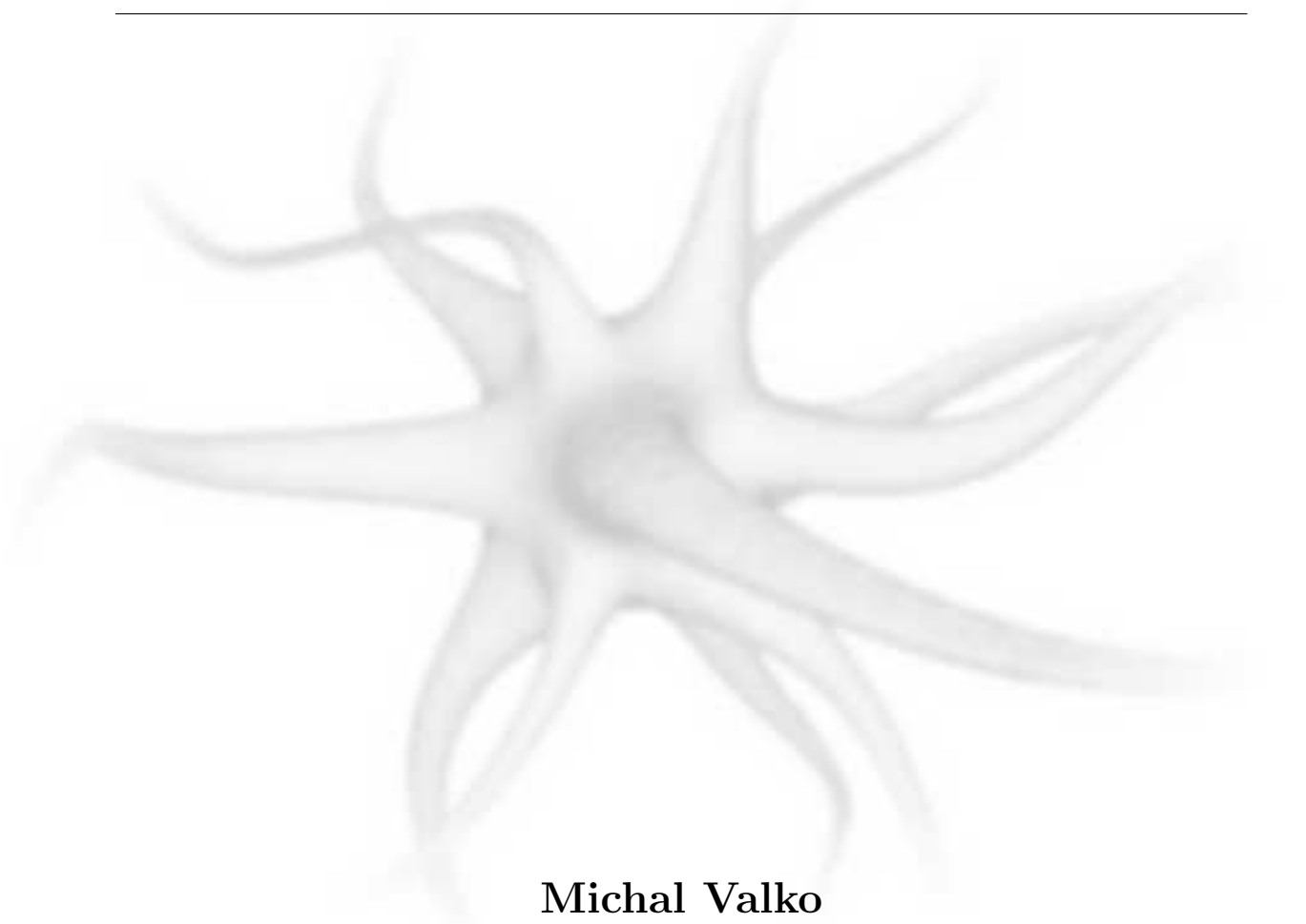
Evolving Neural Networks for Statistical Decision Theory

Michal Valko

Comenius University Bratislava, Slovakia
Faculty of Mathematics, Physics and Informatics
Department of Applied Informatics



Michal Valko

Evolving Neural Networks for Statistical Decision Theory

Advisor: Radoslav Harman

master thesis

Bratislava & Lisbon

April 29, 2005

Acknowledgements

I want to express sincere thanks to my advisor Mgr. Radoslav Harman, PhD. for his careful supervision, stimulating discussions and inspiring advices. I would also like to thank Doc. MUDr. Juraj Pavlásek, DrSc. for introducing me to addressed domain and Nuno Cavalheiro Marques, PhD. for his comments. Finally, I am very grateful to my family and friends for supporting me during all my studies.

Contents

Chapter 1

Introduction

G. B. Shaw: "If you have a good brain, devote yourself to statistics. If not, devote yourself to politics or theater."

Real biological networks are able to make decisions (Mitchell T., 1997). We will show that this behavior can be observed even in some simple architectures of *biologically plausible* neural models (Pavlasek et al., 2003). The great interest of this thesis is also to contribute to methods of statistical decision theory by giving a lead how to evolve the neural networks to solve miscellaneous decision tasks.

In particular, we want our networks to solve decision making task over spike (temporal) trains. As an illustration of practical utilization consider the following: We have a source input with a sequential access to it and our goal is to decide whether the input data satisfy the given constraints, such as 'Is the data–set from a Poisson distribution?'. Network architectures can be also used for comparative decision making: we have two sequential inputs and we are to decide, which one is more frequent, more variable or regular. Not all of these problems have the statistical hypothesis testing algorithms that give us an answer at a chosen significance level according to the features of the input set (Schervish, 1995).

We present and demonstrate a method that deals with this type of tasks. It is based on developing and evolving an artificial, but biologically plausible, tasks–specific neural network. After the evolution, the evolved network is taken as a decision maker and its output is taken as a decision. A beneficial feature of simple plausible models is that we can uncover and explore the information processing.

Unfortunately, the construction of a NN with a given task is not obvious even for the well defined optimization problems (Hopfield and Tank, 1985). That is why we use an evolution to do it for us. Using a genetic algorithm as a replacement for back propagation does not seem to be competitive with the best gradient methods, but it promising when gradient or error information is not available (Schaffer et al., 1992). The difference between the classic statistical and this approach is that we obtain an *adaptive* decision tool: the network can respond faster to simpler inputs. Such an information processing is not (besides the learning phase) computationally hard. For these reasons, this approach also deserve an attention of real–time controllers constructors (Pham and Liu, 1995).

Of course, all the desired properties must affect a fitness function, an optimality quantifier of a concrete NN in a population of NNs. (Kvasnicka et al., 2000) The better decision maker is evaluated with a higher fitness — an ability to survive and breed. Thus the "better genes" are spread into the population (hopefully) providing a better generation (Mitchell M., 1998). After evolution, the best NN in the population is supposed to be good enough for decision maker for the given task.

In the next chapter we will provide a framework of this thesis: the chosen NN model and the probabilistic distributions for the input data. When using genetic algorithms to evolve the required NN, we have to consider the chromosome coding, direct/indirect coding, select the model parameters that will be subject to evolution, evaluation of the chromosome, the fitness function, the way of the crossover and the type of the genetic algorithm. We offer a detailed description of the used methods in chapter 2 and 3. The results on the selected tasks are presented in chapter 4. Conclusions and indications for further work are given in chapter 5.

Chapter 2

Methods

This chapter provides an overview of used methods. Evolution and genetic algorithms are entered into details separately in chapter 3.

2.1 JASTAP Model

2.1.1 Why this model?

The reason why we have chosen this model is its biological *plausibility*. It is a spiking neuron model and it respects physiological aspects of a biologically realistic neuron. Next, it is better configurable than the standard NN models. Moreover, it can describe the ability of the human brain to make decisions. In other words, with this model we will show how can a simple "wetware" solve some statistical problems. The results have shown that in some cases we are able to *decode* what does the evolved network do.

2.1.2 Description

A brief description of JASTAP model follows. The details can be found in (Janco et al., 1994). Every JASTAP model is an artificial NN, which consists of JASTAP neurons (neuroids[1]) as the basic elements. A neuron is described with:

[1] we will use the term "neuron" when referring to the modeling element as well

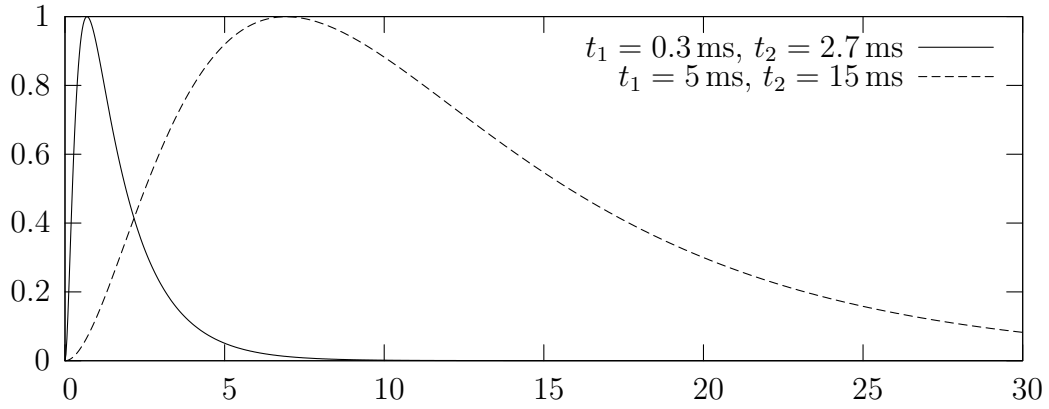


Figure 2.1: Postsynaptic potential

▷ SET OF SYNAPSES The neuron is interconnected with its environment by one or more synaptic inputs and a single output (axon). The output can be connected with one or more neuron synapses in the network.

For each synapse we consider:

- INPUT — can be internal (connected with axon of other neuron) or external (from the outer environment)
- SHAPE OF PSP[2] PROTOTYPE — the waveform evoked by a spike arriving at a synapse is described by

$$\text{PSP}(t) = k \cdot \left(1 - \mathrm{e}^{-\frac{t}{t_1}}\right)^2 \cdot \mathrm{e}^{-\frac{2t}{t_2}} \tag{2.1}$$

  The waveform inter alia emulates whether the synapse is located on a soma or on a dendritic tree. Parameters t_1 and t_2 can vary from synapse to synapse. They determine the potential decay. Redman and Walmsley (1983) mention $t_1 = 0.3\,\text{ms}$ and $t_2 = 2.7\,\text{ms}$ as shown in figure 2.1. As the neuron carries out the time–and–space summation of the input potentials, more moderate decay can cause[3] more sophisticated information processing. Therefore, we used t_1 up to $5\,\text{ms}$ and t_2 up to $15\,\text{ms}$ (figure 2.1).
- LATENCY — the time delay of the synaptic transmission and the axonal conduction.

[2] postsynaptic potential
[3] one should take into account the time discretization during simulation

- SYNAPTIC WEIGHT — a value from $\langle -1, 1\rangle$. This number represents the strength of the synaptic input. We distinguish an excitatory PSP from a synapse with a positive SW[4] and an inhibitory PSP from a synapse with a negative SW.
- PLASTIC CHANGES — Depending of the synapse type, SW can be influenced by some mechanisms, for instance Hebbian learning or heterosynaptic presynaptic mechanism. We do not use this feature of the model in this work, because the learning during information processing phase is not our goal.

▷ INSTANTANEOUS MEMBRANE POTENTIAL — a quantity within the $\langle -1, 1\rangle$ range, determined as the sum of PSPs limited by the non–linear function (figure 2.2)

$$\mathrm{MP}(t) = \frac{2}{\pi} \cdot \operatorname{atan}\left(\sum\nolimits_{\text{synapses}} \mathrm{PSP}(t)\right) \tag{2.2}$$

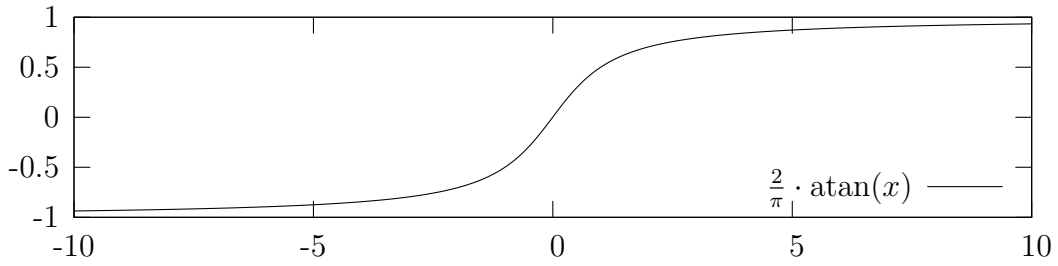


Figure 2.2: Limiting non–linear function

▷ THRESHOLD — θ, a value from $\langle 0, 1\rangle$ — determines the limit for firing

▷ SPIKE FREQUENCY — the spike frequency is restricted by the absolute refractory period. This is managed by setting minimum $I_{\min}$ and maximum $I_{\max}$ inter–spike interval for the firing pattern. However, for statistical purposes we usually do not want to exclude a number of sets of inputs because of their biological implausibility. Hence we put $I_{\min} = 1\,\mathrm{ms}$ for the lowest and $I_{\max} = 10\,\mathrm{ms}$ for the highest value. The actual inter–spike interval I_a is determined as:

$$I_a = I_{\max} - (I_{\max} - I_{\min}) \cdot \frac{2}{\pi} \cdot \operatorname{atan}\left(\frac{\mathrm{MP} - \theta}{1 - \mathrm{MP}}\right) \tag{2.3}$$

The spike frequency condition does not allow the neuron to fire sooner, even if the MP exceeds the threshold.

[4] synaptic weight

2.1.3 Input Processing

To simulate the information processing in a JASTAP model we simultaneously (in each step, for each neuron) do the following:

1. detect whether a spike is present on each synapse
2. with the corresponding weight and latency calculate the actual action potential
3. summarize the potentials from the synapses and calculate the instantaneous MP
4. determine whether the potential exceeds the threshold
5. fire — if the spike frequency conditions allows neuron to do so

2.2 Inter–Spike Distribution

For NN tasks we have used data input patterns similar to real data flows in CNS[5] in the way, that the lengths between successive occurrences of the spikes are modeled as the events of a random variable. Koch (1998) points out that Gamma and double log–normal distributions are very similar to inter–spike intervals distribution observed in the brain. We have chosen the Gamma distribution for our inputs because of large diversity of its instances. Generated inputs will be subject to decision making process performed by NN.

2.2.1 Gamma Distribution

Definition 1. *A random variable Z has the Gamma distribution, if the probabilistic density function of Z is*

$$f(z) = \frac{z^{\alpha-1}\mathrm{e}^{-\frac{z}{\beta}}}{\beta^{\alpha}\Gamma(\alpha)} \qquad \alpha, \beta > 0, \quad z \geq 0, \quad \textit{and we denote it as } \mathscr{G}(\alpha, \beta)$$

α is so–called *shape* parameter due to the changes of α significantly modify the shape of the distribution function. If Y is from $\mathscr{G}(\alpha, 1)$, then $Z = \beta Y$ is from $\mathscr{G}(\alpha, \beta)$. Thus β has the *scaling* property. α and β together define one–to–one particular Gamma

[5] central nervous system

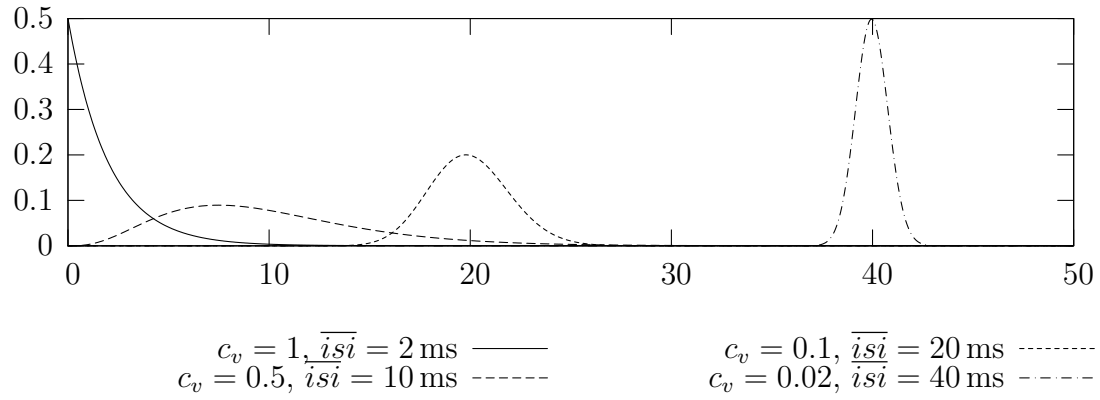


Figure 2.3: Gamma distributions: the probabilistic density functions for various c_vs and $\overline{isi}$s

distribution. However, for our purposes it is better to define it with other pair of parameters: the coefficient of variation and the mean value (here labeled as an $\overline{isi}$ — mean inter–spike interval).

Definition 2. *The coefficient of variation is defined as the ratio of the standard deviation to the mean:* $c_v = \sigma/\mu = \sigma/\overline{isi}$.

Since for $\mathscr{G}(\alpha, \beta)$

$$\sigma^2 = \alpha\beta^2, \qquad \mu(Z) = \overline{isi}(Z) = \alpha\beta \qquad \text{and consequently} \qquad c_v(Z) = \frac{1}{\sqrt{\alpha}} \tag{2.4}$$

we can use $\mathscr{G}\left(c_v^{-2}, \overline{isi} \cdot c_v^2\right)$ given c_v and $\overline{isi}$.

2.2.2 Algorithms

We used Gamma distribution generators GS* and GKM1 from Fishman (1996). These generators use a combination of the acceptance–rejection, composition and inverse transform methods.

Algorithm GS*

The algorithm generates $Z \in_R \mathscr{G}(\alpha, 1)$ for $0 < \alpha < 1$:

$b \leftarrow \alpha/\mathrm{e} + 1$
while not *success*

$U \leftarrow_R (0,1\rangle$
$Y \leftarrow bU$
if $Y \leq 1$ **then**
  $Z \leftarrow Y^{1/\alpha}$
  $W \leftarrow_R -\ln(0,1\rangle$
  if $W \geq Z$ **then** *success*
else
  $Z \leftarrow -\ln\big((b-Y)/\alpha\big)$
  $W_1 \leftarrow_R (0,1\rangle$
  $W \leftarrow W_1^{1/(\alpha-1)}$
  if $W \geq Z$ **then** *success*
return βZ

Inverse transform method for $\alpha = 1$

From definition 1 after plugging $\alpha = 1$ in, we get $f(z) = \mathrm{e}^{-z/\beta}/\beta$ — the exponential distribution with the mean β. So we can use the inverse transform method to generate it from the uniform distribution, by taking $U \in_R \mathscr{U}(0,1\rangle$. By inverting the distribution function for exponential distribution we get that $X = -\beta \ln U$ has the exponential distribution with the mean β. This is fast to compute and therefore it is the most commonly used method.

Algorithm GKM1

Generates $Z \in_R \mathscr{G}(\alpha, 1)$ for $1 < \alpha$:

$a \leftarrow \alpha - 1$
$b \leftarrow \big(\alpha - 1/(6\alpha)\big)/a$
$m \leftarrow 2/a$
$d \leftarrow m + 2$
while not *success*
  $XP \leftarrow_R (0,1\rangle$
  $YP \leftarrow_R (0,1\rangle$
  $V \leftarrow bYP/XP$
  if $mXP - d + V + 1/V \leq 0$ **then** *success*
  if $m \ln XP - \ln V + V - 1 \leq 0$ **then** *success*
return $\beta a V$

2.2.3 Exponential Distribution and Poisson Process

The gamma distributions provide a multitude of different types of distributions. One of the special cases is when $c_v \to 0$ or $\alpha \to \infty$. Then we get a "random" variable with each drawn equal to $\overline{isi}$. Another special case mentioned above occurs when $c_v = 1$. Then the Gamma distribution reduces to the exponential one. The number of spikes generated over a fixed time interval by exponential distribution has the Poisson process property because during generation there is no dependence on preceding events at all and so the events themselves are statistically independent. When the mean firing rate $\overline{isi}$ is fixed, as in our case, we call it a *homogeneous* Poisson process. The Poisson process generates every sequence of n spikes with equal probability. The random variable obtained as a number of spikes during the fixed time has the Poisson distribution. The Poisson process is simple to generate and matches data on neural response activity (Dayan and Abbott, 2001).

2.3 Statistical Decisioning and Binary Decisions

An important use of mathematical statistics is its ability to make decisions in the situations of incomplete information. We will describe the efficiency of such decisions by the means of statistical decision theory (Schervish, 1995).

Suppose that we observe data $\mathbf{X}_1, \mathbf{X}_2, ...$ taking values in a *sample space* $\mathcal{X}$. For all $i = 1, 2, ...$, the data $\mathbf{X}_i$ have a stochastic nature — they originate from a probability distribution P_i defined on the set $\mathcal{X}$. Moreover, we assume that the distributions P_i do not occur arbitrarily, but exhibit a stochastic behavour given by the *prior distribution* π on the set $\mathfrak{P}$ of all probabilities on $\mathcal{X}$. That is, some of the distributions of the data are more probable, while others can be highly unlikely.

Our general aim is to construct a *decision device* that uses the information contained in each $\mathbf{X}_i$ to decide whether the distribution P_i has or does not have a given property of interest. More formally, we partition the set $\mathfrak{P}$ of all distributions on $\mathcal{X}$ into two disjoint subsets $\mathfrak{P}_0$, $\mathfrak{P}_1$ and consider hypothesis $H_0 : P_i \in \mathfrak{P}_0$ vs. $H_1 : P_i \in \mathfrak{P}_1$. Hence, from the point of view of the statistical decision theory, we intend to find a *decision rule* $\delta : \mathcal{X} \to \{0, 1\}$ assigning a value "accept H_0" or "accept H_1" to any $\mathbf{X} \in \mathcal{X}$.

To construct a meaningful decision rule, we need to specify the loss (or benefit) resulting

from the wrong (resp. right) decision. To do this, we usually define a *loss function* $L : \mathfrak{P} \times \{0, 1\} \to \mathbb{R}$. While the loss function can have many different forms, we use the simplest, and most natural loss function for the binary decision problems: $L(P, \delta) = 0$ iff$P \in \mathfrak{P}_\delta$ and $L(P, \delta) = 1$ iff $P \notin \mathfrak{P}_\delta$. In words, we assign the loss 0 to the right and the loss 1 to the wrong decision. When using NN, we also have to define the loss for "no decision" response (see fitness function in 3.5).

Under this setup, any decision rule δ can be characterized by a *mean loss* given by: $\overline{L}(\delta) = E\left(L\left(P, \delta\left(\mathbf{X}\right)\right)\right)$, where E is the operator of expectation (mean value) and is taken with respect to $P \sim \pi$ and $\mathbf{X} \sim P$. A natural aim is to construct a decision rule $\delta^* \in \Delta$ that minimizes expected loss, that is $\overline{L}(\delta^*) = \min_{\delta \in \Delta} \overline{L}(\delta)$.

Notice that the random variable $L(P, \delta(\mathbf{X}))$ has Bernoulli $0-1$ distribution, which means that $\overline{L}(\delta) = p\left(P \notin \mathfrak{P}_{\delta(\mathbf{X})}\right)$. Therefore, minimization of the expected loss is equivalent to minimization of the probability of wrong decision.

In mathematical literature, the set of possible decision rules usually contains *any* function, in purely mathematical meaning, assigning decisions to data. This is appropriate in the case that we do not specify the actual calculation of the decision. Nevertheless, if we want to perform decision making by a real device, we are always limited to decision rules from some space Δ restricted by all possible designs of the device. For example, if we intend to use a NN as a decision device, we are limited by the architecture and possible functional states of the network (see 3.1.2).

We give an example pertaining to the NNs decision making: The data $\mathbf{X}_i$ can be a vector representation of the spike arrival times and the prior distribution π represents frequency with which various modes of spiking (probability distributions P_i) occur. Next, the set of distributions $\mathfrak{P}_0$ can be the set of all distributions that generate mean interspike distance less than a critical value t_θ. Thus, in this example, we want to construct a decision device: a NN $\delta \in \Delta$, which decides whether the (asymptotic) mean firing rate is lower or higher than $1/t_\theta$. The aim is to minimize probability of wrong decision. The set Δ represents the set of all plausible NN with a prescribed architecture.

2.4 Decisioning with Neural Networks

NN models have been successfully used in many domains: function approximation clustering, and various classification tasks (Navrat et al., 2002). They can discover statistical

regularities in input patterns and encode them on the output (Kohonen, 1997).

However, NNs do not have explicitly determined decision making. Neural paths in the human brain could be better described as a continual information flow than a collection of decisions. As we want to interpret neural activity as a simple decision — mostly as a binary decision — we have to find a way how to decode this activity into a statement.

We use this method: Some neurons in the network are designated as the *output* neurons. When any of the output neurons fire, we take it as a *hot–spot* network decision. The meaning of the decision is task–dependent. Thereafter we can stop the simulation in progress, because the subsequent activity has no impact on the final decision.

In *simple* decision making, we are asking a network about the selected property of the input, in particular whether some parameter of the input pattern is below/above the given value. On the other hand, in *comparative* decision problems, a network is given two inputs and the problem to decide is to select which of them has the higher/lower value of the selected parameter.

Chapter 3

Evolution

As well as in other models of NNs we can more or less accurately design a network topology to be able to solve/decide a given problem. Nevertheless, the network (structure) has many parameters to configure in order to get a specific instance of the network topology. Since there are many parameters, searching for the best configuration rides into multidimensional optimization. While the space to explore is enormous, we have to take into account discretization and stochastic methods. Seeing that the *backpropagation* algorithm cannot be straightforwardly[1] in this model, we have chosen *genetic algorithms* as a network optimization, search space exploration and parameter. GAs are also easier to reconcile with biological learning.

3.1 Coding

3.1.1 Feature Coding

Each entity of the proposed model has several features to set up. We will discuss the necessity of their optimization:

synapse

▷ WEIGHT — very important parameter, high weight tells us whether the input

[1] However, we are aware of the possibility for spiking neuron models to adopt the BP algorithm (Bohte et al., 2002).

from a specific source can by itself overcome the threshold or, on the other side, extremely low one signalizes that the input and therefore the whole connection is negligible and useless.

- ▷ LATENCY — this parameter is important for time–related functions. Precise adjustments of time delays can superpose the PSPs from the different synapses with special time patterns.

- ▷ SHAPE OF PSP — can be very well configured by setting t_1 and t_2 parameters of the waveform as defined in equation 2.1. As we have mentioned above, a slower decay of the PSPs results in more superpositions of the MPs in the neighboring[2] synapses. Although evolving accurate values for t_1 and t_2 for each synapse can be helpful, it also expands the search space. The results of our first experiments have shown that the slower decay the better evolved strategies in general. Thus we decided to leave the PSP decay as slow as possible with respect to the boundaries of biological plausibility.

neuron

- ▷ THRESHOLD — together with the synaptic weights indicate the character of the summation properties

- ▷ SPIKE FREQUENCY — However $I_{\min}$ and $I_{\max}$ value can also be evolved, it have not brought any significant improvements. Despite the fact that enabling higher rates in the firing frequency is also useful in time–dependent actions, the same effect can be achieved by leaving $\langle I_{\min}, I_{\max} \rangle$ interval wide and let evolution to set up the thresholds/weights values to obtain an appropriate firing rate.

3.1.2 Chromosome

Even though we have chosen the features to evolve, we definitely do not want to search the whole $(-\infty, \infty)$ interval, because only narrow domains are (biologically) admissible. With respect to JASTAP model definition limits we will search in boundaries described in table 3.1, defining in such a way a Δ set (the search space for the NNs, see 2.3).

[2] synapses connected to the same neuron

feature	min value	max value
synapse weight	−1	1
synapse latency	0 ms	40 ms
synapse waveform t_1 value	5 ms	
synapse waveform t_2 value	15 ms	
neuron threshold	0	1
minimal neuron firing period	1 ms	
maximal neuron firing period	10 ms	

Table 3.1: Parameter boundaries used for evolution

We decided not to evolve a NN topology and for this reason we will not encode it into the chromosome. It is up to us which bits and in what order represent a particular parameter value. In order to fulfill requirements of an effective crossover (3.3), we will encode the parameters corresponding to the same entity (synapse, neuron) together (Kvasnicka et al., 2000). The reason is that the "good units" of the chromosome would be rather preserved and transferred en block during the recombination process.

3.1.3 Precision

Although all values are real numbers, in computer–like neuronal modeling we are forced to work with the discretization and the binary representation of real values. We use binary coding for the chromosomes.

First of all, real NNs work naturally in fluent time. This cannot be achieved with the model and the JASTAP designers recommend to use 0.5 ms as a time step.

Moreover, weights, latencies and thresholds cannot be coded and searched through the whole bounded interval. For this reason we uniformly chose some values (we use 2^6 to 2^{10} different values) within a permitted interval. As the values are chosen uniformly, it suffices to code only the "position" in the gridded interval. With a binary vector of k bits we can code 2^k different values. Suppose we have a vector $\vec{v} = (v_{k-1}v_{k-2}\dots v_0)$, $v_i \in \{0, 1\}$, for $i = 0, 1, \dots, k-1$ and $p_{\min}$ and $p_{\max}$ are the lower and upper bounds for the parameter p. Then the actual value is calculated as

$$x = p_{\min} + \frac{p_{\max} - p_{\min}}{2^k - 1} \left(\sum_{i=0}^{k-1} v_i 2^i \right) \tag{3.1}$$

No one disputes the fact that such a discretization reduces the precision. However, after the time–discretization, the values of latencies are only integer numbers. Furthermore, one can easily see that some combinations of a threshold and the corresponding synaptic weights have a similar effect. For example: without the limiting function we could divide the threshold and weights by two to get exactly the same behavior. Another interesting point is that in most cases the adjacent values of the thresholds have a similar effect. As an illustration: Many low values of the threshold will pass along any kind of the input activity in the same way.

3.1.4 Gray Code

A *Gray code* is a common technique used in genetic algorithms to flatten the fitness function surface. A standard binary coding faces the Hamming barrier problem: a bit flipped during a mutation can radically change the coded value. A beneficial property of a Gray code is that every bit–flip changes the coded value by $+/-$ 1. This is useful in many stochastic optimization techniques and it is also convenient for a seed–based population creating (3.2.2).

How to transform a standard code to a Gray code?

Let $\vec{v} = (v_{k-1} v_{k-2} \dots v_0)$ is the binary vector in a standard binary code. We can obtain the same number in a binary Gray code $\vec{g} = (g_{k-1} g_{k-2} \dots g_0)$ as follows[3]:

$$\begin{aligned} g_{k-1} &= v_{k-1} \\ g_i &= v_{i+1} \oplus v_i \qquad i = 0, 1 \dots k-2 \end{aligned} \tag{3.2}$$

How to transform a Gray code to a standard code?

The inverse transformation to the previous can be achieved by:

[3] $\oplus$ denotes the binary XOR function

$$\begin{aligned} v_{k-1} &= g_{k-1} \\ v_i &= \bigoplus_{j=1}^{k-i} g_{k-j} = \left(\bigoplus_{j=1}^{k-i-1} g_{k-j} \right) \oplus g_i = v_{i+1} \oplus g_i \qquad i = 0, 1 \ldots k-2 \end{aligned} \tag{3.3}$$

3.2 Initial Population

3.2.1 Random Population

The most simple way to introduce an initial population into evolution is to generate a random one. After we calculate the chromosome size, we randomly choose a corresponding number of bits for each individual.

3.2.2 Noising the Seed

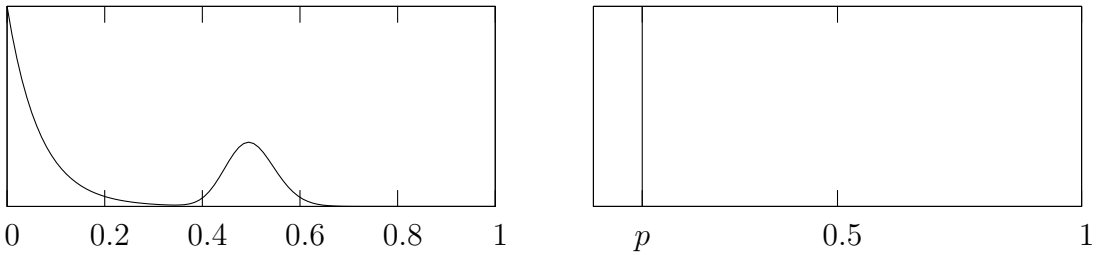
To refine a semi–result rather than evolve a new one, we can use it as a *seed* to generate an initial population for the next evolution. If we expect that the best solution is "near" the seed, we can noise it to spread the values over the fitness surface.

The bit–flipping is used to noise the seed chromosome. We define the largeness of spread, by choosing the probability p and we flip each bit in chromosome with that probability. It is to be recognized that:

- ▷ $p = 0$ remains the chromosome unchanged
- ▷ $p = 0.5$ is similar to random population: results do not depend on the seed
- ▷ $p = 1$ inverts the chromosome

We can generalize this seeding method in the way what "amounts" of generated chromosomes and how "far" away from the seed they appear. We take p as a random variable and define a density function for it. In the process of a population creation we choose a probability p for each new chromosome according to the density function and then with such probability flip each bit of the seed. For example (figure 3.1 (a)) we wish to have some new chromosomes near the seed but also a substantial part of new generation enough away (to avoid a fall into a local extreme). Noising the seed with a parameter

p is the a special case (figure 3.1 (b)). All this techniques can be used in chromosome mutation (3.4).



(a) a example of p.d.f. for the seed **(b)** standard p–mutation

Figure 3.1: Seed techniques

With the usage of a Gray coding to code the chromosomes we can at average expect that the coded number units (thresholds, weights and latencies) in chromosomes will be $p \times$ size (where size is length of number unit) bits away from the seed (in the terms of Hamming distance).

3.3 Crossover

Crossover is a genetic operator used to *recombine* the pair of chromosomes in order to vary generation in the evolution process. As genetic algorithms are an analogy to the real evolution process, a crossover is an analogy to the real biological crossover occurring in reproduction. In stochastic optimization we use it to speed up the searching process. If we eliminate the crossover, the most of genetic algorithms would reduce to hill–climbing heuristics. The idea behind the following principles is that aggregated "good" features of the particular entities can be interchanged during recombination and thereby afford opportunity to create "better" (in the terms of fitness function) chromosomes.

3.3.1 Crossover Techniques

One Point Crossover In this technique one crosspoint is randomly chosen within the size of the parent chromosomes. Then the crossing chromosomes swap the corresponding parts after the crosspoint.

Multi–Point Crossover Multi–point crossover is a generalized case of the previous one. In the recombination, a selected number of crosspoints are chosen uniformly within the chromosome size and then the corresponding parts allocated by the crosspoints are alternately left by and swapped.

3.4 Mutation

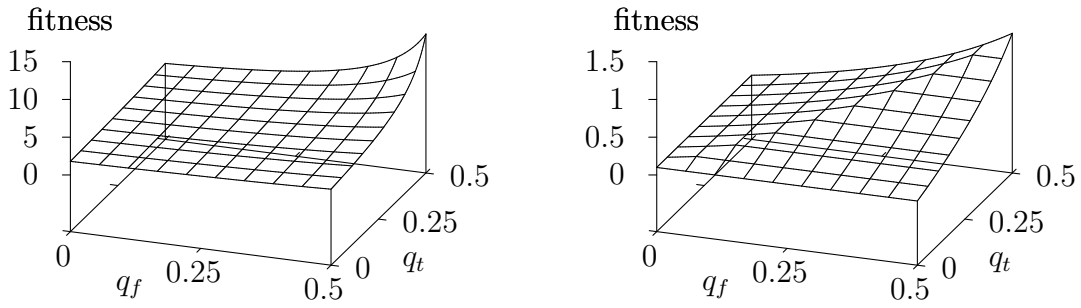
Mutation is another genetic operator and reflects the biological mutation during the reproduction. As an artificial metaphor, we use the bit–flipping on each descendant of the recombination with chosen probability p_{mut} as in the "noising the seed" technique in 3.2.2.

Mutation helps genetic algorithm to avoid local minima. In the earlier phases of the evolution it helps to find better chromosomes, in latter ones we use it to tune the details.

3.5 Fitness Function

In all our simulations a network had to make the true/false hot–spot decisions. As an evaluation of the efficiency we used the same approaches in the fitness definition. The fitness functions as defined here play the role of the cost function, the opposite of the loss function as defined in 2.3. Let q_t denotes ratio of the correct answers from "true" case and q_f denotes the ratio of the correct answers from the "false" one: $q_t + q_f \in \langle 0, 1 \rangle$. If the network does not respond within the assigned time, it is taken as an incorrect answer. Let us also suppose that the amount of the "true" and "false" tests is the same.

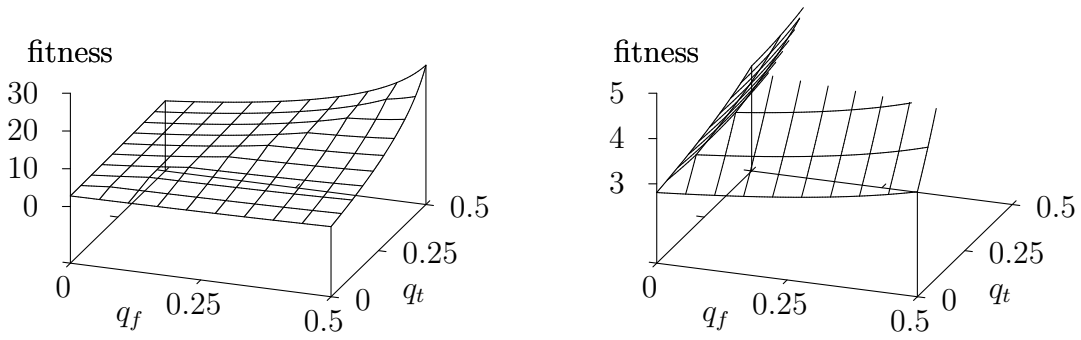
- ▷ OVERALL RATIO FITNESS — figure 3.2 (a) — we evaluate each network with $1/(1 + \varepsilon - q_t - q_f)$ giving more fitness to better networks.
- ▷ ONE–SIDE MINIMUM FITNESS — figure 3.2 (b) — previous fitness function often led evolution to local minimum of 50% of the correct answers ($q_t = 0.5$, $q_f = 0$ and vice versa) so one of the solutions is to prefer such networks that give correct answers in both ways: $1/(1 - \min(q_t, q_f)) - 1$. Notice that in this manner we prefer the networks with the balanced ratios even if the overall ratio is lower.
- ▷ COMBINED FITNESS — figures 3.3 (a) and (b) — the previous fitness definition



(a) overall **(b)** one–side minimum

Figure 3.2: Fitness functions

suffers from the fact that the zero–response networks[4] are evaluated with same fitness as the "half–response" ones. To avoid the problem we combined two previous fitnesses by a linear combination with emphasis to ONE–SIDE MINIMUM.



(a) combined **(b)** combined (detail)

Figure 3.3: Fitness functions

▷ FITNESS WITH SPEED PREFERENCES — a faster decision maker is definitely a better network. However, the correct decision is the higher priority. While we

[4] the networks without any responses or with incorrect decisions

want to obtain the general decision makers, we generate new tests in each evolution step. That causes differences (in absolute numbers) of correct responses. As the speed–related preferences are consequently difficult to apply, we do not include it into the fitness expression.

3.6 Algorithm

We use common genetic algorithm as follows. Chromosomes are chosen for recombination by *roulette–wheel* rule. We have tried two methods for *selection* to the next population: (1) all the new chromosomes replace the old generation and (2) the best chromosomes from the old and recombined generation are selected to the new one. The fact, that in method (2) best individuals are preserved, suggests to use p_{cross} close or equal to 1. Anyway, we have achieved better results with method (2), see 4.4.

$P \leftarrow$ random population
for # tests
  evaluate each chromosome in P with fitness
  $P_{\text{desc}} \leftarrow \emptyset$
  while $|P_{\text{desc}}| < |P|$
    quasi–randomly[5] chose 2 chromosomes χ_1, χ_2 from P
    if $r \in_R \langle 0, 1\rangle < p_{\text{cross}}$ **then**
      $(\chi_1,\chi_2) \leftarrow O_{\text{cross}}(\chi_1, \chi_2)$
      $\chi_1 \leftarrow O_{\text{mut}}^{p_{\text{mut}}}(\chi_1)$
      $\chi_2 \leftarrow O_{\text{mut}}^{p_{\text{mut}}}(\chi_2)$
    $P_{\text{desc}} \leftarrow P_{\text{desc}} \cup \{\chi_1, \chi_2\}$
(1) $P \leftarrow P_{\text{desc}}$
(2) evaluate each chromosome in P_{desc} with fitness
(2) $P \leftarrow$ the $|P|$ best chromosomes from $P \cup P_{\text{desc}}$

[5] by the roulette–wheel rule

Chapter 4

Decision Problems

4.1 More Frequent Input

The foremost problem to analyze is the rate decision making over spike trains. We claim without a doubt that this decision is present in bio–networks. The abilities related to this kind of decision making in human neurophysiology include: when listening to sound, which ear is closer; when two points on one's body are pressed, which one is more painful, and so on. With this in mind, a successful "brain" must demonstrate an efficient performance, when we consider rate coding as an information processing.

Evolution and results of this task have the straightforward applications. First of all, there are applicable in theoretical statistical theory and statistical hypothesis testing of Gamma distributions. Equally important is its utilization in real–time controlling.

Formally, we compare two $\overline{isi}$s by testing hypothesis H_0: $\overline{isi}_1 < \overline{isi}_2$ (more frequent input is the one with the lower $\overline{isi}$) against alternative H_1: $\overline{isi}_1 \geq \overline{isi}_2$. Actual definition of $\mathfrak{P}_0$, $\mathfrak{P}_1$ and π depends on the selected problem. For instance, if we compare two Poisson processes with the fixed $\overline{isi}$s and with the equal chances to occur then

$$\mathfrak{P} = \mathfrak{P}_0 \cup \mathfrak{P}_1, \quad \mathfrak{P}_0 = \{P_0\}, \quad \mathfrak{P}_1 = \{P_1\} \quad \text{and} \quad \pi : p(P_0) = p(P_1) = \frac{1}{2},$$

where, for example, P_0 is defined in the way that the interspike intervals of the spikes arriving to the first input are from $\mathscr{G}(1, \overline{isi}_1)$ and the other ones are from $\mathscr{G}(1, \overline{isi}_2)$ and P_1 is defined in like manner.

4.1.1 Theoretical Strategies

We depict some strategies to solve the problem. First, we describe the copy machine strategy with the success ratios over 50%. This strategy is really simple. Hence we can disregard the products with lower ratios as the not intelligent. Contrary, we will discuss the event counting strategy regarding this as the high–intelligent one. In general, this strategies will frame the results we obtain.

Copy Machine

A *copy machine* strategy appears first to evolve in this task. It is based on the assumption that the a frequent input produces an event earlier than a less frequent one. However, this is not necessarily the best strategy unless the coefficient of variation is zero. To determine the efficiency of this strategy, we take the two inputs from an exponential distribution and calculate whether this strategy provides a correct decision. This is if and only if a random event from more frequent input is smaller than the other one. So we have:

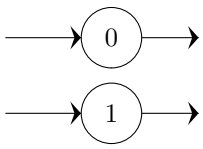
$$X \sim \mathscr{E}(\lambda) \qquad Y \sim \mathscr{E}(\gamma) \qquad \mu(X) = \lambda \qquad \mu(Y) = \gamma \qquad \lambda < \gamma \tag{4.1}$$

As the inputs are independent and the probability density function is $f(x) = e^{-x/\lambda}/\lambda$, we get:

$$\begin{aligned} P[X < Y] &= \int_{x=0}^{\infty} \int_{y=x}^{\infty} \frac{e^{-\frac{x}{\lambda}}}{\lambda} \frac{e^{-\frac{y}{\gamma}}}{\gamma} \mathrm{d}y \mathrm{d}x = \frac{1}{\lambda\gamma} \int_{x=0}^{\infty} e^{-\frac{x}{\lambda}} \left[\int_{y=x}^{\infty} e^{-\frac{y}{\gamma}} \mathrm{d}y \right] \mathrm{d}x = \\ &= \left[a = -\frac{y}{\gamma} \right] = \frac{1}{\lambda} \int_{x=0}^{\infty} e^{-\frac{x}{\lambda}} \left[\int_{a=-\infty}^{-\frac{x}{\gamma}} e^{a} \mathrm{d}a \right] \mathrm{d}x = \frac{1}{\lambda} \int_{x=0}^{\infty} e^{-\frac{x(\lambda+\gamma)}{\lambda\gamma}} \mathrm{d}x = \\ &= \left[b = -\frac{x(\lambda+\gamma)}{\lambda\gamma} \right] = \frac{\gamma}{\lambda+\gamma} \int_{b=-\infty}^{0} e^{b} \mathrm{d}b = \frac{\gamma}{\lambda+\gamma} = \frac{1}{1+\frac{\lambda}{\gamma}} \end{aligned} \tag{4.2}$$

lower $\overline{isi}$	higher $\overline{isi}$	ratio
30 ms	40 ms	57.14 %
20 ms	40 ms	66.67 %
10 ms	40 ms	80.00 %
20 ms	30 ms	60.00 %
10 ms	30 ms	75.00 %
10 ms	20 ms	66.67 %

Table 4.1: Copy machine strategy efficiency, $c_v = 1$



Thus the ratio of correct decided sets depends only on the λ/γ ratio. Efficiency of the copy machine strategy is shown in table 4.1 on selected input trains. The function of the copy machine strategy can be performed by a trivial NN with two neurons connected to both inputs and also taken as the outputs of the network (see right image). Both synapses in the NN should have the same latency and neuron thresholds and the corresponding synaptic weights should be set up in the way that every event is passed along (zero thresholds and positive weights satisfy the condition).

Event Counting

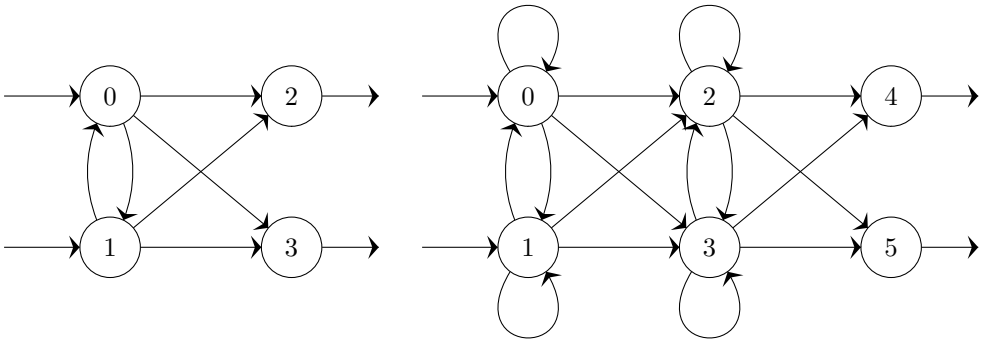
An *event counting* strategy is based on a spike counting during the given period. It is likely the best possible strategy using the sequential observation of an input pattern. So we may consider it the upper bound for our NNs. It works as follows: During the given time period it counts the events (spikes) on the both inputs. After the time elapsed, it makes the decision that the more frequent input is the one with the more events observed. If the numbers of events are equal, it votes, for example, for the first input.

We know that if N is the number of spikes within interval $\langle 0, T\rangle$ with independently exponentially distributed inter–spike intervals with mean $\overline{isi}$ equal to μ the N has the Poisson distribution $N \sim \mathscr{P}(T/\mu)$:

$$N_X \sim \mathscr{P}(\mu_\lambda) \quad N_Y \sim \mathscr{P}(\mu_\gamma), \quad \mu_\lambda = T/\lambda \quad \text{and} \quad \mu_\gamma = T/\gamma; \quad \lambda < \gamma \quad \mu_\lambda > \mu_\gamma \quad (4.3)$$

lower $\overline{isi}$	higher $\overline{isi}$	ratio
30 ms	40 ms	72.45 %
20 ms	40 ms	94.51 %
10 ms	40 ms	99.99 %
20 ms	30 ms	83.97 %
10 ms	30 ms	99.91 %
10 ms	20 ms	98.83 %

Table 4.2: Event counting strategy efficiency after 300 ms, $c_v = 1$



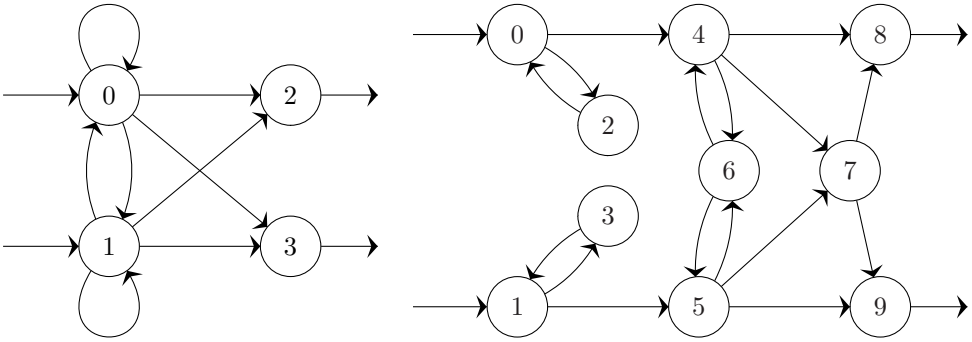
(a) *A*–network **(c)** *C*–network

Figure 4.1: Network topologies

According to this notation, the probability of correct decision is

$$\begin{aligned} P &= P[N_X > N_Y] + \frac{1}{2}P[N_X = N_Y] \\ P[N_X > N_Y] &= e^{-\lambda-\gamma}\sum_{n=0}^{\infty}\sum_{m=n+1}^{\infty}\frac{\lambda^n}{n!}\frac{\gamma^m}{m!} \\ P[N_X = N_Y] &= e^{-\lambda-\gamma}\sum_{n=0}^{\infty}\frac{\lambda^n}{n!}\frac{\gamma^n}{n!} = e^{-\lambda-\gamma}J_0(2\sqrt{\lambda\gamma}) \end{aligned} \tag{4.4}$$

Table 4.2 shows calculated probabilities of the correct decisions on selected $\overline{isi}$s.



(b) *B*–network **(d)** *D*–network

Figure 4.2: Network topologies

4.1.2 Network Topologies

We present our results of the evolution on four network topologies. (figure 4.1 and 4.2). All four are symmetrical, due to symmetric nature of the problem. The difference between A and B network is in the recurrent loop connections of the input neurons. In C network, we added a "hidden"[1] layer. The design of the D network is taken from (Pavlasek et al., 2003), the first mention of JASTAP's capability to cope with rate decision problems.

4.1.3 Results

Interspike intervals

We regard the $10 - 40$ ms $\overline{isi}$s as an appropriate and plausible input for the networks. To exhibit model properties we have chosen 10 ms, 20 ms, 30 ms and 40 ms as the fixed $\overline{isi}$s for mutual comparison. To determine whether one network could be efficient in decision making of various rates we tried to evolve an individual which is able to decide random rates from $\langle 10\,\text{ms}, 40\,\text{ms}\rangle$.

[1] in the terms of multi–layer perceptrons (Rumelhart et al., 1986)

lower $\overline{isi}$	higher $\overline{isi}$	copy	A	B	C	D	event
30 ms	40 ms	57.14 %	60.12 %	<u>62.18</u> %	59.03 %	61.31 %	72.45 %
20 ms	40 ms	66.67 %	<u>88.11</u> %	87.56 %	87.67 %	81.32 %	94.51 %
10 ms	40 ms	80.00 %	<u>99.81</u> %	99.65 %	99.25 %	99.58 %	99.99 %
20 ms	30 ms	60.00 %	66.92 %	<u>71.52</u> %	69.65 %	67.27 %	83.97 %
10 ms	30 ms	75.00 %	<u>99.35</u> %	99.19 %	99.14 %	98.08 %	99.91 %
10 ms	20 ms	66.67 %	<u>95.04</u> %	92.74 %	94.12 %	91.75 %	98.83 %
$\langle 10\,\text{ms}, 40\,\text{ms}\rangle$		60.22[3]%	66.66 %	66.68 %	<u>68.48</u> %	65.75 %	79.68[4]%

Table 4.3: Different network topologies (A, B, C, D — 4.1.2) compared with the theoretical strategies (copy and event columns). The best strategies among the evolved is underlined. Evolved with $p_{\text{mut}} = 0.05$, $p_{\text{cross}} = 1$ and 1 crosspoint — 300 ms, $c_v = 1$.

Set up

All the following experiments were conducted with a population of 200 individuals in two phases of 200 steps. After the first phase of the evolution, the best individual was depicted and used as a seed with $p = 0.05$ for the next (*fine–tuning*) phase. This was meant as a supplement of the BP or other gradient–based methods. The maximal response time was set to 300 ms as an approximation of the biological response time. To evaluate the individuals in the population we used the combined fitness (see 3.5) with 50 tests. At the end of the evolution, we evaluated all with 10^3 tests and picked up the best. Tables show the efficiency ratio[2] of the winners evaluated with 10^4 tests.

Different topologies

Table 4.3 shows the comparison between the different network topologies (4.1.2) and the theoretical strategies (4.1.1) on the same input data sets and evolution parameters. As we expected, none of the results overcame the event strategy. On the other hand, all networks evolved to better than the copy one. The best results for the fixed $\overline{isi}$s were achieved with A and B–networks. In a more complex problem, when the $\overline{isi}$s were chosen randomly, the C–network dominated. One can easily see that the B–network

[2] The best fitness does not necessarily mean the best ratio. Fitness applied here reflects also the balance of correct responses (as defined in 3.5)

[3] tested on the simple network (4.1.1), 10^6 tests

[4] 10^6 tests

crosspoints		1	2	5	3	1	3
p_{cross}		1	0.5	1	0.9	0.7	0.9
p_{mut}		0.05	0.1	0.005	0.01	0.05	0.01
lower $\overline{isi}$	higher $\overline{isi}$	type (2)	type (2)	type (2)	type (2)	type (1)	type (1)
30 ms	40 ms	59.03 %	60.78 %	62.45 %	63.26 %	59.43 %	61.49 %
20 ms	40 ms	87.67 %	83.17 %	85.76 %	83.79 %	75.43 %	80.87 %
10 ms	40 ms	99.25 %	99.06 %	98.12 %	99.69 %	98.87 %	99.59 %
20 ms	30 ms	69.65 %	70.94 %	61.73 %	58.45 %	65.44 %	66.93 %
10 ms	30 ms	99.14 %	97.97 %	98.72 %	92.81 %	94.99 %	98.35 %
10 ms	20 ms	94.12 %	92.18 %	91.58 %	94.25 %	91.38 %	87.66 %
⟨10 ms, 40 ms⟩		68.46 %	68.10 %	72.00 %	67.30 %	60.45 %	68.17 %

Table 4.4: Different evolution parameters: C–network, $c_v = 1$

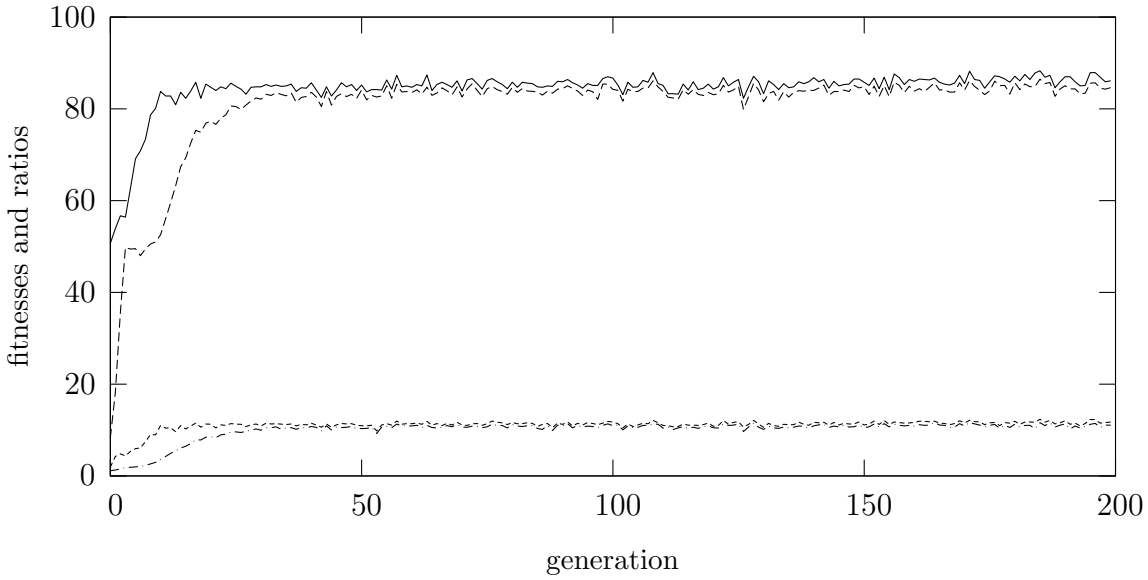
can be hardwired to perform exactly same behavior as the A. The same relation exists between B and C–networks. Although this $A \sqsubseteq B \sqsubseteq C$ relation, proposed results do not follow it due to suboptimal products of the evolution and also the larger search space in the more complex C and D–networks. We decided to use the C–network for further simulations to observe more sophisticated strategies. We also achieved better results with this structure with the different evolution parameters as shown in table 4.4 and decided to use parameters in column 1 for the rest of our experiments.

Evolution process

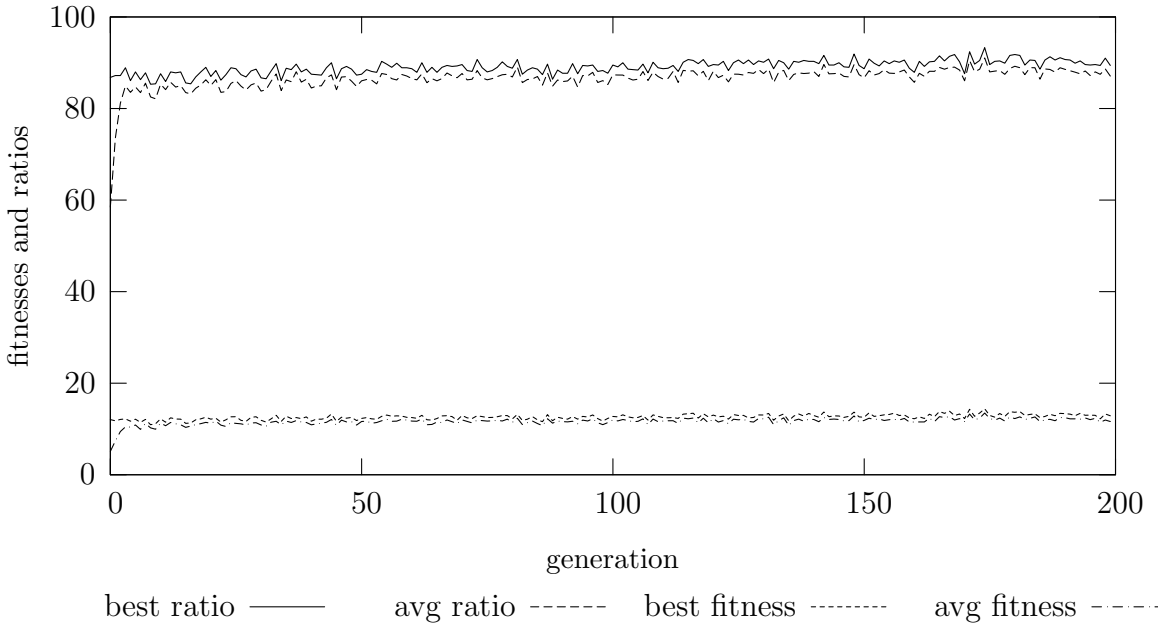
Evolution curves are shown in figures 4.3 (a) and (b). Nondecreasing tendency is caused by elitism–like selection in GA. The limited number of tests causes that calculated fitness is only an approximation. We can observe a slow, but permanent improvement. At the beginning of the fine–tuning phase, the average ratio and fitness dropped down due to a seed procedure, but recovered back in tens of generations.

Interpretation

Figure 4.4 (b) shows the evolved network for 10 vs. 30 ms decision making and figures 4.4 (d) and (e) how does this network deal with the both types of given inputs



(a) First phase: ends with 85.4 % ratio (fitness 11.0) for the best.



(b) Second (fine–tuning) phase: ends with 89.71 % ratio (fitness 12.9) for the best.

Figure 4.3: Evolution curves: best and average ratio and fitness per generation task: decision making of the $\overline{isi}$ 10 vs. 20 ms, $c_v = 1$. Fitnesses and ratios were evaluated with 10^3 test in each step.

lower $\overline{isi}$	higher $\overline{isi}$	$c_v = 0.02$	$c_v = 0.1$	$c_v = 0.5$	$c_v = 1$[6]	$c_v \in_R \langle 0, 1\rangle$
30 ms	40 ms	100.00 %	100.00 %	77.17 %	63.26 %	73.29 %
20 ms	40 ms	100.00 %	100.00 %	98.07 %	87.67 %	95.61 %
10 ms	40 ms	100.00 %	100.00 %	99.99 %	99.69 %	99.85 %
20 ms	30 ms	100.00 %	99.96 %	90.78 %	70.94 %	87.62 %
10 ms	30 ms	100.00 %	100.00 %	99.94 %	99.14 %	99.77 %
10 ms	20 ms	100.00 %	100.00 %	99.85 %	94.25 %	98.60 %
$\langle 10\,\text{ms}, 40\,\text{ms}\rangle$		98.02 %	92.51 %	78.15 %	72.00 %	66.01 %

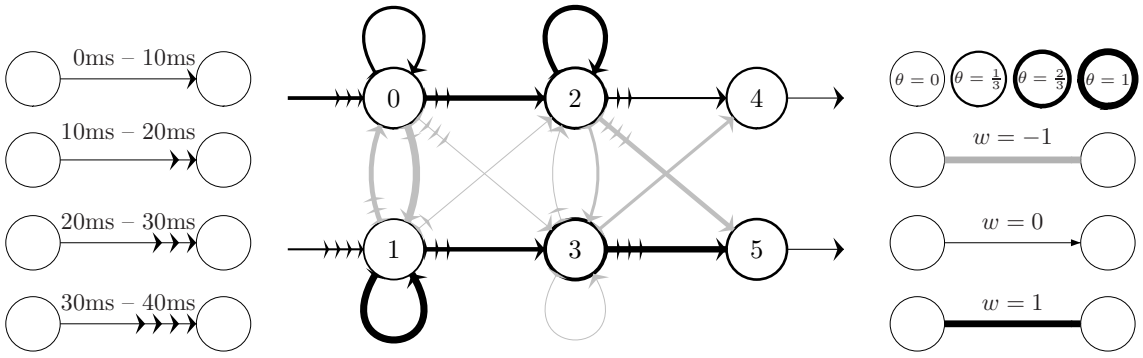
Table 4.5: More frequent input: Comparison of results for different c_vs, 300 ms

(20 vs. 30 ms and 30 vs. 20 ms) when the decision is successful. During the evolution process this network developed two *information paths* and let them to *compete* each other. Shown individual primarily concentrates on exciting more activity in 20 ms train. Although some events in 30 ms path can overcome the thresholds (4.4 (d)–1 at about 100 ms), competing ensures that such activity is diminished. No over–threshold activity is apparent at the middle layer (neurons 2 and 3), and at the output layer, potentials are the whole time above/below zero. Hence these layers represent the degrees of confidence that the decision is correct.

Various variations

Next experiments (table 4.5) compare the complexity of the problem with respect to regularity. For fixed c_vs we chose the values 0.02, 0.1, 0.5 and 1. $c_v = 0.02$ reflects the noise in almost regular input. In last column, c_v was uniformly chosen from $\langle 0, 1\rangle$ interval for both inputs. Results have clearly shown that the more regular input the easier problem to decide. For the fixed regular inputs an evolution gets individuals with 100 % accuracy[5] (for $c_v = 0.02$ even after 4–6 generations). In $c_v = 1$ case table display the best results for different evolution parameters (see table 4.4).

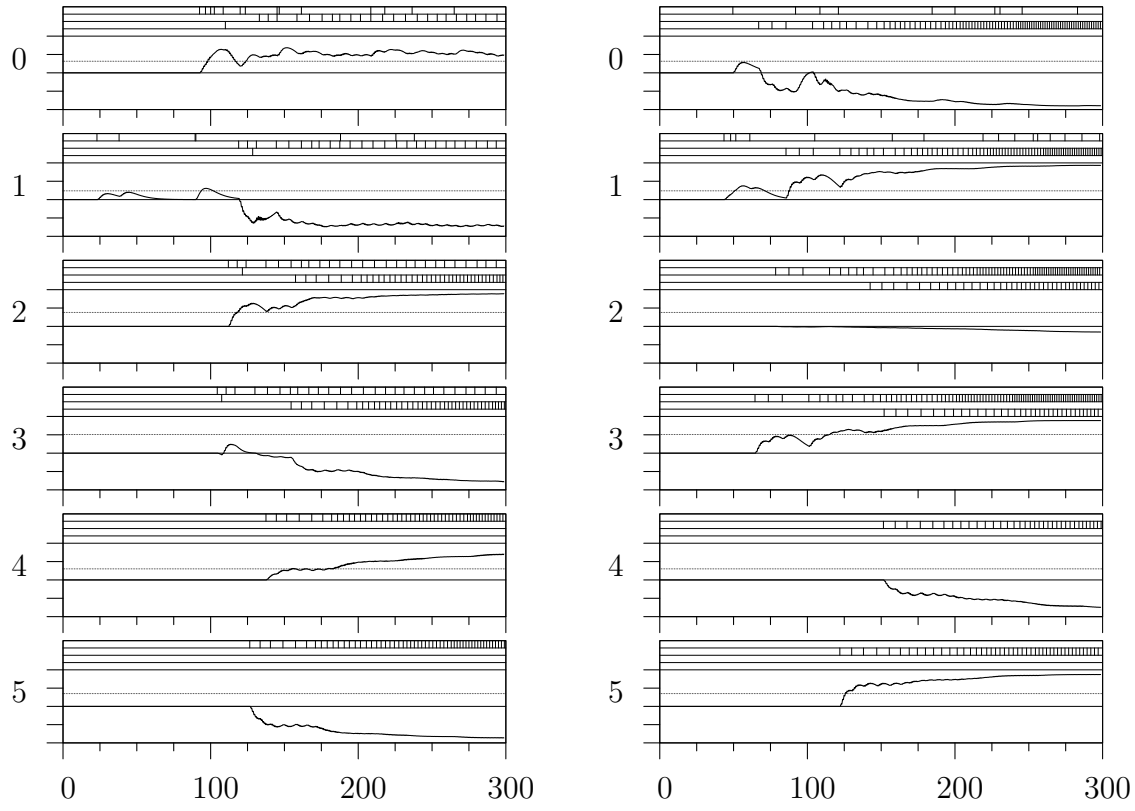
[5] This is not surprising regarding the copy machine strategy.



(a) legend: latencies

(b) evolved network for $\overline{isi}$: 20 vs. 30 ms, $c_v = 1$

(c) legend: weights and thresholds



(d) 0: $\overline{isi} = 20$ ms, 1: $\overline{isi} = 30$ ms

(e) 0: $\overline{isi} = 30$ ms, 1: $\overline{isi} = 20$ ms

Figure 4.4: Evolved network with displayed thresholds, weights and latencies: Network (b) contains information paths: 0—2—4 and 1—3—5, competing connections: $0 \to 1$, $1 \to 0$, $0 \to 3$, $1 \to 2$, $2 \to 3$ $3 \to 2$, $2 \to 5$ and $3 \to 4$, (d) and (e) display information processing. Inputs from synapses are plotted in increasing order after extern input for neurons 0 and 1.

Strategies

Besides the time–and–space summation, our studies of the evolved results have revealed several interesting strategies to help networks to decide. In some cases the network does not even use one of the inputs. Next strategies can be found in the most of evolved results.

- ▷ INFORMATION PATHS As one can see in the examples, the evolution built the information paths mostly like 0—2—4 vs. 1—3—5 in figure 4.4 (b) and 0—3—4 vs. 1—2—5 in figure 4.5 (a) or sometimes the special ones like 0—2—4 vs. 1—2—5 in figure 4.5 (b)

- ▷ COMPETING The information paths are competing each other. Competing is maintained with the negative weights and sometimes also with the minimal latency corresponding to the mutual connection of the paths. It is especially beneficial when $c_v \to 1$, because the networks have to count the events over longer time to be accurate.

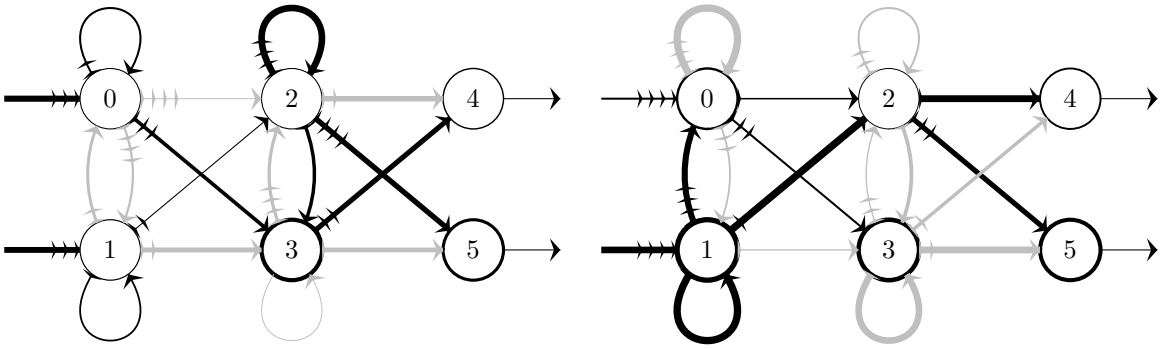
- ▷ HANDLING REDUNDANCY Look at the figure 4.5 (b). We could take out the neuron number 3 without a difference, its potential cannot exceed the threshold in given conditions. Only one synapse is excitatory but under–threshold, others are inhibitory. Therefore, when structure is too complex, evolution gets rid of the redundant elements.

- ▷ COPY MACHINE PRINCIPLE occurs when the neuron has inputs with over–threshold weights. It is an *information pass–by* or an another way how to handle the redundancy. One of the explanation is that the structure is more complex than is needed. It has repeatedly occurred in "small c_v" problems.

- ▷ ACTIVITY ROUTING We will examine figure 4.5 (b) again. The same information flows through the only one neuron number 2. However, this network performs its task with 92.23% accuracy. How is it possible? The strong negative recurrent loop at 0 causes that a frequent input activity is projected to a sparse and low input for 2 (but still over threshold). In addition, inhibition $0 \to 1$ reduces the output on 1, because 1's threshold is high. On the contrary, frequent activity at 1 excites 2, since the 2's connection from 1 is strong. In this case, 1 also excites

[6] Column shows the best results from table 4.4.

0 to produce more activity. High potential at 2 entails the higher firing rate and causes 5 to fire sooner than 4. From what has been said, the potential value of 2 acts as a semaphore and a router: low one means frequent activity in 0 and the high one, in 1 and allows 5 to fire.



(a) $\overline{isi}$: 20 vs. 40 ms, $c_v = 0.5$ **(b)** $\overline{isi}$: 10 vs. 30 ms, $c_v = 1$

Figure 4.5: Evolved networks with displayed thresholds, weights and latencies.

4.2 Hypothesis Testing of Frequency

In this task the network is given just one input train and it has to decide whether its $\overline{isi}$ is higher/lower than selected critical value $\overline{isi}_\theta$ (see 2.3). In particular, network is providing statistical hypothesis testing with the zero hypothesis $H_0 : \overline{isi} < \overline{isi}_\theta$ against alternative. To put it differently, the network does not have an option to compare two inputs. In addition, it is forced to remember a threshold value. Hence this is a more difficult task, than the previous one: Even psychophysical experiments on human behavior support the idea that humans are better at reporting relative comparisons of stimulus features (luminance, contrast, pressure) than reporting exact values

4.2.1 Results

We modified the structure of the C–network as shown in figure 4.6 (a) for this task. Evolution set up and parameters remained equal as in 4.1.3. For the fitness evaluation,

H_0	$c_v = 0.02$	$c_v = 0.1$	$c_v = 0.5$	$c_v = 1$	$c_v \in_R \langle 0, 1\rangle$
$\overline{isi} < 20\,\text{ms}$	98.57 %	94.72 %	89.48 %	57.20 %	73.91 %
$\overline{isi} < 25\,\text{ms}$	99.00 %	95.04 %	85.59 %	60.66 %	73.78 %
$\overline{isi} < 30\,\text{ms}$	98.97 %	94.64 %	67.53 %	56.86 %	74.21 %

Table 4.6: Hypothesis testing of frequency: $\overline{isi} \in_R \langle 10\,\text{ms}, 40\,\text{ms}\rangle$, 300 ms

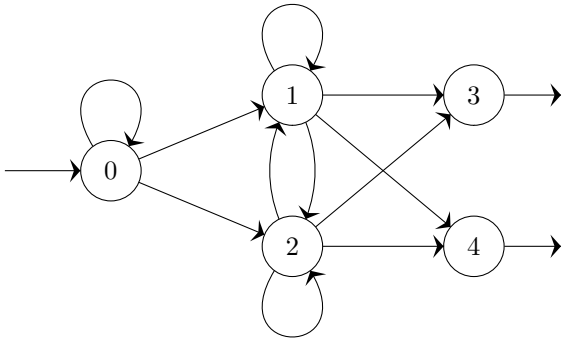
actual $\overline{isi}$ was randomly chosen from $\langle 10\,\text{ms}, 40\,\text{ms}\rangle$. We selected 20, 25 and 30 ms for $\overline{isi}_\theta$. Results are shown in table 4.6. Like in the previous "stereo" input, more regular inputs were easier for frequency decision making. Although the $\overline{isi}_\theta = 20\,\text{ms}$ results can be considered similar to $\overline{isi}_\theta = 30\,\text{ms}$ due to ratio of yes/true tests of the prior distribution π (1/3 vs. 2/3 and 2/3 vs. 1/3), it is to be recognized that a lower $\overline{isi}$ can be determined sooner just because the lower $\overline{isi}$ inputs include more events. Some new strategies have appeared:

Strategies

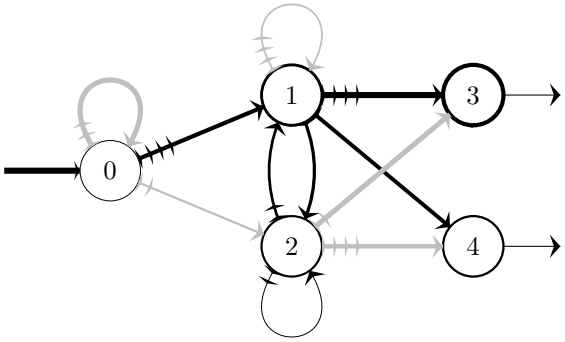
- ▷ GAP DETECTION There is a longer time between the events in the spike trains with the lower frequencies. The network in figure 4.6 (c) detects these gaps in neuron 2 and with the more occurrences causes 4 to fire — figure 4.6 (d).
- ▷ PERFECT TIMING The network in figure 4.6 (e): Neuron 1 is out of the game, 4 fires always. Number 3 is interesting: The parameters are set up to keep 2's firing rate so that 3's potential fluctuates around its threshold. 2's negative recurrent loop emphasizes the difference, when the $\overline{isi}$ is below 25 ms. There is also an evidence that the network remembered 25 ms value by this means.

4.3 More Regular Input

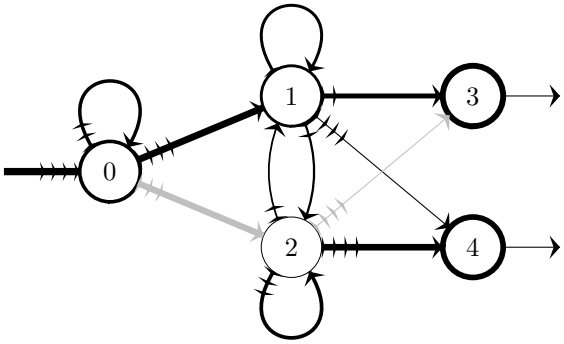
After frequency we focused our attention to another feature: *regularity*. The presence of regularity in biological network paths is also considered the presence of an information transmission. Network has to determine, which one of the input trains is more regular/irregular. Evolution was run on C–networks with the same parameters as in frequency case. Results (table 4.7) have shown that in general more frequent inputs are



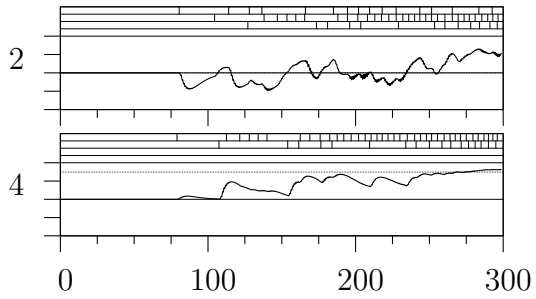
(a) M-network



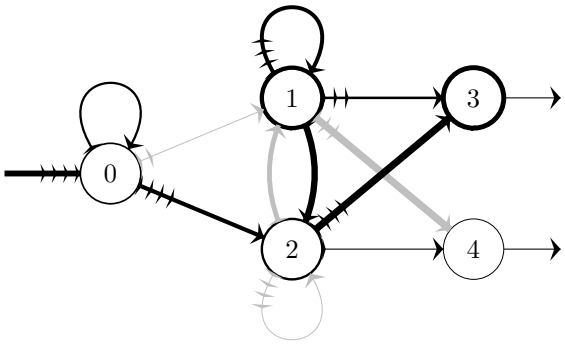
(b) evolved M-network for $\overline{isi} < 25\,\text{ms}$ with $c_v = 0.5$



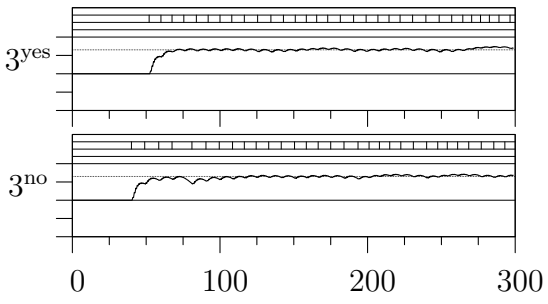
(c) evolved M-network for $\overline{isi} < 25\,\text{ms}$ with $c_v = 0.5$



(d) gap detection — a "yes" decision



(e) evolved M-network for $\overline{isi} < 25\,\text{ms}$ with $c_v \in_R \langle 0, 1\rangle$



(f) "yes" and "no" decision in neuron 3

Figure 4.6: Hypothesis testing of frequency: topology, examples and new strategies

lower c_v	higher c_v	10 ms	20 ms	30 ms	40 ms	$\langle 10\,\text{ms}, 40\,\text{ms}\rangle$
0.02	0.50	99.93 %	99.75 %	98.90 %	98.74 %	74.52 %
0.02	1.00	99.95 %	99.61 %	97.99 %	98.32 %	88.85 %
0.50	1.00	83.05 %	76.55 %	77.44 %	64.32 %	67.08 %
$\langle 0.00, 1.00\rangle$		70.99 %	67.85 %	67.78 %	63.96 %	55.07 %

Table 4.7: More regular input: Comparison of results for different $\overline{isi}$s, 300 ms

easier to decide. We ascribe it again to more events in the spike train. In addition, $c_v = 0.5$ was more distinguishable from $c_v = 0.02$ than from $c_v = 1$. In the former case, the more deterministic input was helpful in decision making.

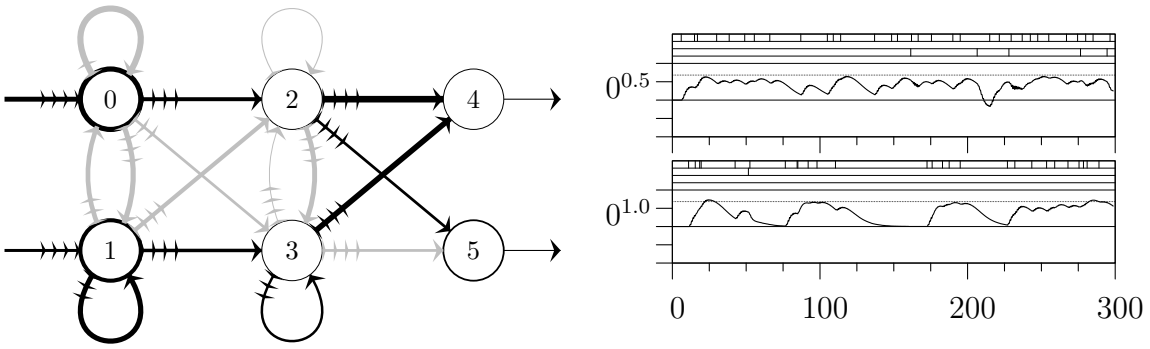
Strategies

Two more strategies appeared and dominated in the regularity decision making.

- ▷ CLOSE EVENTS — principle that has shown here particularly helpful. The resulting set up of network parameters are preset to "wait" for close events. Closer events occur more frequently in the more irregular spike trains. In the most of reviewed examples, thresholds and weights were adapted to raise no activity in the more regular input. See the evolved network in figure 4.7 (a) and corresponding input processing in neuron 0 in figure 4.7 (b). We also noticed it in detecting higher frequency, see figure 4.6 (b).
- ▷ DISTANT EVENTS — a complementary feature to CLOSE EVENTS of irregular input trains. They contain more occurrences of the time intervals with no activity. A decision is then made with the same principle as in GAP DETECTION strategy.

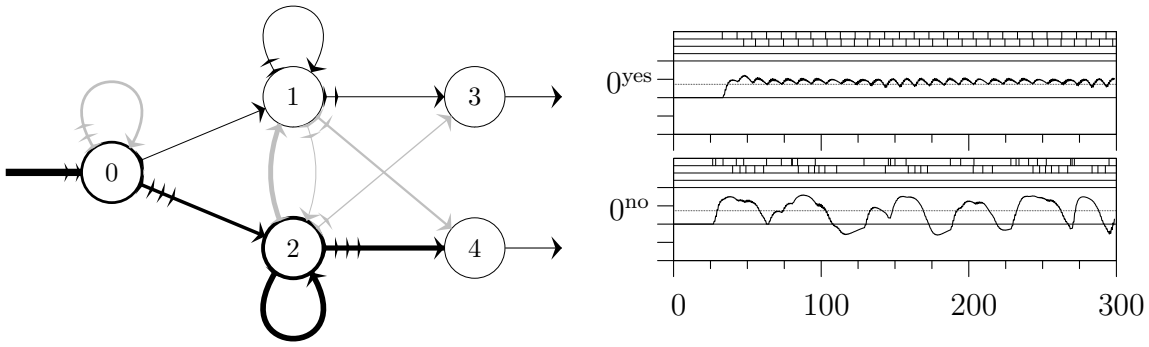
4.4 Hypothesis Testing of Regularity

In this case, the "mono–network" in figure 4.6 (a) tests $c_v < c_{v_\theta}$ against alternative. For the evaluation tests the c_v was randomly chosen from $\langle 0, 1\rangle$. For the threshold value we chose $c_{v_\theta} = 0.5$. Winners (table 4.8) achieved 70 – 80 % accuracy using close events principle in general combined with the following strategies:



(a) evolved network **(b)** "yes" and "no" decision in neuron 0

Figure 4.7: More regular input: evolved network for $c_v = 0.5$ vs. 1 with $\overline{isi} = 20\,\text{ms}$



(a) evolved network **(b)** "yes" and "no" decision in neuron 0

Figure 4.8: Hypothesis testing $c_v < 0.5$ with $\overline{isi} = 20\,\text{ms}$

H_0	10 ms	20 ms	30 ms	40 ms	$\langle 10\,\text{ms}, 40\,\text{ms}\rangle$
$c_v < 0.5$	83.04 %	76.28 %	76.20 %	74.99 %	66.31 %

Table 4.8: Hypothesis testing of regularity for various $\overline{isi}$s: $c_v \in_R \langle 0, 1\rangle$

Strategies

While observing results of hypothesis testing of regularity, we have noticed that the network performs preprocessing at the one–neuron level. A simple setup of neuron values provides various information filtering:

- ▷ MEMORY IN LATENCY — this occurs when recurrent loop has a latency value almost equal to $\overline{isi}$. If the input is rather regular, it gets neuron to highly superpose its potential.
- ▷ FROM REGULARITY TO FREQUENCY — another reasonable use of a latency. With a positive recurrent loop set up to to half of $\overline{isi}$ a neuron stays in the permanent firing when the input is more regular. On the contrary, irregular input causes dropouts in firing. As a result, more regular input is transformed to the more frequent one.
- ▷ IRREGULARITY STOPPING — figure 4.8 (a) and (b) — with immediate negative recurrent loop, neuron cannot fire some time after firing. This eliminates close events and filter more irregular inputs. Then the output is subject to frequency decision making, as well as in the previous case.

Chapter 5

Conclusions

This thesis demonstrated the JASTAP's applicability to model decision making on a neural substratum. The results in more frequent input decision making support the ideas on rate coding abilities. We have tested these capabilities on several topologies and framed them within the theoretical strategies. In like manner we have examined networks whether they can detect and make decisions on regularity patterns in the input trains. Equally important is that we have found the decision makers for rate and regularity decision making.

The advantage of evolution on spiking models like JASTAP over perceptrons is that the results are amenable to analysis. As we have shown, they provide an insight into internal operations on the level of synapses. In the simulations on evolved networks, decision procedures were based on the simple strategies and principles: competing, close events, perfect timing, activity routing, gap detection, from regularity to frequency, the well–known time–and–space summation and others. A network has sometimes found a tricky solution during an evolution, that we had not expected before (as the close events principle), that helped to perform decision making. These strategies are also useful for designing decision makers at NN level.

Indications for further work

Various optimization techniques, evolutionary and traditional, can be used to train NNs. One limitation in our work was the use of fixed topologies. With the structure optimization of a network architecture during an evolution, we could get more accurate

decision makers. Moreover, an evolution of topology can result in as simple individuals as it is needed with respect to given task.

Secondly, we have fixed several parameters to speed up evolution such as: t_1 and t_2 in PSP prototype and $I_{\max}$ and $I_{\min}$ as a firing frequency limitations. In particular, a longer decay in PSP shape had significant positive impact on the results. While including these parameters into evolution entails in more complex search space, it also enable to evolve superior and more descriptive NNs.

Finally, it is possible to include speed preferences into fitness function and get faster decision makers. It is especially meaningful in those tasks, when we got ratios close to 100 %. Nonetheless, more tests in fitness evaluation during evolution reduces the ratio fluctuation and open possibilities to incorporate response time into fitness. Alternatively, one can wish to have promptness as a higher priority than accuracy.

Chapter 6

Abbreviations

$\in_R$	randomly chosen from
c_v	coefficient of variation
μ	statistical mean
σ	standard deviation
σ^2	variance
$\mathscr{E}$	exponential distribution
$\mathscr{G}$	Gamma distribution
$\mathscr{P}$	Poisson distribution
$\mathscr{U}$	uniform distribution
CNS	central nervous system
GA	genetic algorithm
H_0	zero hypothesis
$\overline{isi}$	inter–spike interval
ms	millisecond
MP	membrane potential
NN	neural network
NE	neuro–evolution
PSP	post–synaptic potential
SW	synaptic weight

Bibliography

Blum, A. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271.

Bohte, S. M., Kok, J. N., and Poutré, J. A. L. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37.

Dayan, P. and Abbott, L. F. (2001). *Theoretical Neuroscience*. MIT Press, Cambridge, MA.

Fishman, G. S. (1996). *Monte–Carlo — concepts algorithms and applications*. Springer-Verlag, New York.

Hopfield, J. J. and Tank, D. W. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:144–152.

Janco, J., Stavrovsky, I., and Pavlasek, J. (1994). Modeling of neuronal functions: A neuronlike element with the graded response. *Computers and Artificial Intelligence*, 13(6):603–620.

Koch, C. (1998). *Biophysics of Computation: Information Processing in Single Neurons (Computational Neuroscience)*. Oxford University Press.

Kohonen, T., editor (1997). *Self-organizing maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Kvasnicka, V., Pospichal, J., and Tino, P. (2000). *Evolutionary algorithms*. STU Publishing, Bratislava.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Higher Education.

Navrat, P., Bielikova, M., Benuskova, L., Kapustik, I., and Uger, M. (2002). *Artificial Intelligence*. STU Publishing, Bratislava.

Nolfi, S. (2002). Learning and evolution in neural networks. In Arbib, M. A., editor, *Handbook of Brain Theory and Neural Networks*, volume 2, pages 415–418. MIT Press.

Nolfi, S. and Parisi, D. (2002). Evolution of artificial neural networks. In Arbib, M. A., editor, *Handbook of Brain Theory and Neural Networks*, volume 2, pages 418–421. MIT Press.

Novak, M., Faber, J., and Kufudaki, O. (1993). *Neuron networks and information systems of living orgamisms*. Grada, Praha.

Pavlasek, J. and Jenca, J. (2001). Temporal coding and recognition of uncued temporal patterns in neuronal spike trains: biologically plausible network of coincidence detectors and coordinated time delays. *Biologia, Bratislava*, 56(6):591–604.

Pavlasek, J., Jenca, J., and Harman, R. (2003). Rate coding: neurobiological network performing detection of the difference between mean spiking rates. *Acta Neurobiol Exp (Wars)*, 63(2):83–98.

Pham, D. T. and Liu, X. (1995). *Neural Networks for Identification, Prediction and Control*. Springer–Verlag, London.

Redman, S. and Walmsley, B. (1983). The time course of synaptic potentials evoked in cat spinal motoneurones at identified group ia synapses. *J Physiol (Lond)*, 343(1):117–133.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pages 318–362.

Schaffer, J. D., Whitley, D., and Eshelman, L. J. (1992). A survey of the state of the art. In *Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE.

Schervish, M. J. (1995). *Theory of Statistics*. Springer Series in Statistics. Springer.

Wikipedia (2005). *The Free Encyclopedia*. http://en.wikipedia.org/.