function that is most used in neural networks,

[FBoTDoES98]. Usually the transfer function is chosen in advance and is not incorporated in the genome representation, because evolving the weights already has the desired effect.

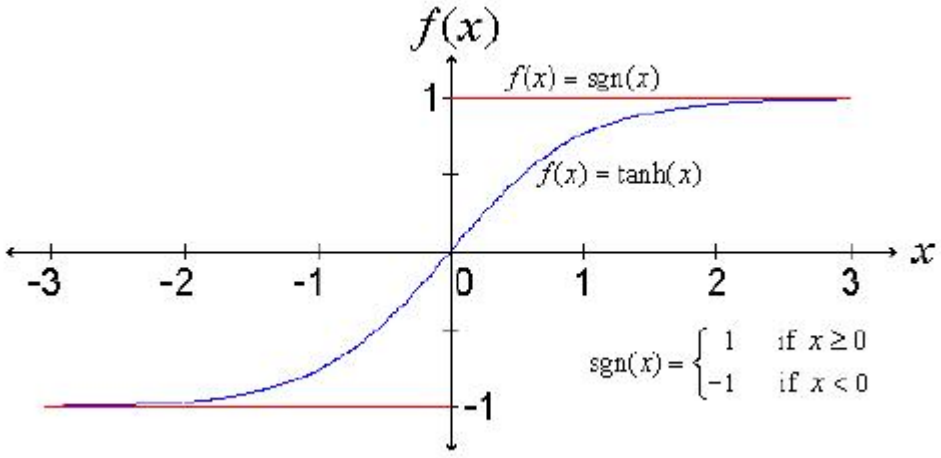

Figure 3.2: Two examples of transfer functions, the sygmoidal transfer function is most used in practice. *Source: Computing with Spiking Neurons, [FBoTDoES98].*

3.1 Feedforward vs. Recurrent

The goal of this section is to give an overview when feedforward networks are used and when recurrent networks are preferred. Clearly, the type of network that is used depends on the representation of the genome and the task for which the neural network is used. While recurrent networks are capable of computing more complex tasks, allowing for recurrent connections also increases the amount of networks that encompass the search space. Most topology evolving algorithms do not have restrictions considering recurrent networks, so when such a structure is used it automatically allows recurrent connections. Some algorithms, like SANE, do not allow for recurrent connections and some, like ESP and CoSyNE, allow for the user to determine himself what kind of fixed topology needs to be used. In general, recurrent networks are more often used in applications that involve artificial agents or robots, because recurrent connections allow for memory to be involved in the decision process, [AbBR01, KG00]. The application that uses the neural network should determine whether or not a recurrent network is necessary

and an appropriate representation needs to be chosen.

3.2 Other types of networks

Spiking neural networks are neural networks that behave more similar to biological neural networks than standard artificial networks because biological neural networks also display spiking behavior, [FBoTDoES98]. It has also been proved that spiking neural networks are computationaly more powerful than some of its original artificial neural network counterparts, [OCBM$^+$11]. Neurons in spiking networks behave differently than neurons in other artificial neural networks. In general, each neuron in a spike network has a *voltage potential* and a *threshold level* and a neuron emits a spike whenever the voltage potential is higher then the threshold level. After a spike a neuron needs some time to cool down before it can emit another spike. The voltage level is raised by incoming spikes or incoming sensory data for input nodes, connections weights are used to calculate the change in voltage potential that a spike causes. This model is called the *integrate-and-fire model* and is most frequently used when considering spiking networks.

When evolving spiking neural networks, usually both the connection weights and threshold levels are evolved. Both of these are real values, so this does not change much for the already existing representations except for adding another dimension. Both fixed topology and topology evolving representations can be applied when using spiking neural networks. O'halloran et. al. [OCBM$^+$11], used the NEAT algorithm for spiking neural networks to classify breast cancer tumours as either benign or malignant. Floreano and Mattiussi, [FM01], used a fixed topology spiking neural network to evolve the behavior of a robot.

Spiking neural networks are not as frequently used as networks that use a sigmoid transfer function. This could be because it is a relatively new technique and more difficult to implement and comprehend. However, it has been able to solve tasks that sigmoid networks were not able to solve, [FDM08, FM01].

Another relatively new type of neural network is the *Echo State Network*, echo state networks use a sigmoidal transfer function and are always recurrent networks, [Jae01, DBS08]. The hidden nodes are fully connected to the input and output nodes. The main idea of echo state networks is that only the weights directly to the output nodes need to be learned, the other nodes are randomly chosen and stay fixed, Figure 3.3 shows this model of an echo state network.. These types of networks have mostly been succesful at supervised learning tasks but have been combined with neuroevolution and

unsupervised learning settings recently, [Jae01, DBS08]. To date, echo state networks have been combined succesfully with the CMA-evolution strategies technique. These studies also conclude that evolving more than just the outgoing weights, like topology or other weights, will probably have a positive effect on the results.

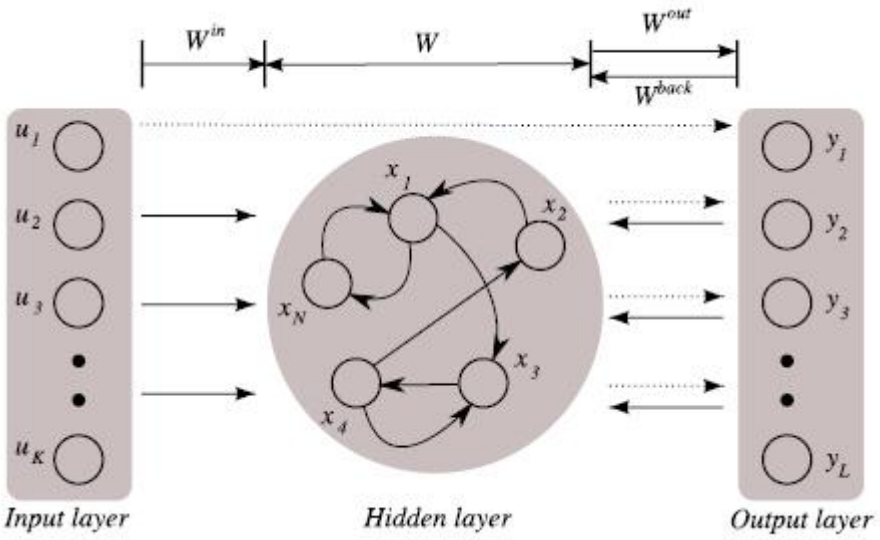

Figure 3.3: The model of an Echo State Network, the weights of the plain connections are randomly chosen and stay fixed, the dashed connections need to be evolved. *Source: Unsupervised learning of Echo State Networks: A Case Study in Artificial Embryogeny, [DBS08].*

Chapter 4

Applications

This chapter will discuss for what kind of problems neuroevolution is particularly suitable. In general, neuroevolution is more suitable for unsupervised learning problemes. Unsupervised learning can be described as the area of machine learning where a training set of of input-output pairs does not exist, [WGP95]. Therefore another performance measure is needed to evaluate an algorithm's performance, this works well together with evolutionary algorithms, which need a fitness measure as well. Many articles present neuroevolution techniques and test them on a couple of benchmark problems to test the algorithms performance against its rivals. While this provides interesting information, certainly from a scientist's point of view, it is also interesting to see on what kind of real-world problems neuroevolution shows promise. This chapter will discuss both the popular benchmark problems and some of the other problems to which neuroevolution has been succesfully applied.

4.1 Benchmark problems

In 1991, Wieland [Wie91], started the trend to use standard control theory problems as benchmark problems for neuroevolution algorithms. He used a genetic algorithm to find the solution for a pole balancing problem. Which consisted of a cart with a pole on top of it that could tilt both left and right. The card could move left or right on a finite track to try to keep the pole balanced. Input variables to this problem are the angle of the pole, the angular velocity of the pole, the velocity of the cart and the position of the cart. Output would be the force that needs to be exerted on the cart. This original problem is not very complicated and can be solved by a neural network where the weights are guessed at random, [GSM06]. For this reason researchers created similar benchmark tests that are harder to

solve, such as balancing two poles on one card that have a different length and balancing poles without knowing all the information available. On these tasks neuroevolution outperforms other reinforcement learning techniques and for the more difficult pole balancing tasks the more effective algorithms to date are NEAT, AGE, CMA-ES and CoSyNE. it is interesting to note that most of these algorithms are direct representations and that half of them use a fixed size and a fixed topology.

4.2 Other applications

While neuroevolution techniques are mostly tested with the help of benchmark problems, there are multiple examples of other applications where neuroevolution performs well. a promising field is perhaps the evolution of agents in computer games. Schrum and Miikkulainen, [SM08], evolved opponents in computergames that were able to distinguish between objectives of different importance in a simple two dimensional game. In [AbBR01], Aharonov-Barki et. al. evolved a controller for an agent that used memory to search for food, and Bryant and Miikkulainen, in 2003, [BM03], set up an experiment where a team of identical agents could work together and perform different tasks to achieve a team objective. While these experiments involve games that are much simpler than the average game that is played nowadays, the need for intelligent behavior of *non-player characters* in computer games is increasing and reinforcement learning techniques like neuroevolution could provide this behavior. Bryant and Miikkulainen performed another interesting experiment in 2007, [BM07] , where the goal was to develop visibly intelligent behavior in NPC's. They argued that good behavior for an NPC does not necessarily need to be optimal behavior, but behavior that mimicks human behavior. They conducted an experiment and concluded that a form of neuroevolution performed best when non-player characters had to mimick human behavior.

Neuroevolution has also been applied to evolve systems that can play specific games. [CF99] used neuroevolution to evolve a controller that could play checkers and that could beat a master player on several occasions. Kaikhah and Garlick, [KG00], used their version of the ESP algorithm to evolve a blackjack player that used card counting strategies to gain an advance of the house. Neurevolution has also been used as a succesful controller in vehicle simulations for a car, [CLL10], and a helicopter, [KW09].

Neuroevolution can also be used for classification tasks, classification is a supervised learning task because the classes are known in advance. Evolutionary algorithms are not specifically suited for these tasks. Johan Hägg,

[HÖ8], compared the CMA-evolution strategies method against the backpropagation algorithm for a gesture recognition task. He concluded that backpropagation was more succesful at the task. However, there have been occasions were neuroevolution did give the best results on classification tasks, Montana and Davis performed an experiment, [MD89], where backpropagation and neuroevolution was used to classify sonar data. In their experiment the genetic algorithm outperformed the backpropagation algorithm on the classification task, they also stated that algorithms that use both techniques were perhaps the most promising. A very different application of neuroevolution was in the field of physics, Aaltonen et. al. [CA09] used neuroevolution to calculate the mass of a quark particle.

Neuroevolution is most succesful in reinforcement learning problems where controllers need to be evolved. especially when considering controlling an object that needs to stay balanced, a helicopter or pole, or controlling non-player characters in computer games or board games. Neuroevolution has been succesful in classification tasks at some occasions, but is not clear what the true potential is for neuroevolution algorithms in this area.

Chapter 5

Conclusions

Neuroevolution representations can be divided amongst three categories:

- Direct representations.
- Developmental representations.
- Indirect or implicit representations.

Direct representations can again be divided into three types, weight vector representations, neuron representations and topology representations. The advantage of a direct representation is that it is understandable and does not need an external set of rules to decode it to a phenotype. The major drawback of this kind of representation is that bigger networks get gradually bigger genotype representations. The NeuroEvolution of Augmenting Topologies (NEAT) algorithm is the most widely used algorithm at this moment, although both CMA-ES and CoSyNE have shown better results at certain benchmark tests. The popularity of NEAT could be ascribed to the fact that it is more robust, because the amount of hidden nodes is not chosen in advance.

Developmental representations are relatively older compared to their rivals and are not frequently used, they were developed to exploit modularity within larger neural networks. But exploiting this characteristic only helped in certain applications, where other measures had to be taken to deal with other problems. Indirect representations are relatively new and are based on Gene Regulatory Networks. The most succesful implicit representation, called Analog Genetic Encoding, reported performances similar to those of the best direct representations.

Recently, a couple of new neural network techniques have been used in combination with neuroevolution. Both spiking neural networks and echo

state networks show promise. Spiking neural networks trained with the NEAT algorithm performed tasks of which normal artificial neural networks were not capable.

Neuroevolution is most suitable in unsupervised learning situations, especially in situations where some sort of artificial controller needs to be evolved. Neuroevolution has been used for supervised learning tasks as well, but does not perform significantly better than other algorithms. But for reinforcement learning problems neuroevolution is top of the line. In the future it might be the key for computers to behave similar to the human brain.

Bibliography

[Abb03] H A Abbass. Pareto neuro-evolution: constructing ensemble of neural networks using multi-objective optimization. *The 2003 Congress on Evolutionary Computation 2003 CEC 03*, 3(4):2074–2080, 2003.

[AbBR01] Ranit Aharonov-barki, Tuvik Beker, and Eytan Ruppin. Emergence of memory-driven command neurons in evolved artificial agents. *Neural Computation*, 13:691–716, 2001.

[ASP94] Peter J. Angeline, Gregory M. Saunders, and Jordan B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5:54–65, 1994.

[ATM05] A. Agogino, K. Tumer, and R. Miikulainen. Efficient credit assignment through evaluation function decomposition. In *The Genetic and Evolutionary Computation Conference*, Washington, DC, June 2005.

[Bis95] C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[BLLS11] R. Batllori, C.B. Laramee, W. Land, and J.D. Schaffer. Evolving spiking neural networks for robot control. *Procedia computer science*, 6:329–334, 2011.

[BM03] Bobby D. Bryant and Risto Miikkulainen. Neuroevolution for adaptive teams. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, pages 2194–2201, Piscataway, NJ, 2003. IEEE.

[BM07] Bobby D. Bryant and Risto Miikkulainen. Acquiring visibly intelligent behavior with example-guided neuroevolution. In

Proceedings of the Twenty-Second National Conference on Artificial Intelligence, Menlo Park, CA, 2007. AAAI Press.

[BMS90] Richard K. Belew, John Mcinerney, and Nicol N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In *In*, pages 511–547. Addison-Wesley, 1990.

[Bon02] J. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02*, CEC '02, pages 1872–1877, Washington, DC, USA, 2002. IEEE Computer Society.

[CA09] CDF Collaboration and T. Aaltonen. Measurement of the top quark mass with dilepton events selected using neuroevolution at cdf. *PHYS.REV.LETT.*, 102:152001, 2009.

[CF99] K Chellapilla and D B Fogel. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE*, 87(9):1471–1496, 1999.

[CF01] Kumar Chellapila and David B. Fogel. Evolving an expert checkers playing program without using human expertise. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 5:422–428, 2001.

[CLL10] L. Cardamone, D. Loiacono, and P. L. Lanzi. Learning to drive in the open racing car simulator using online neuroevolution. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(3):176 –190, sep. 2010.

[cZFLM02] Jean christophe Zufferey, Dario Floreano, Matthijs Van Leeuwen, and Tancredi Merenda. Evolving vision-based flying robots. In *In Bulthoff, Lee, Poggio, Wallraven (Eds.), Proceedings of the 2nd International Workshop on Biologically Motivated Computer Vision (BMCV*, pages 592–600. Springer-Verlag, 2002.

[Dar] Charles Darwin. *On the origin of species*. New York :D. Appleton and Co.,. http://www.biodiversitylibrary.org/bibliography/28875.

[DBS08] Alexandre Devert, Nicolas Bredeche, and Marc Schoenauer. Unsupervised learning of echo state networks: a case study in artificial embryogeny. In *Proceedings of the Evolution artificielle, 8th international conference on Artificial evolution*, EA'07, pages 278–290, Berlin, Heidelberg, 2008. Springer-Verlag.

[DM92] Dipankar Dasgupta and Douglas R. Mcgregor. Designing application-specific neural networks using the structured genetic algorithm. In *In Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96. IEEE Computer Society Press, 1992.

[DM00] Nirav S. Desai and Risto Miikkulainen. Neuro-evolution and natural deduction. In *Proceedings of The First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pages 64–69, Piscataway, NJ, 2000. IEEE.

[DMF06] Peter Durr, Claudio Mattiussi, and Dario Floreano. Neuroevolution with analog genetic encoding. In *PPSN*, volume 4193 of *Lecture Notes in Computer Science*, pages 671–680. Springer, 2006.

[FBoTDoES98] T.S. Frank, J.W. Burdick, California Institute of Technology. Division of Engineering, and Applied Science. *Computing with spiking neurons*. CIT theses. California Institute of Technology, 1998.

[FDM08] Dario Floreano, Peter Durr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[FLM03] James Fan, Raymond Lau, and Risto Miikkulainen. Utilizing domain knowledge in neuroevolution. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 170–177. AAAI Press, 2003.

[FM01] Dario Floreano and Claudio Mattiussi. Evolution of spiking neural controllers for autonomous vision-based robots. In *in: T. Gomi (Ed.), Evolutionary Robotics IV*. Springer-Verlag, 2001.

[GHLM02] Brian Greer, Henri Hakonen, Risto Lahdelma, and Risto Miikkulainen. Numerical optimization with neuroevolution, 2002. Undergraduate Thesis, Department of Computer Sciences, The University of Texas at Austin.

[GM99] Faustino J. Gomez and Risto Miikkulainen. Solving non-markovian control tasks with neuroevolution. Dissertation Proposal, Computer Science Department, University of Texas at Austin, 1999.

[Gru95] Frederic Gruau. Automatic definition of modular neural networks. *Adaptive Behaviour*, 3(2):151–183, 1995.

[GSM06] Faustino Gomez, Juergen Schmidhuber, and Risto Miikkulainen. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning*, pages 654–662, Berlin, 2006. Springer.

[HÖ8] Johan Hägg. Gesture recognition using neuroevolution. In *Proceedings of IDT Workshop on Interesting Results in Computer Science and Engineering*, pages 18–26, 2008.

[HBL10] Gerard David Howard, Larry Bull, and Pier Luca Lanzi. A spiking neural representation for xcsf. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9:159–195, 2001.

[Ige03] Christian Igel. Neuroevolution for reinforcement learning using evolution strategies. In R. Reynolds, H. Abbass, K. C. Tan, B. Mckay, D. Essam, and T. Gedeon, editors, *Congress on Evolutionary Computation (CEC 2003)*, volume 4, pages 2588–2595. IEEE, 2003.

[Jae01] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks with an erratum note 1. *Technical report*, pages 1–47, 2001.

[JBS08] Fei Jiang, Hugues Berry, and Marc Schoenauer. Supervised and evolutionary learning of echo state networks. In *Proceedings of the 10th international conference on Parallel Problem*

Solving from Nature: PPSN X, pages 215–224, Berlin, Heidelberg, 2008. Springer-Verlag.

[KG00] Khosrow Kaikhah and Ryan Garlick. Variable hidden layer sizing in elman recurrent neuro-evolution. *Applied Intelligence*, 12:193–205, May 2000.

[KNG98] Michael Korkin, Norberto Eiji Nawa, and Hugo De Garis. A "spike interval information coding" representation for atr's cam-brain machine (cbm). In *Proceedings of the Second International Conference on Evolvable Systems (ICES'98), volume 1478 of Lecture Notes in Computer Science*, pages 256–267. Springer-Verlag, 1998.

[KW09] Rogier Koppejan and Shimon Whiteson. Neuroevolutionary reinforcement learning for generalized helicopter control. In *GECCO 2009: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 145–152, July 2009.

[Mat05] C. Mattiussi. *Evolutionary synthesis of analog networks*. s.n., 2005.

[MD89] David J Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. *Machine Learning*, 123:762–767, 1989.

[MD09] Jean-Baptiste Mouret and Stéphane Doncieux. Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 627–634, New York, NY, USA, 2009. ACM.

[MM96] David E. Moriarty and Risto Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, (AI94-224):11–32, 1996.

[OCBM+11] M. O'halloran, S. Cawley, R. C. Conceicao B. McGinley, F. Morgan, E. Jones, and M. Glavin. Evolving spiking neural network topologies for breast cancer classification in a dielectrically heterogeneous breast. *Progress in Electromagnetics research letters"*, 25:153–162, 2011.

[RM07] Joseph Reisinger and Risto Miikkulainen. Acquiring evolvability through adaptive representations. In *In Proc. of*

Genetic and Evolutionary Computation Conference, pages 1045–1052, 2007.

[SKR04] Keren Saggie, Alon Keinan, and Eytan Ruppin. Spikes that count: rethinking spikiness in neurally embedded systems. *Neurocomputing*, 58-60:303–311, 2004.

[SM02a] Kenneth O. Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, page 9, San Francisco, 2002. Morgan Kaufmann.

[SM02b] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[SM08] Jacob Schrum and Risto Miikkulainen. Constructing complex npc behavior via multi-objective neuroevolution. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2008)*, pages 108–113, Stanford, California, 2008.

[SM09] Jacob Schrum and Risto Miikkulainen. Evolving multimodal behavior in npcs. In *IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, pages 325–332, Milan, Italy, September 2009. (Best Student Paper Award).

[Sta04] Kenneth O. Stanley. *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2004.

[WGP95] Darrell Whitley, Frederic Gruau, and Larry Pyeatt. Cellular encoding applied to neurocontrol. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 460–467, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[Wie91] Alexis Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks* (Seattle, WA), pages 667–673. Piscataway, NJ: IEEE, 1991.

List of Figures