Potsdam University

Faculty of Human Sciences: Linguistics Department

Master's Program *Cognitive Systems*

Master Thesis

Evolving Recurrent Neural Networks for Active Vision

written by

Simon Untergasser

794948

Submission date:

March, 17th, 2020

1. Reviewer:

Prof. Dr. T. Schaub – Potsdam University

2. Reviewer and Supervisor:

Prof. Dr. M. Hild – Beuth University of Applied Sciences Berlin

Abstract

Imagine a learning game, where a human teacher explains something by drawing on a whiteboard and a robot follows the explanation. To realize this scenario, the robot needs a visual processing module which can identify and categorize drawn symbols. One way of implementing such a module is by using an active vision process, inspired by biological systems. Such a process can be realized using recurrent neural networks. In this thesis, such networks which categorize symbols in images are evolved using artificial evolution. The networks move a small window over the image by actively determining the next position of this window. They "scan" the image and identify the depicted symbol. The implementation of the neural networks and of the artificial evolution process is described in detail. Furthermore, two strategies for finding appropriate networks are presented and compared. The results of the evolutionary process are integrated in a framework for the learning scenario, in which the recurrent neural networks are used to detect and categorize 3 symbols.

Kurzfassung

Stellen Sie sich ein Lernspiel vor, bei dem ein menschlicher Lehrer etwas erklärt, indem er auf ein Whiteboard zeichnet und ein Roboter der Erklärung folgt. Um dieses Szenario realisieren zu können, benötigt der Roboter ein visuelles Verarbeitungsmodul, das gezeichnete Symbole identifizieren und kategorisieren kann. Eine Möglichkeit solch ein Modul zu implementieren, besteht darin, einen von biologischen Systemen inspirierten "Active Vision" Prozess zu nutzen. Dieser Prozess kann mit Hilfe rekurrenter neuronaler Netze realisiert werden. In dieser Arbeit werden solche Netze, die Symbole in Bildern kategorisieren, mit Hilfe von künstlicher Evolution entwickelt. Die Netzwerke bewegen ein kleines Fenster über das Bild indem sie aktiv die nächste Position dieses Fensters bestimmen. Sie "scannen" das Bild und bestimmen das gezeigte Symbol. Die Implementierung der neuronalen Netze und des künstlichen Evolutionsprozesses wird ausführlich beschrieben. Außerdem werden zwei Strategien zum Finden geeigneter Netzwerke vorgestellt und verglichen. Die Ergebnisse des Evolutionsprozesses werden in ein Framework für das Lernspiel eingebaut, in dem die rekurrenten neuronalen Netze für die Identifizierung und Kategorisierung von 3 Symbolen verwendet werden.

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

_____

Date / Signature

Contents

1 Introduction

For a humanoid robot to cope with the complexity of our world, especially human environments, it needs a coherent perception of its surrounding. This includes visual stimuli, auditory input and also proprioception (the sense for body posture and self movement). One step towards the perception of the environment consists of enabling the robot to learn from visual stimuli. Consider the setup of a robot in front of a whiteboard, where a human teacher explains something in a *learning scenario*. In order to follow the explanation, the robot needs to perceive the symbols drawn on the whiteboard. With this learning scenario as long term goal in mind, the first step in this direction is the visual categorization of symbols. This thesis focuses on developing a module for processing these visual symbols by using an *active vision* mechanism. Active vision belongs to the broader field of *active perception*. Active perception means, that an agent, perceiving sensory data from its environment, actively manipulates its behavior to increase the information content it receives through its sensors ([Bajcsy, 1988], [Bajcsy et al., 2018]).

When only the visual channel is used, the perception mechanism is called active vision. This process is studied in humans for example in the field of eye movements [Wurtz, 2015]. Figure 1.1 shows an example of the sequential changes in the direction of gaze of human subjects inspecting a painting. The dots represent fixation points and the lines the trajectories of eye movements.

Similar to the changing direction of gaze in humans, in this work, an agent moves a small window of a few pixels over a much larger image and therefore changes its own sensory input. This input in turn influences the agents state and therefore it't motor outputs (which move again the window). This connection between sensor input and motor output is referred to as sensorimotor coupling [Di Paolo et al., 2017] or sensor-motor-sensor triples (SMS triple as in [Dörner, 1999]). According to the sensorimotor coupling theory knowledge is always a sensory stimuli intertwined with active behavior of an agent.

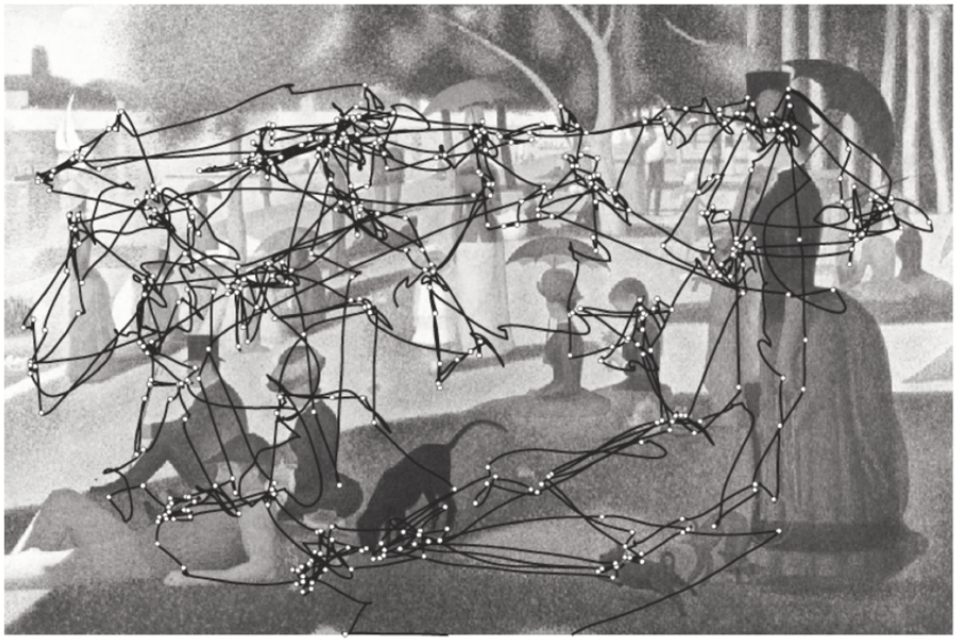

Figure 1.1: A picture of a painting with overlayed trajectories of eye movements as lines and fixation points as white dots. (Taken from [Wurtz, 2015])

In this work, the agent "explores" a given image and determines if there is a particular symbol present and to which category it belongs to. This agent is realized as a *recurrent neural network*. To find such an agent, which is capable of this task, *artificial evolution* is used. Two different approaches are explored and evaluated and finally, a working system for the learning scenario is constructed.

1.1 Structure of this work

In the next section, related work and the literature serving as basis for this thesis is presented. Chapter 2 presents the theoretical foundations including dynamical systems theory, recurrent neural networks and artificial evolution. In chapter 3, the implementation of these networks and the technical details of the artificial evolution is described. Then follows chapter 4, in which the experiments and their main outcomes are presented. This chapter is divided into two sub-parts each describing different approaches, showing the results and discussing them. In chapter 5 the evolved recurrent neural networks are

included into an real world application. Chapter 6 summarizes this work and presents specific ideas for future work and improvements for the presented ideas.

1.2 Related work

The usage of recurrent neural networks as dynamical controllers solving different tasks as described above was demonstrated in a lot of works, from which the most related are presented in this section. Over twenty years ago, Tani used a recurrent neural network and its attractor dynamics to navigate a wheeled robot by using only local sensor information [Tani, 1996]. In later work [Tani and Nolfi, 1999] this framework was extended and it was shown, that the location of the robot (either room A or room B) is represented in the dynamics of the neural network. In more recent works (e.g. [Yamashita and Tani, 2008]), he and colleagues extended the recurrent neural network to multiple timescales, where different parts of the network are updated slower or faster (3 update rates). They showed the usefulness of their approach in experiments with humanoid robots. [Heinrich and Wermter, 2018] used these multiple timescale networks and extended them to incorporate the somatosensory (body), auditory and visual perception into one representation to control a humanoid robot in an multimodal language acquisition scenario.

The combination of recurrent neural networks and artificial evolution was studied in several works by Beer. In [Beer, 2003] he evolved neural agents which can move one dimensional in horizontal direction and have 7 distance sensors. Their task is to categorize vertically falling objects into circles or diamonds and catch the circles but avoid the diamonds. He then provides extensive and exemplary analyses of the evolved agents. This work was later supplemented with even more detailed analyses ([Williams et al., 2008], [Beer and Williams, 2015]). Another very interesting setting is presented in [Agmon and Beer, 2014], where the agents have (simplified) chemical sensors and need to find sources of nutrition. One of the main schemes is, that cognition and behavior arises from interaction and feedback between organism and environment.

Most related to this thesis is the approach of [Floreano et al., 2004]. The authors evolved recurrent neural networks for an active vision task. This task consists of classifying a symbol shown in an image. The image is 320x240 pixels and shows either a black square or a black triangle on white background. They also introduce some noise by switching the value of some pixels with a small probability. Their neural networks get as input a 3x3

pixel window, where each of the pixels has a receptive field containing a varying number of pixels from the original image. One additional input is active, if the window is on the border of the image and not active otherwise. The network has 6 output neurons, two represent square and triangle respectively, one sets the zoom factor (one cell of the retina spans either 5x5, 10x10 or 20x20 pixels) and one selects the activation method of the visual neurons (either average of all input pixels ind the receptive field or the value of the top left pixel). Their experiments showed that evolved agents were able to discriminate between squares and triangles with 100% accuracy. Some ideas from this and the aforementioned works are adopted for this thesis as follows:

In this work, recurrent neural networks for the discrimination of symbols are evolved. Similarly as in [Floreano et al., 2004], the networks also get as input a small window of the total image and can move this window horizontally and vertically. In contrary to [Floreano et al., 2004], the cells of the window have only one pixel as receptive field. The networks cannot change the zoom factor. Additionally they cannot change the activation method. Another difference is the size of the network. In [Floreano et al., 2004] the size is fixed, whereas in this work, the neural networks can grow as required. The analyses of the networks in this thesis take [Beer, 2003] as model to shed some light on the internal dynamics of the evolved agents.

2 Theory of dynamical systems and recurrent neural networks

This chapter deals with the theoretical foundations for the presented thesis. After a brief introduction into the theory of dynamical systems, recurrent neural networks are introduced formally as examples of such systems. Some simple architectures are described, which will be helpful in understanding the functional parts of bigger networks. These simple architectures can be seen as modules with specific functionalities. In the rest of the chapter, artificial evolution as mechanism to find networks, capable of solving a defined task is introduced. In particular, the method of gradually evolving increasingly complex agents is described.

2.1 Dynamical systems theory

Recurrent neural networks can be seen as more or less complex systems having a time dependent state. To describe these networks, along with many other time dependent phenomena, dynamical systems theory is used [Strogatz, 1994]. This section introduces this theory along with it's important terminology. Additionally, the kind of figure used throughout the thesis to analyze neural networks is introduced here.

2.1.1 Mathematical definition of dynamical systems

A dynamic system is the mathematical description of a time varying processes. It is described by a transformation $f : \mathcal{S} \to \mathcal{S}$ which acts on a *state* $\mathbf{s}(t) \in \mathcal{S}$, where $\mathcal{S}$ is the *state space* and $t \in T$ is the time, with either $t \in \mathbb{R}$ or $t \in \mathbb{Z}$ (see below). It is assumed, that the *state variables* in the *state vector* $\mathbf{s}(t) = [s_1(t), s_2(t), ..., s_N(t)]^T$, with N being the order of the system, contain sufficient information to describe the whole systems state

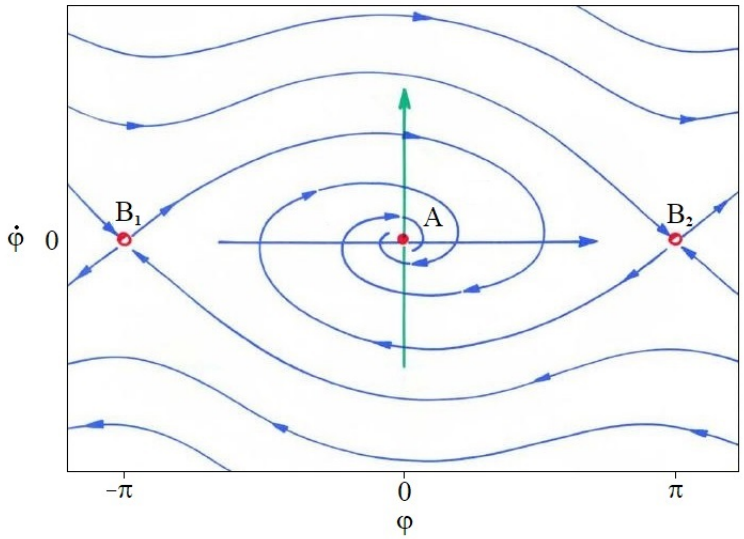

Figure 2.1: Phase diagram of a stiff pendulum with friction. On the horizontal axis is the angle φ, on the vertical axis the velocity $\dot{\varphi}$. Point A is an attractor corresponding to the resting pendulum at the bottom, points B_1 and B_2 represent the very same repellor corresponding to the pendulum in fully upright position (figure from [Abraham and Shaw, 1992] with changes).

at any particular point in time. Formally a dynamical system is the triple $(T, \mathcal{S}, f)$. For a time-continuous system the differential equation

$$\dot{\mathbf{s}}(t) = f(\mathbf{s}(t)), \quad f : \mathcal{S} \to \mathcal{S} \tag{2.1}$$

describes the evolution from one state to the next. This *evolution rule* for time-continuous systems with $t \in \mathbb{R}$ changes to the *update rule*

$$\mathbf{s}(t+1) = f(\mathbf{s}(t)), \quad f : \mathcal{S} \to \mathcal{S} \tag{2.2}$$

for time discrete systems with $t \in \mathbb{Z}$. Since computation in digital hardware is always discrete, the rest of this work only deals with the time discrete case. The set of all possible states is called *state space* $\mathcal{S}$ or *phase space* with $\mathcal{S} \subseteq \mathbb{R}^n$ being a manifold. The description of the system is completed by adding an initial condition $\mathbf{s}(t_0)$ with t_0 as starting time. For every possible initial condition, represented as a point in state space,

exists a *trajectory* defined by this starting point and the update rule f. The set of all possible trajectories is called the *flow* and illustrates the overall behavior of the system.

Of particular interest is the long-term behavior of the system. After enough updates, the state of many dynamical systems ends up in a small subset ob the state space called a *limit set*. When the systems state is in a limit set, it will stay there indefinitely. If all nearby trajectories converge to the limit set, it is called stable limit set or *attractor*. For unstable limit sets (also called *repellors*), the system will move away from the limit set if perturbated. For this work attractors are of particular interest.

Figure 2.1 shows a phase diagram of a stiff pendulum with angle φ on the horizontal and velocity $\dot{\varphi}$ on the vertical axis. The blue lines represent some example trajectories. As can be seen, points B_1 and B_2 are the same repellor (π and $-\pi$ are the same point), since the trajectories lead away from them, point A is an attractor, since all nearby trajectories lead to it. The blue lines going from left to right or from right to left at the top and the bottom respectively are trajectories for the overturning pendulum.

2.1.2 Description of attractors and some of their properties

[Haykin et al., 2009] provides the following definition of an attractor:

A subset (manifold) $\mathcal{M} \subset \mathcal{S}$ of the state space is called attractor if

- *there is an open neighborhood around $\mathcal{M}$ that shrinks down to $\mathcal{M}$ under the flow (basin of attraction);*
- *no part of $\mathcal{M}$ is transient;*
- *$\mathcal{M}$ cannot be decomposed into two non-overlapping pieces;*
- *$\mathcal{M}$ is invariant under the flow;*

With this definition in mind, different attractors can be described. The attractor A in Figure 2.1 is a *stable fixpoint*. As the name suggests the limit set consists of only one point in which the system ends up for $t \to \infty$. Other kinds of attractors are *periodic orbits*, *quasiperiodic orbits* and *chaotic attractors* (with chaotic attractors mentioned here for the sake of completeness, but not further elaborated). When the system ends up in an orbit, the limit set consists of a subset $\mathcal{M} \subset \mathcal{S}$ where the points are visited in a periodic manner. The attractor is called a p-Orbit with period p, when the system visits p points

$$O_p(s_0) = \{s_0, s_1, s_2, \ldots, s_{p-1}\} \tag{2.3}$$

with $s_p = s_0, \forall t < p : s_t \neq s_p$. The attractor is called quasiperiodic if the points are not repeated exactly but the phase portrait clearly shows a periodic behavior.

As the first point in the definition states, for small perturbations the system will always return to the attractor. The points from which the system converges to an attractor over time are called its *basin of attraction*. In the so called *phase portrait* or *phase diagram* all the limit sets and basins of attraction are displayed to fully explain the different dynamical behaviors of the system and which behavior can be found where in its state space.

For a stable environment, the last point of the definition holds. The attractor landscape is invariant under the flow and determines the behavior of the system. When the environment changes, for example changing sensory values provided to a neural network, the attractor landscape can change in different ways. Attractors can move in the state space, resulting in slightly different behaviors. This slight changes of attractors can be desirable. When the attractor landscape changes drastically, with new attractors appearing or others disappearing its called a *bifurcation*. These bifurcations can also be desirable to change between different behaviors. The systematical study of bifurcations by varying one or more values (e.g. weights or inputs) can be graphically evaluated to explain the behavior of a given system. Some examples will be shown in section 4.1.

2.2 Recurrent neural networks

Artificial neural networks are based on a simplified model of biological neurons as processing units [Haykin et al., 2009]. The neuron has inputs $\mathbf{x}(t) \in \mathbb{R}^n$ from sensors, other neurons and possibly from itself. The fire probability of the neuron is modeled by weighting these inputs with the weight vector $\mathbf{w} \in \mathbb{R}^n$ and propagating the summed results through a transfer function $\sigma : \mathbb{R} \to \mathbb{R}$.

2.2.1 The single neuron

Figure 2.2 shows an artificial neuron with n inputs. The weight vector $\mathbf{w}$ and the input signal $\mathbf{x}(t)$ are defined by

$$\mathbf{w} = (w_1, w_2, \ldots, w_n)^T \tag{2.4}$$

$$\mathbf{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t))^T. \tag{2.5}$$

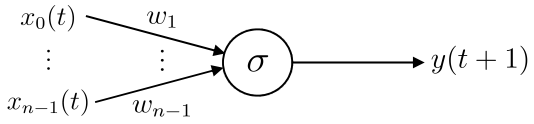

Figure 2.2: Single neuron with input $\mathbf{x}(t) \in \mathbb{R}^n$, activation function σ and output $y(t+1)$.

The activation a of the neuron is then the weighted sum of the input signals:

$$a = \sum_{i=1}^{n} w_i x_i = \mathbf{w}^T \mathbf{x}. \tag{2.6}$$

This is the inner product of the weight vector $\mathbf{w}$ and the input signal $\mathbf{x}(t)$ and therefore represents the similarity of $\mathbf{w}$ and $\mathbf{x}(t)$. The output $y(t+1)$ is then calculated by

$$y(t+1) = \sigma(a+b) = \sigma(\mathbf{w}^T \mathbf{x} + b) \tag{2.7}$$

where $b \in \mathbb{R}$ is called the bias and can be understood as an offset for the activation of the neuron. The bias term can also be included in the weight vector by adding one more weight and concatenating the vector $\mathbf{x}(t)$ with a constant 1.

$$\sum_{i=1}^{n} w_i x_i(t) + b = \sum_{i=0}^{n} w_i x_i(t), \quad \text{with } w_0 = b \text{ and } x_0(t) = 1 \tag{2.8}$$

The *transfer function* σ, also called *activation function*, can be of linear type, truncating (e.g. *rectified linear unit*) or of a sigmoidal type. For this work the *hyperbolic tangent*

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.9}$$

is used, since it combines some useful properties. If the activity of the neuron is in a small range around zero (approx. ± 1) the tanh behaves almost linearly. When the input signals or the weights are very big, the tanh is in saturation and the neuron behaves like the signum function:

$$\operatorname{sgn}(x) := \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{2.10}$$

In case the neuron has a self connection w_s it is called recurrent. Equation 2.7 would then become

$$y(t+1) = \sigma(\mathbf{w}^T\mathbf{x} + w_s y(t) + b). \tag{2.11}$$

Of course, the self connection w_s can be integrated in the weight vector $\mathbf{w}$ and the output $y(t)$ of the last time step can be included in $\mathbf{x}(t)$ similarly as in equation 2.8.

2.2.2 Networks of neurons

Extending from only one neuron to a fully (or only partly) connected network of m neurons the system becomes a parametrized discrete-time dynamical system with the neuron's output state space $\mathcal{S} \subset \mathbb{R}^m$, the parameter space $\mathbf{W} \subset \mathbb{R}^{m \times n}$ and the update rule f given by

$$\mathbf{x}(t+1) = f(\hat{\mathbf{x}}(t)) = \sigma(\mathbf{W}\hat{\mathbf{x}}(t)). \tag{2.12}$$

Here the vector $\hat{\mathbf{x}}(t) \in \mathbb{R}^n$ is the concatenation of sensory inputs, the neuron's outputs of the last time step and a constant term representing the bias term. The weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ with w_{ij} denoting the weight from neuron j to neuron i is given by

$$\mathbf{W} = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n-1} & 1 \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n-1} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n-1} & 1 \end{pmatrix}. \tag{2.13}$$

The weight matrix is often sparse (most entries are zero) since not all neurons are connected to each other and not all neurons necessarily get all the sensory inputs. Therefore it can be computationally more efficient to calculate the network updates not as a matrix multiplication but to process the neurons individually. How this can be done is shown in chapter 3. Additionally a subset of all neurons is declared as output units. It is only this subset which is read out and serves, for example, as a controller for a robot. The sensory

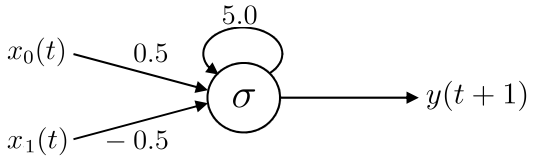

Figure 2.3: One neuron acting as a switch. Depending on the input, the output is either $+1$ or -1.

inputs are often referred to as input neurons. All other neurons are called hidden units. When analyzing recurrent neural networks it can often be observed, that a small subpart of the network exhibits a specific behavior. The next section gives a short overview of possible dynamics of such subparts.

2.2.3 Dynamics/Modules of small neural networks

One important behavior of a neuron is that of a neural switch. Depending on the input, the neuron switches between two states with their corresponding outputs. This is realized for example with two complementary inputs and a strong self connection as shown in figure 2.3. As long as both inputs x_1 and x_2 have the same value, the activation of this neuron stays zero. As soon as one of the inputs has a higher value than the other, the neuron's activation transitions to $+1$ or -1 (or near these values) in a few update steps. This can be observed by plotting the temporal evolution of the neuron's output for different starting values of $y(t_0)$. For the example neuron this is shown in figure 2.4. Here the inputs are slightly out of balance with $x_1 = 0.1$ and $x_2 = 0$. The different lines show the temporal evolution of the neuron's output starting at different values of $y(t_0)$. These range from -1 to $+1$ in steps of 0.05. Here, the time steps are denoted as t_i on the x-axis and the output value of the neuron on the y-axis. This kind of figure is used in later chapters with the same meaning. Additionally at some points the values at these positive and negative attractors are described as $+1$ and -1 for simplicity when they are really only in the vicinity of these extreme values (for example between 0.8 and 0.99 for $+1$).

Another interesting behavior one single neuron can show is that of an oscillator. When the neuron has one input and again a strong self connection with an opposed sign to the input, it starts to oscillate between positive and negative activation. This is very obvious

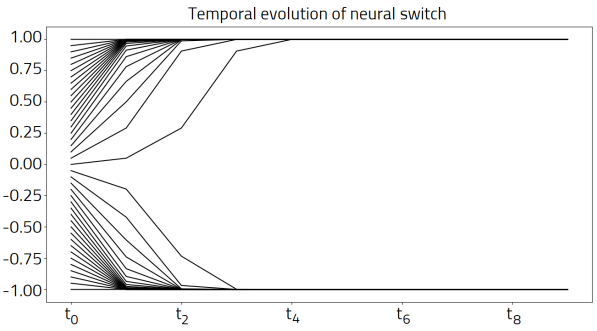

Figure 2.4: The temporal evolution of the aforementioned neural switch. The output transitions to $+1$ or -1 in only a few steps when the activation y is out of balance.

by looking at the equations and some example calculations. Let the equation for the output of the neuron be

$$y(t+1) = \tanh\big(2x(t) - 2y(t)\big). \tag{2.14}$$

Assuming again an input of $x = 0.1$ the output of the neuron for the first few time steps is calculated as:

$$\begin{aligned} y(0) &= \tanh(2 \cdot 0.1 - 2 \cdot 0) = \tanh(0.2) \approx 0.20 \\ y(1) &\approx -0.19 \\ y(2) &\approx 0.53 \\ y(3) &\approx -0.69 \\ y(4) &\approx 0.92 \end{aligned}$$

The switching between positive and negative outputs and the increase of the amplitude are typical for this kind of simple oscillator. Depending on the weights and the input value, it stabilizes on a certain amplitude as shown in figure 2.5. This is an example of a 2-orbit system. As a last example, a network formed by 2 fully connected neurons is shown in figure 2.6. If the weights of this network meet certain conditions, this combination is called a neural SO(2)-Oscillator ([Pasemann et al., 2003]). The weight matrix of such a

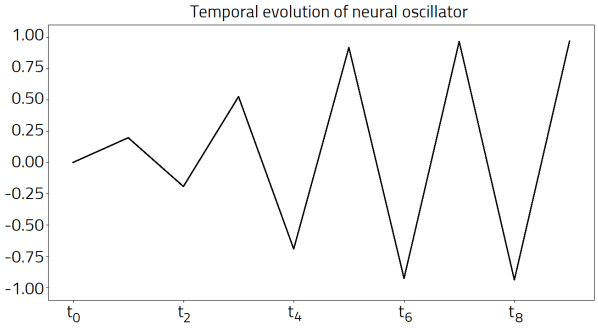

Figure 2.5: One neuron acting as an oscillator. Depending on the input, the output switches between $+1$ or -1.

network is given by

$$\mathbf{W} = \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix} = \alpha \cdot \begin{pmatrix} cos(\phi) & sin(\phi) \\ -sin(\phi) & cos(\phi) \end{pmatrix} \tag{2.15}$$

with $\alpha > 1$. Such a matrix is easily recognized as rotation matrix in the 2D-plane. Since for the stable oscillation a quasi-periodic orbit needs to be established, the factor α is introduced to compensate the slope of the tanh function. The example shown in the figure has the weight matrix

$$\mathbf{W} = \begin{pmatrix} 1.1 & 0.15 \\ -0.15 & 1.1 \end{pmatrix}. \tag{2.16}$$

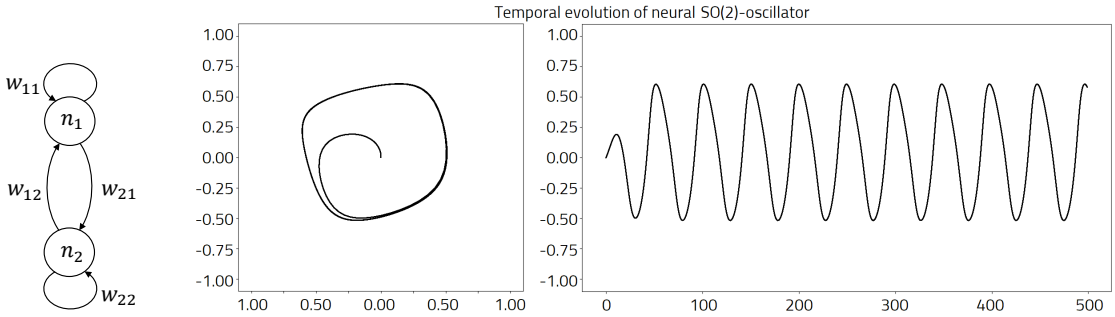

Figure 2.6: Two neurons as given in the leftmost figure form a SO(2) oscillator. In the middle the phase diagram of the two neurons is shown (output of neuron 1 on the x-axis and output of neuron 2 on the y-axis). The figure on the right shows the output of the first neuron plotted over time.

2.3 Artificial Evolution

Adopting some ideas from nature, artificial evolution is a population based optimization algorithm [Nolfi et al., 2000]. It is often used, when classic optimization strategies like gradient methods fail to perform well. Artificial evolution can provide a solution to a problem which in some cases is not optimal, but good enough for the task at hand. The optimization problem needs to be formulated such that a solution can be given in a set of N numerical parameters, often formulated as a vector $\mathbf{v} = (v_1, v_2, \ldots, v_N)$. This vector of numbers is called the *genotype* of an *individual*. A set of K individuals form a *population* P. The individuals in the population are randomly initialized with $v_i := [-4; 4]$ being a good choice for neural network weights. All individuals in the population form a *generation* which is subjected to the following steps.

Each individual is evaluated using a carefully chosen *fitness function*. This function has to include all the important aspects of the given task but should be formulated as simply as possible. Additionally it has to be monotonically increasing, so that a better behavior results in a better fitness. This function is then used to rate the individuals according to their capability to solve the given problem.

Then, biologically inspired operations like *selection*, *recombination* and *mutation* serve as operators and are applied to the individuals in one generation. Each operator is applied with a certain probability. These probabilities are the adjustable hyperparameters for the algorithm. In the selection phase, the best individuals are selected to form the basis of a new generation. If the number of selected individuals is smaller than the population size, the rest of the individuals are created via copying or recombining. It is also possible to create totally random new individuals.

Recombination, also called *crossover*, corresponds to sexual reproduction in natural evolution. Randomly chosen parts of two or more individuals are combined together to form a new genotype. Thus, partial solutions from different individuals can be combined.

When the population is complete, all individuals are mutated to generate previously unseen genotypes. This can be structural mutation like deleting or adding neurons or weights but also changes to the numerical values of the network weights. It is the experimenter's job to control mutation probabilities so that there is enough variability for new solutions to emerge but not too much to destroy already found solutions.

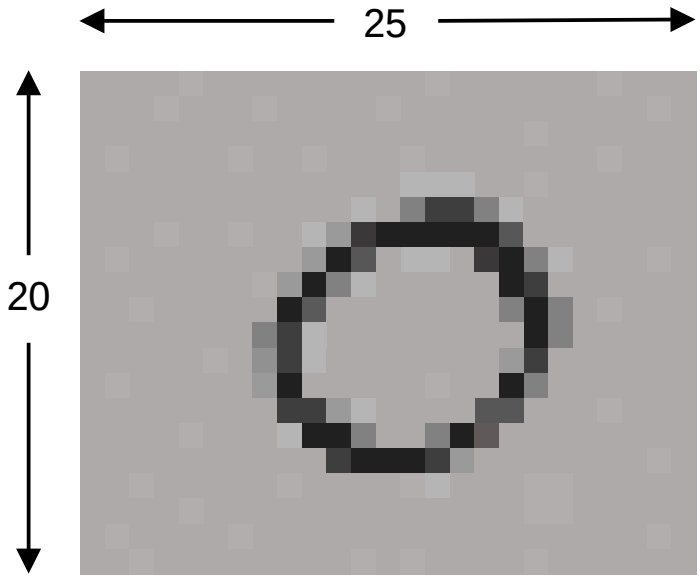

Figure 2.7: An image from the pixelpipeline has 25 times 20 pixels. This image shows the zoomed in image of a circle drawn on the whiteboard.

2.4 The given images and the approach for the task

As described in the introduction, the RNN agents evolved in this work have the task to discriminate between different symbols. In an active vision process, they should determine which symbol is depicted in an image.

2.4.1 How the given images look like

The setting consists of images of a whiteboard with symbols drawn on it. The images are obtained through the visual perception system of a humanoid robot which imposes the following boundary conditions. The images come in the YUV color format with Y being the luminance and U and V the chrominance. Taking human perception into account, the YUV format has a reduced resolution for the chrominance. Two pixels have different Y values but share U and V. The raw image from the camera consists of 360 times 288 pixels. The image processing system of the robot, called pixelpipeline, gets the raw image, performs an user defined affine transformation and provides the result in a reduced resolution of 25 times 20 pixels. The possible transformations include zooming, rotation

and stretching or compressing. Depending on the transformation a certain number of pixels from the raw image are averaged to give a superpixel in the transformed image. These superpixel then form the final image of 25 times 20. In figure 2.7 one reproduced example is shown. Although all color information is available, the RNN agents only use the luminance channel. Since the symbols are black on a white background, the luminance of pixels in the symbol is much lower than the luminance of the background. This makes the separation quite easy. The pixel values of the darker pixels have an average value of 0.18, the bright pixels have an average value of 0.49. These values are used for the binarized versions of the images (see chapter 4).

The real world data collected for this work consists of 96 images of three symbols and 32 images of plain background. Every symbol, namely a square, a circle and an arrow, was drawn 16 times and for every drawing 2 images were taken. These 2 images are taken with a slightly different setting of the pixelpipeline, so that they are not identical. When the zoom factor or the position of the camera relative to the whiteboard is changed slightly, the resulting superpixel are quite different.

2.4.2 Two strategies for the structure of the agents

The RNN agents do not get the whole image as input. Only a small window of 5 times 5 pixels, called the *retina*, is available to the neural network (see figure 2.8). The rows of the retina are concatenated to form the input for the network. This input is then processed and as output the neural network controls how the retina moves along the horizontal and the vertical axis. The first two neurons handle these controls (indexed 0 and 1). The next neurons encode the symbol categories. Two strategies are possible to solve this task.

The first strategy uses one neural network in which one neuron is reserved for each of the symbols. Therefore the neural network would need at least three more neurons. The neuron with the highest activity would indicate the existence of the corresponding symbol. In this work, neuron 2 (as mentioned above, neurons 0 and 1 control movements) encodes a square, neuron 3 a circle and neuron 4 an arrow. The absence of a symbol is encoded as low activity in all three neurons.

In the other strategy, one network only looks for one symbol. In other words, for every symbol one network will be evolved. These networks only need three neurons, two for movement and one which is active if the corresponding symbol is present and inactive

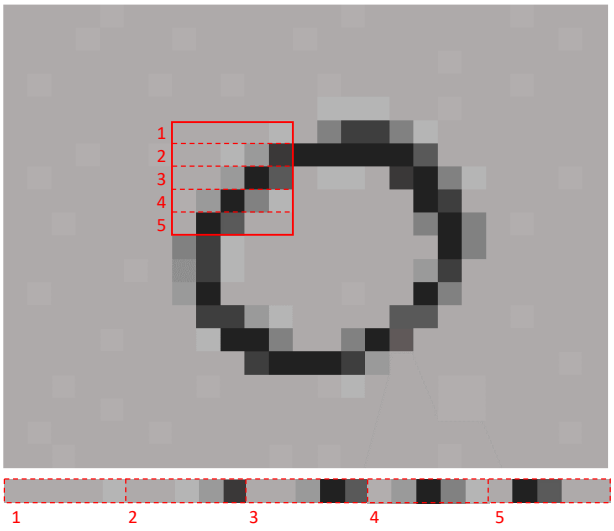

Figure 2.8: The retina consists of a 5 times 5 pixel window. The rows of this window are concatenated to give a size 25 vector, which serves as input to the neural network.

otherwise. Although this strategy requires one additional network for every new symbol, these "specialized" networks can be smaller. Additionally adding new symbols does not require a completely new evolved network, but only a new one for the new symbol. This makes the second strategy a possible candidate for open-ended learning.

Since the task of discriminating these real world images is quite difficult, the first stages of evolution are performed using simpler artificial images. With increasing complexity of the task the RNN agents can be evolved such that they finally are able to handle the real world images. This increase of complexity is described in detail in section 3.2.

3 Implementation of artificial evolution and neural networks

The main goal of this work is the development of recurrent neural networks (RNN) which can be used by a robot to recognize a set of symbols drawn on a whiteboard. This goal is achieved by using the principle of active perception. The RNN actively determines its next step while analyzing a given image. Its dynamic state then ends up in an attractor which unambiguously indicates which kind of symbol is shown in the image. Since the recognition of symbols in such a way is a very hard task, the RNN is gradually evolved. Starting with simple problems, the tasks presented to the networks get increasingly more complex. This chapter starts by describing in detail the implementation of the developed algorithms, which were used to evolve RNNs. The first part of this chapter is deliberately more focused on the technical details to allow others to reimplement these ideas. The second part describes the path of evolution to find RNNs solving complex tasks starting with the simplest task up to the final goal (the original image) and shows the design decisions made on this track.

3.1 The technical details of the implementation

This sections deals with the implementation of recurrent neural networks and artificial evolution. It gives an overview of all needed structures and the chosen parameters. Figure 3.1 shows the most important steps of the implementation presented in a flow diagram. The left part (1) of the figure shows the procedure `processGeneration`. In this procedure all the steps of the artificial evolution, as described in the previous chapter, are executed. The procedure is as follows: The outer loop loops over all the training images. One loop consists of the loading of one image, the evaluation of all individuals (inner loop), then the mutation and the selection. The inner loop evaluates all individuals of the generation

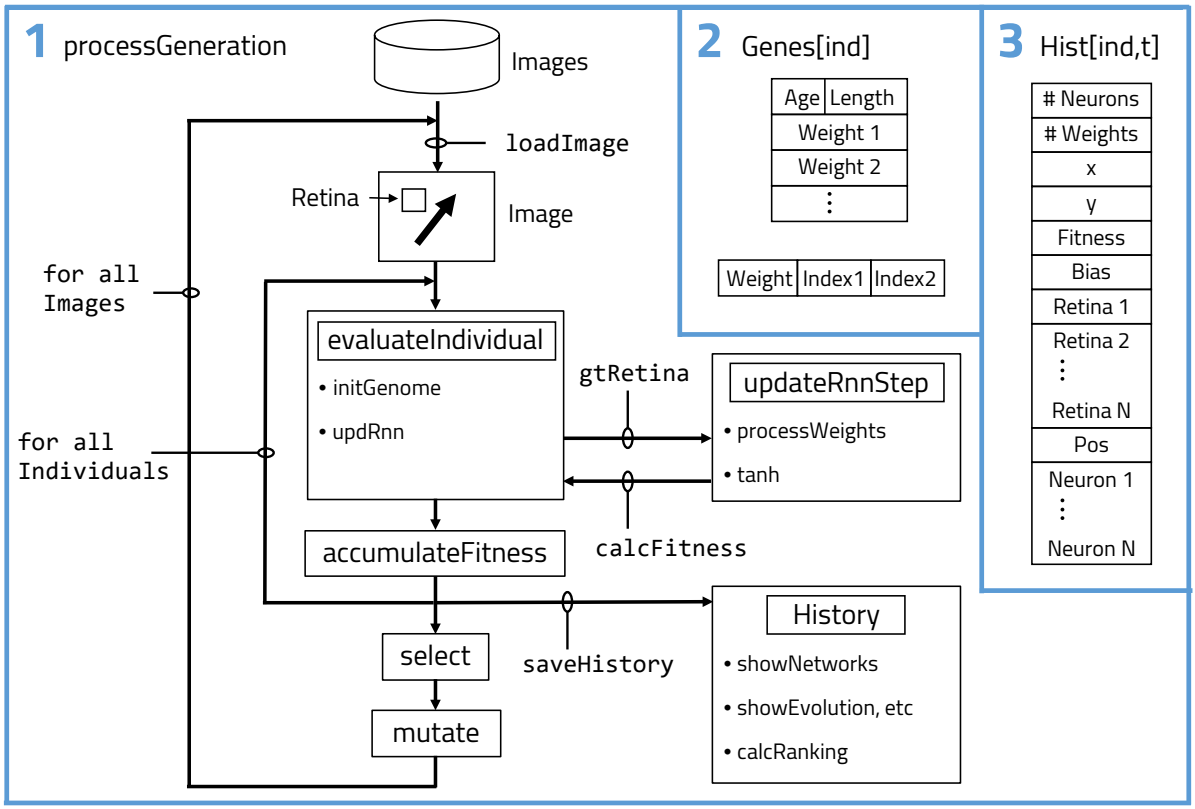

Figure 3.1: The procedure `processGeneration` with the most important steps is shown as flow diagram. On the right hand side, the encoding of the genes and the `History` buffer is shown.

on the given image. Each individuals fitness is saved in the `History` buffer. This buffer is used to display individual networks, their weights, retina movement et cetera. It is also used to calculate the fitness over all images and to rank the individuals according to their fitness values. In the right part of figure 3.1 the encoding of the `History` buffer and the buffer `Genes` for all the genotypes is shown. The following sections describe the implementation of the neural networks and the shown procedures in more detail.

3.1.1 The representation of neural networks in memory

As mentioned in section 2.2 the weight matrix $\mathbf{W}$ of a recurrent neural network is often sparse. So, computing the update by matrix multiplication would involve a lot of unnecessary computation. To compute the update more efficiently and to allow for dynamically changing network structures, the RNN (not shown in figure 3.1) is represented as two

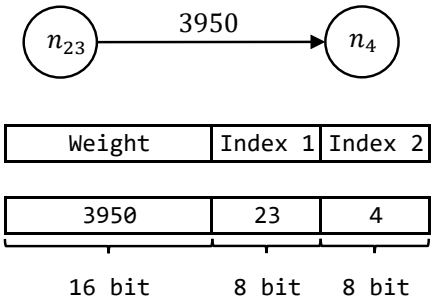

Figure 3.2: The genotype is encoded as a list of triples consisting of the weight and two indices (see text). The weight is represented as a 4Q12 fixed-point number, the indices are two 8 bit numbers, so that the whole list element is a 32 bit number.

arrays, encoded in 4Q12 fixed-point arithmetic. This means, that the top 4 bit encode the digits before the decimal point and 12 bit encode the fractional part.

The first array `RNN` represents the state vector $\mathbf{x}(t)$ for the neurons. All neurons are listed there and accessed by an index. This includes the bias term and all inputs. The first index is the bias term, namely the number 4096 as this is 1 in 4Q12 arithmetic. The next 25 cells are the input from the *retina*. Indices 26 and 27 contain the horizontal and the vertical position of the retina respectively. Starting at index 28 until the maximum index of 255 the neuron activities are stored. The second array `Genome` is a copy of the genotype (from `Genes`) of the currently active network. Its structure can be seen in part 2 of figure 3.1. The first entry encodes the age and the number of weights of the network. Then there are listed the weights, in the figure denoted as "`Weight 1`", "`Weight 2`" and so on. These entries consist of a weight and two indices. The first index is the neuron (or the input) the weight comes from as described above and the second is the neuron the weight goes to (see figure 3.2). As weights can only go to neurons and not to inputs, the second index includes an offset. This offset is the index 28, where the first neuron is encoded in `RNN`. So if the index 2 is 0 the weight really goes to the neuron stored at index 28.

Weights and indices are encoded together in a 32 bit hexadecimal number. The top 16 bit encode the weight in the above mentioned 4Q12 fixed-point notation with $2^{12} = 4096$ representing the 1. The weights are therefore in the range $(-16384, 16383)$ which is $(-8.0, 8.0)$ in floating point representation. The indices range from 0 to 255 allowing for a

maximum number of 228 neurons (since the neuron activations in the array `RNN` start only at index 28, there is only room for $256 - 28 = 228$ neurons). The example in the figure represents a weight going from 23 to neuron 4 with the weight being $3950 \hat{=} 0.96436$ The index 23 is of course an input from the retina image. A self connection of neuron 4 would be indices 32 to 4. Neurons 0 and 1 are used to control the movements of the retina. The outputs are scaled by 4 and determine the change of the x and the y position respectively. Neurons 2 to 4 are used to indicate symbol categories. Neuron 2 stands for a square, neuron 3 for a circle and neuron 4 for an arrow. The recognized symbol is encoded with high activity (near one) of the corresponding neuron and low activity (near negative one) of the others. For easier processing the first entry in the array `Genome` contains the length of the genome in the lower 8 bit and the age in the higher 16 bit. The age is a simple counter which counts the numbers of generations an individual already exists.

3.1.2 Algorithmic details of the artificial evolution

All genotypes of a population are stored in the array `Genes` and the currently processed genotype is copied to `Genome`. Then the procedure `updateRnnStep` (see algorithm 1) is called. It iterates through the array `Genome`, extracts the weight and the two indices and multiplies the weight by the activation of the neuron at index 1 (see lines 4 to 7). The result is added to the neuron at index 2 (at this point in a `Cache`). When all weights are processed, the activation function is applied to the neuron activations in the `Cache`. Then, to conclude one update of the RNN, the results in the `Cache` are copied back to `RNN`.

As can be seen in Algorithm 1, the `Cache` is first filled with NaN (Not a Number). Then, in line 8 to 12, the NaN is overwritten by the *result* at the first appearance of a specific index. Using this method, it can be easily determined which neurons are active and which were deleted by the evolutionary algorithm. One has only to look for NaN in the `RNN` to find the deleted neurons, since these NaN are not overwritten by any *result*. These free slots can than be filled by new neurons.

To evaluate an individual, the procedure `updateRnnStep` has to be called repeatedly. The procedure `evaluateIndividual` in algorithm 2 shows the whole process: First the neurons in `RNN` and the current fitness value have to be set to zero. After that, the current genotype is loaded. Then, for the predefined number of update steps $NrUpdates$,

Algorithm 1 One update step of the recurrent neural network

```
1:  procedure updateRnnStep
2:      Cache ← NaN                                   ▷ fill Cache with NaN
3:      for i ← 0, len(Genome) do
4:          weight ← Genome[i]_{31:16}                ▷ top 16 bit
5:          index1 ← Genome[i]_{15:8}
6:          index2 ← Genome[i]_{7:0}
7:          result ← weight× RNN[index1]              ▷ interim result
8:          if Cache[index2] = NaN then
9:              Cache[index2]← result
10:         else
11:             Cache[index2]←Cache[index2]+result
12:         end if
13:     end for
14:     for i ← 0, len(Cache) do
15:         Cache[i] ← tanh(Cache[i])
16:     end for
17:     RNN ← Cache                  ▷ copy neuron activities from Cache to RNN
18: end procedure
```

the following steps are computed. `getXYPosition` reads the activity of neurons 0 and 1 and computes the new x and y position of the retina. Next, `getRetinaImage`, as the name suggest, uses these new positions to load the corresponding pixel from the full image and writes them into the retina cells (1 to 25) in `RNN`. Then the above described procedure `updateRnnStep` is called.

When the new state of the neural net is calculated, its $fitness$ can be determined. The procedure `calcFitness` described in detail in the next section evaluates the current genotype. After the loop is finished and all update steps are done, the summed $fitness$ value is normalized with $NrUpdates$ to make sure the fitness value is in the range 0 to 1 (provided `calcFitness` returns a value in that range). The values can be of arbitrary range, but normalizing is applied for the convenience of analysis. The final $fitness$ value of the given individual is returned. This value is used by the artificial evolution to rank the individuals.

In algorithm 3, showing the procedure `processGeneration`, every individual in the population is evaluated with every test image. The fitness values are normalized (line 7) with the number of images $NrImages$. Then the best individuals are selected, recom-

Algorithm 2 Evaluation of one individual

```
1: procedure EVALUATEINDIVIDUAL(i)                    ▷ Index i of current genotype
2:     RNN[28:255] ← 0                                ▷ fill RNN with zeros
3:     fitness ← 0
4:     Genome ←Genes[i]
5:     for i ← 0, NrUpdates do
6:         GETXYPOSITIONS          ▷ get position of retina from neuron activity
7:         GETRETINAIMAGE          ▷ retrieve part of image as input to neural net
8:         UPDATERNNSTEP
9:         fitness ← fitness+CALCFITNESS              ▷ according to fitness function
10:    end for
11:    fitness ← fitness/NrUpdates
12:    return fitness
13: end procedure
```

Algorithm 3 Evolving one Generation

```
1: procedure PROCESSGENERATION
2:     Fitness ← 0                                    ▷ fill Fitness with zeros
3:     for i ← 0, NrImages do
4:         LOADIMAGE                                  ▷ Load new test image
5:         for i ← 0, PopulationSize do
6:             fitness ← EVALUATEINDIVIDUAL(i)
7:             Fitness[i] ← Fitness[i]+fitness/NrImages
8:         end for
9:     end for
10:    SELECT                                         ▷ according to selection policy
11:    MUTATE                                         ▷ according to mutation policy
12: end procedure
```

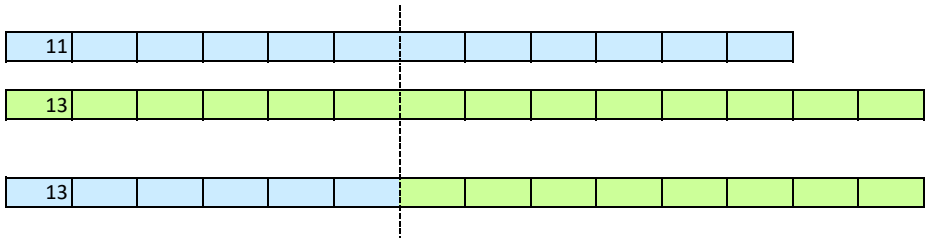

Figure 3.3: Two genotypes are recombined. They are split at a random position indicated by the dashed line. The length of the genome, stored in the first cell, is copied from the individual which provides the part to the right.

bined (`Select`) and mutated (`Mutate`) as described below. The selection and mutation can alternatively be done in an extra step. This can be convenient for analysis of the current generation. After all individuals have been evaluated, results can be displayed and analyzed before "destroying" the population with selection and mutation.

Selection

The selection in the evolutionary algorithm can be handled in different ways (see for example [Nolfi et al., 2000]). For this work a rank based selection of the best 40% was used. These individuals are each copied twice into the new generation. The remaining 20% is filled up with randomly generated individuals which have the same number of neurons and similar number of weights like the best individual of the last generation. After the new generation is filled up with new individuals, the crossover is applied. A small fraction of individuals is chosen, they are divided at some random point and recombined with a fraction of another individual as shown in figure 3.3. During this recombination of individuals, the average of their age is propagated to the new individual.

Mutation

The mutation is perhaps the most integral component of the evolutionary algorithm, since it introduces the changes in the genotype which eventually lead to improvements. It includes 5 actions each with a certain probability of occurrence. These probabilities are listed in table 3.1. The first 4 probabilities handle how probable it is to delete/insert *one* neuron/weight. The mutation in one generation allows only one change of numbers in neurons and weights. Mutation of weights on the other hand can include all weights.

Action	Parameter	Start Value
Delete neuron	μ_delNrn	0.1
Delete weight	μ_delWgt	0.2
Insert neuron	μ_insNrn	0.05
Insert weight	μ_insWgt	0.1
Mutate genome	μ_Gen	0.3
Mutate weight	μ_Wgt	0.3
Variance of mutation	$MutVar$	0.2

Table 3.1: The probabilities for mutation of a genotype. These parameters are manipulated to guide the evolution. Also given are some values to start with.

The probability μ_Gen determines if a certain genotype is mutated. If it is, μ_Wgt is the percentage of weights being mutated. The mutation of one weight consists of adding a normally distributed number with mean 0 and variance $MutVar$. In later stages of the evolution $MutVar$ can be reduced to 0.01 which then is called parameter optimization.

3.2 Increasingly complex tasks for gradual evolution

As mentioned in section 2.4 the RNN agents are evolved using increasingly complex tasks. The first task includes two artificially created symbols, namely a square and an abstract circle, which basically has the same symbol as the square but with cutoff corners (see figure 3.4). The retina starts in the top left corner of the image. This task is solved very

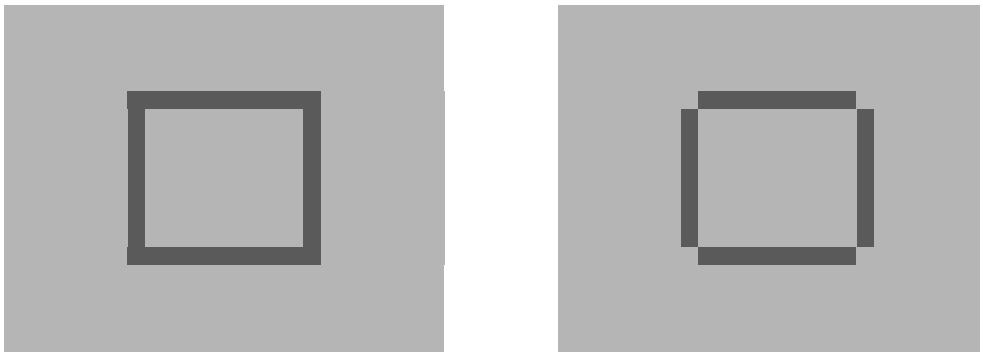

Figure 3.4: Artificially produces images to evolve the first population. A square on the left and a simple circle on the right.

quickly as described in section 4.1. The solution found depends heavily on the constant environment. The next task consists of the same symbols but moved one pixel to the side. As soon as solutions for this slightly more challenging tasks are found, the complexity is increased further by adding 4 images with the symbols now moved once to the right, once down and once both. This gives in total 8 different configurations. This task is referred to as "8 symbols task" even though, there are only 2 symbols in 4 different locations. An analysis of the best network which categorizes these 8 images correctly is given in section 4.1.2.

Now the stage is set to move from artificial examples to more complex images. For this purpose, the mean luminance value of the real world images is calculated and used as a binary threshold to binarize the images as shown in figure 3.5. The binary values used are the average values of the dark and the bright pixels respectively, as they were found in the original images (dark pixels have 0.18, bright pixels have 0.49). These values were also used for the artificial images. Therefore the neural networks evolved with the artificial images already work with nearly the same range of luminance values that they later "see" in the real world images.

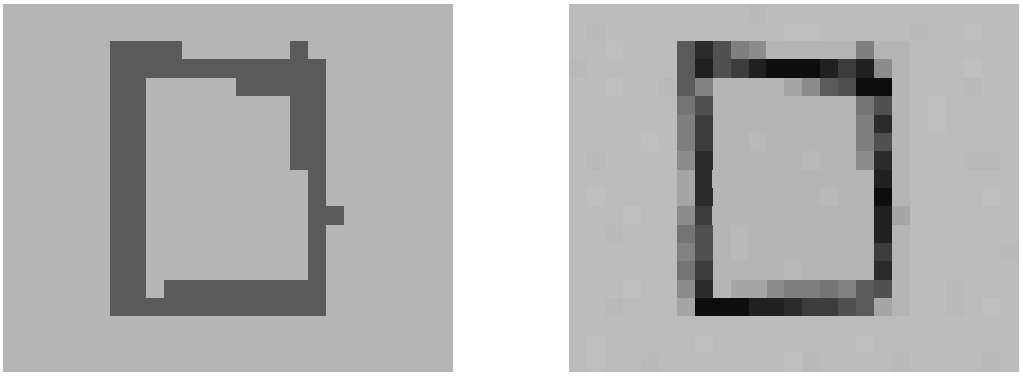

Figure 3.5: A real world image of a drawn square is binarized by taking the mean luminance value as threshold (left). The original image is shown on the right.

The last stage of complexity before getting to the real images is the addition of noise to the luminance values of the image. For this purpose, the variance of the luminance values in the real world images is analyzed and used to scale the noise. Then a normally distributed number with mean 0 and the variance from the real world data is added to the pixels (in the experiments a variance of 0.12 was used). This rather easy method increases

robustness of the found solutions, since networks which depend on fixed luminance values of certain pixels for their behavior will then be sorted out quickly.

3.2.1 Fitness function

The above mentioned `calcFitness` function returns a value between 0 and 1. This value is calculates in two different ways for strategy one and strategy two as follows. For strategy one it is important, that only the neuron indicating the symbol shown in the image has a high activity. The other neurons should have low (negative) activity. Therefore, the fitness function incorporates all of these requirements. For strategy one two cases have to be distinguished. The first case is the presence of a symbol. Here the corresponding neuron needs a high activation and all others a low activation. If there is no symbol, then all 3 neurons should have a low activation. This leads to the following fitness function for a given network depending on the symbol in the image.

$$\text{fitness}_{\text{network}}(\text{symbol}) = \sum_t \frac{1}{6}(3 + O_t(\text{symbol})) \tag{3.1}$$

with

$$O_t(\text{symbol}) := \begin{cases} o_{2,t} - o_{3,t} - o_{4,t} & \text{if symbol = square} \\ o_{3,t} - o_{2,t} - o_{4,t} & \text{if symbol = circle} \\ o_{4,t} - o_{2,t} - o_{3,t} & \text{if symbol = arrow} \\ -o_{2,t} - o_{3,t} - o_{4,t} & \text{if symbol = None} \end{cases}. \tag{3.2}$$

Here t denotes the time step and $o_{2,t}$ is the output of neuron 2 at time step t. For strategy two, the fitness is even simpler. Here the fitness function depends on the purpose of the network. As an example, the fitness function of the square network would be

$$\text{fitness}_{\text{network}}(\text{symbol}) = \sum_t (O_t(\text{symbol})) \tag{3.3}$$

with

$$O_t(\text{symbol}) := \begin{cases} o_{2,t} & \text{if symbol = square} \\ -o_{2,t} & \text{if symbol = not square} \end{cases}. \tag{3.4}$$

3.2.2 How the accuracy of the networks is measured

The evolution on the real world images, binarized and original is done on less then half of the available images. The networks only see 60 images (15 for every symbol and 15 empty ones). The rest (68 images) is left for testing and evaluation. These 68 images will be used to measure the accuracy of the networks. For this purpose, they see these images as input and then are evaluated for 20 time steps. The outputs of the corresponding neurons are analyzed and the final "voting" is determined. Here only unambiguous outputs are counted as right. This means, that the networks output is only correct, if the neurons activation is high which indicates the presence of the given symbol and the others are low (for strategy one). Then the correct votes are counted and divided by the number of trials, which gives the final accuracy.

4 Experiments and analysis of evolved agents

In this chapter the experiments are described and the results are analyzed. The track of increasingly complex tasks described in section 3.2 is carried out and the RNN agents solving the tasks are presented and discussed.

Part one consists of the strategy to evolve one RNN agent to recognize all symbols. It starts with the very simple task of two artificial images showing two symbols. The RNN agent which solves this task best is analyzed. Then the agent solving the 8 symbols task is analyzed and the differences are highlighted. Gradually getting more complex, the best agent for the real world image is compared to the one for the binarized version.

Part two presents 4 agents each evolved to indicate the presence of either one of the four cases. The main similarities and differences are highlighted and it is discussed, where the found solutions could be improved.

4.1 Strategy one: One network for all symbols

The first strategy is to evolve an RNN agent which can detect one of three symbols or the absence of symbols in a given image. As described in section 2.4 this is achieved by using five neurons as output neurons whose activity determine the movement of the retina and encode the presence of the symbols. As a first stage of evolution, RNN agents are being evolved, that can distinguish between two artificial symbols. These agents already have neurons 0 and 1 as output neurons determining the movements. The outputs can vary between -1 and +1. They are then scaled by a factor of 4. So the retina can move at most 4 cells in one time step. Neurons 2 and 3 are reserved for encoding the square and the circle respectively. The neuron encoding the arrow is not yet considered, but will be added in the binarized task.

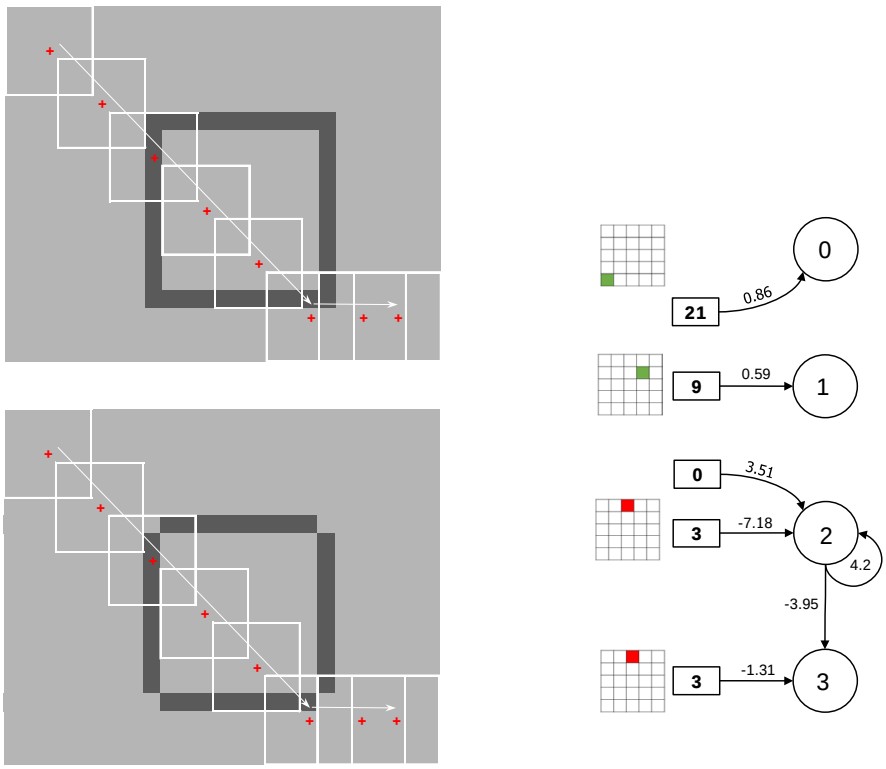

Figure 4.1: The recurrent neural network which solves the task of categorizing two artificial symbols. Neuron 1 moves the retina horizontally and neuron 1 moves it vertically. Neuron 2 encodes the square and neuron 3 encodes the circle.

4.1.1 Two artificial symbols

The two symbols shown in figure 4.1 on the left represent a square at the top and a circle at the bottom. On the right hand side in the same figure, one agent which solves this task is shown. It is a fairly simple network with 4 neurons and 7 weights. As the movement of the retina shows, the agent relies heavily on the static environment. Neurons 0 and 1 receive a small constant input which resolves in a constant movement from the left top corner to the bottom right corner. It uses input fields 21 and 9 for positive activation, which is only varying slightly, depending on the input values. Neuron 2 has only one input, a self connection and a bias and acts as a switch. Its equation looks as follows:

$$y(t+1) = \tanh\left(4.2y(t) - 7.18x(t) + 3.51\right)$$

where $y(t)$ is the neuron's current output value and $x(t)$ is the current input. One can see, that in the circle task, input field 3 always gets the same value of the background color (which is 0.49, the darker pixels are 0.18). This input is almost exactly compensated by the bias term with a small perturbation to negative values:

$$\begin{aligned} y(1) &= \tanh\big(4.2 \cdot 0 - 7.18 \cdot 0.49 + 3.51\big) \\ y(1) &\approx -0.0082 \\ y(2) &\approx -0.043 \\ y(3) &\approx -0.185 \\ y(4) &\approx -0.656 \end{aligned}$$

This negative tendency results in a strong negative activity after a few iterations. Through the connection from neuron 2 to neuron 3, neuron 3 then gets a strong positive activation. This results in the final decision for circle. If, on the other hand, input field 3 hits a darker pixel at update step 2, the activation changes to positive values, which then results in the opposite decision:

$$\begin{aligned} y(2) &= \tanh\big(4.2 \cdot (-0.0082) - 7.18 \cdot 0.18 + 3.51\big) \\ y(2) &\approx 0.967 \end{aligned}$$

In this first task, the evolutionary algorithm found a simple solution to the given task, without the need for actively exploring the symbols.

4.1.2 Eight artificial symbols

To enforce a more stable solution, the artificial symbols are now moved one pixel to the side or to the bottom or both, resulting in 8 different configurations. Now the agents can no longer rely on a static environment and another solution has to be found. One successful agent is shown in figure 4.2 and its behavior for two configurations in figure 4.3 and 4.4.

The only difference in this first configuration is the pixel at input field 3 in the third iteration. As before, neuron 2 acts as a switch and its activation becomes $+1$ for the square and -1 for the circle as can be seen in the plots of the neuron outputs. As the right plot

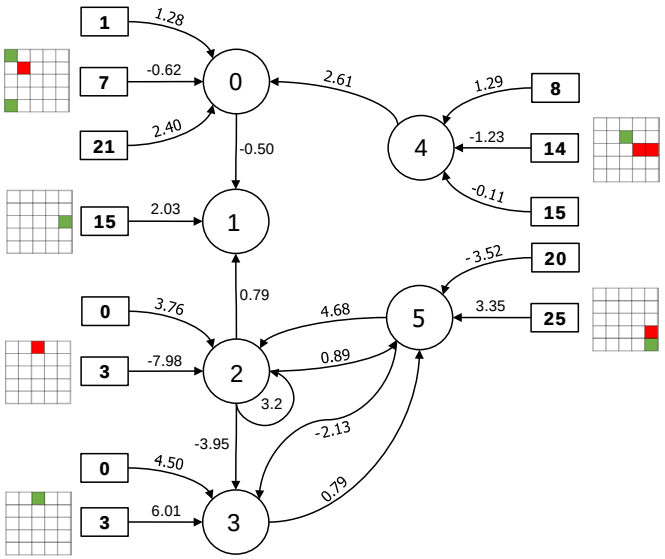

Figure 4.2: One agent solving the eight symbols task successfully.

shows, the output of neuron 2 gets more negative with every iteration until it ends up at -1 which then influences the activation of neuron 3 to become positive. When on the contrary input field 3 shows a dark pixel, neuron 2 switches immediately to a positive activation where it remains. Neuron 2 also influences the movement in the y-direction. In the square environment, neuron 2 forces neuron 1 to stay positive. Otherwise neuron 1 slowly changes from $+1$ to -1 resulting in the turning behavior.

By far the most interesting behavior in this setting is shown in figure 4.4. Here input field 3 at update step 3 does not help to differentiate between the two symbols, so that another solution evolved. The occurrence of the darker pixel still results in the strict positive output of neuron 1 and therefore in the movement downwards. Here the behavior shows a movement to the right bottom corner and then to the left bottom corner, so that the network can explore if the symbol has missing edge pixels. It is then update step 9 where input field 3 again sees either a darker pixel or not indicating the square or the circle respectively. In the output plots one can see, that at this point the decision is already made and the second turn on the left corner would not be necessary. Interestingly in the circle case, the output of neuron 1 gets positive in the last update steps which again would result in the movement upwards on the right side of the image as in the case before.

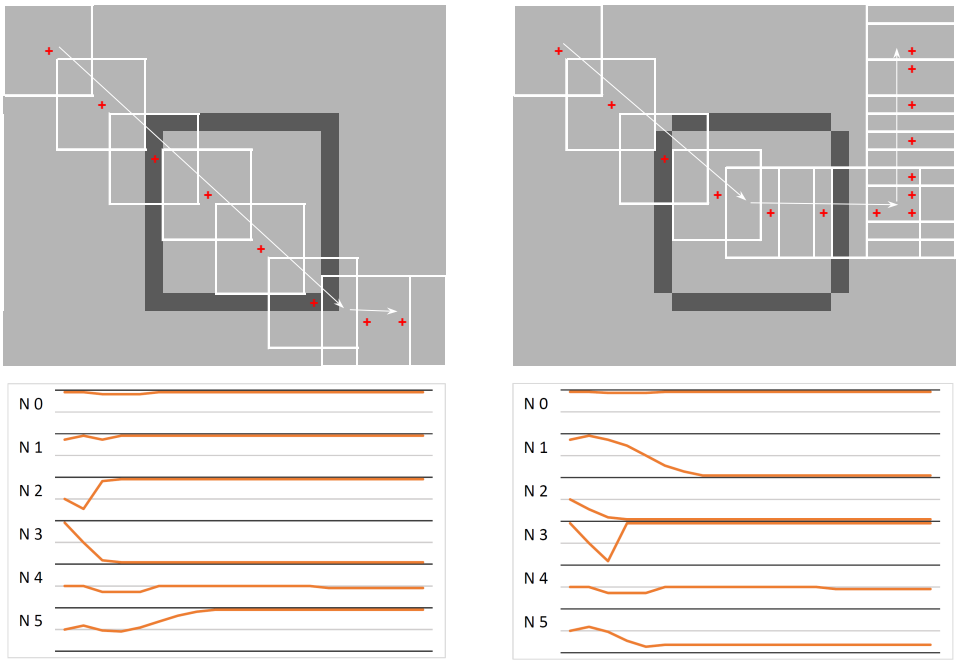

Figure 4.3: The first one of two different behaviors in the eight symbols task. On the top the artificial images along with the movement of the retina is shown. On the bottom the outputs of the 5 neurons are shown.

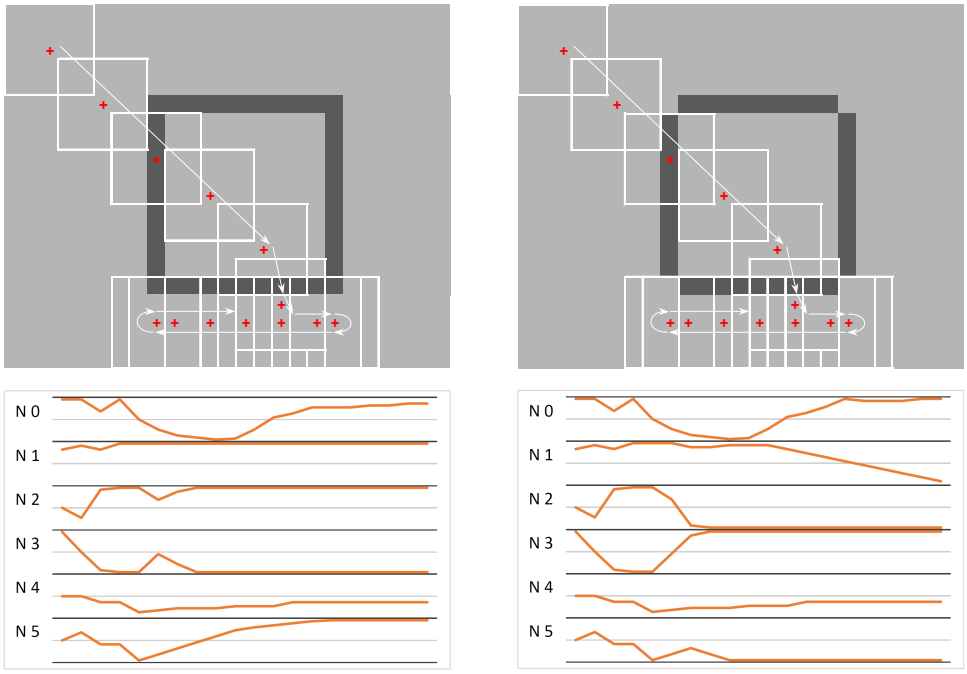

Figure 4.4: The second behavior is quite different. The retina now actively explores the bottom of the symbol.

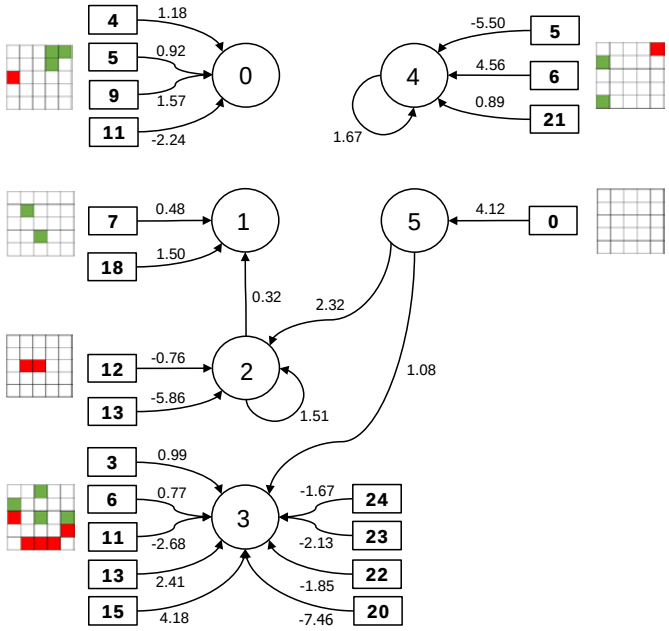

Figure 4.5: The RNN agent solving the binarized real image task. It consists of only 6 neurons and 25 weights.

The evolution of these RNN agents which solve the eight symbols task has shown, that very small and compact solutions are possible. These solutions would be very hard or even impossible to engineer. After these experiments many different solutions were found from which 30 were picked. These served as basis for the next task. Of course the real images are very different from the artificial ones, therefore the solutions presented in the next sections have very little in common with the ones presented above. Additionally there is now one more symbol, which should be encoded by neuron 4. Nevertheless, starting with a simpler tasks like shown in this section sped up the evolutionary process. It took 800 to 1000 generations on each task to find the final solutions presented in the next sections. Experiments without simpler task, starting the evolution directly with the original images have shown, that 60 percent of the experimental runs yielded no solution at all. In 40 percent it took 6000 to 10000 generations to achieve comparable results.

4.1.3 Binarized images of real symbols

From the solutions which were found from the previous task, the 30 solutions which solved the task with the highest accuracy yet by using the fewest neurons and weights were picked. These formed the first generation for the next task, described in this section. The real images were binarized as described in section 3.2.

One agent solving this task for the square and arrow is shown in figure 4.5. It is only slightly more complex than the agent solving the 8 symbols task. It has only 3 more weights. Its obvious, that the two agents have only very little in common. What seems necessary is the self connection of neuron 2, which makes it behave like a switch if a certain threshold of activity is reached (strong positive feedback). The connection from neuron 2 to neuron 1 is also similar, influencing the movement in y direction. Apart from that, everything has changed. Neuron 3 has now 12 inputs which seems to make analysis hard. Figure 4.6 shows though, that neuron 3 is always inactive for the two cases shown (a third case will be discussed below). Neuron 2 and 4 on the other hand lead to the decision for square or arrow. It is now discussed, what conditions lead to a positive activation of these neurons. The weight vector and the input vector of neuron are respectively:

$$w_{\mathrm{N2}} = \begin{pmatrix} -0.76 & -5.86 & 1.51 & 2.32 \end{pmatrix}^T$$

$$x_{\mathrm{N2}} = \begin{pmatrix} x_{12} & x_{13} & y_{\mathrm{N2}} & y_{\mathrm{N5}} \end{pmatrix}^T$$

Correspondingly for neuron 4:

$$w_{\mathrm{N4}} = \begin{pmatrix} -5.50 & 4.56 & 0.89 & 1.67 \end{pmatrix}^T$$

$$x_{\mathrm{N4}} = \begin{pmatrix} x_5 & x_6 & x_{21} & y_{\mathrm{N4}} \end{pmatrix}^T$$

Here the time dependency was omitted for brevity. The bias is zero (non existing weights) for both neurons (although the connection to neuron 5 serves as bias, as will be discussed below). The activations of neurons 2 and 4, a_{N2} and a_{N2} are the dot products of w_{N2} and x_{N2} and w_{N4} and x_{N4} respectively.

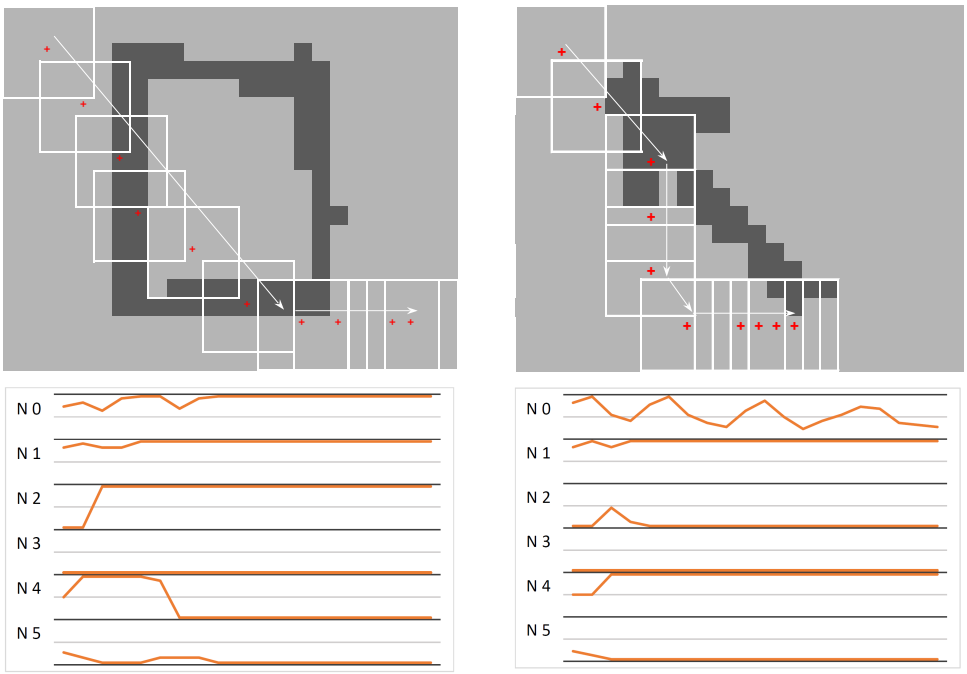

Figure 4.6: As an example, two binarized images of a square and an arrow are shown on top of this figure. Overlayed is again the movement of the retina. On the bottom, the activations of neurons 0 to 5 is shown.

$$a_{\mathrm{N2}} = -0.76x_{12} - 5.86x_{13} + 1.51y_{\mathrm{N2}} + 2.32y_{\mathrm{N5}}$$
$$a_{\mathrm{N4}} = -5.50x_{5} + 4.56x_{6} + 0.89x_{21} + 1.67y_{\mathrm{N4}}$$

To understand the behavior of the agent, the inputs can be examined. In the first time step, all input fields of the retina are showing the background color (0.49). As can be seen in the graph of the neural network in figure 4.5, neuron 5 has only one input coming from index 0. This input field represents the bias. Therefore it is always 1 and the output of neuron 5 is:

$$y_{\mathrm{N5}}(t) = \begin{cases} 0 & \text{if} \quad t = 0 \\ \tanh(4.12) \approx 0.99 & \text{else} \end{cases}$$

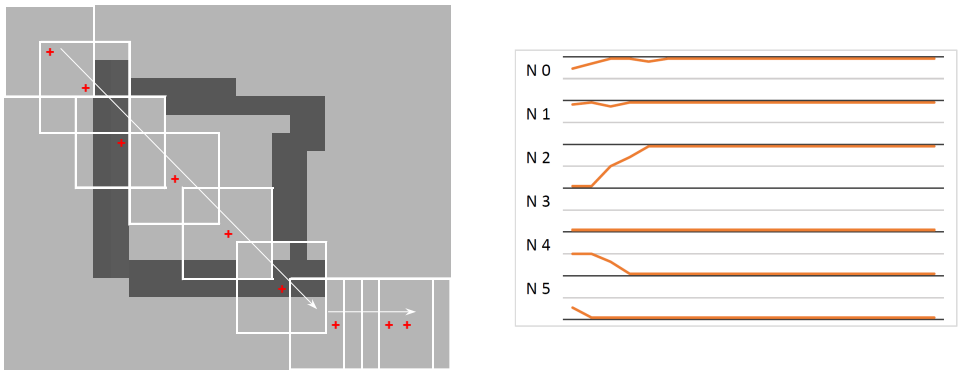

Figure 4.7: A second example of the retina movement and corresponding neuron outputs for a square image.

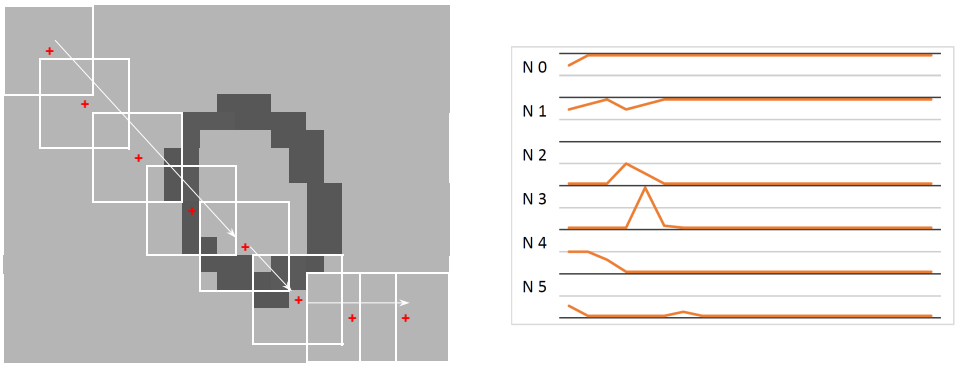

Figure 4.8: The retina movement and corresponding neuron outputs for a circle image. It shows a small spike at time step 4.

The activation after the first time step is therefore:

$$\begin{aligned} a_{\mathrm{N2}}(t=0) &= -0.76 \cdot 0.49 - 5.86 \cdot 0.49 + 1.51 \cdot 0 + 2.32 \cdot 0 \\ &= -3.24 \end{aligned}$$

The output is then

$$y_{\mathrm{N2}}(t=0) = \tanh(-3.24) \approx -0.99$$

In the next time step the input field 5 sees a darker pixel with a value of 0.18. Additionally the activation of neuron 2 from the last time step and the activation of neuron 5 now contribute to the activation.

$$a_{N2}(t=1) = -0.76 \cdot 0.49 - 5.86 \cdot 0.49 + 1.51 \cdot (-0.99) + 2.32 \cdot 0.99$$
$$y_{N2}(t=1) = \tanh(-2.44) \approx -0.98$$

In step 3 the inputs for neuron 2 are still the same for both cases square and arrow. The output of neuron 2 at $t = 2$ is:

$$a_{N2}(t=2) = -0.76 \cdot 0.49 - 5.86 \cdot 0.18 + 1.51 \cdot (-0.98) + 2.32 \cdot 0.99$$
$$y_{N2}(t=2) = \tanh(-0.62) \approx -0.55$$

In step $t = 3$, the decision is made. While both input fields of neuron 2 see darker pixels in the square image, they both see bright ones in the arrow image. The output of neuron 2 is:

$$y_{N2}(t=3) = \tanh(0.27) \approx 0.27$$

while for the arrow it is:

$$y_{N2}(t=3) = \tanh(-1.78) \approx -0.94$$

Now the self connection of neuron 2 determines the further development of its activity. The following table (5.1) shows the values for $y_{N2}(t)$ for the next few time steps. The pixel

t	square	arrow
4	0.90	−0.94
5	0.95	−0.98
6	0.99	−0.99

Table 4.1: Activations of neuron 2

at input fields 12 and 13 cause the decision and neuron 2 lands in two different attractors depending on the input image.

The same analysis can be done for neuron 4, which indicates the presence of an arrow. As can be seen in figure 4.5, the weights coming from the input fields 5, 6 and 21 are well balanced. The absolute value of the positive weights are almost the same as from

t	square	arrow
0	-0.02	−0.02
1	0.93	−0.07
2	0.91	−0.92
3	0.90	−0.91
4	0.90	−0.90
5	0.07	−0.90
6	−0.86	−0.90
7	−0.99	−0.90

Table 4.2: Activations of neuron 4

the negative ones. So if all input fields see the same input, the activation of this neuron stays the same as before. The self connection maintains the activation of the last time step. Additionally the weights of input fields 5 and 6 are very strong, so if their inputs are not the same, they force the neuron's output in positive or negative saturation. Table 4.2 shows the neuron's output for 8 time steps. As shown, as soon as input field 5 sees a dark pixel, the neuron's activation rises very fast. The activation is then maintained for 4 time steps in the square case and until the end in the arrow case. Input field 6 sees the dark pixel in time step 5 and 6 in the square case. One could assume, that this strategy would probably not work in case of a smaller square. But a careful analysis of various different cases showed, that several different strategies evolved in this rather simple neural network. An example is shown in figure 4.7, where the dark pixel at input field 21 gives the final push to negative activation of neuron 4.

Similarly there exist several strategies for the arrow image. In the given example above, the neural network depends on the absence of the dark left bar of the square.

Now the network for the circle output at neuron 3 could be analyzed in the same way. This neuron seems the most complicated one, since it has 10 different inputs. Unfortunately this neuron is almost never active. Even in the circle images is stays at negative activation levels. There are several cases though, where it shows activation spikes, as shown in figure 4.8. The outputs of neurons 2 and 4 are zero, so they correctly indicate the absence of a square and a circle but the output of neuron 3 also stays zero. It shows only one spike at time step 5. The weights of neuron 3 are very unbalanced with the sum of absolute values of the negative weights being almost double the amount of the positive ones. Therefore the neuron has a strong negative bias when all inputs see the same

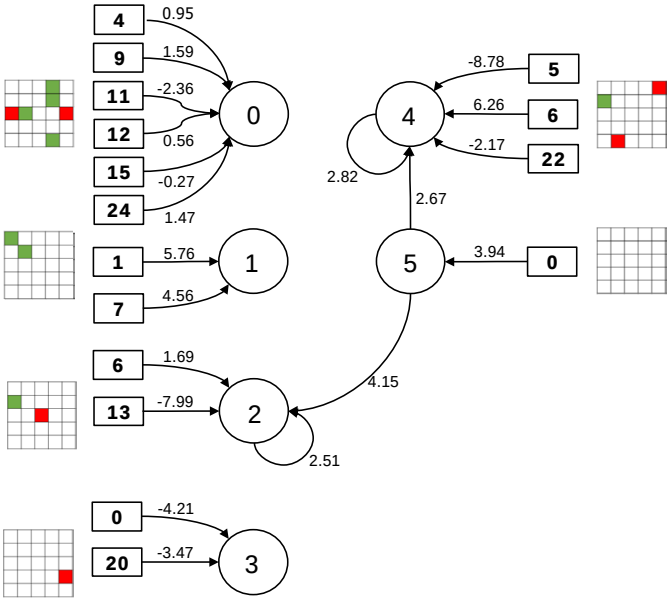

Figure 4.9: This recurrent neural network solves the real image task to a satisfactory degree. It consists of 6 neurons and 21 weights.

pixel values. So even if the neuron's activation switches to positive values triggered by a specific pixel arrangement like in the circle image, the activation can not be maintained. In the next section, it is discussed, how the network can be changed to solve the task successfully.

4.1.4 Original images of real symbols

Again 30 individuals were chosen to form the first generation for the next task. Now the agents see the original images and have to decide if there is a square, an arrow or a circle in the image. One agent, which is descended from the agent presented above is shown in figure 4.9. Some details like the structure of neuron 4 and its inputs remained the same, other parts are completely different. The rather complicated structure of neuron 3 almost entirely vanished leaving it only with the bias and one connection to input field 20. It will be shown, that this makes the recognition of a circle impossible. The number of connections of neuron 0 increased which results in a more "sophisticated" movement

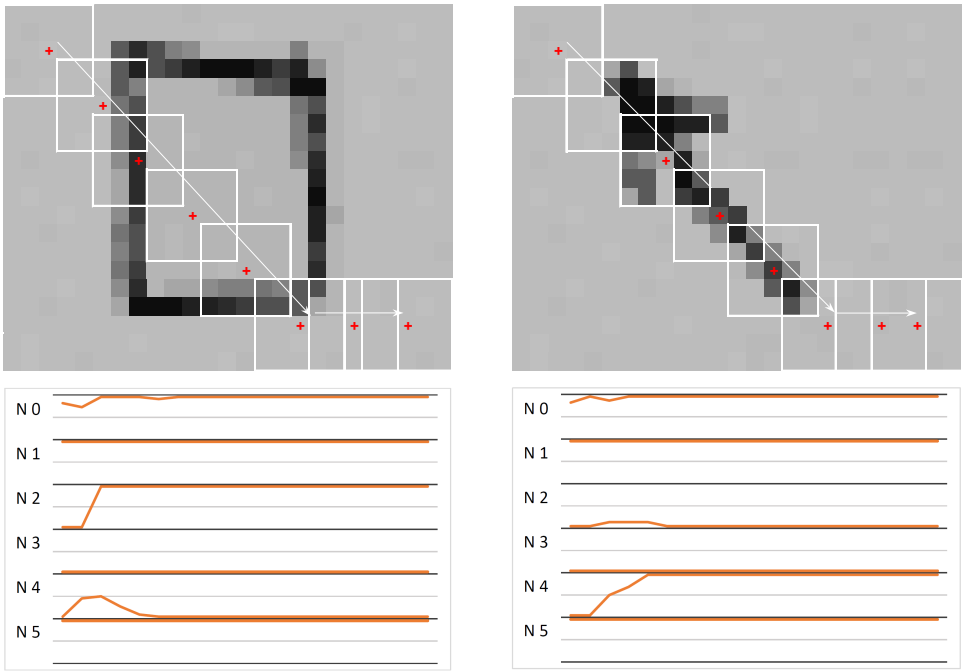

Figure 4.10: The final task consists of original images. The square and the arrow from the example before are used again for this network.

pattern. The same two example images as before, now without binarization, are shown in figure 4.10.

Compared to the binarized task, the movement of the retina is now less influenced by the pictured symbol. Only in the square case, the horizontal movement (which corresponds to the output of neuron 0) is slowed down once in the second time step. Apart from that, there is a straight movement from top left to bottom right with the decision for square or arrow made on the way. In the following paragraphs, the neural network is examined from a different perspective. Although the values of -1 and $+1$ cannot really be reached by the neuron's output, due to the tanh, in the next paragraphs, it is referred to these values as the minimum and maximum values of the output.

Looking at the temporal evolution of a given neuron taking various conditions into account (in this case the input values that come from brighter or darker pixels), the network dynamics can be better understood. From figure 4.11 it is obvious, that neuron 2 has 2 attractors depending on the inputs and on the current state. The temporal evolution of the neuron's output ends up in either -1 or $+1$. It is obvious that once the neuron

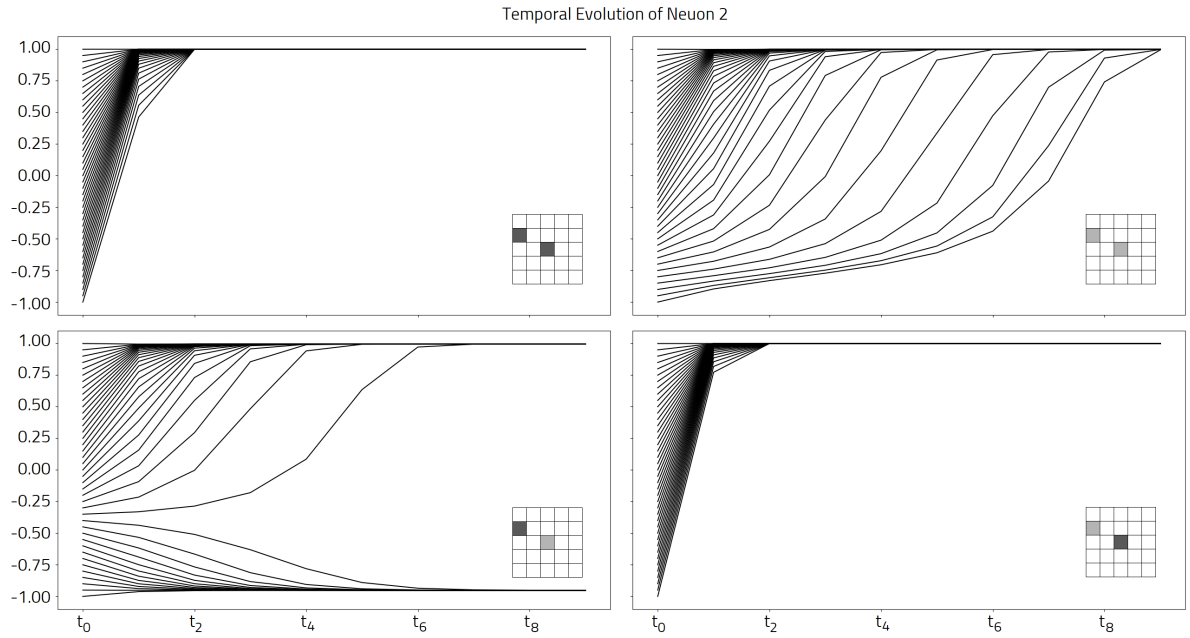

Figure 4.11: The figure shows the output of neuron 2 over the range of 10 update steps. These update steps are not starting at 0. This means, that the output of neuron 5 is already at 0.99. The different lines in the plots correspond to 21 different starting values ranging form -1 to $+1$ in steps of 0.05. The top two figures show the temporal evolution for the same pixel value on both input fields (dark on the left, bright on the right). The bottom left corresponds to a dark pixel at input field 6 and a bright one at input field 13 and vice versa is shown on the right.

is in the positive realm, it ends up in the $+1$ attractor. Only for the situation shown in the bottom left, a dark pixel at input field 6 and a bright pixel at input field 13, the activation can get negative and the neuron ends up in the -1 attractor. So only if the neuron's activation is already negative and it then sees this specific pixel configuration the -1 attractor can be reached. In figure 4.11 the first time step is omitted. Instead 21 possible activations are considered. But the first time step is important to understand the patterns in figure 4.10, since the activation of neuron 5 is 0 at this point. Therefore neuron 2 does not get the strong positive bias from neuron 5, which leads to the following activation:

$$\begin{aligned} o_{\mathrm{N2}}(t=0) &= \tanh(1.69 \cdot 0.49 - 7.99 \cdot 0.49 + 2.51 \cdot 0 + 4.15 \cdot 0) \\ &= \tanh(-3.09) \\ &\approx -0.99 \end{aligned}$$

In the next time step both the input fields see again bright pixels which lead to:

$$\begin{aligned} o_{\mathrm{N2}}(t=0) &= \tanh(1.69 \cdot 0.49 - 7.99 \cdot 0.49 + 2.51 \cdot () - 0.99) + 4.15 \cdot 0.99) \\ &= \tanh(-1.46) \\ &\approx -0.90 \end{aligned}$$

The value of -0.90 is therefore the starting point with which the further development using figure 4.11 can be examined. In the square image, input field 6 sees a bright pixel while input field 13 sees a dark one. This corresponds to the bottom right graph. Clearly the neuron ends up very fast in the $+1$ attractor. For the arrow image, input field 6 sees a darker pixel than input field 13, which keeps the activation in the negative realm. In the next step, it is the other way around. Now one would expect to see a strong positive activation. But since the values of the pixels are somewhere in the range between 0.15 and 0.52 for the original image (note that 0.18 and 0.49 for the binarized images were mean values), the behavior of the neuron is somewhere between the bottom left and the top right graph. This is exactly what can be observed in figure 4.10, where the output of neuron 2 gets a little bit more positive, but then goes back to the attractor in -1.

Unfortunately the behavior of this network for the circle image is even worse than the binarized version. As figure 4.9 shows, neuron 3 has only 2 inputs. The first one is a strong negative bias and the second one is a negative input from input field 20. Clearly this neuron can only be at negative one all the time. Since the neural network analyzed in this section is one of the more successful ones, further ideas to handle these tasks had to be developed. One other approach is described in the next section.

4.1.5 Results and discussion

The neural networks presented in this section were evolved using increasingly complex tasks. In a total of 70 experimental runs with an average of approximately 5000 generations, only 8 experiments ended with satisfying results. These 8 runs ended with networks which ranged in size between 6 neurons with 20 weights and 21 neurons with 92 weights. The success of these networks ranged between a 0.52 and 0.73 accuracy in predicting the right symbol in the test images. Most of the time they were able to detect 3 of the 4 symbols with very good accuracy but failed to detect the last one. A part of the remaining 62

experimental runs were already stopped at the binarized task, because their performance was not promising. But the majority did perform well on the "training" images but were not able to generalize to the test images. This problem could probably be solved by using a lot more training images to evolve the networks.

4.2 Strategy two: One network for each symbol

In the course of exploring the evolution of neural networks which can discriminate all 4 symbol types, namely square, circle, arrow and void (in the the rest of this thesis, the word void is used to describe the absence of symbols in the image). It was found, that only a few networks evolved, which solve this task to a satisfactory degree. As a matter of fact, the number of trials with no useful result exceeded the number of trials with useful results by a factor of nearly 10. One way to counteract this, would be to tune the hyperparameters. Another completely different approach is presented in this section.

For an learning scenario it would be very useful to be able to learn new symbols without the need to evolve totally new networks. To get around this, smaller networks which are specialized on only one symbol has been evolved. These specialized agents discriminate only one symbol against all others. In that way, if the agents are robust enough, new symbols can be added and only the agent responsible for recognizing this new symbol has to be evolved. For every new image, all agents give a vote on how much they "are certain" that their corresponding symbol is shown.

In the next paragraphs, 4 small networks are presented, which recognize void, square, arrow and circle quite well. Neuron 0 and 1 handle again the horizontal and vertical movement. Now only one additional neuron is required to indicate the presence or absence of the symbol at hand. Neuron 2 is dedicated to this task. So for all the following networks, a positive activation of neuron 2 indicates the presence of the symbol for which the network was evolved and a negative activation indicates its absence.

4.2.1 When no symbol is present (void)

The first case which will be analyzed is void, or the absence of any symbol. Figure 4.12 shows one of the agents solving the task perfectly. For all test patterns the output of neuron 2 either is $+1$ if no symbol is in the image or -1 if there is a symbol. Two

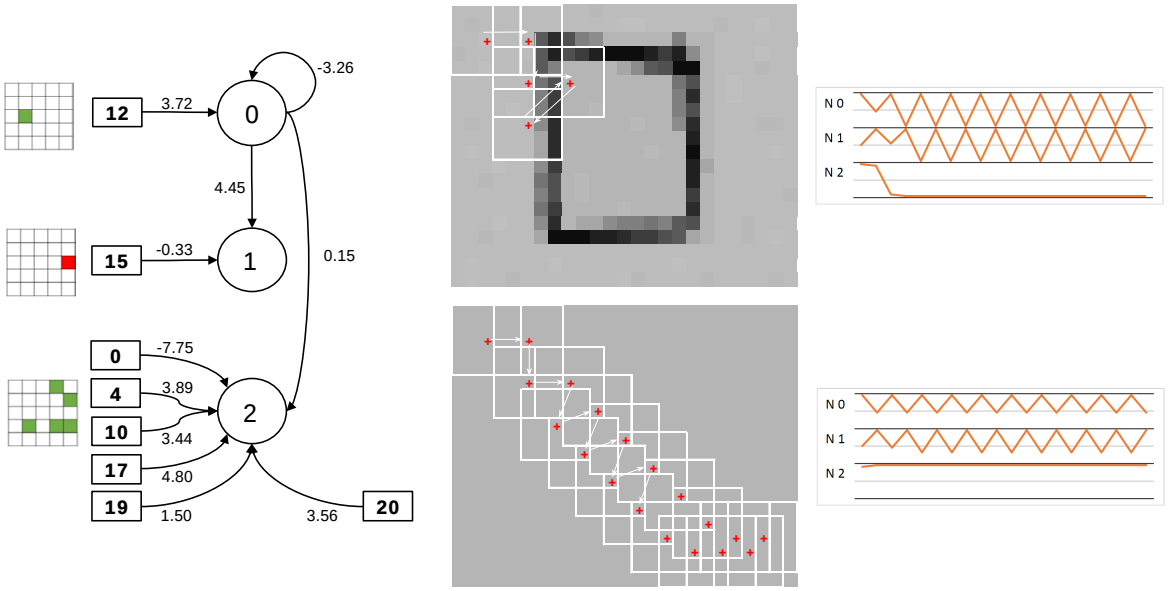

Figure 4.12: The neural network evolved to indicate the absence of any symbol is shown on the left. On the right are shown two corresponding typical movements of the retina and the outputs of the neurons. On the top right for a square image, which results in a negative output of neuron 2 and on the bottom void which results in a positive output.

example images are shown in the right part of the figure. Neurons 0 forms a simple oscillator by having a strong negative self connection and a positive input. The negative feedback connection leads to a switch from positive activation to negative activation and vice versa in every time step. The positive input keeps the oscillation between $+1$ and approximately -0.2 for bright pixels, which results in a movement to the right. When a dark pixel is found, this input is weakened so that the oscillation reaches negative one. This oscillating pattern is forced onto neuron 1 over a strong connection. This leads to the same, but phase shifted oscillation in neuron 1. This neuron also has a small input connection (negative in this case), which leads to an increase of the amplitude in negative direction. This can be seen in the bottom figure, where the output value of neuron 1 oscillates between 1 and approximately -0.3. Together this results in a movement from top left to bottom right with an oscillating "searching" pattern until some dark pixels are found. If dark pixels are found both neurons oscillate between $+1$ and -1 which leads to an oscillation of the retina between two positions. Neuron two has a strong negative bias and a lot of positive inputs. While on bright pixels, the inputs are much stronger

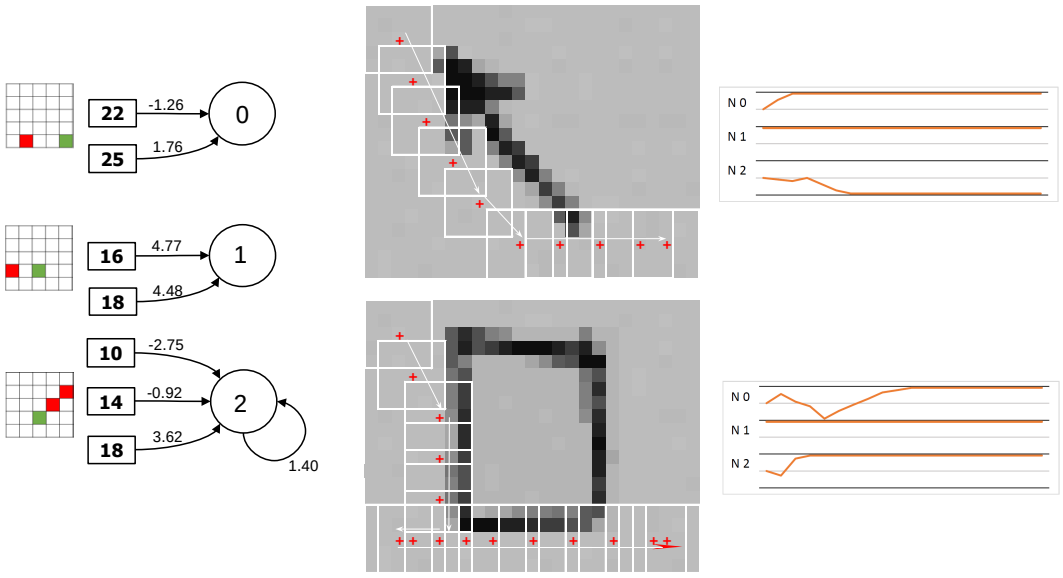

Figure 4.13: The RNN agent which scans the image and indicates the presence of a square with high activation of neuron 2. Again on the left side the network, in the middle to images with the movement of the retina and the corresponding neuron activations on the right.

than the bias, so the output is at +1. As soon as some fraction of the input fields see darker pixels, the bias gets stronger than the positive inputs and the activation switches to the negative side. Neuron 2 has no self connection, so the state cannot be maintained. Its activation is fully determined by the current inputs. Thus the whole functionality of this solution depends on the movement pattern to stay on the dark pixels of a detected symbol.

4.2.2 The square detecting network

The agent for the detection of a square on the other hand has exactly this self connection of neuron 2 which enables it to maintain its activation. The agent, along with two examples of its behavior, are shown in figure 4.13. The movement in vertical direction is always the same, since both inputs to neuron 1 are very strong positive. The movement in horizontal direction on the other hand depends on the symbol. With every time step the two connected input fields get the same input and the retina moves to the right. Only when the right input field (25) sees a pixel which is $\frac{1.76}{1.26} = 1.40$ times darker then the pixel at input field 22, the movement in horizontal direction stops. This is the case in

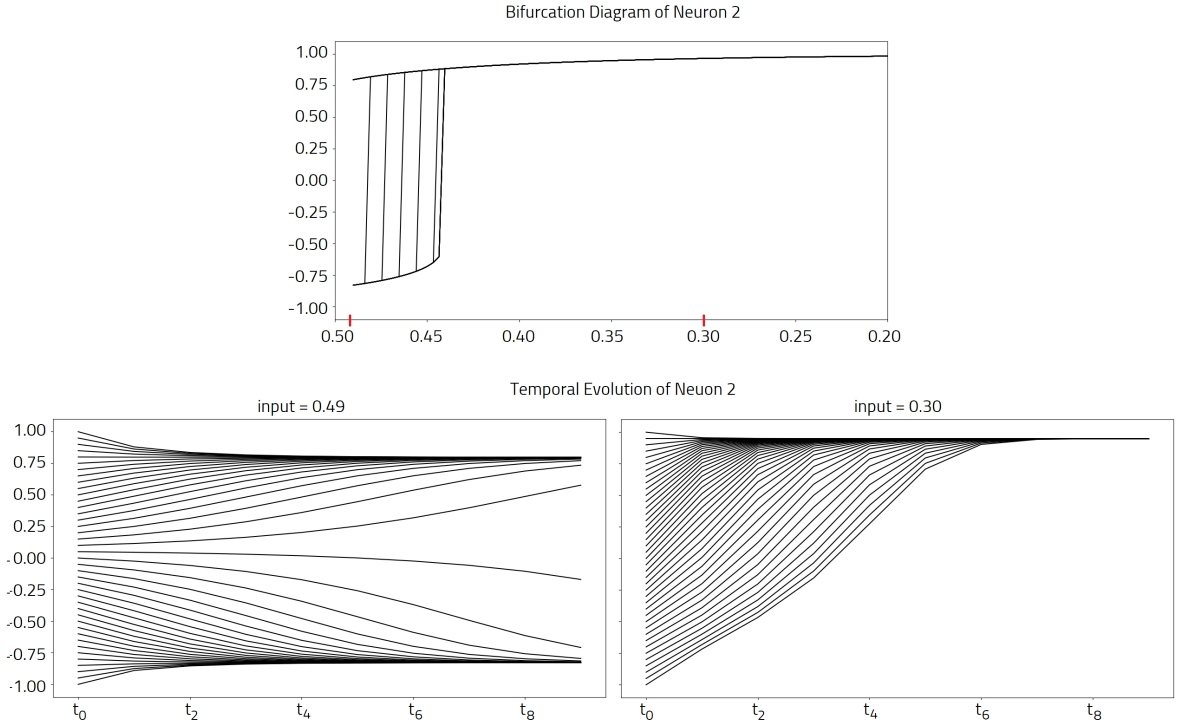

Figure 4.14: The figure shows the bifurcation diagram of neuron 2 with the pixel value on input field 10 varied from 0.49 to 0.18 on the x-axes. There are two attractors present until the pixel value of 0.44 below which only one attractor remains. The bottom figures show the phase diagram for different starting values of the neuron's activation with a pixel value of 0.49 in the left graph and 0.30 in the right one.

the square image. There is even a movement to the left, when the very dark pixel at the bottom left of the square is reached. The behavior of neuron 2 can be explained best by looking at phase and bifurcation diagrams. Figure 4.14 shows some examples. A good start is the graph in the bottom left, which shows the temporal evolution of the neuron's output when all input fields see bright pixels with value 0.49. Clearly there exist two attractors at approximately $+0.8$ and -0.8. When the neuron's activation starts at 0, it slightly transitions to negative activation over a long period of time. This can also be seen in the first few steps in the arrow image in figure 4.13. What happens, when the value of the pixel at input field 10 gets darker, can be seen in the top of figure 4.14. For pixel values of 0.49 down to 0.44 there exist two attractors, but below that threshold only one attractor survives and the output of the neuron ends up there. This is also shown in the bottom right for an example pixel value of 0.30.

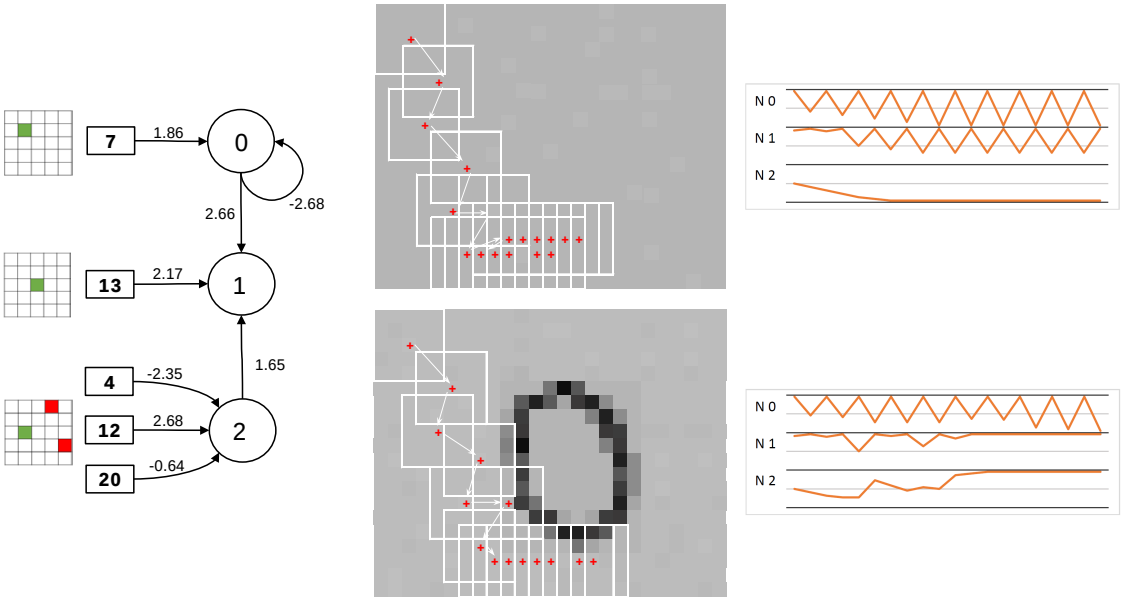

Figure 4.15: As before, this figure shows the neural network on the left side, two examples of the retina movement in the middle and the corresponding neuron activities on the right.

The figure would look very similar, if the pixel value of input field 14 is varied. Since the corresponding weight is smaller, the bifurcation happens at darker pixel values with the threshold at 0.30. In the square image after the third time step, both these input fields see dark pixels for a few steps. And when the retina moves horizontally at the bottom of the image, input field 10 sees dark pixels all the time, which leads to a fast transition to the positive attractor. It also works for much smaller squares, since the amount of dark pixel the retina hits always are enough. Only for a few cases, a circle is confused for a square, if the left side of the circle is drawn to straight, looking like a square.

4.2.3 The circle detecting network

The solution for the circle image (figure 4.15) is a descendant of the void solution presented above. The structure of the neural network is completely the same. Neuron 0 also has a strong negative self connection leading to an oscillating behavior. This oscillation is forced onto neuron 1, which also has one connection to an input field, although a positive one this time. As opposed to the void agent, where the negative input helped to increase

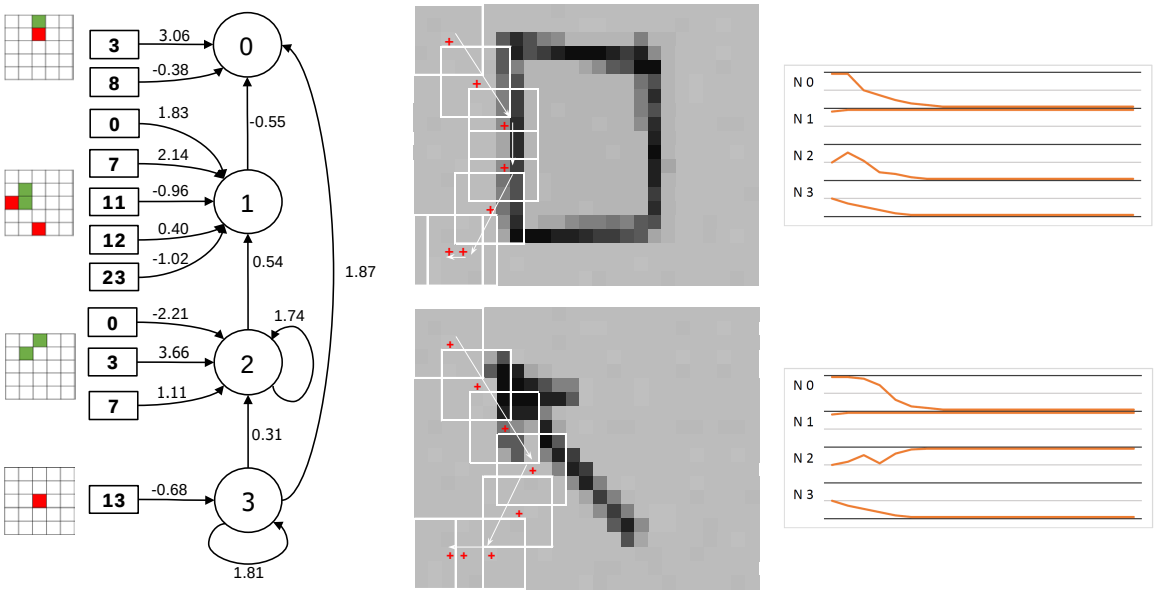

Figure 4.16: The neural network solving the arrow detection is shown on the left. The middle row shows two example images with retina movement and the corresponding neuron outputs are displayed on the right.

the amplitude of the vertical oscillation in negative direction, the positive connection in this case decreases the amplitude. It even suppresses the oscillation completely, when the amplitude of neuron 0 is too small (for example in the bottom right graph of neuron activations in figure 4.15). The activation of neuron 2 depends only on the inputs. When all pixel values it sees, are the same, its activation slowly transitions to negative saturation. When on the other hand a dark pixel hits input field 4 or 20, it slowly transitions to positive values. These two fields are positioned in such a way, that they are both at dark pixels when the retina is positioned at the left bottom of a circle. Its obvious, that this solution is easily confused, since a lot of symbols will lead to a positive activation of neuron 2. For the void case described above, this solution is sufficient, since any symbol should lead to a switch in activation, but here, this strategy is not working in many cases.

4.2.4 The arrow detecting network

The most complex network is the one solving the arrow task shown in figure 4.16. It consists of 4 neurons and 17 weights. Again, neuron 2 has its self connection and a

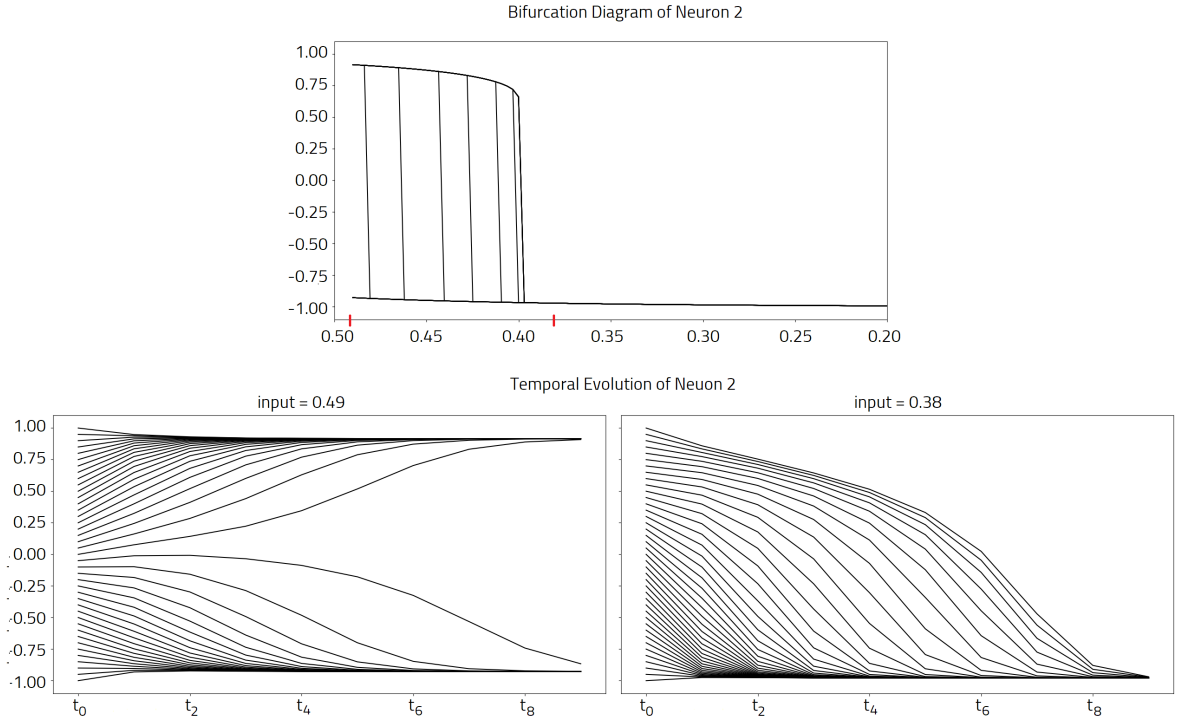

Figure 4.17: This figure shows again the bifurcation diagram of neuron 2. Here are also two attractors present until the positive one disappears around pixel value 0.39. In the bottom again the phase diagrams for different starting values of the neuron's activation with a pixel value of 0.49 in the left graph and 0.30 in the right one.

negative bias. Although it has the most weights and looks like it would have complicated dynamics, neuron 1 is the simplest to analyze. This is because it has a very strong positive bias and the sum of the positive weights is greater than the sum of the negative weights (absolute values). It therefore has always a positive activation around 0.9 or higher. The next neuron to look at is number 3. It has a small negative input connection and a self connection. It therefore acts as an integrator with the output slowly transitioning from 0 at $t = 0$ to negative one. It depends on the pixel values how fast this transition is, but it ranges between 4 and 7 time steps for all images used. The negative activation of neuron 3 influences neuron 2 and neuron 0. Since the output of neuron 1 is already determined, neuron 0 can now be analyzed. In the first time step, the strong positive weight from input field 3 dominates the neuron's activation. Beginning at time step 2, the inputs coming from neuron 1 and neuron 3 overwhelm the input values from the retina fields. With neuron 3 getting more negative, the activation of neuron 0 also transitions to negative

values. This results in the curve like movement of the retina first a few steps to the right and then the turn to the left again. The movement to the right stops earlier if the input fields of neuron 0 sees darker pixels, because then, the connections from the other neurons dominate even more. To understand the behavior of neuron 2, its again useful to look at the bifurcation diagram and some phase diagrams. Figure 4.17 shows three graphs as before, now associated with the arrow detecting agent. The phase diagrams in the bottom look almost exactly like the diagrams in figure 4.14, except that the remaining attractor in the bottom right diagram is now at negative one. Another more subtle difference can be observed in the bottom left diagram. Here the curve corresponding to a starting value of -0.05 first gets more positive before then moving towards the negative attractor. The maximum value of this curve is the point, where the influence of neuron 3 overwhelms the overall activation. As before, in the bifurcation diagram at the top one can observe, that there are two attractors for input values from 0.49 down to approximately 0.39. Below that, only the negative attractor remains. Compared to the square detecting agent, this agent has a bigger tolerance to darker pixels. Before, the threshold was at 0.44 which meant, that for only slightly darker pixel values than the very bright ones, the decision for the neuron to end up in the positive attractor was made. Here the pixels have to be much darker or they need to be presented longer for this decision. This can be seen on the course of the activations in figure 4.16. In the arrow image, input field 3 sees a dark pixel at time step 3, which results in a negative dip of the activation. But this is not enough to kick the neuron into the negative attractor. In the square image on the other hand, input field 3 sees dark pixels in at least two consecutive time steps, which seems to be enough, so that the output ends up in negative one.

4.2.5 Results and discussion

The networks presented in the above section were evolved to discriminate only one symbol against all others. This task is easier to solve and allows for more flexibility at the same time. The system can be more open ended, for example, if new symbols would be added to the discrimination task. The networks solving these tasks are smaller then the networks presented in section 4.1. They therefore need less computations (see also next subsection). The networks for detecting void, the square and the arrow are very good at their tasks. For the test set, they have accuracies of 0.91, 0.86 and 0.81 respectively. There are some

cases for the arrow detection network, where it confuses a square for an arrow, when the square is sheared to one direction. The circle detection network on the other hand is not satisfactory at all. As mentioned above, the strategy to depend only on the movement and the occurrence of dark pixels at the ideal positions is too error-prone. For a lot of circles in the training images, this strategy worked, but in the test set, a lot of false positives occurred for squares and even some arrows. This network is still one of the best performing ones with an accuracy of 0.72.

4.2.6 Evaluation of computational complexity

One important aspect in developing algorithms for robots is the computational complexity. In this section the amount of computations needed is estimated and compared to a feed forward neural network which was trained using the backpropagation algorithm. The neural network agents, which were presented in this chapter are very small and lightweight with respect to the number of required computations. To decide what symbol is shown, all 4 networks were evaluated for 20 time steps. Table 4.3 lists the number of neurons and weights for these networks. The 20 time steps are a very high number, since in all

Network	No. of neurons	No. of weights
void	3	11
square	3	8
circle	3	8
arrow	4	17

Table 4.3: The number of neurons and weights of the networks presented in the last sections

experiments, the decision was made after 5 time steps on average. Only the circle network needs up to 12 time steps. By stopping the updates when the network is in an attractor, and the output does not change anymore, the necessary time steps can be reduced. This would probably lead to an average of 10 time steps. The total amount of computations for one time step include one multiplication and one addition for every weight and one tanh for every neuron. Multiplied by 10 time steps the following numbers of computations are needed.

$$\begin{aligned} \text{No. of multiplications} &= 10 \cdot 44 = 440 \\ \text{No. of additions} &= 10 \cdot 44 = 440 \\ \text{No. of tanh} &= 10 \cdot 13 = 130 \end{aligned}$$

To compare the number of computations to a standard fully connected neural network, different topologies were trained and evaluated. There was no topology found, which gave comparable results. No feed forward network could classify all test symbols correctly. One network with an accuracy of 0.78 had 500 inputs, 100 hidden units in the first layer and 20 hidden units in the second layer and 4 output units. These values result in a total number of neurons of 124 and a total number of weights of 52080. To evaluate one image, this approach needs the following computations.

$$\begin{aligned} \text{No. of multiplications} &= 52080 \\ \text{No. of additions} &= 52080 \\ \text{No. of tanh} &= 124 \end{aligned}$$

While the evaluation of tanh has to be performed just as often as in the present approach, the multiplications and additions are higher by a factor of about 120. Of course this is only a very broad comparison, since the feed forward network does not perform so well. Using a convolutional network would fit the task way better. Furthermore there was no regularization, no dropout and no sophisticated loss function implemented. Another reason for the poor performance of the feed forward network is certainly the small number of training images.

Nonetheless, the presented approach using small recurrent neural networks in an active vision setting clearly is advantageous regarding computation time. Additionally they seem to generalize well after using only 60 "training" images for their evolution.

5 Real world application in the learning scenario

As was described in the introduction of this work, the evolved recurrent neural networks can be used in a learning game played by a human and a humanoid robot. This chapter describes the framework for such a learning game and how the neural networks can be used in it. Although the system was developed and is running on a humanoid robot, for the framework which will be described here, the robot is not necessary. A camera is sufficient.

5.1 Game setting and game script

The description of the bigger picture in the introduction assumes a learning game for a human teaching a robot something on a whiteboard. In a reduced setting, a first step to this bigger picture was implemented in form of a game. The game process is detailed in figure 5.1. The robot (or a camera) is positioned in front of a whiteboard, such that the image consists of the area of the whiteboard where the human teacher is about to draw some symbols. The different situations of the game setting are handled using state machines. In the following paragraphs, these situations are described in more detail.

State 1: Game start

In the beginning the image coming from the camera shows a white background. The start of the game is initiated by initializing the game state machine. This can be triggered using the keyboard, but can also be triggered by another external event like a voice command. When the game starts, it is assumed, that only the background is in the image. Everything that is already on the whiteboard will be treated as such. This allows elements, like a

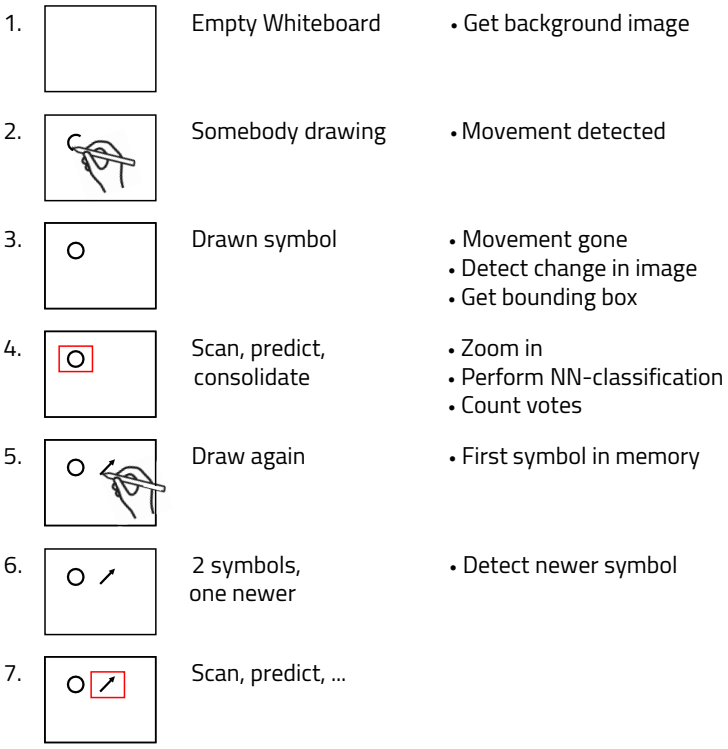

Figure 5.1: The process of the learning game. On the left side is sketched the situation the camera sees. In the middle are key points describing the situation and on the right side the procedures the system is using.

colored whiteboard eraser, magnets or the colored surrounding wall to be in the image. These things will be treated as background and do not disrupt the game unless they are altered in some way. The first step is then to save the current image as background image.

State 2: Drawing a symbol

Now the teacher enters the stage and draws a symbol on the whiteboard. This situation is handled as a waiting state, as the symbol is not yet ready to be examined. The waiting state is implemented by using motion in the image as indicator, that something is happening. When the motion stopped, the wait state will be left to go to the analysis of the symbol.

The motion detection is realized using difference images. The camera image is updated with $12.5Hz$. Every pixel $I_{t,i}$ of the new image I_t is compared to the corresponding pixel

$I_{t-1,i}$ in the last time steps image I_{t-1} and the difference is accumulated. This gives the total difference $D_{t,t-1}$:

$$D_{t,t-1} = \sum_{i=N} d(I_{t,i} - I_{t-1,i}) \tag{5.1}$$

where the difference in one pixel $d(p_1, p_2)$ is the sum of the sum of absolute differences of all color channels:

$$d(p_1, p_2) = abs(p_{1,Y} - p_{2,Y}) + abs(p_{1,U} - p_{2,U}) + abs(p_{1,V} - p_{2,V}) \tag{5.2}$$

As long as the total difference $D_{t,t-1}$ is bigger than some threshold, the situation is classified as 2. in figure 5.1. When the movement is gone, that means, the difference falls below that threshold, the system stays in the waiting state for another 2 seconds and then transitions to the third state.

State 3: Finding the symbol on the whiteboard

In this state, the current image is again still and the change in this current image in relation to the background image, saved in step 1, is found. This is done via the difference in pixel values. All pixels that changed significantly are marked as foreground. Using these pixel values, the bounding box can be found. Assuming the set of pixels P containing all pixels belonging to the symbol, the bounding box is found by setting the top left corner with the minimal x and the minimal y coordinate (which not necessarily come from the same pixel) and the bottom right corner with the maximum x and the maximum y coordinate. These coordinates are used to configure the pixelpipeline to zoom in and get an image of the symbol as shown in the previous chapters.

State 4: Classification of the symbol

Now the neural networks developed in this work are used to classify the symbol. All networks of strategy two described in the last chapter get 20 time steps to "look" at the image. For evaluation a simple voting system was implemented: For every time step in which the output value of neuron 2 is higher than 0.7, the corresponding symbol categories votes are increased by 1. This is done for all (current) 4 networks. The symbol category

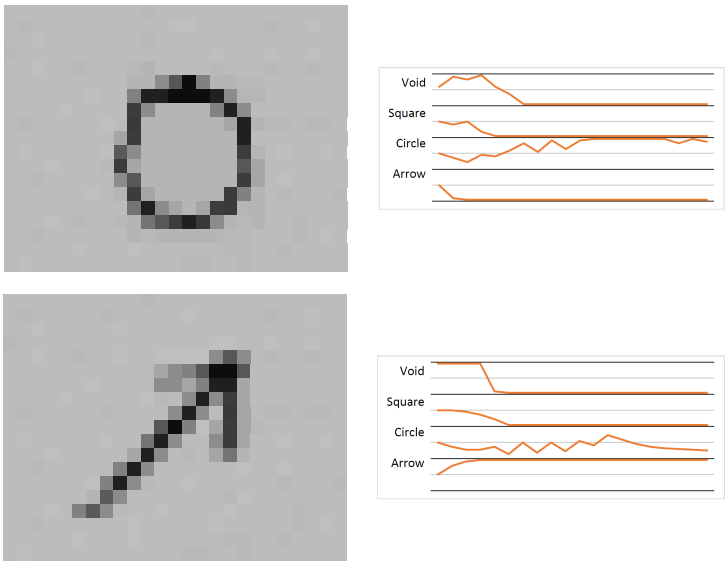

Figure 5.2: Two example images with the activities of neuron 2 for the 4 different classifying networks.

with the most votes (highest number) wins. This means, the longer one neural network indicates the presence of its corresponding symbol by high activation of neuron 2, the more votes it collects. Networks which are sure that there symbol is present will stay in high activity and get a high voting. On the other hand if the network is not sure, its activation could oscillate between high and low activity and would therefore get a lower voting. One example is shown in figure 5.2 for a circle on top and an arrow at the bottom. The corresponding votes are listed in table 5.1 in columns *circle 1* and *arrow*. The votes show a clear tendency to the right answer in both columns. The votes give

Network	circle 1	arrow	circle 2
Void	3	5	3
Square	0	0	16
Circle	9	0	13
Arrow	0	17	0

Table 5.1: Votes of the networks

a correct answer for 64 of the 68 test symbols. This gives an accuracy of this combined system of 64 out of 68, which is 0.94. In figure 5.3 one case is shown, where the answer is not correct. The votes are listed in 5.1 in column *circle 2*. Admittedly, it is not a nice

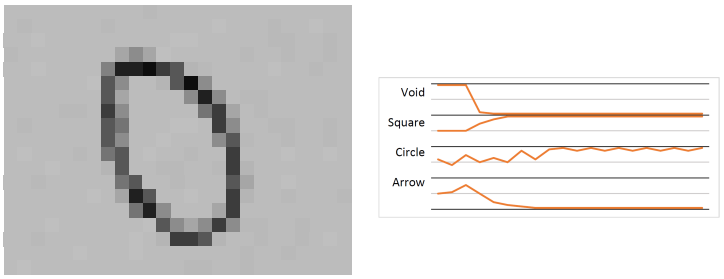

Figure 5.3: One example image with the outputs of the 4 networks. Clearly this voting failed as the square network gets a high vote, whereas the circle network gets a lower vote.

circle, but here the confusion between network *square* and network *circle* is obvious. As described in the above chapters, the circle network is easily fooled. It is not sure but manages to oscillate itself into the positive activation and therefore collects only 13 votes. The square network on the other hand goes to high activation almost immediately and collects 16 votes. Here a mechanism to handle these close cases could solve the problem but is left for future work.

States 5 to 7: How the game proceeds

After the voting, the system returns to a waiting state. Here it waits again for a movement to occur and to stop again, which will trigger the whole process again. Currently, when returning to the waiting state, the current image is saved as background. In that way, new symbols can be drawn and only the new ones get analyzed. In this state, a memory process can be included (see section 6.2) which allows for a higher cognitive process to use the information. States 6. and 7. have then the same steps as described above.

6 Summary and future work

This section summarizes the thesis by revisiting the main topics and results. It then gives some ideas for future work including new ideas and suggestions for further development.

6.1 Summary

This thesis began with the picture of a humanoid robot in front of a whiteboard. In an learning scenario a human teacher draws symbols on this whiteboard explaining something and the robot is following the teacher. For realizing such scenarios, a module for processing the drawn symbols was developed in this thesis. This was done using recurrent neural networks as dynamic controllers. The networks determine the symbols shown in an image through an active vision process. The attractor landscape of these networks leads to a decision for a symbol, depending on the input image. To identify capable networks, an artificial evolution procedure was implemented. Two different strategies have been pursued to solve this task.

In the first strategy, neural networks which can distinguish between all symbols have been evolved. It was found, that these networks are hard to evolve, even by using increasingly more complex tasks to evolute gradually. The accuracy of the best performing networks reached 0.73 for the test images. To increase this rather unsatisfactory accuracy a second strategy was developed and tested.

The second strategy consisted of evolving four different networks. One network for each symbol and one for the absence of symbols (plain background). This approach yielded better accuracies between 0.81 and 0.91 for the detection of void, squares and arrows respectively. The agent detecting circles only achieved an accuracy of 0.72.

The 4 agents evolved in the second strategy were then integrated in a real life application. The outputs of all networks are computed and the decision for a symbol is made by

a voting mechanism, in which the outputs higher than 0.7 are counted. With this voting mechanism, a learning scenario has been implemented.

The overall computational complexity of this voting mechanism was compared to a feed forward neural network trained on the same task (accuracy of 0.68). While both approaches have a comparable number of tanh (or any other nonlinearity) executions, the number of multiplications and additions of the feed forward network is 120 times higher then for the voting mechanism. This is of course due to the high number of weights involved. The usage of recurrent neural networks in an active vision scheme as described in this thesis is therefore advantageous, because it needs considerably less computation and therefore less energy while performing even better.

6.2 Future work

The neural networks for the detection of void and for the circle depend only on the visual input and are therefore not very robust. As was mentioned in section 4.2.1, the correct output of the network depends highly on the movement pattern of the retina. If there is a symbol present, the retina needs to stay on the symbol for neuron 2 to have low activity (which is the correct response). This behavior would be way more robust, if neuron 2 had a self connection. Then, the current state of neuron 2 could be maintained and a detected symbol would push neuron 2 in the negative attractor where it would stay, even if the retina is not on the symbol anymore.

Similarly for the circle network, a self connection of neuron 2 could be advantageous. To identify and proof possible advantages of the self connection mechanism more experiments are necessary. Surprisingly, the solution presented in 4.2.3 had the highest accuracy in all experiments. It by no means a robust solution. With some more evolutionary runs, a more accurate and robust solution can probably be found.

The learning scenario presented in chapter 5 can be extended in the following way: In the current implementation, the image at game start is taken as background image. One extension would be, to recognize already drawn symbols. This extension can be implemented easily, since most pixel have the background color (white) and only a few pixel deviate from this color. The color information would be sufficient to create a bounding box and look closer. Another additional skill would be the tracking of lighting conditions. Depending on the day time and light sources, the colors (also the white of the background)

are considerably different. Over time, the robot could memorize different conditions and adapt to them appropriately. In the current implementation, every time a symbol was recognized, the current image is taken as background. A memory would be helpful to know about the different symbols drawn at different points in time. This would enable higher cognitive processes to reason about the temporal and causal relations.

For the learning scenario to be open ended it would also be desirable, if the teacher could choose to introduce new symbols when needed. When a so far unknown symbol is drawn, the voting should result in a low activation of all networks. Then it would be clear that none of the three already known symbols is present and there is no void either. The low activation of all networks could trigger a new evolutionary run, with the new symbol as target. At least 5 to 10 images of a new symbol are needed. Using different settings for the pixelpipeline, the robot can obtain these different images of the new symbol. Another approach would be that the teacher introduces a new symbol by drawing it a few times. With these images a new neural network can be evolved using the already present images of the other symbols as non target examples. Then the voting mechanism also needs to be adapted.

To make the learning scenario truly interactive and more flexible, the possibility of feedback should be integrated. For example if a symbol was incorrectly categorized, the human teacher could correct the categorization.

List of Figures

List of Tables

Bibliography

[Abraham and Shaw, 1992] Abraham, R. and Shaw, C. (1992). *Dynamics: The Geometry of Behavior*. Addison-Wesley.

[Agmon and Beer, 2014] Agmon, E. and Beer, R. D. (2014). The evolution and analysis of action switching in embodied agents. *Adaptive Behavior*, 22(1):3–20.

[Bajcsy, 1988] Bajcsy, R. (1988). Active perception. *Proceedings of the IEEE*, 76(8):966–1005.

[Bajcsy et al., 2018] Bajcsy, R., Aloimonos, Y., and Tsotsos, J. K. (2018). Revisiting active perception. *Autonomous Robots*, 42(2):177–196.

[Beer, 2003] Beer, R. D. (2003). The dynamics of active categorical perception in an evolved model agent. *Adaptive behavior*, 11(4):209–243.

[Beer and Williams, 2015] Beer, R. D. and Williams, P. L. (2015). Information processing and dynamics in minimally cognitive agents. *Cognitive science*, 39(1):1–38.

[Di Paolo et al., 2017] Di Paolo, E., Buhrmann, T., and Barandiaran, X. (2017). *Sensorimotor life: An enactive proposal*. Oxford University Press.

[Dörner, 1999] Dörner, D. (1999). *Bauplan für eine Seele*.

[Floreano et al., 2004] Floreano, D., Kato, T., Marocco, D., and Sauser, E. (2004). Coevolution of active vision and feature selection. *Biological cybernetics*, 90(3):218–228.

[Haykin et al., 2009] Haykin, S. S. et al. (2009). *Neural networks and learning machines/Simon Haykin*. New York: Prentice Hall,.

[Heinrich and Wermter, 2018] Heinrich, S. and Wermter, S. (2018). Interactive natural language acquisition in a multi-modal recurrent neural architecture. *Connection Science*, 30(1):99–133.

[Nolfi et al., 2000] Nolfi, S., Floreano, D., and Floreano, D. D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press.

[Pasemann et al., 2003] Pasemann, F., Hild, M., and Zahedi, K. (2003). So (2)-networks as neural oscillators. In *International Work-Conference on Artificial Neural Networks*, pages 144–151. Springer.

[Strogatz, 1994] Strogatz, S. (1994). *Nonlinear dynamics and chaos*. Addison-Wesley.

[Tani, 1996] Tani, J. (1996). Model-based learning for mobile robot navigation from the dynamical systems perspective. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(3):421–436.

[Tani and Nolfi, 1999] Tani, J. and Nolfi, S. (1999). Learning to perceive the world as articulated: an approach for hierarchical learning in sensory-motor systems. *Neural Networks*, 12(7-8):1131–1141.

[Williams et al., 2008] Williams, P. L., Beer, R. D., and Gasser, M. (2008). An embodied dynamical approach to relational categorization. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 30.

[Wurtz, 2015] Wurtz, R. H. (2015). Brain mechanisms for active vision. *Daedalus*, 144(1):10–21.

[Yamashita and Tani, 2008] Yamashita, Y. and Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment. *PLoS computational biology*, 4(11).