Evolving autonomous agents with simulated brains using L2L and Netlogo

A thesis submitted for the degree of
Bachelor of Science

Jessica Yu

Submission date: 2021-09-20

Supervisor:	Prof. Dr. Abigail Morrison
Advisors:	Dr. Cristian Jimenez-Romero
	Alper Yegenoglu
Examiners:	Prof. Dr. Abigail Morrison
	Prof. Dr. Peter Rossmanith

Department of Computation in Neural Circuits
RWTH Aachen University
Germany

Contents

1 Introduction **4**
1.1 Motivation 5
1.2 Overview 6
1.3 Related work 7
1.4 Outline 9

2 Genetic Algorithms **10**
2.1 Evolution-inspired algorithms 10
2.2 A genetic algorithm cycle 10
2.3 Hyperparameters 12

3 Model **13**
3.1 Virtual insect environment 13
3.2 Spiking neural network 14
3.2.1 Architecture 14
3.2.2 Neural dynamics 16
3.3 Fitness function 17

4 Optimisation Tools **18**
4.1 BehaviorSearch 18
4.2 Learning-to-learn (L2L) 20
4.3 Comparison 22

5 Search Configurations **24**
5.1 Hyperparameter prefiltering 24
5.2 Main investigation 25
5.3 Static and dynamic world 25
5.4 Varying network topology 26

6 Results and Evaluation **27**
6.1 Hyperparameter prefiltering 27
6.2 Main investigation 28
6.2.1 BehaviorSearch 28

6.2.2 L2L 34
6.2.3 Comparison 36
6.3 Static and dynamic world 38
6.4 Varying network topology 39

7 Network Analysis **40**
7.1 2 hidden-layer neurons 40
7.2 6 hidden-layer neurons 42
7.3 1 hidden-layer neuron 44
7.4 0 hidden-layer neurons 46

8 Multi-Ant Model **47**

9 Conclusion **49**
9.1 Summary 49
9.2 Outlook 50

References **51**

Appendices **58**

1 Introduction

Artificial neural networks (ANNs) are machine learning algorithms based on the architecture and computations of the human nervous system [1]. As ANNs have become more widespread, their computational power has also continued to evolve. Spiking neural networks (SNNs) represent the third generation of neural networks [2] and, in addition to previous generations, offer the ability to reproduce spatio-temporal dynamics [3]. By using spiking neurons as main computational units, SNNs are considered to be more biologically realistic, since information is transmitted through action potentials or spikes. This makes them particularly attractive for modelling biological systems, including the simulation of biological organisation.

Spiking neurons can be modelled using various abstractions [4]. The Leaky Integrate-and-Fire (LI&F) model [5] is widely used among neuroscientists to analyse the behaviour of SNNs [6, 7, 8]. It reproduces the basic properties of a biological nerve cell using the following mechanism: when input currents cause an increase in the membrane potential until a certain threshold is reached (otherwise it decays towards the resting potential), a spike is initiated (firing event). Immediately afterwards, the neuron enters an absolute refractory state in which all incoming pulses are neglected. Once the refractory period has elapsed, the neuron is open to incoming pulses again. An illustration of this process is shown in Figure 1.

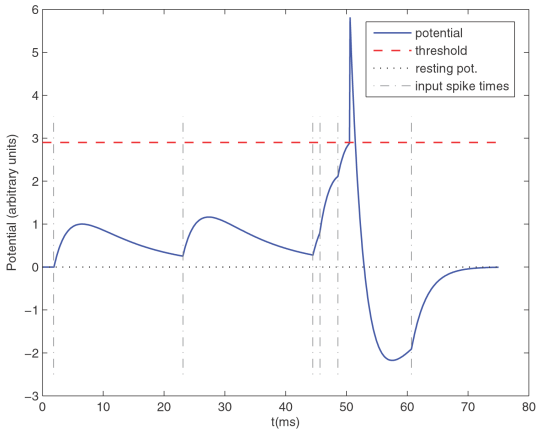


Figure 1: Membrane potential of a Leaky Integrate-and-Fire neuron [9]

As with many neural models, the difficulty in achieving the desired performance is finding the appropriate parameter settings, e.g. synaptic characteristics such as weight and learning rate, etc. Typically, these models contain a large set of parameters, each with its own data type and admissible range. This results in a huge search space, which makes the identification of value assignments that lead to acceptable solutions a very time-consuming task. In order to locate adequate settings more efficiently, many autonomous approaches use heuristic search methods for parameter optimisation. Among them, genetic algorithms (GAs) provide evolution-based search techniques inspired by the processes of natural adaptation [10]. As with the model to be optimised, the challenge is to choose the appropriate hyperparameter settings. If another autonomous approach were to be applied at this point, it would ultimately lead to an endless chain of parameter optimisations. For this reason, the main purpose of this study is to (manually) investigate the settings of GA hyperparameters that lead to satisfactory performance of the considered model.

1.1 Motivation

Computational neuroscience is concerned with understanding the principles and procedures of neural systems through the study of biologically realistic mathematical models [11]. In nature, organisation is found at different levels and complexity, from molecular structures to populations of organisms. Among these, insect colonies exhibit a complex adaptive self-organising pattern [12]. Division of labour, coordination of activities, temporally synchronised behaviour and the ability to distribute information processing make these creatures particularly interesting for computational neuroscientists. SNNs are not only biologically more realistic computational systems, but most importantly offer the capability to reproduce the occurrence of temporal coupling in insect societies [13]. This makes them particularly suitable for simulating and studying those creatures.

As the complexity of behavioural traits in insect colonies increases, so does the complexity of neural models used for simulating their behaviour.

Consequently, it becomes not only more difficult to achieve a realistic simulation, but also to understand and explain the observed behaviour based on the model. For this reason, reducing the complexity and limiting it to the most basic behavioural properties provides the opportunity to study the model in more detail.

In order to obtain a simulation that reproduce biologically realistic behaviour, the use of parameter optimisation methods is often required. Genetic algorithms present meta-heuristic search techniques, which use a predefined fitness function for the evaluation of solution candidates. As evolution-inspired algorithms, GAs provide many benefits [14], some of which are listed below:

- consider multiple solutions in each cycle (population-based search)
- easily parallelized (evaluation distributed over several processors)
- maintain diversity in population (stochastic operators e.g. mutation)
- ability to escape from local optima
- support multi-objective optimisation
- compatible with mixed type encodings
- easy to implement, various variants

Genetic algorithms generally contain a set of hyperparameters that influence the progress of the search. In particular, these parameters affect the convergence speed and the best fitness achieved during the search process. The latter directly corresponds to how well a solution solves the considered problem. Taking into account the limited available resources, there is the necessity to identify hyperparameters that lead to the best possible performance of the applied GA, which forms the main motivation for this work.

1.2 Overview

This section provides an introduction to the topic and a general overview of the content covered in this thesis. A multi-agent simulation model embedded in NetLogo [15] is examined. It simulates an insect, more specifically an

ant, moving through a virtual world with many obstacles in search of food. During this process, the ant is controlled by a spiking neural network whose parameters (weights and delays) are optimised using genetic algorithms. For this purpose, two optimisation tools (L2L [16] and BehaviorSearch [17]) are applied and compared with each other.

As part of this investigation, a series of optimisation searches were performed. A more detailed description of the search configurations can be found in Chapter 5. First, several short search runs were conducted to become familiar with the parameters of genetic algorithms and to limit the settings to promising ones. Subsequently, these parameter values were examined in more detail through longer searches carried out on a high performance computer (HPC). Chapter 6 is primarily concerned with evaluating the data obtained through the search runs in terms of achieved fitness and parameters of the applied GAs to determine which settings deliver the best results. Afterwards, an analysis of the optimised networks is presented. In particular, SNNs with variable topologies are considered to understand and explain the behaviour of the insect during the simulation. In the end, a modified version of the considered model is examined, which simulates multiple ants that try to avoid collisions between each other, while maintaining their behavioural characteristics. It is particularly interesting here to observe whether the ants can also cope in a world with other moving agents.

1.3 Related work

Even though the application of spiking neural networks for controlling autonomous robots [18, 19, 20] has proven to be very successful, there is not much research done regarding the simulation of insect colonies using SNNs. Chevallier et al. proposed a model called SpikeAnts [21], in which an ant colony is controlled by a sparsely connected SNN. Each ant makes its own decision of action (foraging, sleeping, self-grooming) through two different spiking neurons, a Leaky and a Quadratic Integrate-and-Fire neuron. Another study is presented by Ahmadi et al. [22] in which a framework that models an artificial creature and a colony of creatures is considered. The

creatures are controlled by an SNN and placed in a virtual environment with randomly positioned food. Main differences compared to the model examined in this study are found in the network architecture (a single creature is modelled with 108 neurons), varying strength of visual stimulation (depending on the distance of the food) and the addition of an energy level that decreases with each action and increases when food is found. The aim of this research was to achieve a high survival chance (energy level over zero) of the creatures through a genetic algorithm optimisation.

Several studies use spike timing dependent plasticity (STDP) [23, 24, 25] and reinforcement learning [26, 27] to train spiking neural networks. While these methods reflect synaptic plasticity, there are several other techniques to obtain a functional network. Meta-heuristic search is a common method to optimise large sets of parameters in an autonomous way. Genetic algorithms applied to spiking neural networks have gained popularity as this combination has proven successful in practice [28, 29, 30]. Although there have been many applications of genetic algorithms in various fields like economics [31, 32], medicine [33, 34, 35] and social science [36, 37] etc., not many work can be found that focuses on the optimisation of hyperparameters. In fact, most GA optimisation approaches only use fixed hyperparameter settings. Usually, they are either manually chosen [38, 39, 40, 41] or based on literature suggestions [42, 43]. Only rarely is an examination of the hyperparameters carried out in order to improve the results of the genetic algorithm and therefore the model to be optimised. Some research has been done on the general study of genetic algorithm parameters. In the work of Hassanat et al. [44], the crossover and mutation probabilities were investigated by dynamically increasing and decreasing the ratios. Experimental results show that one of the most commonly chosen setting did not perform well in any of the trials. Another study was conducted by Sipper et al. [45] in which 25 different problems were investigated thoroughly regarding parameter assignments of GAs. Likewise, this research concluded with the fact that no common settings could be extracted that would deliver general solid performance. This emphasises the importance of case-specific optimisation of GA hyperparameters.

1.4 Outline

- **Chapter 2** provides an introduction to genetic algorithms. It gives a brief summary of evolutionary concepts, followed by a more detailed explanation of the algorithmic structure and settings of GAs.
- **Chapter 3** describes the simulation model embedded in NetLogo. This includes the construction of the insect environment as well as the architecture and neural dynamics of the implemented spiking neural network.
- **Chapter 4** gives an overview of the utilised optimisation tools L2L and BehaviorSearch. In particular, the respective realisation of the genetic algorithm as preparation for subsequent search analysis is explained and compared.
- **Chapter 5** reports which search experiments were performed with a description of the parameter and search settings, and clarifies some underlying thoughts during the procedure.
- **Chapter 6** evaluates the search results. The parameter settings of the two algorithms are investigated and compared with respect to the best fitness achieved and the development over the generations.
- **Chapter 7** deals with the optimised networks in more detail. It provides an analysis of the internal dynamics and computations regarding the agent's behaviour.
- **Chapter 8** presents a modified version of the model which simulates multiple insects. The performance and achieved fitness as well as the architecture of the network are examined and discussed.
- **Chapter 9** provides a summary of the work and suggestions for future research.

2 Genetic Algorithms

Genetic algorithms (GAs) are meta-heuristic search methods that belong to the class of evolutionary algorithms. As an abstraction of evolutionary processes based on Charles Darwin's original theory on natural selection [46], GAs apply evolutionary operators (selection, recombination, mutation) to solve optimisation problems.

2.1 Evolution-inspired algorithms

Biological evolution describes the development of organisms at the level of populations through the mechanism "survival of the fittest". As stated by Sivanandam and Deepa [47] as well as Kramer [48], the evolutionary process takes place encoded in the genetic information of living beings. In particular, it involves operations on chromosomes and how this information decoded affects the survival chance of its carrier.

In nature, selection defines the mechanism that organisms with a better adaptability to their environment have a better chance of survival and therefore reproduce more frequently. As a result, chromosomes with advantageous expressions are passed on to subsequent generations more often than disadvantageous ones. Evolution mainly takes place during the reproductive phase of living beings. Several mechanisms contribute to this process. Among them, recombination and mutation are the most common. The former involves the combination of parental chromosomes to produce the genetic information of the next generation, i.e. the children or offspring. Mutation, on the other hand, describes the random alteration of chromosome segments resulting in new encoded information within a population. Together, these three operations - selection, recombination and mutation - form the basis of the class of evolutionary algorithms.

2.2 A genetic algorithm cycle

Among evolutionary algorithms, GAs form the most widely applied technique in practice. The concept of genetic algorithms was first introduced by Holland

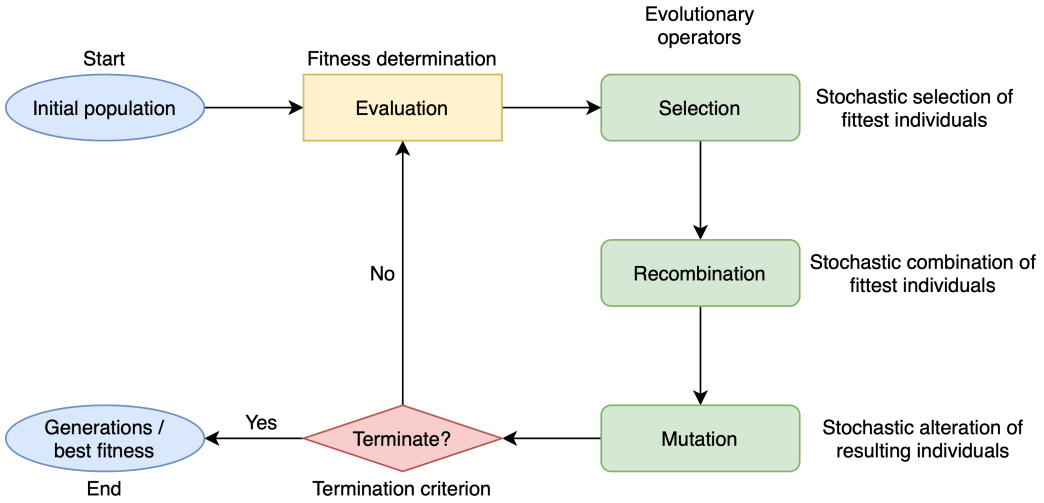


Figure 2: Typical cycle of genetic algorithms

[49] in 1975 and further elaborated by Goldberg [50].

A genetic algorithm is an iterative and stochastic search method that operates on a population of individuals. Each individual represents a candidate solution to the given optimisation problem. Since each solution is encoded in an individual, it is often simply referred to as a chromosome. Figure 2 shows the cyclic process of genetic algorithms along with the applied evolutionary operators. The algorithm starts with a randomly initialised population of individuals. Each of them is evaluated in the first step of the iterative process based on a predefined fitness function. This function determines how well an individual solves the given problem. Therefore, the construction of a suitable fitness function is often the most challenging task when implementing GAs [51, 52]. After the evaluation, the evolutionary operators are applied to the population. Depending on the implementation, each operator can be realised in different ways. Selection involves stochastically choosing the fittest individuals to pass on to the next population. Common variants [53] include roulette selection, tournament selection and rank selection. Afterwards, the best chromosomes are combined or cloned to produce offspring which are added to the subsequent population. This process is referred to as

crossover, which can be realised through single-point crossover, multi-point crossover, uniform crossover and blend crossover etc. [54, 55, 56] Mutation is then performed, often with a certain probability, on the entire population of resulting individuals. Depending on how the solutions are encoded in the chromosomes, different options [57] such as bit-flip mutation or uniform mutation can be used. After applying the evolutionary operators, the resulting population forms the new generation on which the GA operates in the next cycle. Before the algorithm enters the next iteration, the termination criterion is checked. Usually, the number of produced generations or a certain fitness score to be achieved is chosen for this purpose. Results of interest from genetic algorithms are primarily the best fitness achieved and the generated solutions over the course of generations.

2.3 Hyperparameters

In nature, the progression of evolution varies between organisms. Some species evolve very slowly, while others go through a much faster evolutionary process. On the one hand, the difference in reproduction rate contributes to this observation. On the other hand, evolution is a strongly stochastic process, i.e. how and when selection, recombination and mutation take place is completely random.

Genetic algorithms model the stochasticity through a set of control parameters. Generally, this involves setting the number of individuals contained in each population and the probabilities with which each operation is executed. Depending on the implementation, the hyperparameters may vary between GAs. In this study, the genetic algorithms of two different optimisation tools are applied to a parameter optimisation problem, which is explained in the next chapter. The algorithms are compared in terms of performance, i.e. achieved fitness and speed of convergence with respect to the hyperparameter settings. A more detailed description of the hyperparameters and implemented genetic algorithms can be found in Chapter 4.

3 Model

A modified version of the model proposed by Jimenez-Romero et al. [58] is examined in this study. It simulates an artificial ant navigating through a virtual world full of food and obstacles (c.f. Section 3.1). In this model, the ant receives external visual stimuli from objects placed at a fixed position in the world, as well as reward or punishment sensation when it comes into contact with these objects. The insect is controlled by a spiking neural network, whose synaptic parameters are optimised using genetic algorithms (see Chapter 4). The model including the SNN is implemented in NetLogo [15]. NetLogo is a programming environment specialised in the simulation of multi-agent models. It offers the possibility to explore the behaviour of agents through instructive commands during the simulation.

In the following sections, the virtual world in which the insect is located is introduced, followed by a detailed explanation of the implemented spiking neural network.

3.1 Virtual insect environment

NetLogo offers several types of agents, including turtle, patch and link agents, which can be used for the simulation. In this model, the insect agent as well as the spiking neurons are modelled as turtle agents which are able to move around in the world. Each patch represents a fixed position in the NetLogo virtual environment. Patches can be programmed with own characteristics and are able to interact with other agents. Link agents are used to model the synaptic connections between the neurons in the SNN.

A detail of the NetLogo interface visualising the insect environment is shown in Figure 3. In the beginning of each simulation, the yellow insect is located in the middle of the virtual world. White patches form a boundary around the world and additional vertical walls are drawn throughout the area creating a maze like environment. In addition, about 200 of each red and green patches are distributed all over the world. Red patches represent harming objects, while green patches symbolise food or rewards.

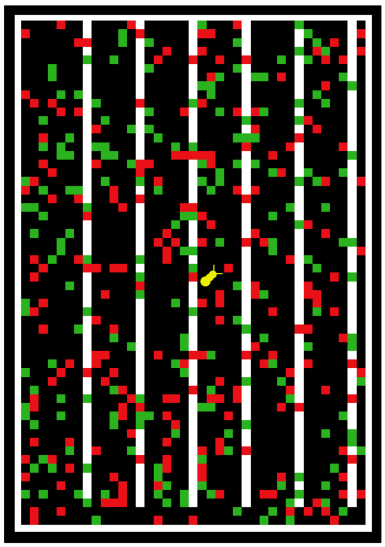


Figure 3: A yellow insect is located in a virtual maze simulated in NetLogo

In the ideal case, the ant should behave in such a way that it moves around in the maze while collecting as many food as possible without colliding with obstacles.

3.2 Spiking neural network

The spiking neural network implemented in this simulation is based on simplified version of a Leaky Integrate-and-Fire (I&F) model [5]. In the following sections, the architecture of the SNN is explained followed by a summary of the internal neural dynamics. A more detailed description of the utilised SpikingLab framework can be found in Jimenez-Romero et al. [59].

3.2.1 Architecture

A screenshot of the modelled SNN captured from the NetLogo interface is shown in Figure 4. The network consists of three layers, with the first layer responsible for handling external stimuli through sensors and afferent neurons, which propagate the input pulses to the next layer. The artificial insect is able to process three types of sensory information. Visual perception is realised via three photoreceptors, each of which is sensitive to a specific colour

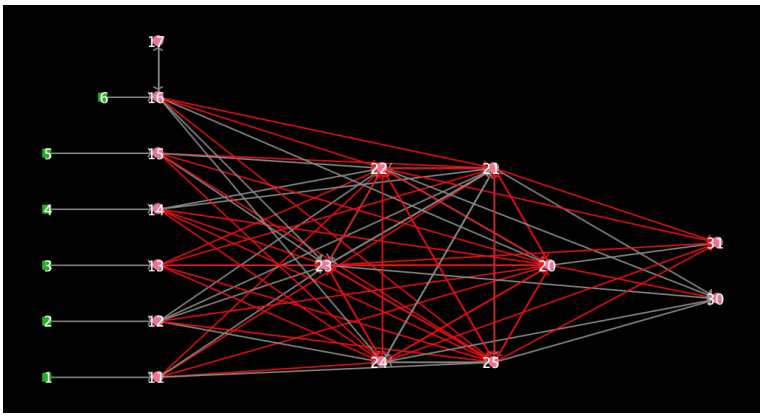


Figure 4: Spiking neural network simulated in NetLogo

(white, red and green). In addition, the network has two receptors which allows the insect to perceive pain and rewarding sensation. To enable the ant to continue moving even in the absence of external stimulation, an artificial heartbeat is modelled by two interconnected neurons (labelled 16 and 17). This heartbeat is started at the beginning of each simulation through a pulse initiated by an external input current (i.e. voltage clamp). Neurons in the middle layer carry out the main computations. By default, six interconnected neurons form this layer. Experiments with fewer and more neurons are examined and discussed in more detail in Chapter 7. The decision about which action the ant will perform at each step is made by two motoneurons or actuators. The insect is able to execute two types of actions, rotation and forward movement. If the spiking neuron labelled 30 is firing, the ant rotates 4 degrees to the right. On the other hand, if the motoneuron with label 31 fires, the insect moves 0.4 steps forward.

Since each neuron in this network is connected to every neuron in the following layer along with the interconnected neurons in the middle, the number of connections adds up to 78 synapses for six second-layer neurons. Each synapse has a weight and a delay value, resulting in a total of 156 parameters to be optimised. For the weight parameters a range of -20 to 20 is chosen and for the delays every integer value between 1 and 7 is a valid setting.

3.2.2 Neural dynamics

The virtual insect can perceive its environment up to a certain distance (see Table 5). When a photoreceptor senses the corresponding colour in front of the ant, it starts sending pulses to the afferent input neuron. An example is shown in Figure 5. In graph a), the membrane potential of the neuron sensitive to the colour green is illustrated. In this moment, the ant has its focus on a green patch, and the colour sensor causes the connected neuron to fire. As it approaches the food, the neuron continues to fire until the ant reaches and collects the food. At this point, the green neuron stops firing and diagram b) shows that the reward receptor has sensed the food and triggers its afferent neuron to fire once.

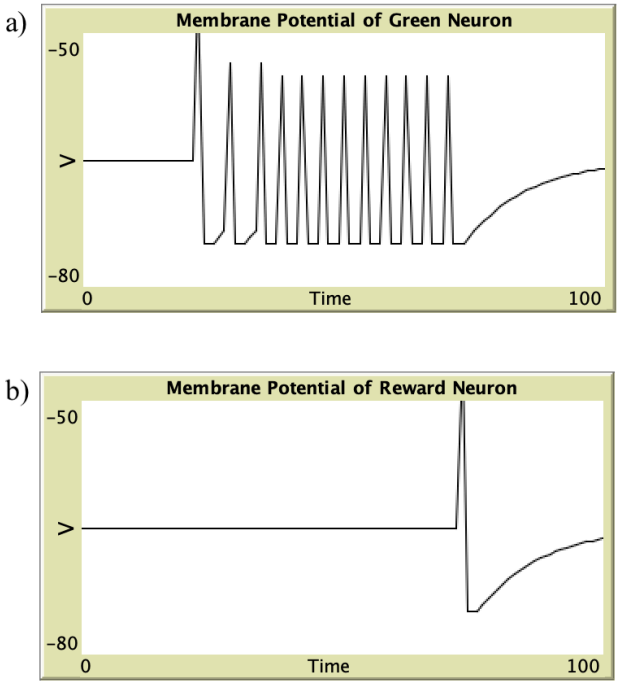


Figure 5: Membrane potential of selected input neurons

a) shows the membrane potential of the neuron processing green colour stimulation when the corresponding patch is sensed.
b) shows the membrane potential of the neuron processing rewarding sensation when the ant collects food.

As mentioned above, the SNN is realised according to the basic principles of an LI&F model. Each spiking neuron receives signals exclusively from presynaptic neurons, i.e. no external stimuli influence the membrane potential. A modelled spiking neuron has the following characteristics: membrane potential, resting potential, firing threshold, decay rate, refractory potential and duration as well as spikes sent per stimulus. The exact settings can be found in Table 5. During the simulation and the optimisation, these settings remain unchanged. In [59] a state-transition machine of the simulated spiking neurons and an illustrative presentation of the membrane potential can be found.

3.3 Fitness function

As stated in Section 2.2, the definition of a suitable fitness or objective function is crucial to guide the genetic algorithm into the right direction. Since each candidate solution is evaluated based on the fitness value, this function must encode the desired behaviour of the ant in the simulation. The difficulty is to find a good balance of included constraints without constructing an overly complex fitness function. In this model, the ant should move through the world collecting food while avoiding the obstacles. The formula for calculating the fitness of an individual I is shown in (1). For each reward found by the ant, the fitness is increased by 1. If the ant hits an obstacle, i.e. a wall or a harmful object, the fitness is reduced by 1.5.

$$fitness(I) = 1 \cdot n\_found\_food - 1.5 \cdot n\_collisions \tag{1}$$

Hence, the goal is to maximize the fitness of the insect. Here, the penalty for a collision is set slightly higher than the reward for collecting food. It is worth mentioning that these values were determined through experiments and tests in the simulation. If the penalty is set too high, the ant would stop moving as soon as it perceives an obstacle. On the other hand, the ant should prefer avoiding a collision to collecting food.

4 Optimisation Tools

In this study, the genetic algorithms of two optimisation tools are applied and compared. In particular, the hyperparameters are examined with regard to the achieved fitness and speed of convergence. In order to analyse and compare the software tools more adequately, the following sections introduce the parameters and describe the implemented algorithms in more detail.

4.1 BehaviorSearch

BehaviorSearch [17] is a software tool specialised in the automated optimisation of agent-based models. It interfaces with NetLogo models and offers several meta-heuristic search techniques including genetic algorithms. Five parameters of the GA can be set, which are listed below:

- **population-size** defines how many individuals form a population. Each generation consists of exactly that many individuals.
- **tournament-size** determines the size of the group of individuals considered during a selection iteration. In each round, the fittest individual from this group is chosen to be included in the next population.
- **crossover-rate** specifies the number of parental chromosomes selected during the recombination phase and thus how many children are included in the next population.
- **mutation-rate** indicates the probability with which each parameter value of an individual is changed.
- **population-model** determines how the next population is composed. In this case, the parameter is set to ”generational”, which means that the current population will be completely replaced at once by the new one.

A description of the implemented algorithm along with the applied parameters is shown in Algorithm 1. As with typical GAs, the algorithm starts by creating the initial population and evaluates the fitness of each individual (lines 1-4). In the main cycle, the number of crossover pairs is calculated

Algorithm 1: Genetic algorithm of BehaviorSearch

Input: GA parameters
Output: best fitness, generations of individuals

```
1  for i = 1 to population-size do
2  |  cur_pop ← add(create_chromosome());
3  end
4  fitness ← compute_fitness(cur_pop);
5  while search_unfinished() do
6  |  crossover_pairs ← (crossover-rate · population-size) ÷ 2;
7  |  for i = 1 to crossover-pairs do
8  |  |  parent_1 ← select(cur_pop, fitness, tournament-size);
9  |  |  parent_2 ← select(cur_pop, fitness, tournament-size);
10 |  |  new_pop ← add(mate(parent_1, parent_2));
11 |  end
12 |  for i = crossover-pairs + 1 to population-size do
13 |  |  new_pop ← add(select(cur_pop, fitness, tournament-size));
14 |  end
15 |  for i = 1 to population-size do
16 |  |  new_pop ← mutate(new_pop, mutation-rate);
17 |  end
18 |  cur_pop = new_pop;
19 |  fitness = compute_fitness(cur_pop);
20 end
```

based on the set crossover-rate (line 6). Afterwards, the new population is filled with offspring which result from mating the parents that are chosen through tournament-selection (lines 7-11). During this recombination process, the chromosomes are combined via single-point crossover (see Figure 6). Recombination in this form involves the random selection of a crossover point within the parent chromosomes. The two separate parts are then swapped to create new chromosome strands that form the children. Tournament selection is applied again to fill the new population with individuals from the current population until the population-size is reached (lines 12-14). Before the new population is evaluated again and the next cycle is entered, mutation is applied to each value of each individual according to the mutation-rate (lines 15-17).

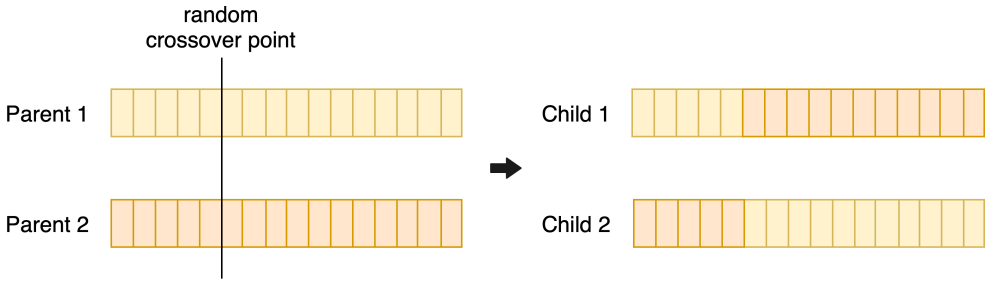


Figure 6: Single-point crossover (modified from [60])

4.2 Learning-to-learn (L2L)

Learning to learn (L2L) [16] is an optimisation framework that provides several gradient-free optimisation algorithms, including GAs. The algorithm along with the applied operators are executed using the DEAP framework [61], which specialises in evolutionary operations and computations. A description of the GA parameters can be found below:

- **pop_size** defines the number of individuals in each population.
- **tourn_size** determines how many individuals are considered during each selection round. The fittest individual is included in the next population.
- **cx_prob** indicates with which probability two individuals are combined to create new offspring.
- **mate_par** is the exploration factor used in blend crossover (Figure 7).
- **mut_prob** describes the probability with which a chromosome is mutated or, more precisely, enters the mutation phase.
- **ind_prob** indicates the probability with which a parameter of an individual (of a possible solution) is mutated.
- **mut_par** specifies the standard deviation of the Gaussian distribution considered during the mutation phase.
- **n_iteration** determines the number of generations created before the algorithm terminates.

Algorithm 2: Genetic algorithm of L2L

```
   Input: GA parameters
   Output: best fitness, generations of individuals
 1 for i = 1 to pop_size do
 2 |  cur_pop ← add(create_chromosome());
 3 end
 4 fitness ← compute_fitness(cur_pop);
 5 for i = 1 to n_iteration do
 6 |  for j = 1 to pop_size do
 7 |  |  new_pop ← add(select(cur_pop, fitness, tourn_size));
 8 |  end
 9 |  for j = 1 to pop_size by 2 do
10 |  |  if random() < cx_prob then
11 |  |  |  new_pop ←
   |  |  |    add(mate(new_pop[j], new_pop[j + 1], mate_par));
12 |  |  end
13 |  end
14 |  for j = 1 to pop_size do
15 |  |  if random() < mut_prob then
16 |  |  |  new_pop[j] ← mutate(new_pop[j], ind_prob, mut_par);
17 |  |  end
18 |  end
19 |  cur_pop ← new_pop;
20 |  fitness ← compute_fitness(cur_pop);
21 end
```

As with BehaviorSearch, the genetic algorithm in L2L begins with the creation of the initial population in which random candidate solutions are generated. Each individual is then evaluated in the simulation to determine its fitness. In the first step of the main cycle, the new population is filled with individuals from the current population via tournament selection (lines 6-8). In lines 9-13, this group of individuals is then iterated through and with a probability of cx_prob two individuals are paired up to produce new offspring. In this algorithm, recombination is realised using blend crossover, which is shown in Figure 7. With this type of recombination, the corresponding parameter values x and y of the parental chromosomes are taken to create a range (exploitation). A factor α (mate_par in case of L2L) is

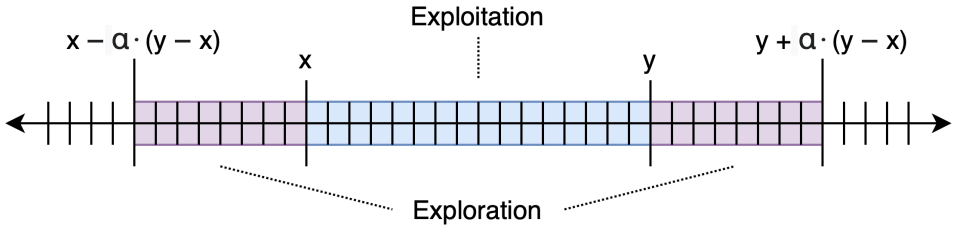


Figure 7: Blend crossover (modified from [62])

applied to this range in order to expand it (exploration). A random value is then selected out of this area for each of the two children. This procedure is repeated until complete chromosome strands are generated. Afterwards, uniform mutation is applied through the Gaussian distribution according to parameters mut_prob, ind_prob and mut_par (lines 15-18). This cycle is repeated until a certain number of generations, indicated by n_iteration, has been generated.

4.3 Comparison

In order to make an accurate comparison between both algorithms, it is necessary to point out the similarities and differences in the realisation of the GAs. First of all, both genetic algorithms have a very similar structure. In both cases, the proposed solutions are encoded directly according to the assigned values. An important difference concerns the step-size of the weight parameters. While L2L uses the full range, i.e. every real number is a possible assignment, the parameter range in BehaviorSearch is limited to a step-size of 0.2. The intention behind setting this restriction is further elaborated in Section 5.1. During the evolutionary operations, tournament selection is used in both implementations to choose the parent individuals as well as to fill in the new population with existing individuals. The main difference occurs in the application of recombination. While BehaviorSearch use single-point crossover, the genetic algorithm implemented in L2L applies blend crossover.

As mentioned earlier, L2L has a larger set of hyperparameters. For a more reasonable comparison of both algorithms in terms of performance,

the parameter settings should be chosen as similar as possible. Table 1 presents a comparison between the hyperparameters and clarifies which ones correspond to each other. Both hyperparameter sets include parameters to define the size of a population as well as the number of individuals considered in each tournament round. In both cases, crossover is performed with a certain probability. Therefore, parameters cx_prob and crossover-rate can be treated in the same way. As previously mentioned, blend crossover extends the considered area by a factor α. In L2L, this is defined by the additional parameter mate_par. Since the realisation of recombination varies between the algorithms, this parameter needs to be considered separately. During the mutation phase, L2L considers three parameters. mut_prob defines with which probability an individual is mutated or, more precisely, enters the mutation phase. In the case of BehaviorSearch, every individual goes through mutation. To achieve maximal similarity, in this study, mut_prob is therefore set to 1.0. ind_prob is the parameter that corresponds to mutation-rate in BehaviourSearch as both indicate the probability with which a value encoded in an individual is altered. Thus, these parameters can be treated equally. Lastly, mut_par defines the standard deviation of the Gaussian distribution considered during mutation. As in BehaviorSearch this value is set fixed to 0.1, the same number is chosen in the case of L2L.

Category	BehaviorSearch	L2L
Population	population-size	pop_size
Selection	tournament-size	tourn_size
Recombination	crossover-rate	cx_prob
		mate_par
Mutation		mut_prob
	mutation-rate	ind_prob
		mut_par

Table 1: GA parameter comparison of BehaviorSearch and L2L

5 Search Configurations

A set of search results is required in order to analyse the hyperparameters of the genetic algorithms. As a complete search of all possible settings is both time- and resource-consuming, a more efficient search approach is performed to extract and investigate well-functioning parameter settings. In the following sections, the search procedure and configurations as well as some underlying considerations are explained.

5.1 Hyperparameter prefiltering

Before conducting longer optimization runs, it is important to get familiar with the effects of the hyerparameters first. In order to be able to perform a broader but still efficient search, the runs during this stage were kept shorter. For this purpose, the genetic algorithm of BehaviorSearch is used for two main reasons. On the one hand, BehaviorSearch applies a "classical GA", which mainly means that the hyperparameters are limited to most essential ones. While this makes the exploration easier to follow, it also allows for a better understanding of how the parameters influence the search progress. On the other hand, since the search space is more restricted due to the defined step-size, the search process and the results will be more stable with different hyperparameter settings.

As listed in Section 4.1, the GA of BehaviorSearch has five adjustable parameters, four of which are of greater interest: population-size, tournament-size, crossover-rate and mutation-rate. First, the crossover-rate and the mutation-rate are examined through a short grid search. Then the best values are selected to examine the population-size in relation to the tournament-size. Each search runs for 1000 evaluation iterations (number of fitness determination) and for each setting 10 different seeds are tested. In case of the grid search, values from 0.0 to 1.0 with a step-size of 0.1 were tested with the population-size set to 32 and the tournament-size set to 3. In the first trial, the mutation-rate is explored with a fixed crossover-rate of 0.7. The following searches are ran with the best performing setting from previous

trials. It should be mentioned here that a full grid search would require a lot of resources, even if the searches are kept short. In this phase, the goal is not to identify the best performing configuration, but rather a well-performing range for each parameter. A further investigation of these ranges is then carried out through longer search runs. Hence, for this purpose, only a partial grid search is performed to extract the promising ranges. For the second group of parameters, the goal is to find a well-functioning ratio between population-size and tournament-size. The complete search results can be found in the Appendices and will be evaluated in Section 6.1.

5.2 Main investigation

The main purpose of this study is to compare the two genetic algorithms in terms of the best fitness achieved and the parameter settings that lead to these fitness scores. In the previous phase, the well-performing ranges of the hyperparameters were extracted. This part focuses on a deeper investigation through longer search runs performed on a cluster. At this point, it should be mentioned that the search executions vary between the two optimisation tools due to the differences in duration of the searches. With BehaviorSearch much longer runs were possible within an acceptable amount of time. Hence, for each search, the number of fitness evaluations was set to 60000 iterations and each setting was tested with 3 different seeds. On the other hand, the algorithm offered by L2L could achieve 100 generations within approximately the same time. In addition, due to the more variant search results with L2L, a more broader (regarding the ranges) and thorougher investigation (4 different seeds) was performed with the crossover and mutation parameters, while reducing the investigation on the population size and tournament size. An evalutation of the search results can be found in Section 6.2.

5.3 Static and dynamic world

With training any computational model, there is always the risk of overfitting [63]. During the long search runs, the simulation model considered

while evaluating the fitness of a solution was always kept the same. To be more precise, every object in the insect environment was placed at the exact same position for each simulation. On the one hand, a fixed world leads to a more stable search, which would make the analysis of the fitness development more comprehensive. On the other hand, an individual could achieve a high fitness value by over-fitting to the world, especially with more complex networks that allow more memory capacity. Hence, to prevent over-fitting, search experiments with variant world setting in each fitness evaluation were performed. With the aim to further counteract over-fitting, experiments were conducted with reduced simulation lengths, i.e. less simulation iterations (ticks). The underlying intention is to achieve a better performance of the ant faster. Individuals with a high fitness score would imply that the insect can find food more quickly, i.e. find more in fewer simulation time. For this investigation, three different simulation lengths were tested, that is, 20000 (standard length), 10000 and 5000 ticks. A total of 8 search settings with a fixed and a non-fixed world were applied for each length. Among these, 4 settings were selected from previous searches that resulted in high fitness values and the other 4 were performed with settings that yielded comparatively low fitness values.

5.4 Varying network topology

All searches described in the previous sections were performed with an SNN containing six neurons in the middle layer. However, it is educational to know how a network with fewer or more neurons would influence the behaviour of the insect. In particular, it would be informative to observe whether the insect can achieve the same behaviour with fewer than six neurons and whether it can perform even better with more neurons, for example, find food faster or more efficiently. For this purpose, tests were carried out with 0 to 14 second layer neurons. More than 14 neurons were not a practicable in the NetLogo simulation due to the overhead caused by the increasing number of parameters. It should be added here that in the case of the network with 0 neurons, i.e. without a middle layer, the input neurons are directly

connected to the actuators. In order to enable these motoneurons to influence each other, two synapses were added between them. A total of 8 searches were performed for each number of neurons, 4 with fixed world settings and another 4 with variant world to exclude the risk of over-fitting as described in the previous chapter.

6 Results and Evaluation

In this chapter, the results of the searches are evaluated and discussed. Following the procedure of the search runs described in Chapter 5, the data obtained from the short runs are first examined and then a more detailed investigation of the longer runs is presented. In particular, it involves the best fitness achieved and the parameter settings that contribute to this outcome.

6.1 Hyperparameter prefiltering

During the short search runs, three trials were performed for each crossover and mutation rate. For each trial, the best value from the previous trials was chosen for the corresponding fixed parameter. Mutation rates of 0.2, 0.4 and 0.5 were set for the exploration of the crossover rate and correspondingly 0.6, 0.7 and 0.8 were assigned to crossover rate for the exploration of the mutation rate. The search results can be found in Figures 22 - 27. For each setting, the average, median and standard deviation were calculated from the 10 different initial populations. The best performing settings are marked for each trial. For each trial, the settings with the best performance are marked. Best performance here means not only an overall high fitness, but also a stable fitness, as the setting should not only work well with certain initial populations. Therefore, a high mean and median as well as a low standard deviation is preferred.

From these experiments it can be concluded that overall lower values for the mutation rate and higher probabilities for crossover lead to good results. Specifically, a range of 0.2 to 0.5 for mutation rate and 0.6 to 0.8 for crossover rate can be derived, which will be further investigated in the

following section. Regarding population-size and tournament-size, the goal is to find a well-functioning ratio between them. For this purpose, population sizes of 16, 32, 48, 64 and 80 were tested in relation to tournament sizes of 3, 4 and 5. Figures 28 - 30 contain the corresponding search results. Overall, a ratio of 0.06 to 0.1 between tournament size and population size leads to the best results.

6.2 Main investigation

With the long search runs, the optimisation tools are first examined one after the other and then a comparison is made with regard to the hyperparameter settings and the achieved fitness.

6.2.1 BehaviorSearch

From the short searches, well-functioning ranges for crossover rate (0.6 to 0.8) and mutation rate (0.2 to 0.5) have been determined. In terms of population size and tournament size, 0.06 to 0.1 has proven to be a good ratio. Since the longer searches are performed on a cluster, it offers the opportunity to use larger populations. With additional experiments and the results from the short searches, the combinations 64-5, 80-6, 96-7 (population size to tournament size) were chosen to investigate crossover and mutation probabilities. Note that all of these pairs have a ratio of 0.07 to 0.08, which is within the range found.

In order to compare the results with the experiments conducted with L2L, a search duration of 100 generations performed by the GA of BehaviorSearch is first examined. Figure 8 shows the highest fitness achieved during the 100 generations in terms of crossover and mutation rate. The corresponding generation in which the best fitness first appeared is presented in Figure 9. Both the fitness values and the corresponding generation number represent the average of three different search runs for each setting. At first sight, it can be seen that with increasing population size, there is also an increase in the fitness score. More specifically, a population size of 64 individuals mostly achieves fitness values of 30 to 50, 80 individuals can reach a fitness of 40

to 55 and a group of 96 candidate solutions obtained a score of about 50 to 60. With regard to the crossover and mutation rate, it can be seen that for all tested population sizes, low values of both parameters lead to overall low fitness scores. For 64 and 80 individuals, higher fitness could be achieved with a high crossover rate and low mutation rate. However, with a population size of 96 individuals, the highest fitness values are obtained with larger values for both parameters. From the diagrams in Figure 9 it can be inferred that for larger populations, a higher number of generations is required to reach the highest fitness within 100 generations. Thus, 100 generations are not enough to make the searches converge, especially for larger populations.

Nevertheless, a first investigation is conducted on the peak points of the fitness scores. Table 2 shows the four highest fitness values for each population size along with the corresponding crossover and mutation rate and generation in which score has been achieved. It can be observed that the combination of 0.7 for the crossover rate and 0.2 for the mutation rate leads to peak values for all three population sizes. Furthermore, the following pairs 0.8 - 0.2, 0.8 - 0.3, 0.6 - 0.3 (crossover - mutation) occur twice in the three diagrams. In average, the generations in which the peak points first occur are overall after approximately 80 cycles. As previously mentioned, this indicates that there is still room for improvement when when running

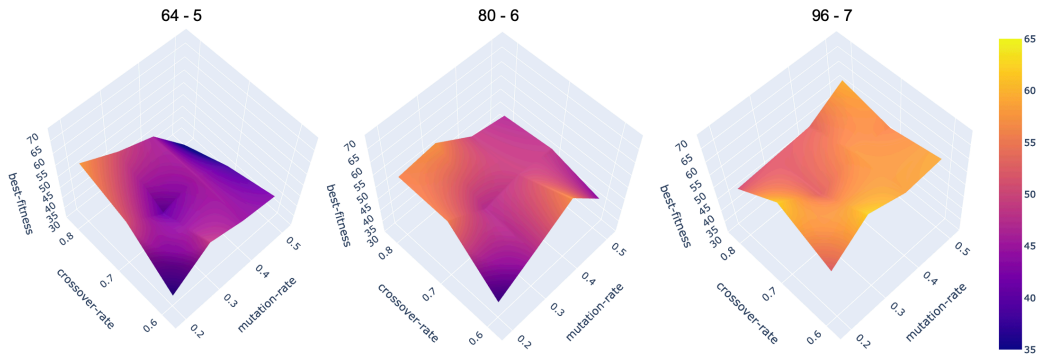


Figure 8: Best fitness achieved in 100 generations

Diagrams labelled with population-size - tournament-size

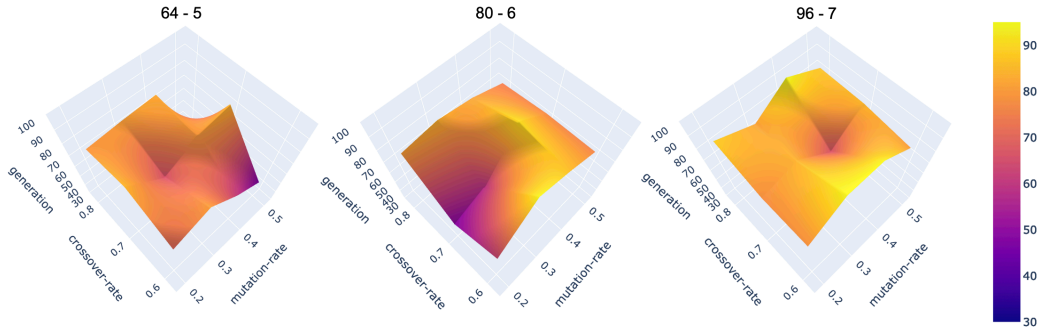


Figure 9: Generation in which best fitness first achieved in 100 generations

Diagrams labelled with population-size - tournament-size

pop-size	64	80	96
tourn-size	5	6	7
settings	cx - mut - fit - gen	cx - mut - fit - gen	cx - mut - fit - gen
peak points	0.8 - 0.2 - 58 - 79	0.8 - 0.3 - 57 - 83	0.7 - 0.2 - 62 - 78
	0.8 - 0.3 - 51 - 81	0.6 - 0.4 - 57 - 86	0.6 - 0.3 - 62 - 94
	0.7 - 0.2 - 51 - 84	0.8 - 0.2 - 56 - 80	0.8 - 0.5 - 60 - 85
	0.6 - 0.3 - 49 - 80	0.7 - 0.2 - 55 - 49	0.6 - 0.5 - 59 - 80

Table 2: Peak points of achieved fitness in 100 generations

the searches for more generations.

For this reason, results from searches ran for 625 generations are examined in the following sections. As before, Figures 10 and 11 show the graphs in which the fitness scores and the corresponding generations are plotted. As in the previous case, it is evident from the fitness plots that a higher population size leads to a higher fitness on average. This observation is expected as a higher population size corresponds to a larger number of solutions being considered in each iteration and therefore, it is more likely that a good solution can be found within the optimized population. Compared to 100 generations, a general improvement in the fitness can be observed. With 64 individuals, a fitness between 60 and 75 is achieved. With 80 individuals, the fitness values are roughly in the same range, with the difference that they appear more stable across the different parameter settings. With a population

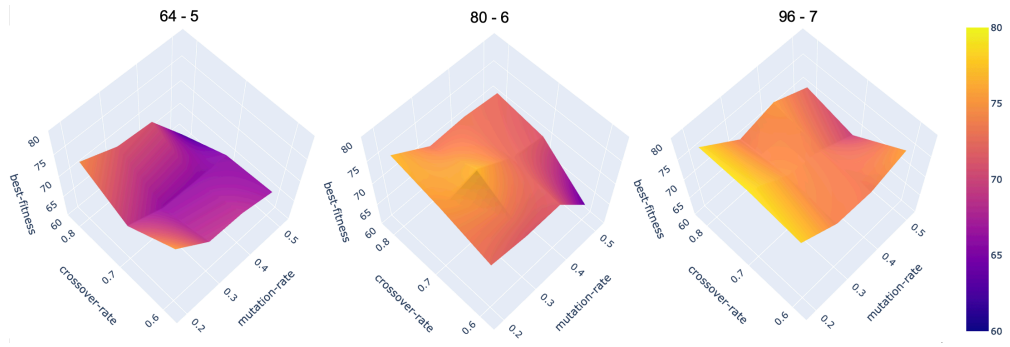


Figure 10: Best fitness achieved in 625 generations
Diagrams labelled with population-size - tournament-size

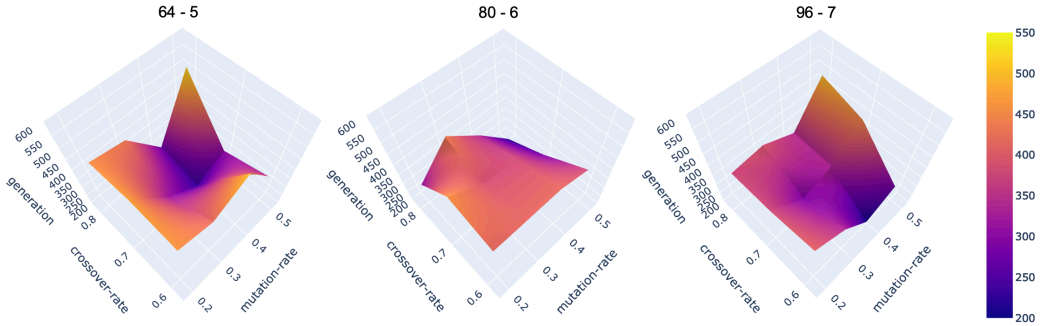


Figure 11: Generation in which best fitness first achieved in 625 generations
Diagrams labelled with population-size - tournament-size

size of 96, the highest fitness values could be achieved, i.e. values between 70 and 80. In the generation plots, a much greater variation can be observed through the different parameter settings. Overall, it can be noted that at least an average of 200 generations is required to assume that the development of the fitness has converged. In most cases, however, the highest fitness was reached at about 400 generations.

Just as before, it is interesting to observe precisely which parameters cause the peak points of the fitness. For this purpose, Table 3 contains the peak points. According to this table, 0.8 for the crossover rate and 0.2 for the mutation rate lead to the highest peak values for all population sizes. In addition, the combinations 0.8 - 0.4, 0.7 - 0.2 and 0.6 - 0.2 appear twice in

pop-size	64	80	96
tourn-size	5	6	7
settings	cx - mut - fit - gen	cx - mut - fit - gen	cx - mut - fit - gen
peak points	0.6 - 0.2 - 75 - 461	0.8 - 0.2 - 77 - 311	0.8 - 0.2 - 79 - 376
	0.8 - 0.2 - 74 - 450	0.7 - 0.3 - 77 - 386	0.7 - 0.2 - 79 - 381
	0.8 - 0.3 - 71 - 277	0.7 - 0.2 - 75 - 459	0.6 - 0.2 - 78 - 409
	0.8 - 0.4 - 71 - 430	0.8 - 0.4 - 73 - 333	0.6 - 0.5 - 76 - 280

Table 3: Peak points of achieved fitness in 625 generations

the peaks. Looking at the generations, roughly 400 generations are required to achieve these peak values.

As previously mentioned, it is noticeable and comprehensible that with a higher population size an overall higher fitness can be achieved, as more possible solutions are considered. Thus, it would be interesting to observe how fitness evolves if the number of search simulations, i.e. fitness evaluations, remains the same for the different population sizes. For this purpose, 12, 13 and 14 show the best fitness achieved after 20000, 40000 and 60000 evaluation iterations. It can be observed that after 20000 iterations the achieved fitness values are generally very similar between the 3 charts. To be more precise, an average fitness of 60 to 70 can be achieved with all tested population sizes and settings.

After 40000 simulation evaluations a difference between the diagrams can be seen. With 64 individuals in a population, the fitness values remain about the same as after 20000 iterations, only the settings 0.8 - 0.2 and 0.6 - 0.2 lead to a fitness of about 75. With a population size of 80, an increase in fitness can be observed with the different crossover and mutation pairs. Starting from low crossover rates in combination with a high mutation rate, only a fitness of 64 could be achieved. With increasing crossover and decreasing mutation probabilities, an increase in the fitness values can be detected. In particular, the highest fitness of 77 could be achieved with 0.8 as the crossover rate and 0.2 as the mutation rate. In the right-hand diagram (96 individuals), a more even fitness occurs with different crossover and mutation settings, i.e. the average fitness values achieved are approximately between 70 and 77.

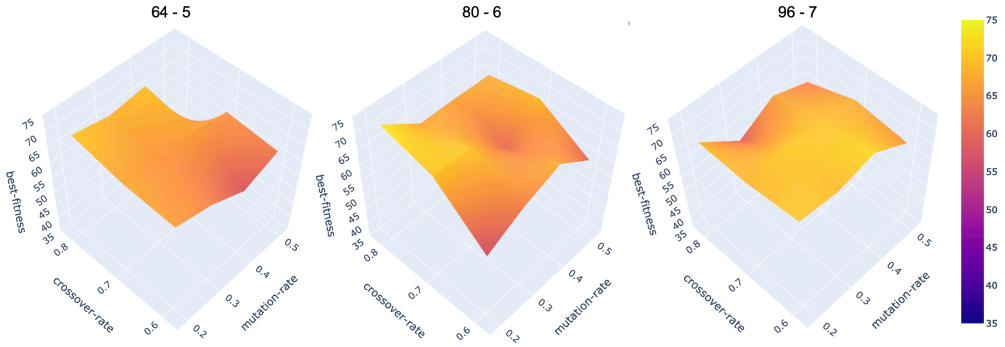


Figure 12: Best fitness achieved in 20000 evaluation iterations

Diagrams labelled with population-size - tournament-size

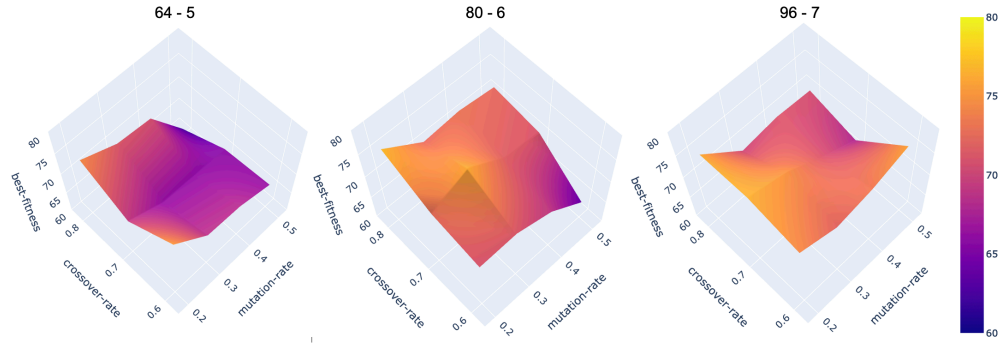


Figure 13: Best fitness achieved in 40000 evaluation iterations

Diagrams labelled with population-size - tournament-size

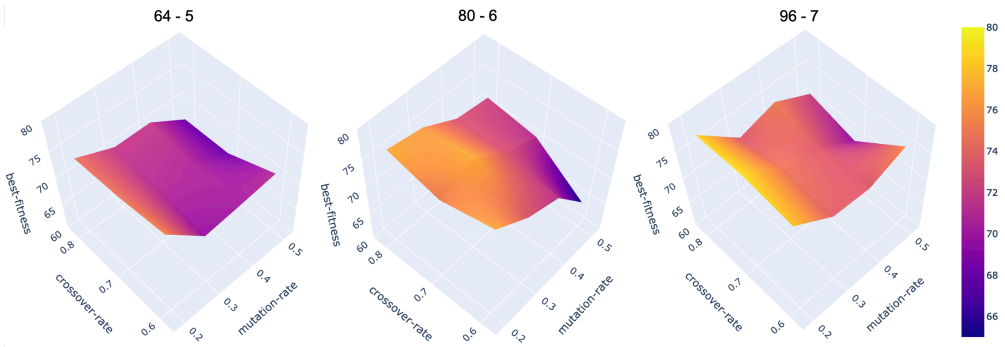


Figure 14: Best fitness achieved in 60000 evaluation iterations

Diagrams labelled with population-size - tournament-size

Figure 14 shows the average fitness achieved within 60000 iterations. As with the diagrams in Figure 13, a better overall fitness was still achieved with larger population sizes. From these plots, a clearer tendency can be seen as to which settings works best. Precisely, the highest fitness values, approximately 75 to 80 in average, can be achieved with low mutation rates, i.e. 0.2 to 0.3.

Considering population sizes specifically, it can be concluded that with a 30% larger population, a 10% better fitness can be achieved within the same number of evaluation iterations.

6.2.2 L2L

Using the genetic algorithm of the L2L framework the search was carried out for 100 generations. After initial runs and testing of different parameter settings, the best fitness achieved varied considerably. In order to focus on the crossover and mutation probabilities and examine them more thoroughly the population and tournament size were set to 96 and 7. This combination provided the best results for the simulations with BehaviorSearch. While performing more experiments using L2L, results have shown that increasing both parameters does not lead to good fitness values. Therefore, the ranges were expanded and the probability values were set lower, i.e. crossover rates of 0.5 to 0.8 and mutation rates of 0.1 to 0.5 were investigated. As mentioned in Section 4.3, blend crossover uses a factor that extends the search range. Since the GA of L2L applies this type of crossover, this parameter needs to be set for the searches. When experimenting with very high and very low rates, no performance improvements were obtained. Hence, the rate of 0.5 recommended by literature was taken. In order to be able to make more accurate statements and conclusions, 4 different searches were carried out for each parameter combination.

Both the average of the best fitness achieved and the average of the generation in which this fitness occurred for the first time are shown in Figures 15 and 16 in relation to the crossing and mutation probabilities. At first glance, it is noticeable that one parameter setting achieves by far the highest fitness

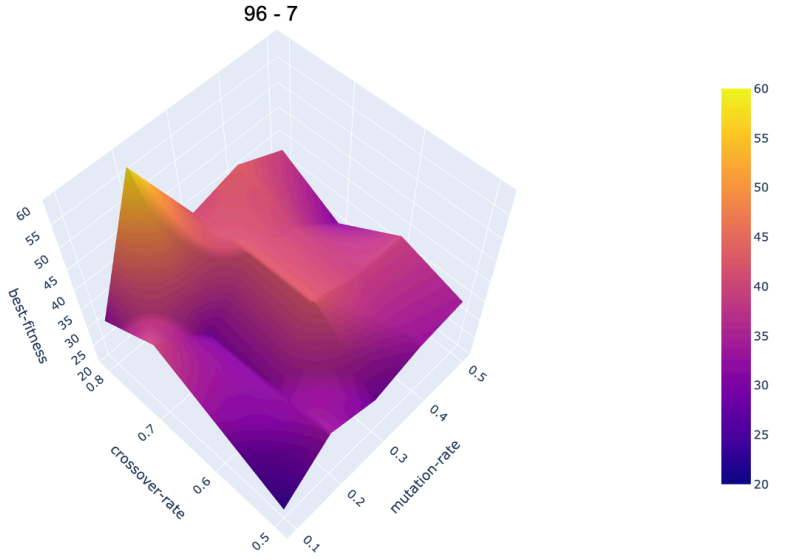


Figure 15: Best fitness achieved in 100 generations

Diagram labelled with population-size - tournament-size

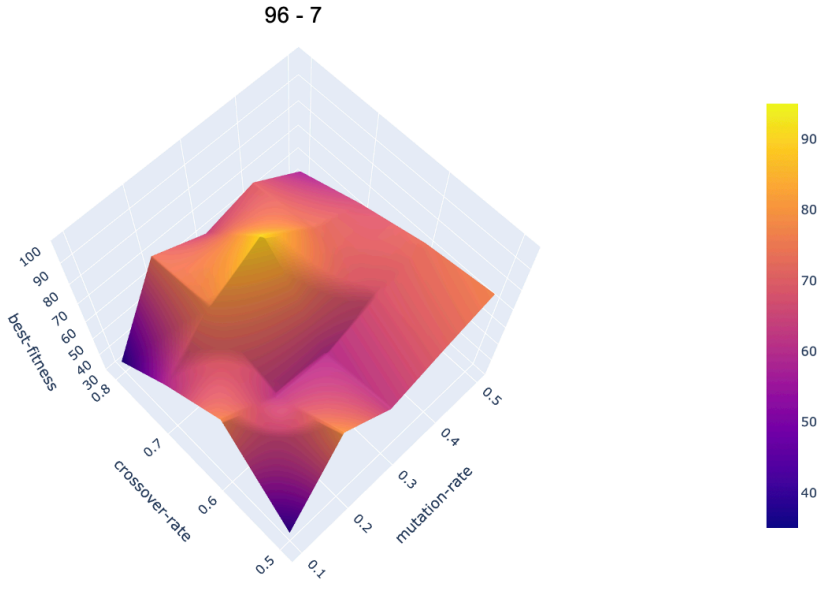


Figure 16: Generation in which best fitness first achieved in 100 generations

Diagram labelled with population-size - tournament-size

values, that is, 0.8 for crossover and 0.2 for mutation rate. With this setting, an average fitness of 59 could be achieved. Apart from that, the range with the mutation rate 0.3 and 0.4 together with the crossover 0.6 to 0.8 gives a comparatively better fitness value. The exact fitness values are listed in Table 4. It shows that apart from the highest peak value of just about 60, the best achieved fitness values are around 40. Figure 16 shows that about 60 to 70 generations are required to reach these fitness values. However, as with BehaviorSearch, this indicates that the search has not yet converged after 100 generations and running the searches for some more generations would likely achieve even higher fitness values.

pop-size	96
tourn-size	7
settings	cx - mut - fit - gen
peak points	0.8 - 0.2 - 59 - 74
	0.6 - 0.3 - 43 - 61
	0.8 - 0.4 - 43 - 70
	0.7 - 0.3 - 42 - 90

Table 4: Peak points of achieved fitness in 100 generations

6.2.3 Comparison

In this section, the results of the two optimisation tools are compared in terms of the fitness achieved as well as the crossover and mutation settings. For a reliable comparison, the graph on the right in Figure 8 (data from BehaviorSearch) together with Figure 15 (data obtained from L2L) are considered. As a reminder, the data is obtained by using a population size of 96 and a tournament size of 7, and the search is performed for 100 generations. When comparing the average fitness values, the data obtained with L2L show more variation overall with different parameter settings. Furthermore, BehaviorSearch achieves an average fitness value between 50 and 60, while L2L's fitness data is mostly between 30 and 40. Within 100 generations, there is thus a difference of about 20% between the algorithms in

terms of fitness. When looking at the generations in which the best fitness could be achieved, it emerges that the values of BehaviorSearch are reached on average after 80 generations. With L2L, the best fitness can be achieved after about 70 generations. Thus, the best fitness values can be reached about 13% earlier with L2L than with BehaviorSearch. It should be noted here that these statements can only be concluded on the basis of data from 100 generations. As genetic algorithms are very stochastic the percentages can change significantly when running the searches for longer iterations. In terms of crossover and mutation probabilities, the settings with which the peak values can be achieved are similar for both tools. In particular, the combinations 0.8 - 0.2, 0.7 - 0.2 and 0.6 - 0.3 lead to the highest fitness values across the different population sizes and search tools. Especially with L2L, the 0.8 - 0.2 setting showed significantly higher fitness than all other settings tested. In summary, a higher crossover rate in combination with low mutation rates gives the best fitness values.

Another point to investigate is the number of generations the search is executed. For example Figure 12 shows that overall a very high fitness of 60 to 70 can already be achieved within 20000 iterations. In this context, it is important to also observe how well the ant behaves in the simulation when it reaches a certain fitness. After testing and observing the ant's behaviour in the simulation, it can be seen that the magnitude of fitness corresponds directly to how well the ant navigates through the world. With a fitness of less than 40, at best, the ant moves very slowly through the world, while constantly spinning. During this process, it can only gather a very small amount of food. With a fitness between 40 and 60, the ant can gather more food and moves around through the world with only few collisions. However, once the ant is trapped, i.e. surrounded by obstacles, it is constantly rotating and does not manage to escape from the trap. Thus, from this point on, the fitness value no longer increases. If the SNN is set with parameters obtained from a run with an achieved fitness of 60 up to approximately 80, the ant shows a much better behaviour. It is able to collect a lot of food in a short amount of time with only very few collisions. Nevertheless, the ant is still easily trapped and either takes a very long time to get out again

or has to encounter many harmful objects to escape the trap. Furthermore, when changing the settings of the world, i.e. the position of the food and rewards, the insect behaves worse and sometimes even crashes randomly with an obstacle. With a fitness of over 80, the ant shows an excellent behaviour. In most cases, it is able to find about half of the available food in less than 50000 simulation iterations without a single collision. Furthermore, the ant has developed a trap escape mechanism that allows it to quickly get out of traps with only few collisions. When changing the world setting, the ant still shows good behaviour, indicating that no over-fitting has occurred, even if a very high fitness has been achieved.

In conclusion, the ant can only display excellent behaviour if the searches are ran for sufficient amount of time, i.e. about 400 generations. However, an acceptable behaviour can be still achieved within approximately 200 to 250 generations.

6.3 Static and dynamic world

In this section, the results of the research with fixed and changing world settings are considered. The resulting fitness score for each search can be found in Figure 31. With 20000 iterations, a changing world setting could not achieve a better fitness with settings that provide a better performance, but obtained a higher fitness with comparatively poorly functioning settings. In the simulation, however, no improvement in the behaviour of the insect could be observed in either case. In case of 10000 ticks, a slightly better fitness was achieved in all 8 settings with a constantly changing insect world. But as in the previous case, no improvements of the insect could be observed during the simulation. Furthermore, unlike with 20000 ticks, the trap escape mechanism did not occur anymore with half the number of simulation iterations. A significant difference in fitness scores can be observed when the simulation was run for only 5000 ticks during the search. Approximately 20% better fitness scores could be achieved with a non-fixed insect environment setting. However, the insect is not only caught very easily, as in the previous case, but also cannot even escape from traps that are not too difficult to overcome.

In summary, the default setting of 20000 simulation ticks during the fitness evaluation is required to achieve the desired behaviour of the ant including the mechanism to escape from the obstacle traps.

6.4 Varying network topology

When examining the different network topologies, i.e. different number of hidden-layer neurons, overall a very high fitness, i.e. between 70 and 80, could be achieved with most of the networks (see Figure 32). In some cases, the SNNs with more than 10 neurons in the middle layer only achieved a fitness of about 40. Since the set of network parameters to be optimised is much larger, it is reasonable that a longer search is probably required to achieve a higher fitness. In addition, as expected, the fitness values of SNNs with only one second-layer neuron are relatively low. Since both actuators cannot be controlled separately via this neuron, the importance of the weight and especially the delay parameters becomes more significant. An exemplary network is investigated in more detail in Section 7.3.

When testing the optimised SNNs in the NetLogo simulation, a difference in the insect's performance is observed at different levels of network complexity. With 2 up to 6 second-layer neurons, the ant shows a very good behaviour. It moves quickly through the world and collects many rewards with only few collisions. A similarly good performance could be achieved with networks in which the motoneurons are directly influenced by the input neurons. However, with more than six neurons in the middle layer, especially with double-digit numbers of neurons, the insect's performance in the simulation has decreased. This is referring to the fact that the ant sometimes does not seem to know which movement to perform or that it hesitates before performing an action. It can be explained by the fact that the network performs more calculations in every situation, even if when the situation is not that difficult to handle. Besides, it cannot be dismissed that a network optimised by longer searches could lead to an improvement in this behavioural observation.

7 Network Analysis

In this chapter, the optimised spiking neural networks are examined in more detail. As previously mentioned, the insect performs very well with a small number of second-layer neurons when controlled by an optimised network. For this purpose, four exemplary networks with varying number of neurons are examined in more detail in the following sections to find out how the ant's good behaviour is achieved. The graphical representation of all optimised networks with up to 6 second-layer neurons can be found in the Appendices.

In order to better understand how the network dynamics work, a SNN with two neurons in the middle layer is considered first. Afterwards, a more complex network with six neurons is examined, in particular, to investigate how the neuronal dynamics and the division of labour between the neurons occur in this case. Next, as indicated in Section 6.4, an analysis of the network with only one neuron in the second layer is presented. Finally, for the sake of interest, the last network to be considered in more detail is an example of a SNN without a middle layer.

7.1 2 hidden-layer neurons

Figure 17 shows an example network with two middle-layer neurons. In order to understand how the actuators are controlled by these neurons, the network parameters between layers two and three are analysed first. It can be noticed that each of these neurons is responsible for controlling one of the actuators. More specifically, the neuron labelled "1" is strongly excitatory (weight of 20) for rotation and strongly inhibitory for forward movement. On the other hand, the neuron labelled "2" is strongly inhibitory (efficacy of -20) for rotation and strongly excitatory for movement. It can be concluded from this that the actions are mutually exclusive, i.e. the ant only performs one action at a time. This is further reinforced by the fact that the neurons of the middle layer are mutually inhibiting. In fact, this pattern, i.e. the coordination of rotation and movement and the mutual inhibition, can be found in each of the optimised networks of this complexity (see Appendices).

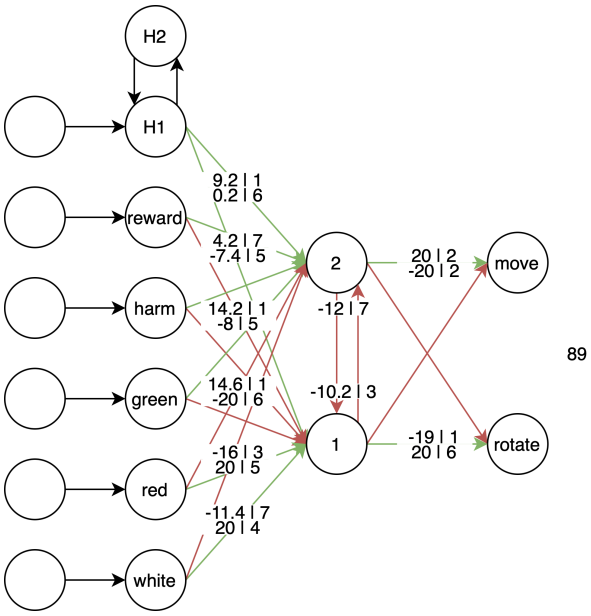


Figure 17: Optimized network with 2 neurons in middle layer

Excitatory synapses are coloured green, inhibitory synapses are coloured red

Considering the input layer, it can be seen that the neurons responsible for processing white and red colour sensations have a very strong excitatory effect on the neuron labelled ”1”, i.e. rotation. At the same time, both have an inhibitory effect on neuron ”2” (movement). Hence, the insect rotates when it observes an obstacle. It is noticeable that the inhibitory effect is a bit stronger when perceiving a red patch than when sensing a white patch. This can be explained by the fact that red patches are located all over the virtual world, while white patches only appear as part of a border or wall. Therefore, the ant behaves a little more cautiously when it perceives harm than when it sees a wall. When the ant discovers a reward, it moves towards it. In the network, the responsible neuron (marked ”green”) has a relatively strong excitatory effect (14.6) on neuron ”2” and a very strong inhibitory effect on the neuron controlling rotation. A particularly surprising discovery can be observed in the influence of the harm-processing neuron on the actuators. If the ant encounters harm, the movement neuron is triggered immediately, while rotation is inhibited. This observation explains the ant’s

ability to quickly escape from traps with only a few collisions. Because in a situation where the ant is surrounded by harmful objects, it decides to move forward instead of rotating. Although it risks some punishments from collisions, it is able to escape the traps and search for more food from there on. This behaviour is more beneficial than an otherwise constant rotation. Because in that case, the insect would avoid punishments, but would not be able to improve its fitness again, as it cannot collect further rewards. In the case where the ant collects a reward, the neuron that triggers the movement actuator is slightly excited and rotation is moderately inhibited. A simple explanation for this behaviour would be that after collecting food, the ant moves forward to locate other rewarding objects. Regarding the influence of the heartbeat, there is a moderate excitatory effect on the neuron controlling the movement action. Consequently, the insect constantly moves small steps forward if neuron "2" or the movement actuator is not inhibited. This explains why the ant is overall moving very quickly throughout the world.

By examining the delay values, a predictive ability of the insect can be detected. An example is given by the outgoing synapses of neuron "1". Primarily, this neuron is triggered when the photoreceptors perceive an obstacle. If the neuron "1" fires, the movement actuator is immediately inhibited (delay of 1). In contrast, the incoming excitatory efficacy of the rotation neuron is strongly delayed (delay of 6). It shows that although the ant perceives a harm, it does not turn around immediately, and instead makes a turn a few iterations later. Since the insect has a visual distance of up to four patches, it knows that the obstacle is still a few steps away, and therefore a turn is not necessary until a few steps have been taken in that direction.

7.2 6 hidden-layer neurons

After examining a well-functioning network with 2 middle layer neurons in the previous section, this chapter analyses an optimised network with six neurons in the second layer (see Figure 18). As this network is much more complex, containing more connections, the internal network calculations are influenced by a larger set of parameters. Therefore, investigating the network

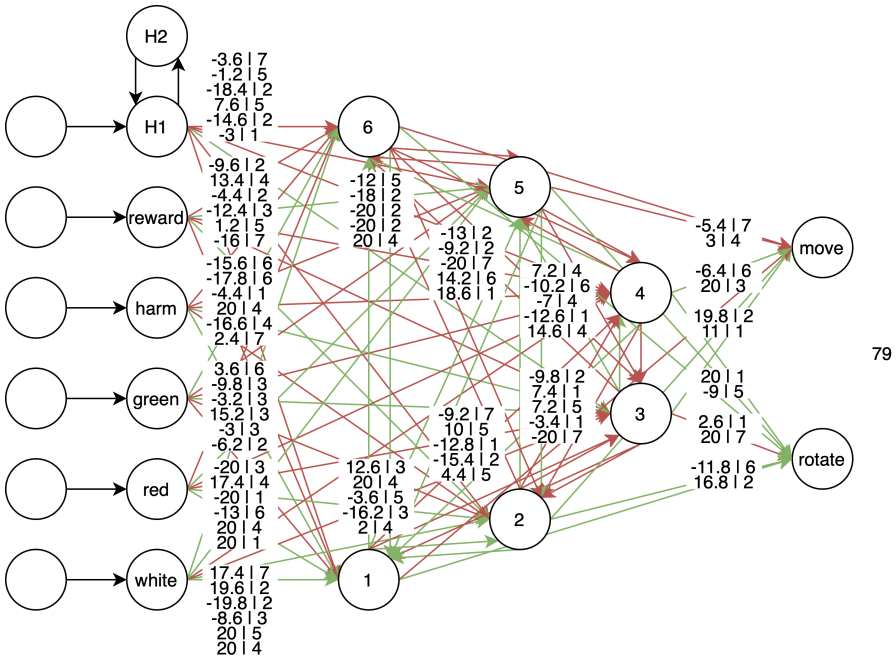


Figure 18: Optimized network with 6 neurons in middle layer

Excitatory synapses are coloured green, inhibitory synapses are coloured red

in layers allows for a more comprehensive analysis. First, the tasks of the middle neurons regarding the control of the motoneurons are determined. From the weight parameters it can be derived that the neurons labelled ”1”, ”2” and ”5” are strongly excitatory for the rotation actuator. Neuron ”3”, on the other hand, is responsible for controlling locomotion. Neuron ”4” has an excitatory effect on both rotation and movement, but much more strongly on the latter. Neuron ”6”, however, has no significant influence on either of the motoneurons.

Next, an investigation on the efficacy of the input neurons is presented to understand how the network responds to a particular stimulation. The outgoing synapses of the neuron responsible for processing white colour sensation reveal that mainly the neurons that trigger rotation are strongly excited, while others are inhibited. This is reinforced by the fact that these neurons are mutually excitatory. However, during this process, neuron ”6” is also triggered, which strongly inhibits each of these neurons except for neuron

”1”. This indicates that neuron ”6” is responsible for reducing the effectiveness of these neurons, which results in the ant not rotating too much. When a red patch is detected, the same neurons are triggered causing the ant to rotate. However, in this case, neuron ”6” is slightly inhibited, revealing that the ant behaves more cautiously when facing red objects. Given the situation in which the insect has its focus on a green object, it moves towards it. In the network, neuron ”3” is triggered, which is responsible for locomotion.

As with the SNN considered in the previous section, the trap escaping mechanism is found with the neuron labelled ”harm”. A collision with an obstacle is handled by triggering neuron ”3” (forward movement) and mostly inhibiting the other middle-layer neurons. However, when looking at the computations of the network after collecting a reward, it becomes apparent that mainly neuron ”5” (strongly excitatory for rotation) is activated. This observation suggests that the ant changes its direction after collecting a reward in order to look out for more nearby. Finally, the efficacy of the heartbeat is exclusively excitatory for the neuron that influences forward movement. As already mentioned, this explains why the ant can move fairly quickly through the virtual environment.

7.3 1 hidden-layer neuron

As mentioned before, the networks containing only one neuron in the middle layer achieved a comparatively low fitness value, since both actuators cannot be controlled separately. Nevertheless, it is remarkable that the ant was still able to collect over 20 rewards when controlled with the network shown in Figure 19 without a single collision. From this network it can be noticed that neuron ”1” has a significantly stronger excitatory effect on the rotation actuator than on the movement actuator. In addition, the signals arriving at the ”rotate” motoneuron are strongly delayed, whereas the signals sent to the movement actuator are effective immediately. Combining both weight and delay values, it shows that the ant the rotation actuator is triggered constantly, but with small breaks in between due to the strong delay. Due to the low weight of the synapse controlling the ”movement” motoneuron,

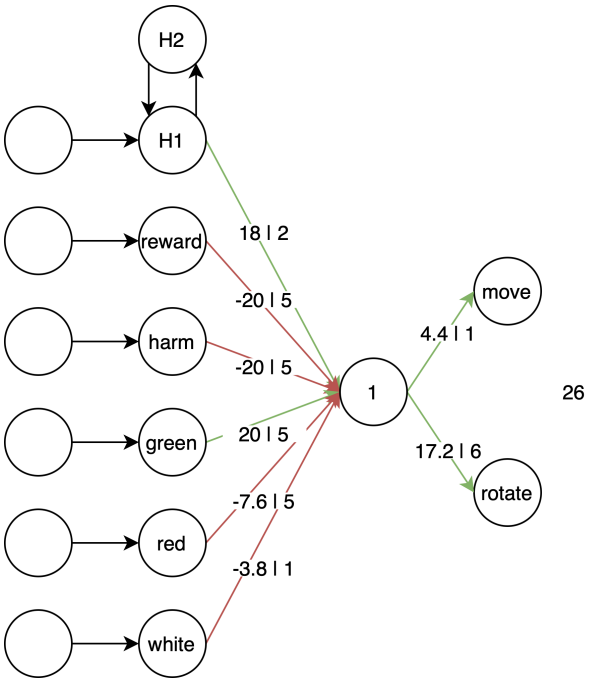


Figure 19: Optimized network with 1 neuron in middle layer

Excitatory synapses are coloured green, inhibitory synapses are coloured red

combined with the decay mechanism, the membrane potential increases very slowly but steadily because of the immediate excitatory effect. Therefore, this motoneuron fires at longer intervals.

In terms of the neurons in the first layer, it can be seen that the only neurons that can trigger neuron ”1” to fire are the neuron that processes green colour sensation and the heartbeat neuron. Both have a strong excitatory effect, while the other synapses have an inhibitory effect. When an obstacle, i.e. a white or red spot, is perceived, the middle layer neuron is moderately inhibited. As a result, the overall firing rate of neuron ”1” is slowed down. This has the consequence that it is harder for the membrane potential of the movement actuator to reach the firing threshold due to the long decay time. Hence, the ant mainly rotates in this situation. When the insect comes into contact with a reward or harm, neuron ”1” is strongly inhibited. Consequently, neuron ”1” hardly fires at all, which means that the movement actuator can no longer reach the firing threshold. As for the ant’s behaviour,

it exclusively rotates when it collides with an obstacle or collects a reward.

Overall, the ant moves through the world while constantly rotating and avoids forward movement by lowering the firing rate of the middle layer neuron. With this network, not only the importance of weight and delay parameters, but also the properties and benefits of a spiking neural network become more obvious.

7.4 0 hidden-layer neurons

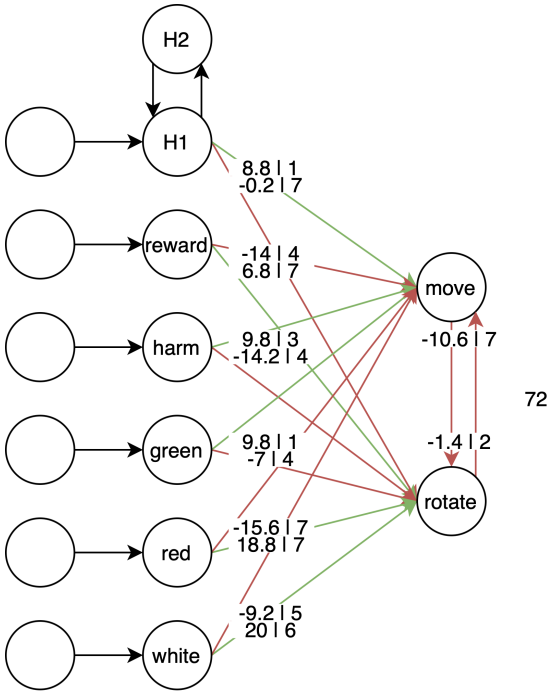


Figure 20: Optimized network with 0 neurons in middle layer

Excitatory synapses are coloured green, inhibitory synapses are coloured red

In the last section of this chapter, an optimised network without middle-layer neurons is considered, i.e. the input neurons directly influence the actuators. Mutual inhibition can be found with the two motoneurons. There is a much stronger inhibitory effect coming from the neuron labelled ”move” than from the neuron labelled ”rotate”. When the ant perceives a white or red coloured patch, i.e. an obstacle, the rotation actuator is strongly excited

and due to the short delay, the movement actuator is almost immediately inhibited during this process. When the ant perceives a white or red coloured patch, i.e. an obstacle, the rotation actuator is strongly excited and the neuron labelled "move is inhibited, particularly strong for red visual sensation. The high delay values, as in the example network discussed in Section 7.1, indicate the ant's ability to predict future events and behave accordingly. When recognising a reward, the movement neuron is only moderately triggered. This reflects the ant's overall cautious behaviour when moving through the world. A collision with harm is handled in the same way as all the networks examined in the previous sections, i.e. it is moving forward in order to escape the harm quickly. After collecting food, the insect rotates slightly to look for more food that is in reach. As also previously indicated, the excitatory efficacy of the artificial heart on the movement actuator enables the insect to move quickly throughout the world.

Overall, a very similar internal computation compared to the network with two middle-layer neurons (Figure 17) can be observed. This is further confirmed based on the achieved fitness as well as when testing the optimised network in the simulation, as the ant shows a likewise excellent behaviour.

8 Multi-Ant Model

In this section, a modified version of the model described in Chapter 3 is discussed. Instead of simulating a single ant, this version models several ants, precisely 10 of them, that move through the same virtual world. Besides gathering food and avoiding obstacles, another goal of the ants is to avoid collision with each other. Since in the simulation the ants are coloured yellow, this model is extended by another photoreceptor and an afferent neuron that is responsible for processing this colour. An exemplary network is shown in Figure 21.

Since the ants should avoid each other, a collision leads to a punishment, i.e. a reduction in the fitness score. As mentioned earlier, the challenge in using a GA for optimisation is to define a suitable fitness function. In

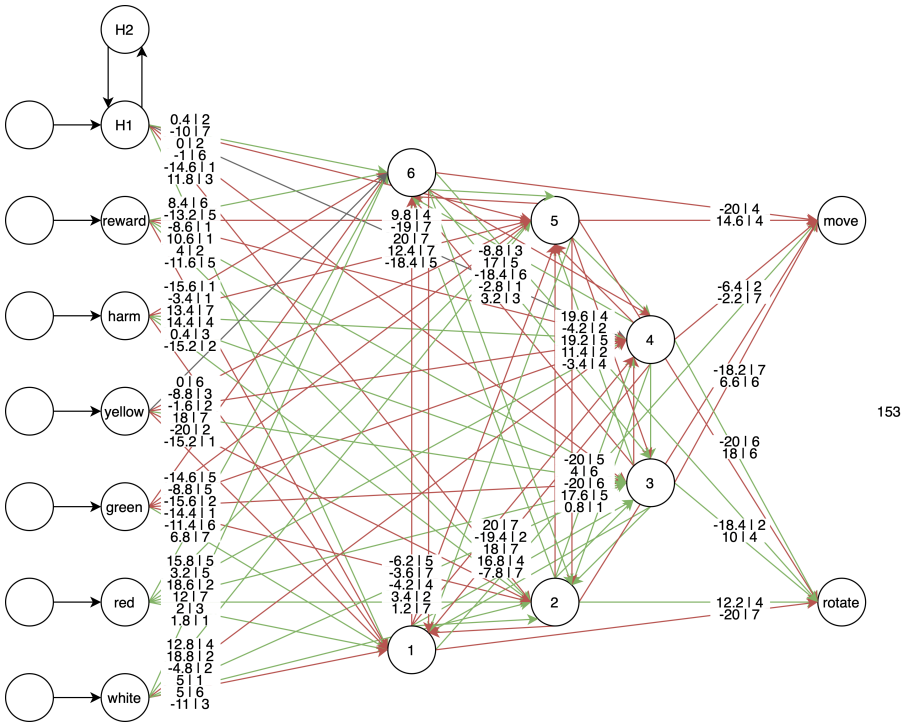


Figure 21: Optimized network of Multi-Ant model

this simulation, the ants should prefer to avoid each other over collecting food and dodging obstacles. First, the optimisation was performed with a penalty of -5 for ants crashing into each other. However, when testing the optimised network in the simulation, the ants stop moving when they perceive each other and remain in that position until the simulation finishes. This observation indicates that the penalty was chosen too high. Therefore, penalties of -2 and -3 were tested in the following optimisation searches. By decreasing the penalty values, this behaviour no longer occurs. The insects manage to find many rewards in a short time without colliding with a single obstacle or bumping into each other. To be more precise, over 150 of approximately 200 rewards were collected by the ants. As depicted in Figure 21, the ants turn slightly when they perceive a yellow object, i.e. another ant.

9 Conclusion

9.1 Summary

The aim of this work is to analyse genetic algorithm hyperparameters and extract settings that lead to the best performance of the GAs. For this purpose, a model that simulates an artificial ant navigating through a virtual environment is considered and evolved. Ideally, the ant behaves in such a way that it finds lots of food as quickly as possible while avoiding any obstacles. During this process, it is controlled by an SNN in which each connection (synapse) has a weight and a delay parameter. To achieve the desired behaviour of the ant, it is necessary to optimise these parameters. For this purpose, GAs from two different optimisation tools (L2L and BehaviorSearch) are applied. Like many meta-heuristic search methods, genetic algorithms contain a set of hyperparameters that influence the progress of the search.

In this work, both optimisation tools are analysed in terms of best hyperparameter settings and achieved fitness. From several search experiments and obtained data sets, the general conclusion is that a high crossover rate and a low mutation rate achieve the best results in terms of fitness score. To be more precise, the settings 0.7 - 0.2, 0.8 - 0.2, 0.6 - 0.3 (crossover rate to mutation rate) have shown to perform very well for both tools. In particular, the pair 0.8 - 0.2 achieved by far the highest fitness value on average for L2L. In terms of population size, a larger group of individuals generally achieves better search results with a tournament to population size ratio of 0.07 - 0.08. Furthermore, at least 200 generations are required until the search converges to a stable fitness. However, since experiments have shown that the fitness value directly corresponds to how well the ant behaves in the simulation, the searches need to be performed for about 400-500 generations to achieve the desired behaviour of the ant. In addition, experiments were conducted with simulation iterations of different lengths when evaluating the fitness values in order to reduce the risk of overfitting. Fortunately, the simulation tests have shown that as long as the achieved fitness is high enough, i.e. above 80, overfitting does not seem to occur. Furthermore, the experiments have

shown that at least 20000 iterations are required for the ant to obtain the ability to efficiently escape from traps. When investigating different network architectures, the results reveal that only two neurons in the middle layer are necessary to obtain a well performing ant. A similar SNN consisting of only the input and output layers show similarly good performance. Finally, using a model that simulates several ants, it was possible to show that the ants exhibit a satisfying behaviour in a world with not only fixed obstacles, but also with moving agents that are supposed to be avoided.

9.2 Outlook

In this work, mainly the parameters of the genetic algorithms were investigated, while other model parameters remained unchanged. With the possibility of achieving an even better performance of the insect, the optimisation of the parameters influencing the spiking neurons listed in Table 5 offer a possible suggestion for future research. In addition, the focus of this work was mainly on the achieved fitness values, however, the data sets obtained from the searches also contain candidate solutions that were considered throughout each search run. Thus, possible research suggestions would be a) a deeper investigation of individuals evolving throughout the generations, i.e. development of solution candidates and b) the identification of patterns of synaptic parameter settings that achieved good or bad performance. During the investigation of the GA parameters, one limitation in the influence of the search process became obvious. That is, after setting these parameters, they remain fixed during the whole search run. Even though genetic algorithms have the ability to escape local optima, it can be observed that the searches will converge, i.e. a further improvement of the fitness score is very unlike. To counteract this occurrence, Kamoi et al. [41] have proposed an improved genetic algorithm which adapt the GA parameter settings during the search process, whenever it converges. In particular, the diversity is measured based on the ratio between the fitness of the best individual and average fitness in the population. Hence, the application of an improved genetic algorithm to the considered model could be a suggestion for future research as well.

References

[1] C. C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, 2018.

[2] W. Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural Networks*, 10(9):1659–1671, 1997.

[3] S. R. Nandakumar, S. R. Kulkarni, A. V. Babu, and B. Rajendran. Building Brain-Inspired Computing Systems: Examining the Role of Nanoscale Devices. *IEEE Nanotechnology Magazine*, 12(3):19–35, 2018.

[4] A. Abusnaina and R. Abdullah. Spiking Neuron Models: A Review. *International Journal of Digital Content Technology and its Applications*, 8:14–21, 2014.

[5] W. Gerstner and W. M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[6] S. Scarpetta, I. Apicella, L. Minati, and A. de Candia. Hysteresis, neural avalanches, and critical behavior near a first-order transition of a spiking neural network. *Physical review. E*, 97(6), 2018.

[7] A. Ororbia. Spiking Neural Predictive Coding for Continual Learning from Data Streams. *ArXiv*, abs/1908.08655, 2019.

[8] A. van Meegen and S. J. van Albada. A Microscopic Theory of Intrinsic Timescales in Spiking Neural Networks. *ArXiv*, abs/1909.01908, 2019.

[9] T. Masquelier, R. Guyonneau, and S. J. Thorpe. Leaky Integrate-and-Fire (LIF) neuron. `https://plos.figshare.com/articles/figure/_Leaky_Integrate_and_Fire_LIF_neuron_/609821/1`, 2015.

[10] M. Mitchell. Genetic Algorithms: An Overview. *Complexity*, 1(1):31–39, 1995.

[11] H. Mallot. *Computational Neuroscience: A First Course*. Springer International Publishing, 2013.

[12] E. Bonabeau. Social Insect Colonies as Complex Adaptive Systems. *Ecosystems*, 1(5):437–443, 1998.

[13] L. E. Quevillon, E. M. Hanks, S. Bansal, and D. P. Hughes. Social, spatial and temporal organization in a complex insect society. *Scientific Reports*, 5(1), 2015.

[14] S. Katoch, S. S. Chauhan, and V. Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, 2021.

[15] U. Wilensky. NetLogo. `http://ccl.northwestern.edu/netlogo/`, 1999. Accessed: 2021-09-10.

[16] A. Subramoney, S. Diaz-Pier, A. Rao, F. Scherr, D. Salaj, T. Bohnstingl, J. Jordan, N. Kopp, D. Hackhofer, and S. Stekovic. IGITUGraz/L2L: v1.0.0-beta. `https://doi.org/10.5281/zenodo.2590760`, 2019. Accessed: 2021-09-10.

[17] F. Stonedahl and U. Wilensky. BehaviorSearch. `http://behaviorsearch.org`, 2010. Accessed: 2021-09-10.

[18] T. S. Clawson, S. Ferrari, S. B. Fuller, and R. J. Wood. Spiking Neural Network (SNN) Control of a Flapping Insect-scale Robot. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 3381–3388. IEEE, 2016.

[19] G. Tang and K. P. Michmizos. Gridbot: An autonomous robot controlled by a Spiking Neural Network mimicking the brain's navigational system. In *Proceedings of the International Conference on Neuromorphic Systems*, pages 1–8. Association for Computing Machinery, 2018.

[20] Z. Bing, C. Meschede, G. Chen, A. Knoll, and K. Huang. Indirect and Direct Training of Spiking Neural Networks for End-to-End Control of a Lane-Keeping Vehicle. *Neural networks : the official journal of the International Neural Network Society*, 121:21–36, 2020.

[21] S. Chevallier, H. Paugam-Moisy, and M. Sebag. SpikeAnts, a spiking neuron network modelling the emergence of organization in a complex system. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pages 379–387. Curran Associates, Inc., 2010.

[22] E. Eskandari, A. Ahmadi, S. Gomar, M. Ahmadi, and M. Saif. Evolving Spiking Neural Networks of artificial creatures using Genetic Algorithm. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 411–418. IEEE, 2016.

[23] O. Zahra, S. Tolu, and D. Navarro-Alarcon. Differential Mapping Spiking Neural Network for Sensor-Based Robot Control. *Bioinspiration & Biomimetics*, 16(3), 2021.

[24] S. A. Lobov, A. N. Mikhaylov, M. Shamshin, V. A. Makarov, and V. B. Kazantsev. Spatial Properties of STDP in a Self-Learning Spiking Neural Network Enable Controlling a Mobile Robot. *Frontiers in Neuroscience*, 14, 2020.

[25] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll. A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks. *Frontiers in Neurorobotics*, 12, 2018.

[26] H. Asgari, B. M. Maybodi, R. Kreiser, and Y. Sandamirskaya. Digital Multiplier-Less Spiking Neural Network Architecture of Reinforcement Learning in a Context-Dependent Task. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):498–511, 2020.

[27] A. S. Lele, Y. Fang, J. Ting, and A. Raychowdhury. Learning to Walk: Spike Based Reinforcement Learning for Hexapod Robot Central Pattern Generation. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 208–212. IEEE, 2020.

[28] J. Pu, V. P. Nambiar, A. T. Do, and W. L. Goh. Block-Based Spiking Neural Network Hardware with Deme Genetic Algorithm. In *2019 IEEE*

International Symposium on Circuits and Systems (ISCAS), pages 1–5. IEEE, 2019.

[29] I. Ezenwe, A. Joshi, and K. Wong-Lin. Genetic Algorithmic Parameter Optimisation of a Recurrent Spiking Neural Network Model. In *2020 31st Irish Signals and Systems Conference (ISSC)*, pages 1–6. IEEE, 2020.

[30] H. Sasaki and N. Kubota. Distributed Behavior Learning of Multiple Mobile Robots based on Spiking Neural Network and Steady-State Genetic Algorithm. In *2009 IEEE Workshop on Robotic Intelligence in Informationally Structured Space*, pages 73–78. IEEE, 2009.

[31] S. H. Mousavi-Avval, S. Rafiee, M. Sharifi, S. Hosseinpour, B. Notarnicola, G. Tassielli, and P. A. Renzulli. Application of multi-objective genetic algorithms for optimization of energy, economics and environmental life cycle assessment in oilseed production. *Journal of Cleaner Production*, 140(2):804–815, 2017.

[32] A. C. Brooks. Genetic Algorithms and Public Economics. *Journal of Public Economic Theory*, 2(4):493–513, 2000.

[33] E. Rodrigues, L. Rodrigues, L. S. N. Oliveira, A. Conci, and P. Liatsis. Automated Recognition of the Pericardium Contour on Processed CT Images Using Genetic Algorithms. *Computers in Biology and Medicine*, 87:38–45, 2017.

[34] N. Bochud, Q. Vallet, Y. Bala, H. Follet, J. Minonzio, and P. Laugier. Genetic algorithms-based inversion of multimode guided waves for cortical bone characterization. *Physics in medicine and biology*, 61(19):6953–6974, 2016.

[35] N. Brill and D. Tyler. Optimizing Nerve Cuff Stimulation of Targeted Regions Through Use of Genetic Algorithms. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5811–5814. IEEE, 2011.

[36] M. M. Ahmed and Y. M. Wazery. Genetic Algorithms for Discovering Community Cores Hidden within Multidimensional Social Networks. In *2019 International Conference on Innovative Trends in Computer Engineering (ITCE)*, pages 142–148. IEEE, 2019.

[37] D. Bucur and G. Iacca. Influence Maximization in Social Networks with Genetic Algorithms. In *Applications of Evolutionary Computation*, pages 379–392. Springer International Publishing, 2016.

[38] A. Cachón and R. Vázquez. Tuning the parameters of an integrate and fire neuron via a genetic algorithm for solving pattern recognition problems. *Neurocomputing*, 148:187–197, 2015.

[39] N. G. Pavlidis, D. K. Tasoulis, V. P. Plagianakos, G. Nikiforidis, and M. N. Vrahatis. Spiking Neural Network Training Using Evolutionary Algorithms. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, pages 2190–2194. IEEE, 2005.

[40] F. C. Su and W. L. Wu. Design and testing of a genetic algorithm neural network in the assessment of gait patterns. *Medical engineering & physics*, 22(1):67–74, 2000.

[41] S. Kamoi, R. Iwai, H. Kinjo, and T. Yamamoto. Pulse Pattern Training of Spiking Neural Networks Using Improved Genetic Algorithm. In *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics and Automation for the New Millennium (Cat. No.03EX694)*, pages 977–981. IEEE, 2003.

[42] M. S. Farahani and S. H. R. Hajiagha. Forecasting stock price using integrated artificial neural network and metaheuristic algorithms compared to time series models. *Soft Computing*, pages 1–31, 2021.

[43] J. Piri, B. Pirzadeh, B. Keshtegar, and M. Givehchi. Reliability analysis of pumping station for sewage network using hybrid neural networks -

genetic algorithm and method of moment. *Process Safety and Environmental Protection*, 145:39–51, 2021.

[44] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath. Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach. *Information*, 10(12), 2019.

[45] M. Sipper, W. Fu, K. Ahuja, and J. H. Moore. Investigating the parameter space of evolutionary algorithms. *BioData Mining*, 11(1), 2018.

[46] C. Darwin. *On the Origin of Species*. John Murray, London, 1859.

[47] S. N. Deepa S. N. Sivanandam. *Introduction to Genetic Algorithms*. Springer International Publishing, 2008.

[48] O. Kramer. *Genetic Algorithm Essentials*. Springer International Publishing, 2017.

[49] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[50] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[51] S. A. Forhad, S. Hossain, M. O. Rahman, M. M. Rahaman, M. Haque, and M. K. H. Patwary. An improved fitness function for automated cryptanalysis using genetic algorithm. *Indonesian Journal of Electrical Engineering and Computer Science*, 13(2):643–648, 2019.

[52] S. Renjith and C. Anjali. Fitness Function in Genetic Algorithm based Information Filtering - A Survey. 2016.

[53] J. Khalid. Selection Methods for Genetic Algorithms. *International Journal of Emerging Sciences*, 3(4):333–344, 2013.

[54] S. Nitasha and T. Kumar. Study of Various Crossover Operators in Genetic Algorithms. *International Journal of Computer Science and Information Technologies*, 5(6):7235–7238, 2014.

[55] A. Jenkins, V. Gupta, A. Myrick, and M. Lenoir. Variations of Genetic Algorithms. *ArXiv*, abs/1911.00490, 2019.

[56] A. B. A. Hassanat and E. Alkafaween. On Enhancing Genetic Algorithms Using New Crossovers. *International Journal of Computer Applications in Technology*, 55(3):202–212, 2017.

[57] S. L. Siew Mooi, A. B. Md Sultan, M. Sulaiman, A. Mustapha, and K. Y. Leong. Crossover and Mutation Operators of Genetic Algorithms. *International Journal of Machine Learning and Computing*, 7:9–12, 2017.

[58] C. Jimenez-Romero, D. Sousa-Rodrigues, J. H. Johnson, and V. Ramos. A Model for Foraging Ants, Controlled by Spiking Neural Networks and Double Pheromones. *ArXiv*, abs/1507.08467, 2015.

[59] C. Jimenez-Romero and J. H. Johnson. SpikingLab: modelling agents controlled by Spiking Neural Networks in Netlogo. *Neural Computing & Applications*, 28:755–764, 2016.

[60] M. Ferreira, J. Bagarić, J. Lanza-Gutierrez, S. Mendes, J. Pereira, and J. A. Gomez-Pulido. On the Use of Perfect Sequences and Genetic Algorithms for Estimating the Indoor Location of Wireless Sensor. *International Journal of Distributed Sensor Networks*, 11:1–12, 2015.

[61] F. Fortin, F. De Rainville, M. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, 2012.

[62] M. Abido. Multiobjective Evolutionary Algorithms for Electric Power Dispatch Problem. In *IEEE Transactions on Evolutionary Computation - TEC*, pages 47–82. IEEE, 2009.

[63] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

Appendices

Parameter	Value
Spiking Neurons	
Resting potential	-65
Firing threshold	-55
Decay rate	0.08
Refractory potential	-75
Refractory duration	3
Spike per stimulus	1
Simulation	
View distance	4
Steps forward	0.4
Rotation degrees	4
Fitness	
Reward	1
Punishment	-1.5

Table 5: Fixed model parameters

Population-Size: 32 Tournament-Size: 3 Mutation-Rate: 0.2 Evaluation Limit: 1000 (=32 Generations)													
				Seed									
Crossover-Rate	Mean	Median	Standard Deviation	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
0.0	13.6	15.0	8.8	27	19	20	15	3	1	15	0	15	21
0.1	15	18.0	11.1	15	25	24	27	5	0	29	1	3	21
0.2	16.1	18.5	8.6	15	20	25	15	21	0	17	1	27	20
0.3	16.4	17.5	10.1	0	36	23	15	14	0	21	20	16	19
0.4	12.5	17.5	10.6	0	30	21	18	18	0	20	1	0	17
0.5	15.0	15.0	11.0	16	35	28	16	2	10	14	1	3	25
0.6	21.8	19.5	9.2	16	31	22	17	16	28	30	4	17	37
0.7	13.2	15.0	8.2	0	20	17	13	25	11	18	7	0	21
0.8	17.9	19.0	6.6	12	23	21	14	11	6	17	24	24	27
0.9	18.5	21.5	9.0	1	32	22	14	21	6	16	28	22	23
1.0	13.6	13.0	10.8	1	34	21	2	18	7	17	1	9	26

Figure 22: Fitness scores of crossover-rate exploration 1

Fixed parameters: population-size 32, tournament-size 3, mutation-rate: 0.2

Population-Size: 32 Tournament-Size: 3 Mutation-Rate: 0.4 Evaluation Limit: 1000 (=32 Generations)													
				Seed									
Crossover-Rate	Mean	Median	Standard Deviation	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
0.0	16.4	16.0	6.7	16	22	16	29	16	3	19	8	16	19
0.1	17.0	17.0	10.6	6	16	20	33	21	7	14	0	18	35
0.2	13.5	15.0	9.5	1	21	24	17	16	7	14	4	1	30
0.3	13.4	13.0	6.6	6	6	23	16	21	9	22	5	12	14
0.4	18.2	17.0	7.8	27	22	17	16	16	10	17	2	28	27
0.5	14.0	15.0	6.0	6	20	16	9	18	6	14	8	21	22
0.6	18.5	18.0	9.3	17	19	41	8	14	22	21	5	15	23
0.7	18.4	19.0	5.5	20	22	20	7	18	21	18	29	13	16
0.8	17.5	17.5	8.8	28	33	19	16	10	21	22	1	14	11
0.9	16.0	17.0	9.7	10	24	17	29	6	1	17	30	5	21
1.0	17.1	16.5	10.5	10	18	43	15	20	16	17	0	10	22

Figure 23: Fitness scores of crossover-rate exploration 2

Fixed parameters: population-size 32, tournament-size 3, mutation-rate: 0.4

Population-Size: 32 Tournament-Size: 3 Mutation-Rate: 0.5 Evaluation Limit: 1000 (=32 Generations)													
				Seed									
Crossover-Rate	Mean	Median	Standard Deviation	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
0.0	15.3	16.0	7.4	4	16	25	17	17	4	16	10	16	28
0.1	14	15.5	8.1	18	16	21	21	1	15	14	1	7	26
0.2	17.6	17.0	9.1	22	38	23	17	14	3	17	16	6	20
0.3	15.1	15.5	8.2	4	31	16	3	16	15	24	15	8	19
0.4	15.6	16.5	5.2	20	14	21	17	7	14	16	6	19	22
0.5	15.7	17.0	6.6	17	17	23	3	21	15	17	4	22	18
0.6	20.1	16.0	10.8	8	23	27	16	13	30	16	8	15	45
0.7	16.7	18.0	4.8	14	5	21	20	15	21	14	16	20	21
0.8	14.2	15.5	8.2	6	28	22	16	16	0	14	4	15	21
0.9	12.9	14.0	6.2	3	14	20	5	12	16	23	6	14	16
1.0	15.1	16.0	7.4	6	17	27	14	17	6	15	4	24	21

Figure 24: Fitness scores of crossover-rate exploration 3

Fixed parameters: population-size 32, tournament-size 3, mutation-rate: 0.5

Population-Size: 32 Tournament-Size: 3 Crossover-Rate: 0.6 Evaluation Limit: 1000 (=32 Generations)													
				Seed									
Mutation-Rate	Mean	Median	Standard Deviation	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
0.0	5.0	4.0	5.5	0	6	6	3	5	0	14	0	0	16
0.1	13.1	17.0	10.3	0	16	24	27	18	0	21	4	0	21
0.2	21.8	19.5	9.2	16	31	22	17	16	28	30	4	17	37
0.3	17.5	17.0	7.7	20	28	18	15	16	1	16	10	24	27
0.4	18.3	17.0	9.3	17	17	41	8	14	22	21	5	15	23
0.5	20.1	16.0	10.8	8	23	27	16	13	30	16	8	15	45
0.6	12.9	14.0	7.1	14	9	17	16	19	6	14	2	5	27
0.7	16.3	16.5	5.7	20	20	17	15	6	16	25	16	7	21
0.8	17.1	16.5	6.3	15	31	16	19	6	17	20	10	16	21
0.9	12.7	14.5	6.6	15	18	17	10	3	3	14	24	6	17
1.0	13.5	14.0	4.2	19	15	9	9	6	13	16	13	16	19

Figure 25: Fitness scores of mutation-rate exploration 1

Fixed parameters: population-size 32, tournament-size 3, crossover-rate: 0.6

Population-Size: 32 Tournament-Size: 3 Crossover-Rate: 0.7 Evaluation Limit: 1000 (=32 Generations)													
				Seed									
Mutation-Rate	Mean	Median	Standard Deviation	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
0.0	5.7	4.5	5.8	0	17	7	3	6	0	14	0	1	9
0.1	12.4	13.0	10.0	33	6	19	16	10	0	17	0	3	20
0.2	13.2	15.0	8.2	0	20	17	13	25	11	18	7	0	21
0.3	14.8	17.5	9.0	0	16	19	23	13	1	20	7	20	29
0.4	18.4	19.0	5.5	20	22	20	7	18	21	18	29	13	16
0.5	16.6	17.5	4.8	14	5	21	20	15	21	14	15	20	21
0.6	16.3	17.0	5.5	20	26	20	19	16	10	14	5	15	18
0.7	12.1	14.0	6.8	15	15	20	1	7	5	14	6	14	24
0.8	12.6	14.0	4.9	13	15	15	16	6	1	14	14	14	18
0.9	14.6	14.0	7.9	14	13	6	27	6	3	15	14	24	24
1.0	12.3	15.0	6.0	7	21	17	10	1	16	15	15	5	16

Figure 26: Fitness scores of mutation-rate exploration 2

Fixed parameters: population-size 32, tournament-size 3, crossover-rate: 0.7

Population-Size: 32 Tournament-Size: 3 Crossover-Rate: 0.8 Evaluation Limit: 1000 (=32 Generations)													
				Seed									
Mutation-Rate	Mean	Median	Standard Deviation	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
0.0	5.5	3.0	7.2	0	6	2	0	4	0	14	0	6	23
0.1	15.5	17.5	9.9	0	6	21	18	15	30	17	0	27	21
0.2	16.0	15.5	7.3	12	23	21	14	11	6	17	24	5	27
0.3	11.6	12.0	7.6	16	8	21	17	4	3	22	1	6	18
0.4	17.5	17.5	8.8	28	33	19	16	10	21	22	1	14	11
0.5	14.2	15.5	8.2	6	28	22	16	16	0	14	4	15	21
0.6	14.3	14.5	7.8	12	19	20	14	6	3	15	17	6	31
0.7	13.3	14.5	6.1	2	23	20	16	8	15	14	10	7	18
0.8	11.8	12.0	7.1	4	13	17	11	22	0	14	7	7	23
0.9	13.2	14.5	7.7	15	24	22	20	13	2	14	3	3	16
1.0	14.8	14.5	6.6	15	13	6	6	27	9	14	21	15	22

Figure 27: Fitness scores of mutation-rate exploration 3

Fixed parameters: population-size 32, tournament-size 3, crossover-rate: 0.8

Mutation-Rate: 0.4 Tournament-Size: 3 Crossover-Rate: 0.6 Evaluation Limit: 1000 (=32 Generations)													
				Seed									
Population-Size	Average	Median	Ratio	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
16	10.1	7.0	0.188	8	18	5	3	24	6	27	0	8	2
32	18.5	18.0	0.094	17	19	41	8	14	22	2	5	15	23
48	17.6	17.5	0.063	22	8	17	15	11	23	14	29	19	18
64	15.7	17.0	0.047	20	6	24	22	4	6	21	5	14	35
80	12	11.5	0.038	19	10	6	12	15	11	14	9	7	17

Figure 28: Fitness scores of population-size exploration 1

Fixed parameters: tournament-size 3, crossover-rate: 0.6, mutation-rate: 0.4

Mutation-Rate: 0.4 Tournament-Size: 4 Crossover-Rate: 0.6 Evaluation Limit: 1000 (=32 Generations)													
				Seed									
Population-Size	Average	Median	Ratio	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
16	16.2	17.0	0.250	14	2	0	21	22	34	17	17	30	5
32	21	21.0	0.125	8	28	27	20	30	34	21	5	16	21
48	16.3	18.5	0.083	17	20	21	16	16	22	15	4	22	20
64	20.6	19.5	0.063	20	18	21	18	18	15	28	17	32	21
80	16.7	16.0	0.050	18	16	20	16	16	17	17	16	9	25

Figure 29: Fitness scores of population-size exploration 2

Fixed parameters: tournament-size 4, crossover-rate: 0.6, mutation-rate: 0.4

Mutation-Rate: 0.4 Tournament-Size: 5 Crossover-Rate: 0.6 Evaluation Limit: 1000 (=32 Generations)													
				Seed									
Population-Size	Average	Median	Ratio	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
16	18.7	20.5	0.313	27	21	20	4	23	19	18	3	23	29
32	18.3	21.0	0.156	1	17	21	31	26	6	16	22	22	21
48	23.2	21.5	0.104	23	35	17	20	31	16	31	25	19	15
64	21.6	21.5	0.078	23	35	30	28	6	28	14	18	14	20
80	17.6	17.5	0.063	23	23	27	15	4	9	14	15	20	26

Figure 30: Fitness scores of population-size exploration 3

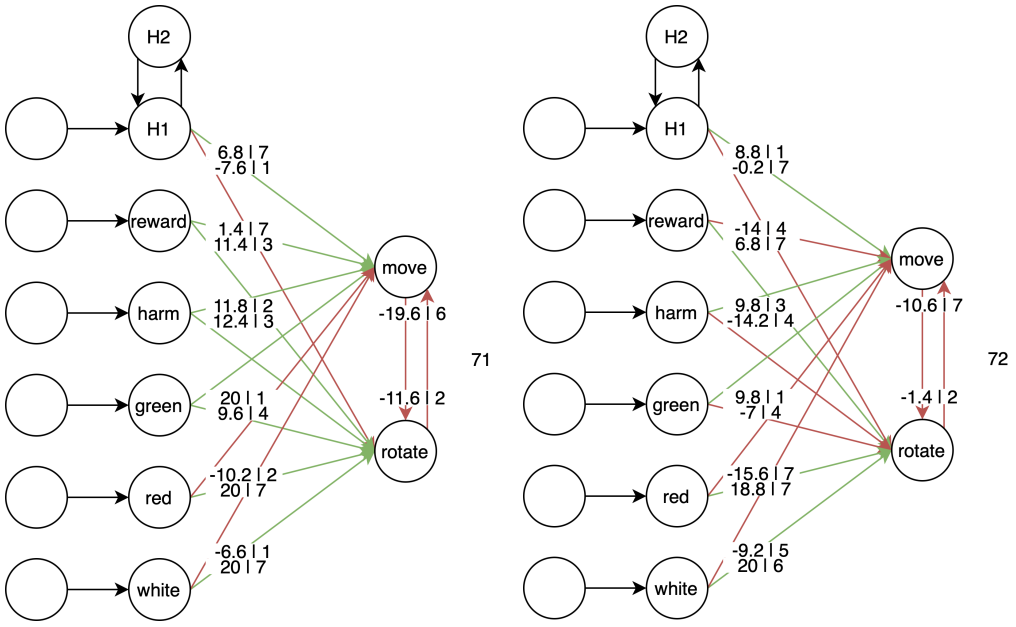
Fixed parameters: tournament-size 5, crossover-rate: 0.6, mutation-rate: 0.4

			Good Settings				Bad Settings			
Simulation Iterations	20000	Static World	84	83	82	80	64	65	67	68
		Dynamic World	84	75	81	78	77	79	66	77
	10000	Static World	52	48	52	47	50	45	45	42
		Dynamic World	54	56	56	48	54	56	46	52
	5000	Static World	28	27	30	28	29	28	24	29
		Dynamic World	35	35	37	40	39	32	31	34

Figure 31: Fitness scores of simulation iteration exploration

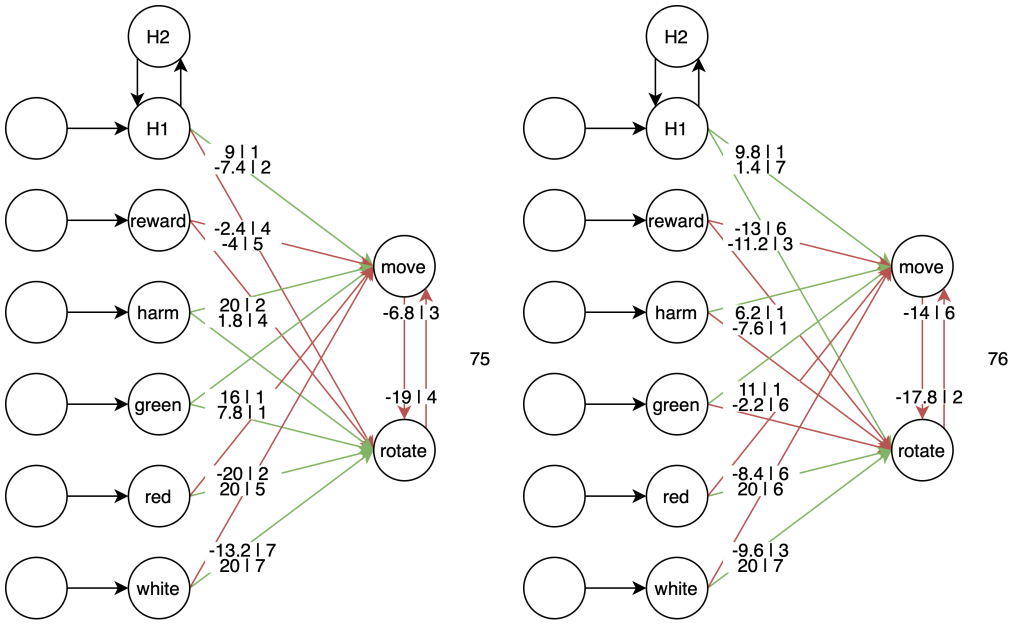
		Static World				Dynamic World			
Number of Second-Layer Neurons	0	71	72	75	76	86.5	83	87	82.5
	1	18	17	19	18	27	22	20	26
	2	72	75	76	66	78	80	89	83
	3	68	77	70	72	82	78	78	76
	4	70	75	74	73	74	78	80	74
	5	72	75	77	70	81	88	85	86
	6	74	73	73	74	82	79	77	85
	7	71	76	75	74	86	80	84	77.5
	8	77	70	72	74	83	73	71	76
	9	72	67	69	77	78	82	78	81
	10	70	68	77	69	81	75	85	62
	11	71	64	71	70	45	74	72	70
	12	68	68	62	76	50	75	48	42
	13	64	67	69	61	67	73	41	73
	14	60	40	67	34	50	43	61	46

Figure 32: Fitness scores of varying number of second-layer neurons



(a) (b)

Figure 33: Networks with 0 hidden-layer neurons



(a) (b)

Figure 34: Networks with 0 hidden-layer neurons

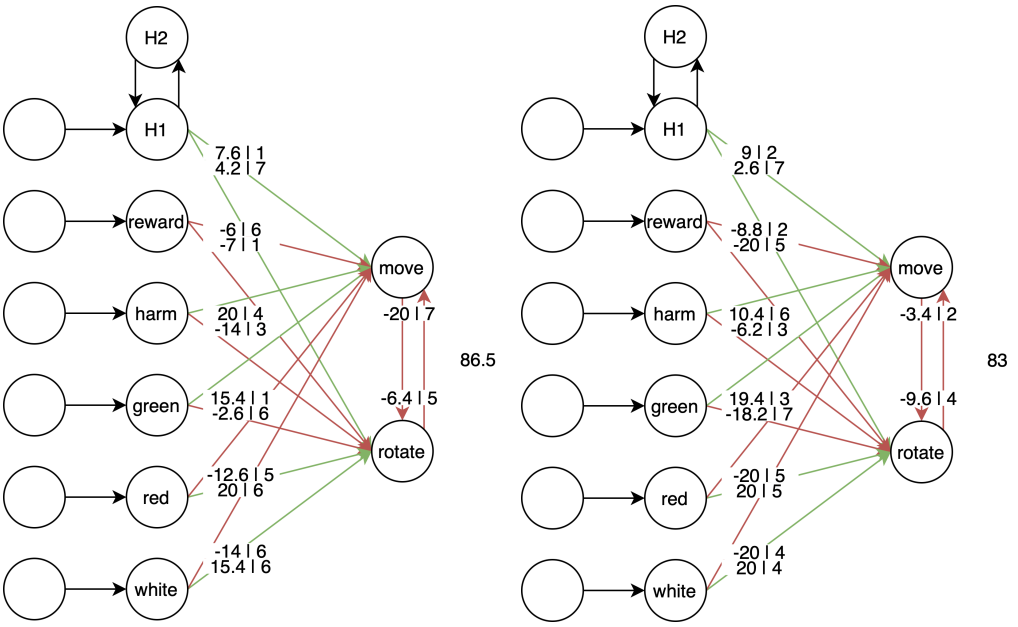


Figure 35: Networks with 0 hidden-layer neurons

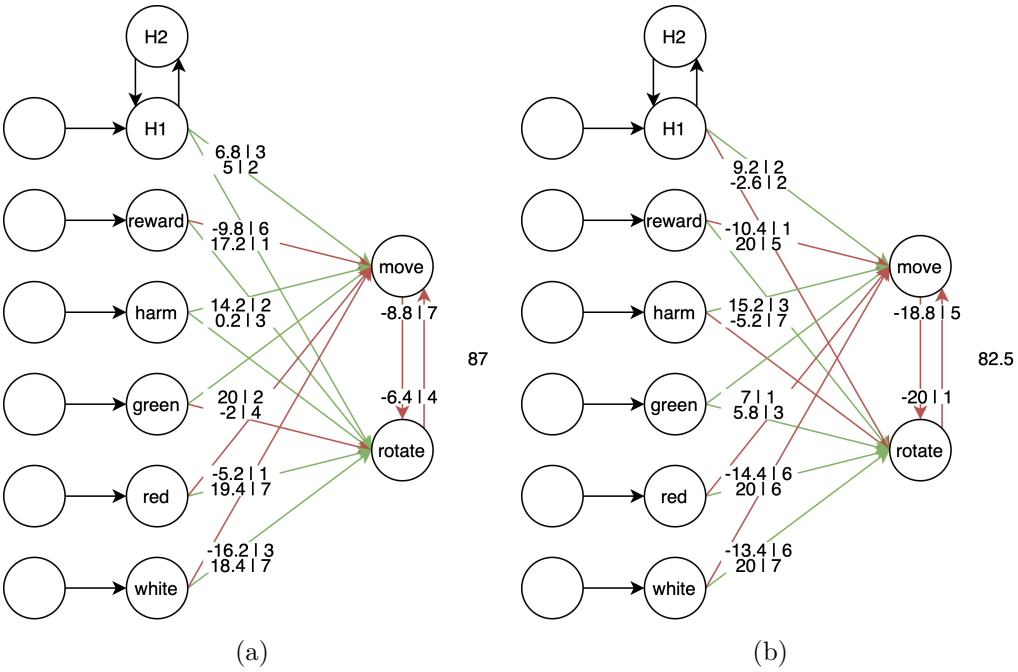
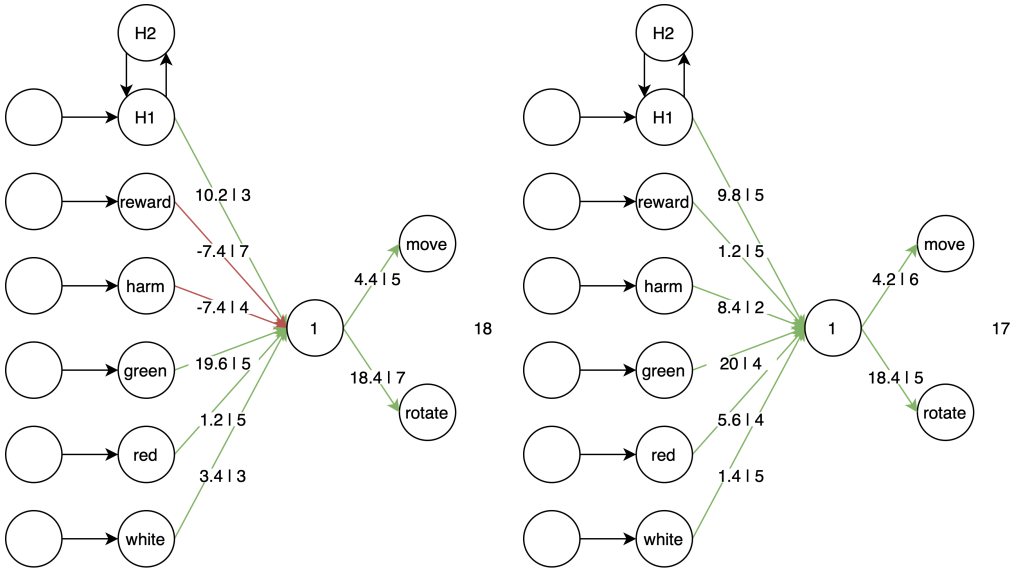
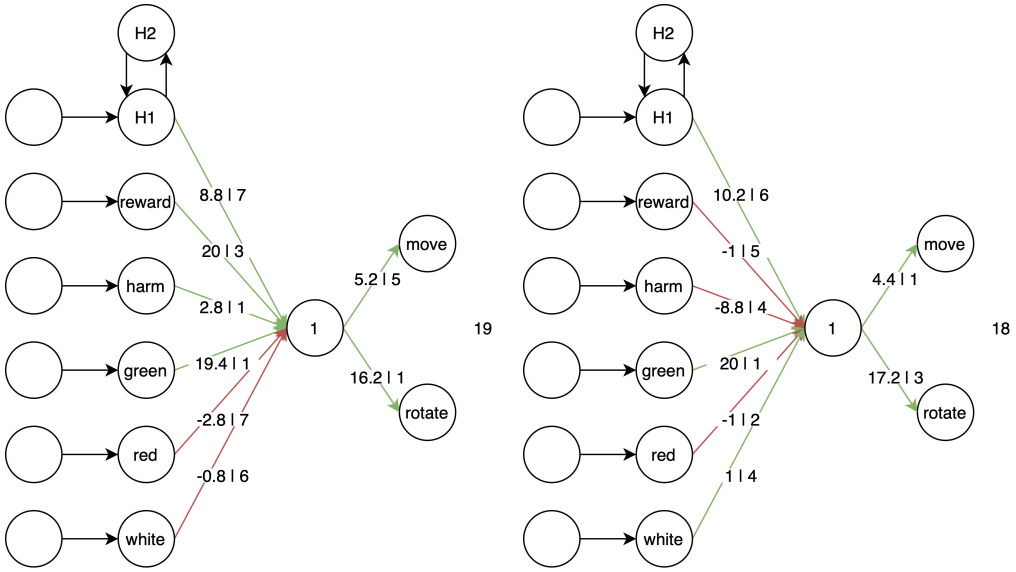


Figure 36: Networks with 0 hidden-layer neurons



(a) (b)

Figure 37: Networks with 1 hidden-layer neuron



(a) (b)

Figure 38: Networks with 1 hidden-layer neuron

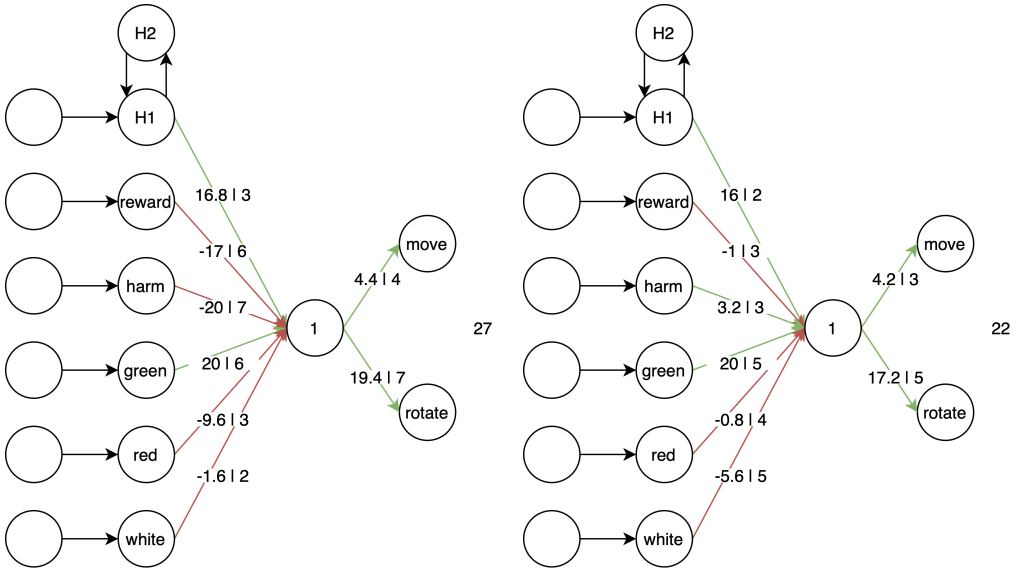


Figure 39: Networks with 1 hidden-layer neuron

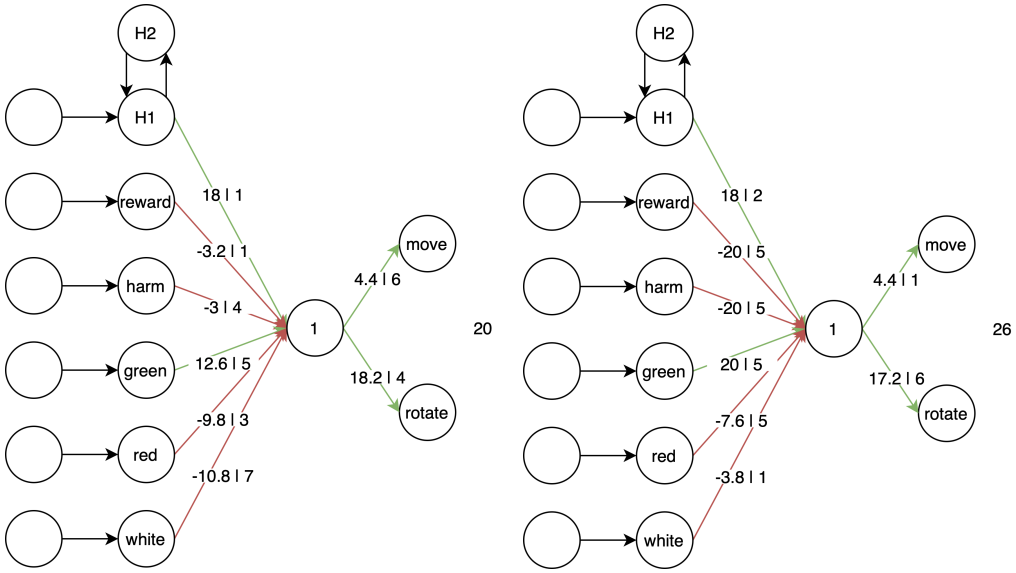


Figure 40: Networks with 1 hidden-layer neuron

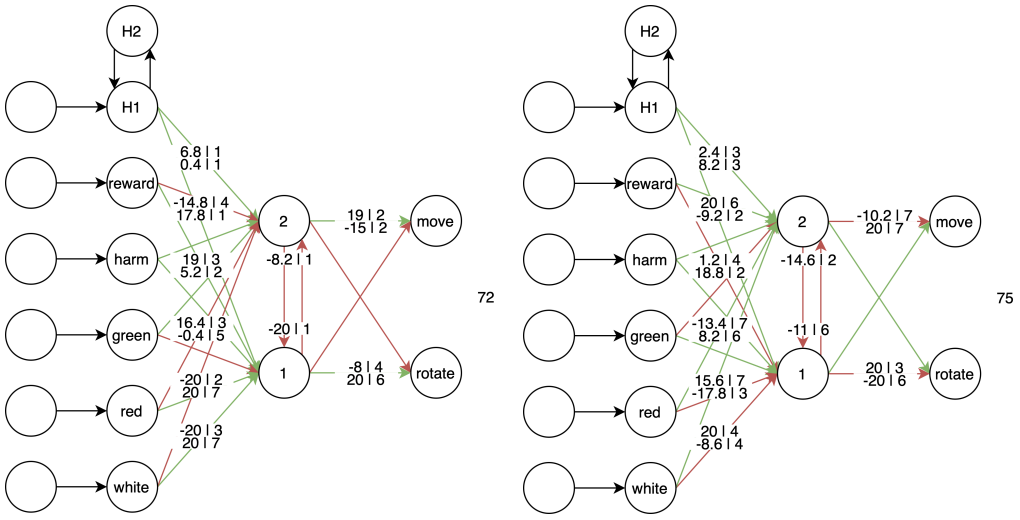


Figure 41: Networks with 2 hidden-layer neurons

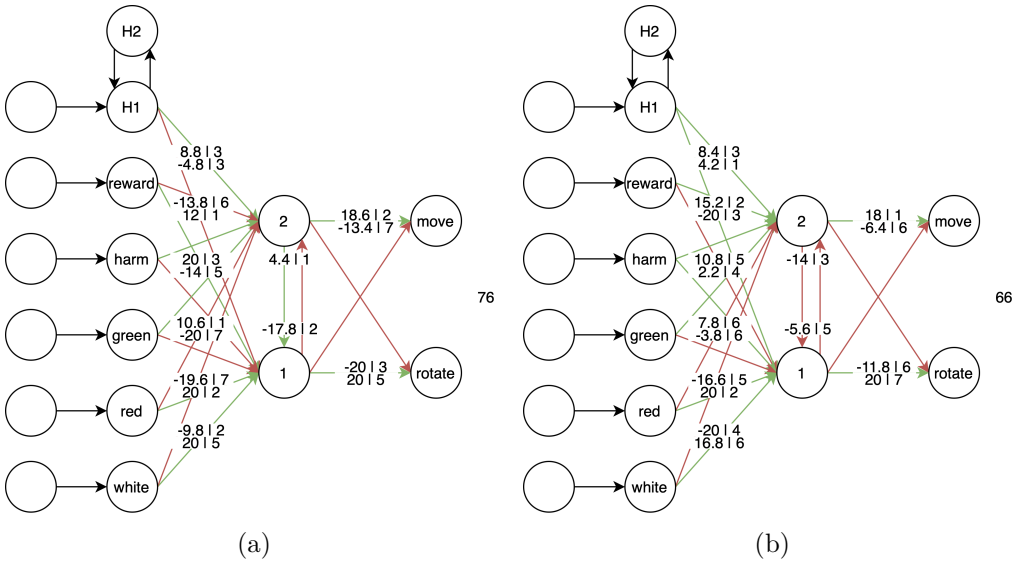


Figure 42: Networks with 2 hidden-layer neurons

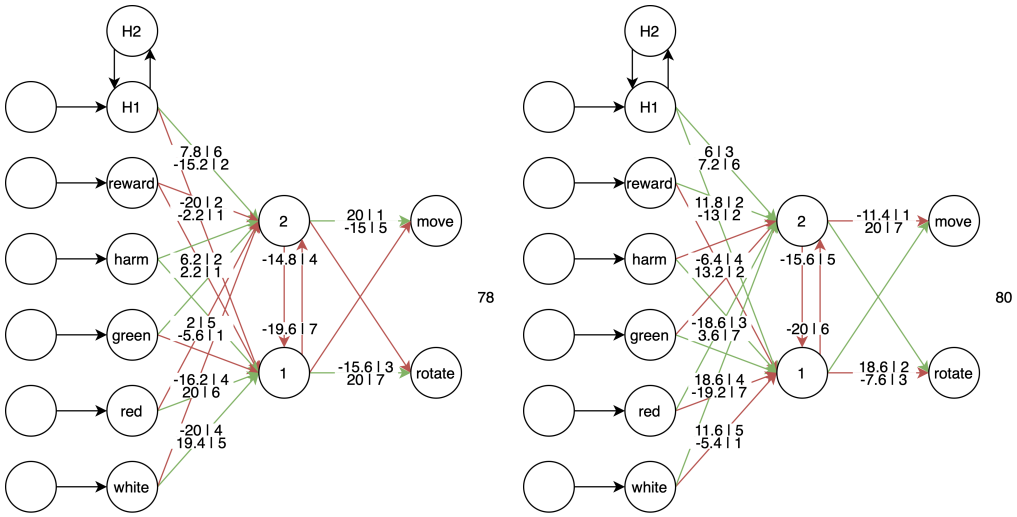


Figure 43: Networks with 2 hidden-layer neurons

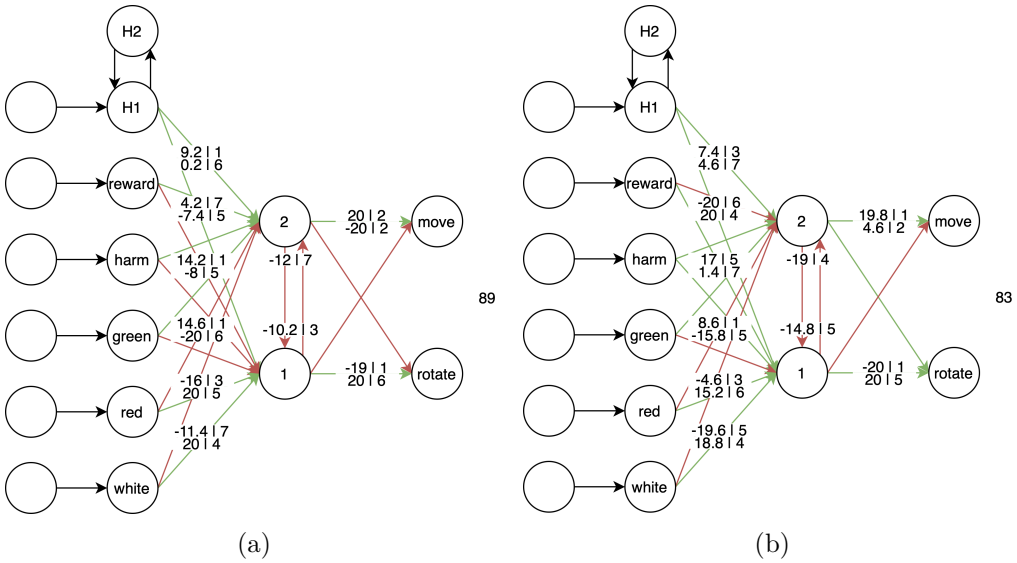
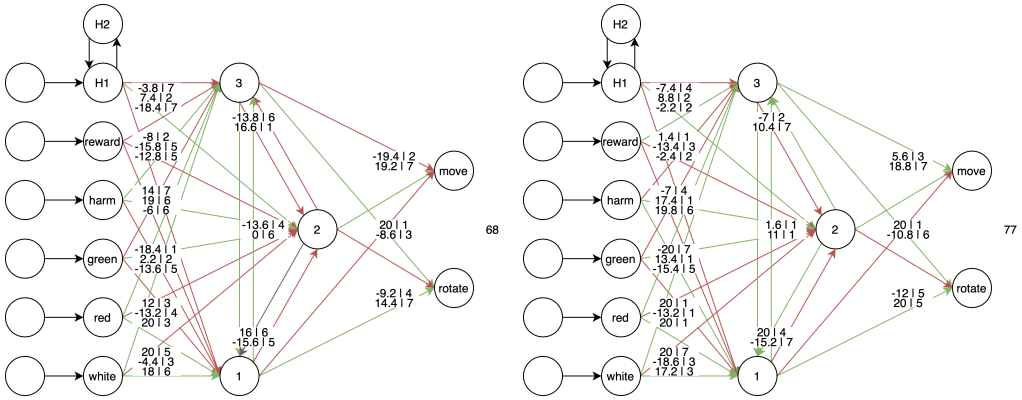
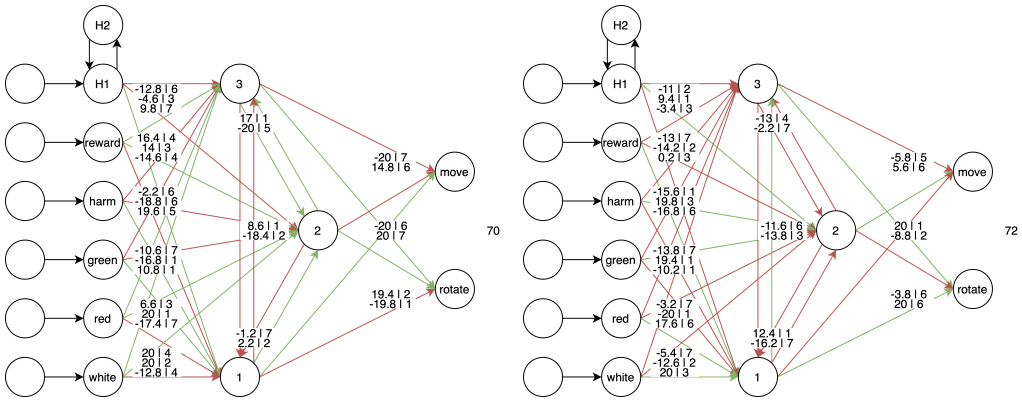


Figure 44: Networks with 2 hidden-layer neurons



(a) (b)

Figure 45: Networks with 3 hidden-layer neurons



(a) (b)

Figure 46: Networks with 3 hidden-layer neurons

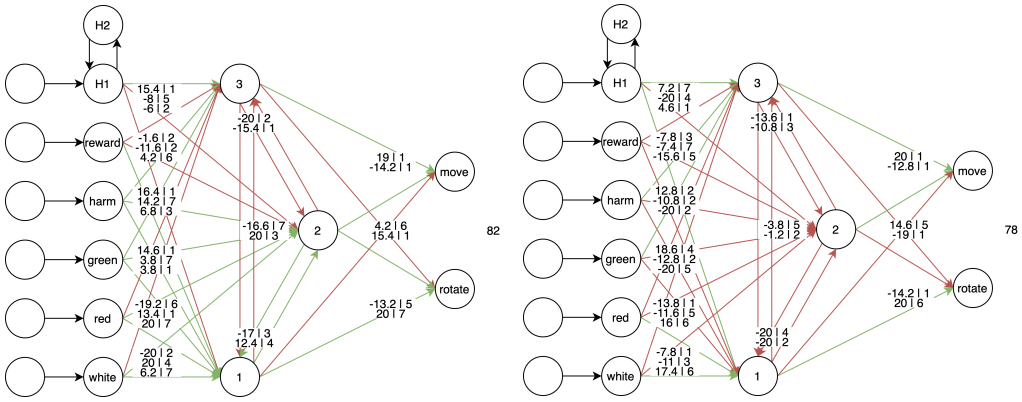


Figure 47: Networks with 3 hidden-layer neurons

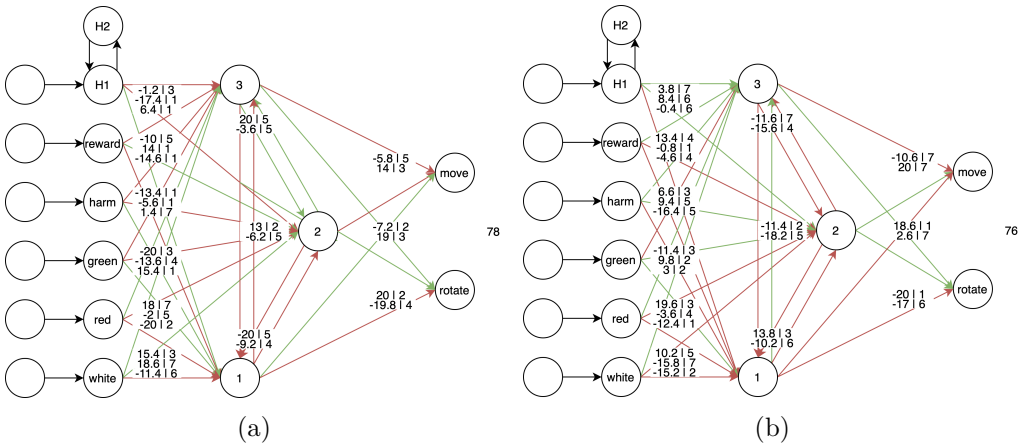


Figure 48: Networks with 3 hidden-layer neurons

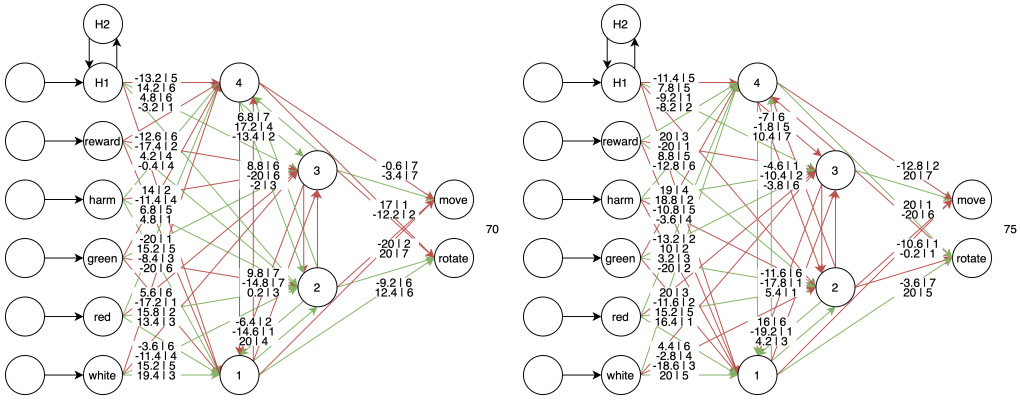


Figure 49: Networks with 4 hidden-layer neurons

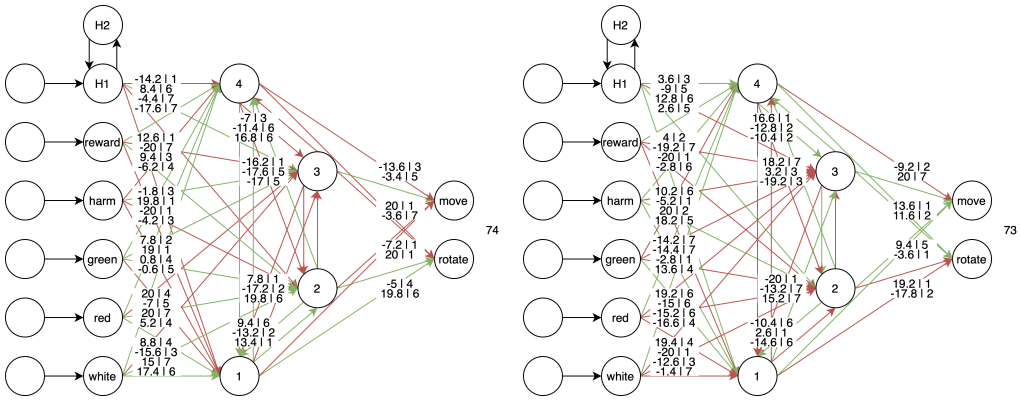
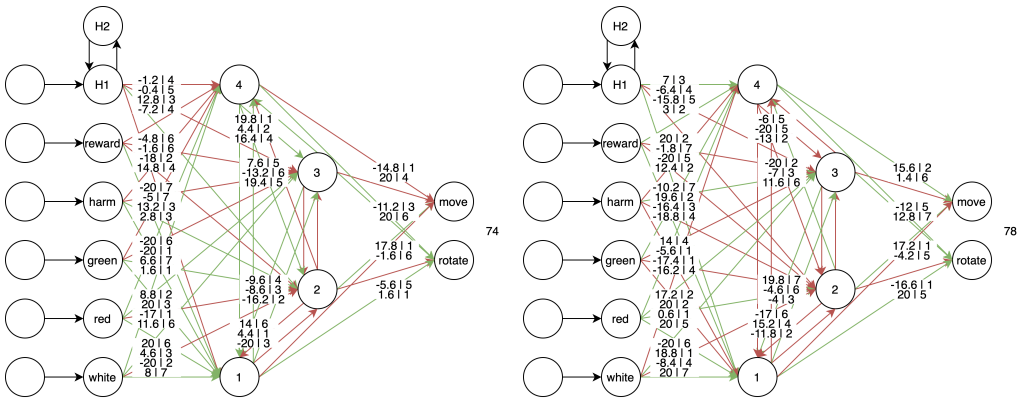
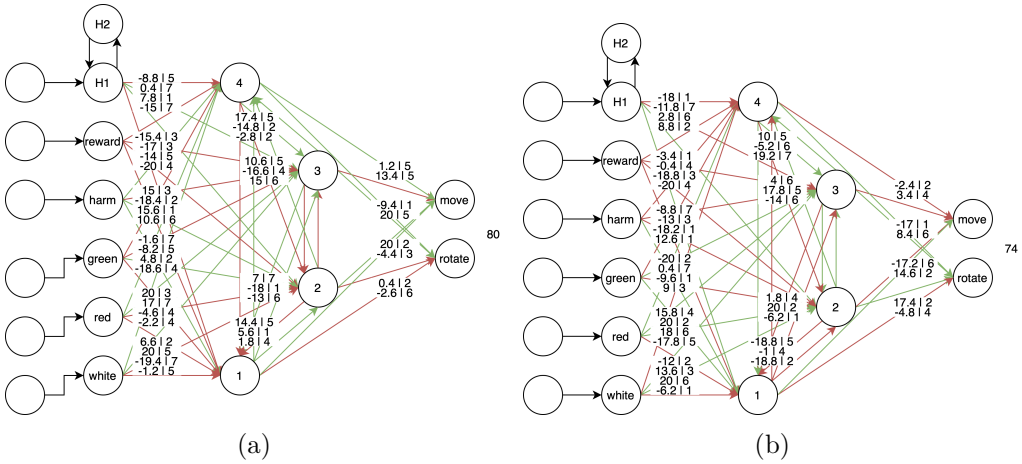


Figure 50: Networks with 4 hidden-layer neurons



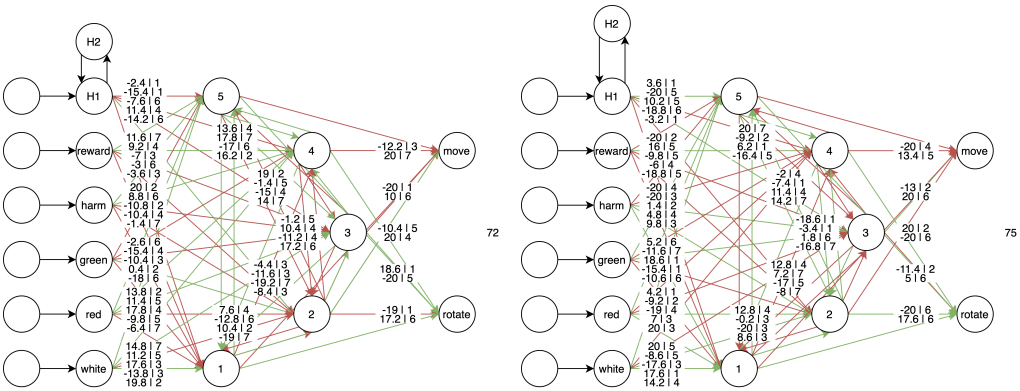
(a) (b)

Figure 51: Networks with 4 hidden-layer neurons



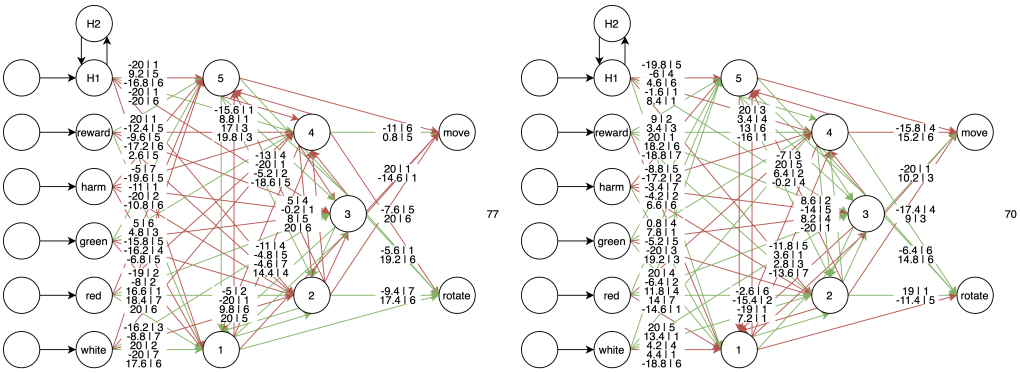
(a) (b)

Figure 52: Networks with 4 hidden-layer neurons



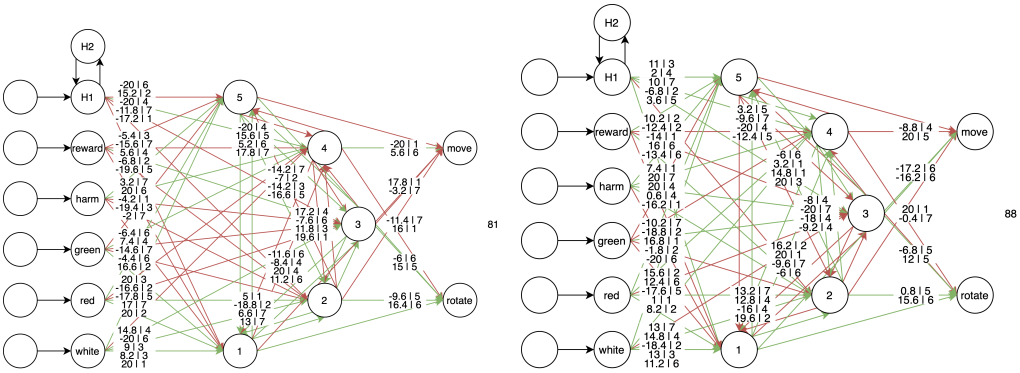
(a) (b)

Figure 53: Networks with 5 hidden-layer neurons



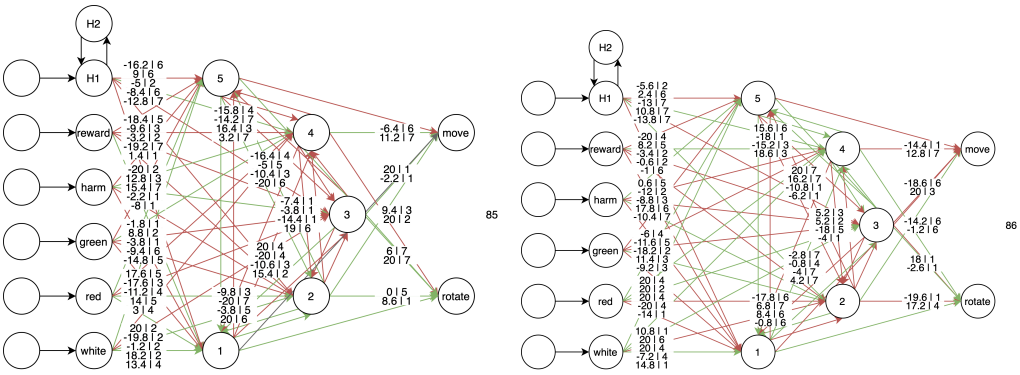
(a) (b)

Figure 54: Networks with 5 hidden-layer neurons



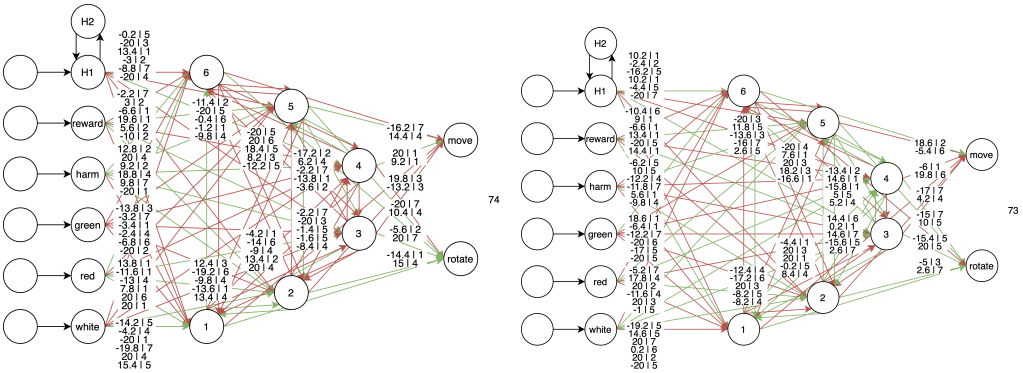
(a) (b)

Figure 55: Networks with 5 hidden-layer neurons



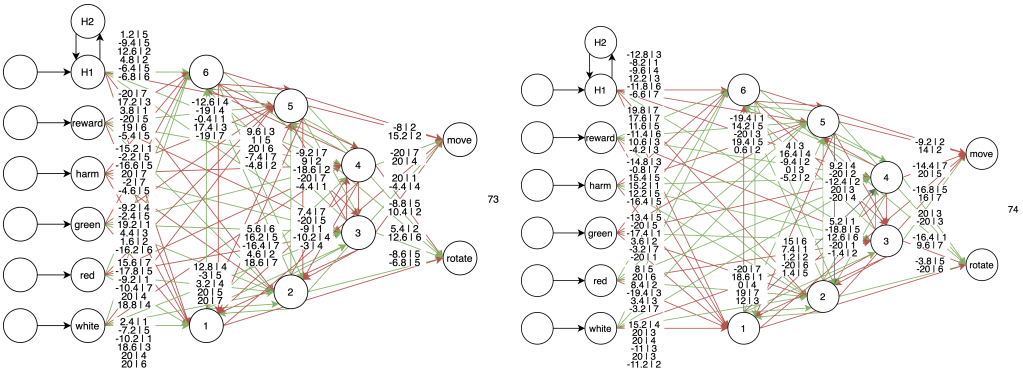
(a) (b)

Figure 56: Networks with 5 hidden-layer neurons



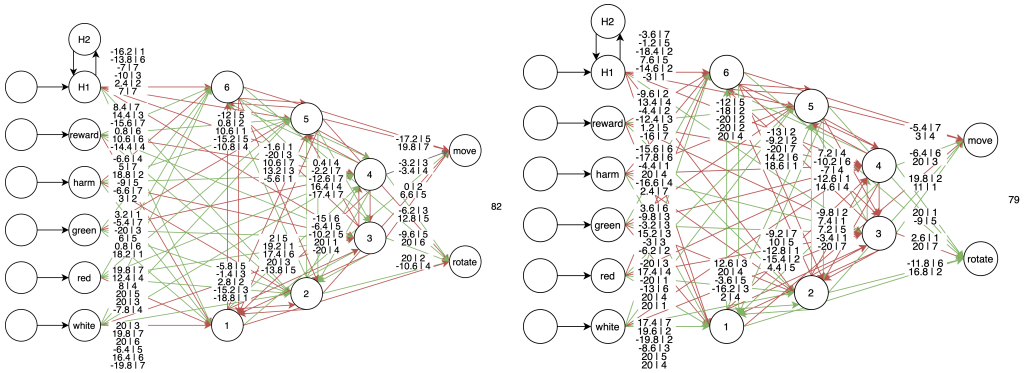
(a) (b)

Figure 57: Networks with 6 hidden-layer neurons



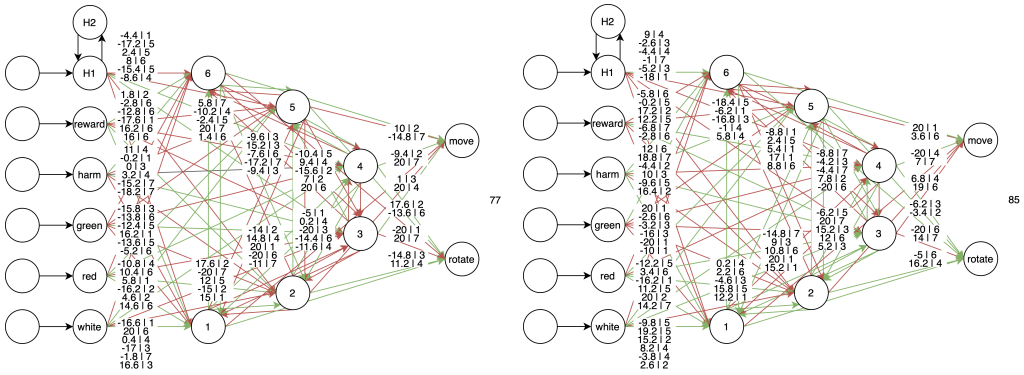
(a) (b)

Figure 58: Networks with 6 hidden-layer neurons



(a) (b)

Figure 59: Networks with 6 hidden-layer neurons



(a) (b)

Figure 60: Networks with 6 hidden-layer neurons