Diploma Thesis

Neural Networks with Dynamic Synapses

Harald M. Fuchs

November, 1998

Institute for Theoretical Computer Science
Graz University of Technology, Austria

—

Adviser: Prof. Wolfgang Maass

Abstract

In this diploma thesis I investigate new models of synapses called "dynamic synapses" (DS) and a possible implementation in analog electronics which could be realized in an analog very large scale integrated silicon chip (aVLSI). These mathematical models of DS were first introduced by Wolfgang Maass and Anthony Zador in 1998 [Maass and Zador, 1998], after recent results in neurobiology suggested that synapses – in their computational meanings – could be more than just static "weights" [Dobrunz and Stevens, 1997],[Zador and Dobrunz, 1997]. It soon became obvious that this model exhibits a behavior very close to their biological counterparts.

For a better understanding of the later presented material a few essential models of neurons and also the typical use of synapses are discussed in the beginning. The pages following after the introduction are dedicated to the original model of the DS [Maass and Zador, 1998]. Then a newer and slightly different model of a dynamic synapse is introduced which is easier to implement in aVLSI. I present results of simulations that reveal the computational power in comparison to the original model. At the end a possible implementation in analog electronics and results of simulations of such implementations are shown. In future work a small neural network with these types of synapses could be realized in aVLSI and applied in real world studies.

Inhaltsangabe: Abstract in German

In dieser Diplomarbeit werden relativ neue Modelle von Synapsen – genannt "Dynamische Synapsen" (DS) und eine mögliche Implementierung in analoger Elektronik, die auch auf einem aVSLI Chip (analog very large scale integration) realisiert werden könnte, untersucht. Diese mathematischen Modelle von DS wurden erstmals 1998 von Wolfgang Maass und Anthony Zador veröffentlicht [Maass and Zador, 1998], nachdem neue Ergebnisse aus der Neurobiologie vermuten ließen, daß Synapsen – in ihrer Bedeutung als Rechenelemente – mehr sein könnten als lediglich statische "Gewichte" [Dobrunz and Stevens, 1997],[Zador and Dobrunz, 1997]. Es wurde schnell augenscheinlich, daß DS ein den biologischen Vorbildern sehr ähnliches Verhalten zeigen.

Zu einem besseren Verständnis des später präsentierten Inhalts werden am Anfang einige derzeit essentielle Modelle von Neuronen gezeigt, ebenso die typische Verwendung von Synapsen. Die der Einführung folgenden Seiten sind dem ursprünglichen Modell der dynamischen Synapsen [Maass and Zador, 1998] gewidmet. Danach wird ein neueres, leicht verändertes Modell einer dynamischen Synapse vorgestellt, welches einfacher in aVLSI zu implementieren ist. Ich präsentiere Ergebnisse von Simulationen, die im Vergleich mit ursprünglichen Modell dessen "computational power" zeigen. Am Ende werden eine mögliche Umsetzung in analoger Elektronik und Simulationen dazu gezeigt. In weiterer Arbeit könnte ein kleineres neuronales Netzwerk mit dieser Art von Synapsen in aVLSI realisiert und in "real world" Studien eingesetzt werden.

Acknowledgments

First of all I would like to say "Thank you!" to Gabriele Burgsteiner who helped me during the last weeks and gave me (nearly) all the support I needed.

I thank my adviser Prof. Wolfgang Maass for all the generous support, the discussions and the inspirations he gave me.

Vicarious for all the people at the Neuromorphic Workshop in Telluride 1998 I thank especially Giacomo Indiveri and Jörg Kramer for the lessons in aVLSI and Chris Diorio for showing me elegant and powerful ways of computation in analog electronics.

I thank all my colleagues at the Institute for Theoretical Computer Science at the Graz University of Technology: Oswin Aichholzer, Thomas Natschläger, Berthold Ruf, Franz Aurenhammer, Ingrid Preininger and Heike Graf for various discussions (also off topics ;) and all the endorsement up to the finishing of my thesis. Special thanks to the system administrator of the Institute, who kept the servers running and prevented any data loss ...

Last but not least thanks to all my friends in Graz who made my stay enjoyable and made my work appear more pleasant: Oliver Friedl, Peter Sabaini (both from NIIL: Network-, Inter- and Intranet Solutions) and all the others whose names would not fit onto this page ...

Contents

List of Figures

1. Introduction

From the early days of the scientific *re*discovery of the functionality of the nervous system [Katz, 1966] and the view of neural networks as computational units [McCulloch and Pitts, 1943], various models of neural networks have been introduced. A few essential models will be discussed in this chapter.

1.1. Simple common models of neurons

1.1.1. McCulloch-Pitts Neuron

One of the simplest but nonetheless powerful components of neural networks is the McCulloch-Pitts neuron [McCulloch and Pitts, 1943]. Because of it's importance for the basic understanding of neural computation I will describe it here in brief. A McCulloch-Pitts neuron $G_{\underline{w}}$ with n inputs and a weight vector $\underline{w} = \langle w_0, \ldots, w_n \rangle$ is described by the function

$$G_{\underline{w}}(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i \cdot x_i + w_0 \geq 0 \\ 0 & \text{else} \end{cases} \tag{1.1}$$

This neuron, which is illustrated in Fig. 1.1, demonstrates in a simplified form a general understanding of nowadays "popular" neurons:

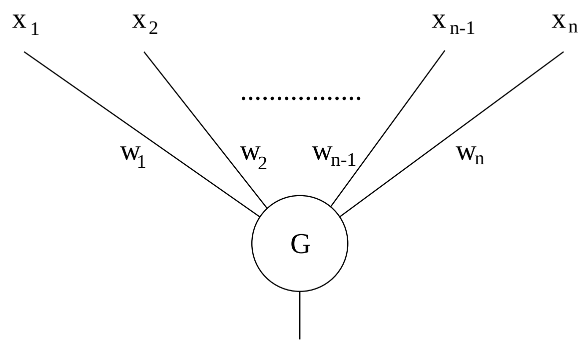

Figure 1.1.: Simplified structure of a neural model

- The main part of an artificial neuronal model consists of a summation point, in this case $G_{\underline{w}}$, described by the equation (1.1). The biological counterpart is the integrating soma. A comparison with a threshold – in this case w_0 – decides about the output of the neuron: 1 or 0 depending on if the sum of inputs times the weights is above or below the certain threshold. In biology the so called *refractory period* determines the maximum spiking rate – or in other words limits the speed of calculation so that other chemical and electrical effects do not get into account.

- n inputs contribute to the computation, where n can be any natural number. In biology, e.g. in certain parts of the brain, 10000 inputs to a single neuron are realistic [Abbott et al., 1997]. The importance of the contribution of each input is measured by it's weight, which can be a positive or a negative number $w \in \mathcal{R}$. The value of each weight determines it's type:
  - $w > 0$: *Excitatory input*
  - $w < 0$: *Inhibitory input*
  - $w = 0$: Not connected

In the McCulloch-Pitts neuron – as well as in most other models – the weights are static on a short time scale. Learning is done by changing these weights and the threshold, after a certain amount of training examples have been evaluated and compared to the results of learning data[1]. By replacing $G_{\underline{w}}$ with a smoother *squashing function* one obtains a linear threshold gate.

1.1.2. Linear threshold gate

Neurons with a smooth squashing function as the activation function have the advantage that more deductive learning algorithms can be employed, because the mathematical descriptions are derivable. This enables the "teacher" of a neural network to use learning algorithms like *back propagation* of the error signal – or *back prop* in short.

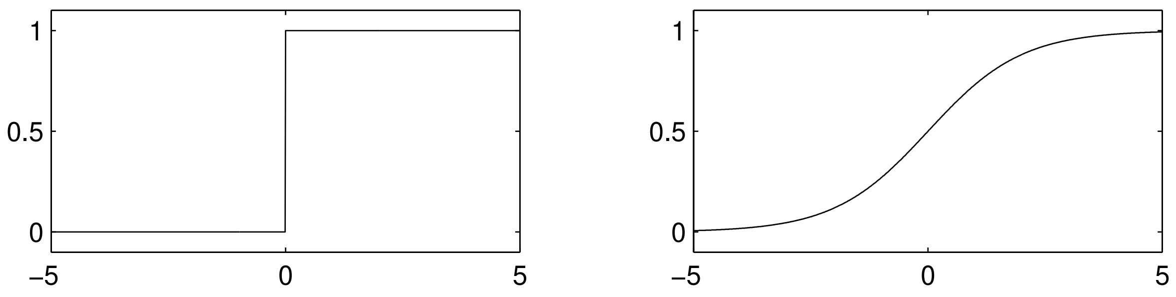

Figure 1.2.: Comparison of different *activation functions*

[1]This is being referred to as "supervised learning"

Figure 1.2 illustrates the difference between the two activation functions and that the left one is not derivable because it is not continuous at point zero on the x-axis. An example for a squashing activation function as used in Figure 1.2 is

$$y = \frac{1}{1 + e^{-x}} \tag{1.2}$$

This kind of activation is better comparable to a biological neuron in the sense of *rate coding*, where the output of a neuron is no more all or nothing but given by the height of the output which is coded in the average firing rate. This was a central dogma in neuroscience for more than half a century. According to Zador et al. recent work in flies, salamanders and several other preparations has shown that at least under some conditions, the timing of individual action potentials[2] can convey substantial information.

1.2. Time domain computation

1.2.1. Spike Response Model

The spike response model (SRM) [Gerstner et al., 1993a], [Gerstner et al., 1993b], [Gerstner, 1995] is different from the neural models mentioned so far. The two latter neurons both operate in analog variable domain and discrete time steps. This means that the inputs are represented as the values of variables on certain discrete points on the time scale. Contrary to this point of view Gerstner et al. use the time domain for coding the input with information carriers called *spikes*. Spikes just mark certain points on the time axis but do not contain any information neither in their height nor in their shape – to say it briefly: all spikes look neglectably different. In mathematical notion a spike is given by a delta pulse at a certain time point $\delta(t - t_i)$.

Keeping in mind that a neuron still has n inputs, one of these spike trains of length F looks like

$$S_j(t) = \sum_{i=1}^{F} \delta(t - t_i) \tag{1.3}$$

At the synapse – the contact between two neurons – still a fixed value (again on a short time scale) J_{ij} determines the influence of the i^{th} input to neuron j. More precisely J_{ij} defines the height and the type of the *post synaptic pulse* (PSP) as the factor the function $\epsilon(t)$ with the axonal delay[3] Δax (see Fig. 1.3).

[2] *Action potentials* are also called *spikes*

[3] This is the time that passes during the transmission of a PSP from the synapse to the soma

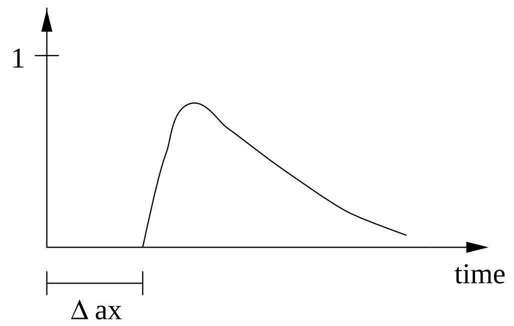

Figure 1.3.: Shape of a typical PSP: $\epsilon(t)$

Again depending on the sign of J the synapse can emerge an excitatory or an inhibitory PSP (EPSP or IPSP). Summing over n inputs reveals the potential at the soma

$$h_i^{syn} = \sum_{j=1}^{n} J_{ij} \int_0^\infty \epsilon(s) \cdot S_j(t-s)ds \tag{1.4}$$

which corresponds to a spatial and temporal summation. Figure 1.4 illustrates this summation for one input S_j where Θ denotes the threshold.

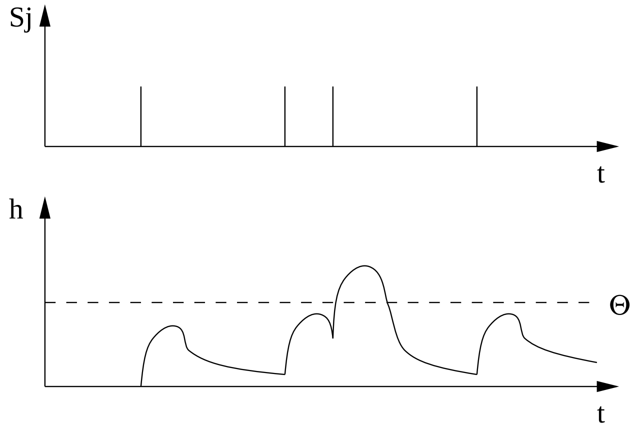

Figure 1.4.: Summation of one input S_j

A relaxed, deterministic neuron generates an action potential whenever h_i^{syn} exceeds the threshold Θ. After a neuron has triggered a spike the *refractory period* determines a delay within which no further action potential can be generated. This refractory period can be modeled by an additional potential

$$h_i^{ref}(t) = \int_0^\infty \eta^{ref}(s) \cdot S_i(t-s)ds \tag{1.5}$$

where

$$\eta^{ref}(s) = \begin{cases} \infty & s \leq \gamma^{ref} \\ \frac{\eta_0}{s-\gamma^{ref}} & s > \gamma^{ref} \end{cases} \tag{1.6}$$

For an illustration of the refractory period see Figure 1.5

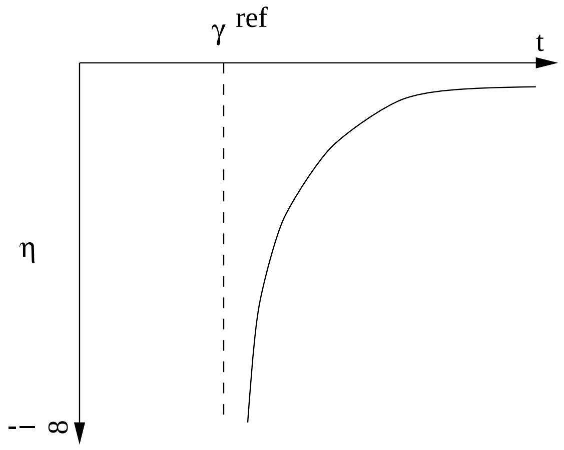

Figure 1.5.: Sketch of the refractory period of the SRM

If we consider other possible influences to the potential in the soma, like an external stimulation, one gets the total sum

$$h_i(t) = h_i^{syn}(t) + h_i^{ref}(t) + h_i^{ext}(t) \tag{1.7}$$

In biology we can observe that even if no spikes arrive at a neuron in the considered time interval – the potential at the soma does not rise – still an action potential is generated from time to time. In the SRM this is not possible so far. The final step is to move from the deterministic to a stochastic model. Therefore noise is being introduced. Hence, the condition for the generation of a spike – if it is not in the refractory period – is no longer:

$$\text{Fire, if } \sum_i h_i(t) > \Theta$$

It is replaced by the conditional probability

$$P_F[S_i(t+1) = 1 | h_i(t)] = \frac{1}{2}(1 + \tanh[\beta(h_i(t) - \Theta)]) \tag{1.8}$$

where β is the noise parameter. For $\beta \to \infty$ one gets back to the deterministic model. The difference between the soma potential at the time t and the threshold Θ is the main influence of the probability that a spike at time $t+1$ is being generated.

With the SRM, no online learning is done by changing the weights after a certain set of training examples. J_{ij} is much more fixed according to the rule

$$J_{ij} = \frac{2}{N(1-a^2)} \sum_{\mu=1}^{q} \xi_i^\mu (\xi_i^\mu - a) \tag{1.9}$$

with

ξ	...	N dimensional vector
q	...	number of vectors or patterns to learn
a	...	average activity: $a = \frac{1}{N \cdot q} \sum_{i=1}^{N} \sum_{\mu=1}^{q} \xi_i^\mu$

For more details see [Gerstner et al., 1993b].

2. Dynamic Synapses

Looking back at McCulloch-Pitts neuron and the spike response model one recognizes that the weights w_i respectively the modifiers of any PSP J_{ij} are still rather static values compared to the high dynamics of incoming spike trains S_j.

Recent results from neurobiology [Dobrunz and Stevens, 1997] and work done by [Abbott et al., 1997] suggest that synapses underlie a very high dynamical process. Synapses get a different role than before [Zador and Dobrunz, 1997]. They are no longer static values that only modify the height of a PSP, but:

- change their characteristics on a very short time scale from one spike to the other.
- do not trigger a release every time a spike arrives at the presynaptic terminal [Katz, 1966].
- the height of PSPs comply with a stochastic process.

The fact that not every arriving spike triggers a release is the main mechanism of dynamic synapses. The *probability* that a PSP evolves due to the arrival of an action potential at the presynaptic terminal is a function of the recent history of the synapse respectively of the spike train $S(t)$.

2.1. Biological background

Figure 2.1 illustrates the mechanisms of a biological synapse. A gap of about 20nm separates the presynaptical terminal from the postsynaptic membrane. Simplified one can describe the relevant mechanisms for the mathematical model as follows: The presynaptic terminal accommodates various Ca^{2+} ion channels that are voltage dependent. They are closed in the relaxed state, so there is no influx of Ca^{2+} ions in the absence of arriving spikes. If an action potential reaches the presynaptic terminal the channels are opened and Ca^{2+} flows in, increasing the potential in the inside. This initiates a movement of packages filled with neurotransmitter – which are called *vesicles* – towards the gap. When a vesicle melts into the presynaptic membrane, the neurotransmitter is released into the small gap and diffuses across the gap to the postsynaptic terminal, where it binds to receptors causing ion channels to open for a short time and produce the

PSP. As the size of this PSP depends on the number and the type of neurotransmitters that are bound to different receptors in a certain time interval the height of the pulse varies from one to the next.

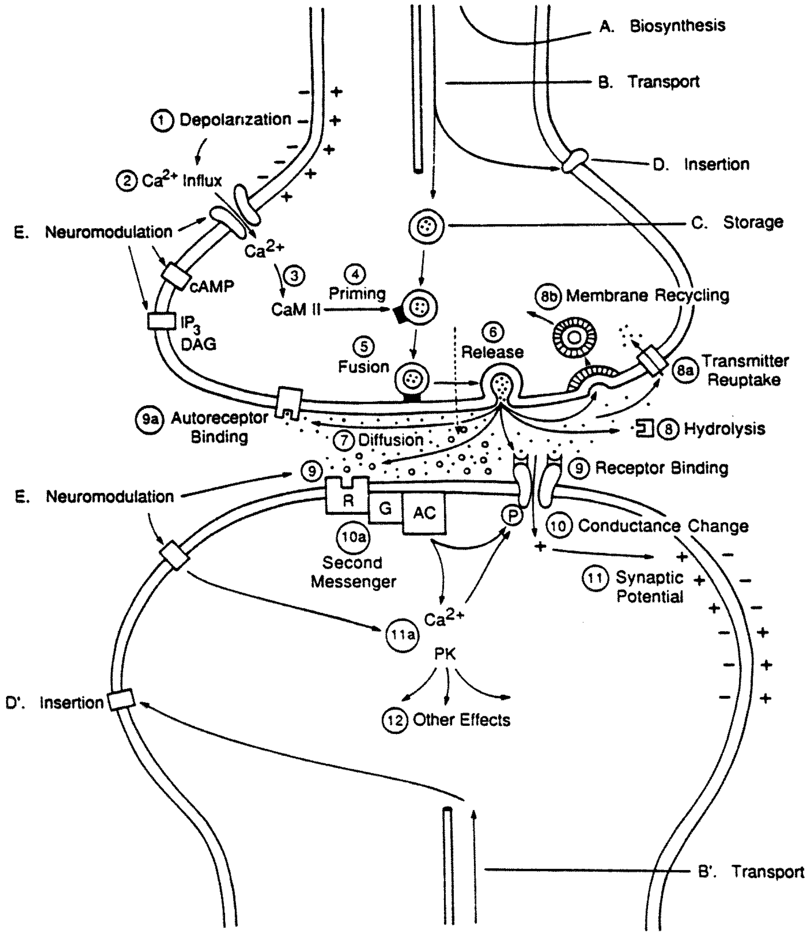

Figure 2.1.: Structure of a biological synapse, from [Churchland, P. and Sejnowski, T., 1992]

As mentioned in the previous chapter a synapse which generates an EPSP (this *increases* the probability of spike generation) is called excitatory. If it reduces this probability by contributing an IPSP, it is inhibitory. Simplified, these two types of synapses can usually distinguished by their reversal potential E_{syn} in relation to the threshold Θ of the postsynaptic neuron. When E_{syn} exceeds Θ the synapse is excitatory, otherwise it is inhibitory.

There exist different types of neurotransmitters which bind to a specific receptor type. A receptor that binds to the neurotransmitter called glutamate is the N-methyl-D-aspartate (*NMDA*) receptor, which is very important for learning. The influx of Ca^{2+} ions – like described above – is due to the opening of ion channels caused by the activation of NMDA receptors.

For every incoming spike at most one vesicle is being released. The concentration of Ca^{2+} ions decreases with time – down to a varying resting potential.

As one can imagine at least two limitations of this electro-chemical process are observable:

- The amount of *readily releasable vesicles* is limited. Which means that after the "pre-produced" vesicles are done, no further release can happen until new neurotransmitter packages have been rebuilt. In biological systems the average amount of readily releasable vesicles is about 15 to 20.
- The concentration of Ca^{2+} ions is depending on the time constants of the ion channels, in other words: how fast the concentration can be raised and how slow the potential decreases in the interval between two spikes.

Referring to [Dobrunz and Stevens, 1997] the probability changes due to these limitations over a short time scale. Figure 2.2 displays biological datas[1].

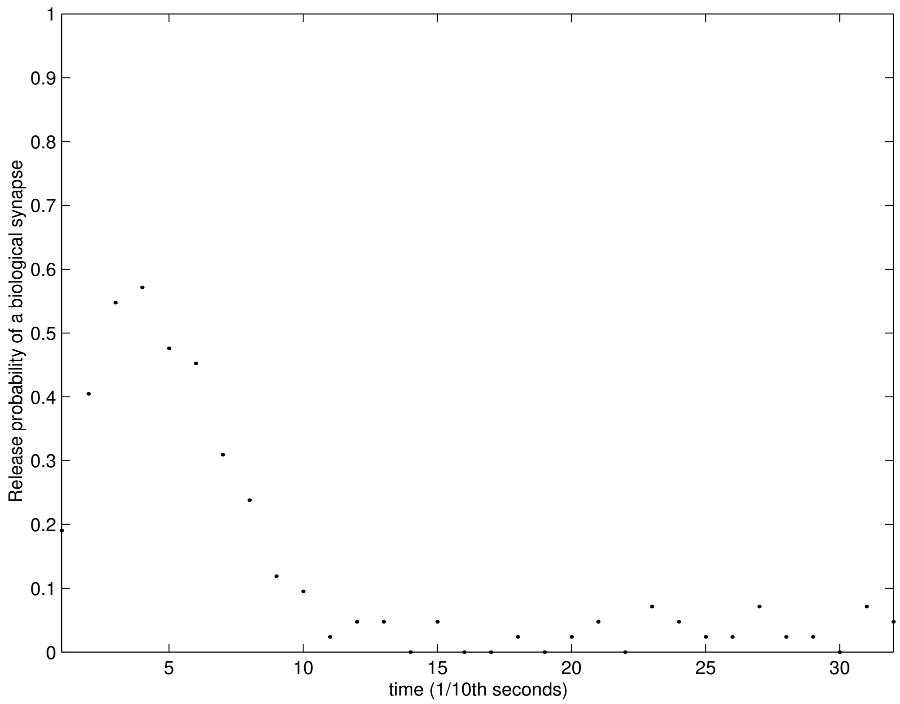

Figure 2.2.: Release probabilities for different spikes in a biological synapse

The probabilities were calculated by averaging releases over 42 trials of spike trains with 32 spikes each with interspike intervals of 100ms with 20 seconds resting time in

[1] From Lynn Dobrunz, The Salk Institute

between each trial on the same synapse. Figure 2.3 shows 8 of the 42 release patterns[2] where one can see, that the process of releasing PSPs is not deterministic. $1s$ mark releases, $0s$ mark "failures" of the synapse on an incoming spike.

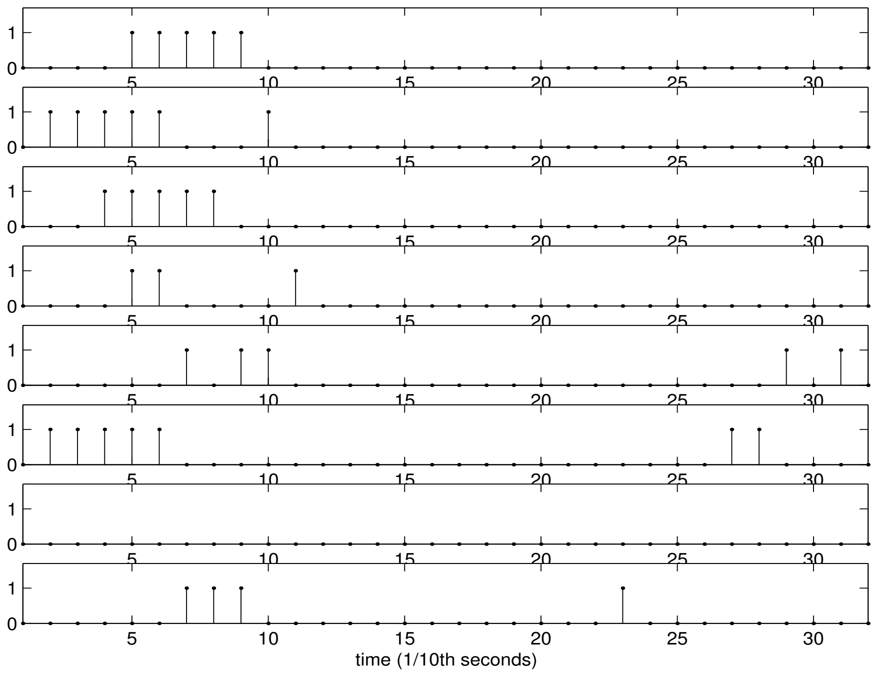

Figure 2.3.: Eight selected release trains of biological datas

[2]From Lynn Dobrunz, The Salk Institute

2.2. Stochastic Synapse

[Abbott et al., 1997] described a model of intracortical synaptic short-term depression to provide dynamic gain-control mechanisms for increasing the sensitivity of a neuron to subtle changes in the firing patterns of an incoming spike train. From the main mechanisms of a biological synapse, Maass and Zador derived a mathematical model of a dynamic synapse [Maass and Zador, 1998]. Using a similar definition of a spike train like in Formula 1.3, where

$$S = \{t_1, t_2, \ldots, t_n\} \tag{2.1}$$

we can describe the *facilitation* of the synapse in principle as the *simplified* Ca^{2+} concentration:

$$C(t) = C_0 + \sum_{(t_i<t)\wedge(t_i\in S)} c(t - t_i) \tag{2.2}$$

where C_0 is a constant greater than or equal to zero. I represents the *resting concentration or resting potential* of calcium ions in the synapse like explained in Section 2.1. $c(t - t_i)$ models the concentration of Ca^{2+} in the presynaptic terminal and can be for example

$$c(t - t_i) = \alpha \cdot e^{-\frac{t-t_i}{\tau c}} \tag{2.3}$$

where α describes the influx of Ca^{2+} through the ionic channels at the arrival of an action potential and τc is responsible for the decay.

The amount of readily releasable vesicles – the *depletion of a synapse* – is modeled in a similar way. Let us assume that V_0 is a constant greater than zero, that models the amount of vesicles that rest in the presynaptic terminal and have been pre-produced during a long inactive period and R is set of time points for which the synapse has released a vesicle: $R \subseteq S$. Then the development of readily releasable vesicles can be described as

$$V(t) = max(0, V_0 - \sum_{(t_i<t)\wedge(t_i\in R)} v(t - t_i)) \tag{2.4}$$

and again $v(t - t_i)$ can be a function similar to

$$v(t - t_i) = e^{-\frac{t-t_i}{\tau v}} \tag{2.5}$$

If we further assume that the probability that a synapse releases a vesicle filled with neurotransmitter depends on the states of the Ca^{2+} concentration and the amount of readily releasable vesicles in the presynaptic terminal, then in our model

$$p(t) = 1 - e^{-V(t) \cdot C(t)} \tag{2.6}$$

which is illustrated in Figure 2.4.

The dynamics of equations 2.1 to 2.6 are illustrated in Figure 2.5.

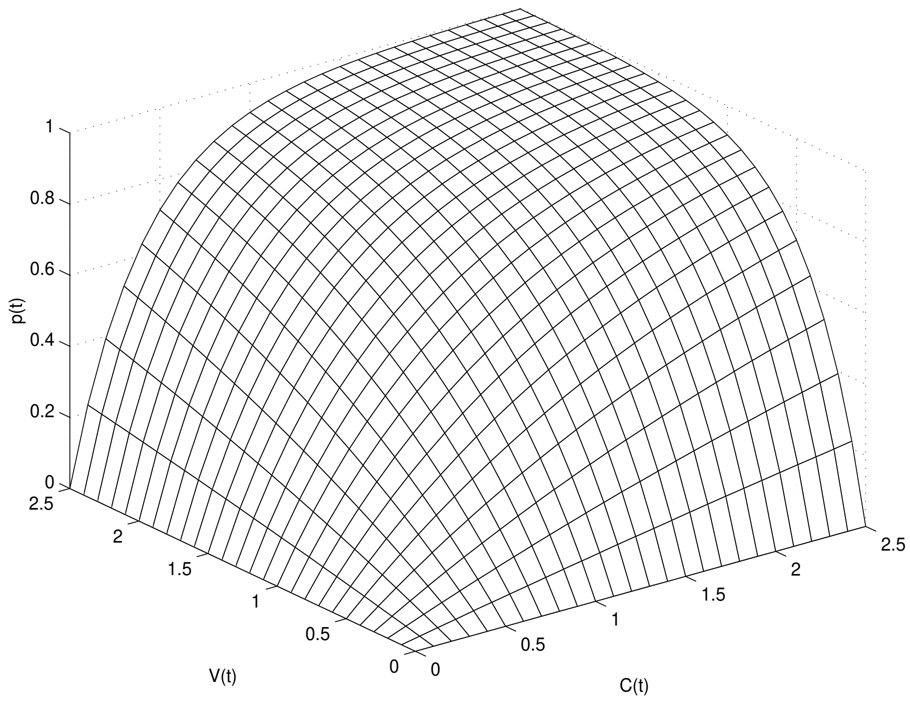

Figure 2.4.: Dependence of $p(t)$ from $C(t)$ and $V(t)$

Notice that $p(t)$ is defined only on times $t \in S$, so no spontaneous release can occur. $R's$ mark examples of releases of a vesicle, $F's$ mark failures. So the complete mathematical model of the synapse can be described by five variables: C_0, V_0, τ_c, α and τ_v.

Further computational implications of these dynamics are discussed in [Maass and Zador, 1998]. The most important consideration for the next chapters is that in this model the "weights" of a synapse are *not* constant. Contrary, they change even from one incoming spike to the next.

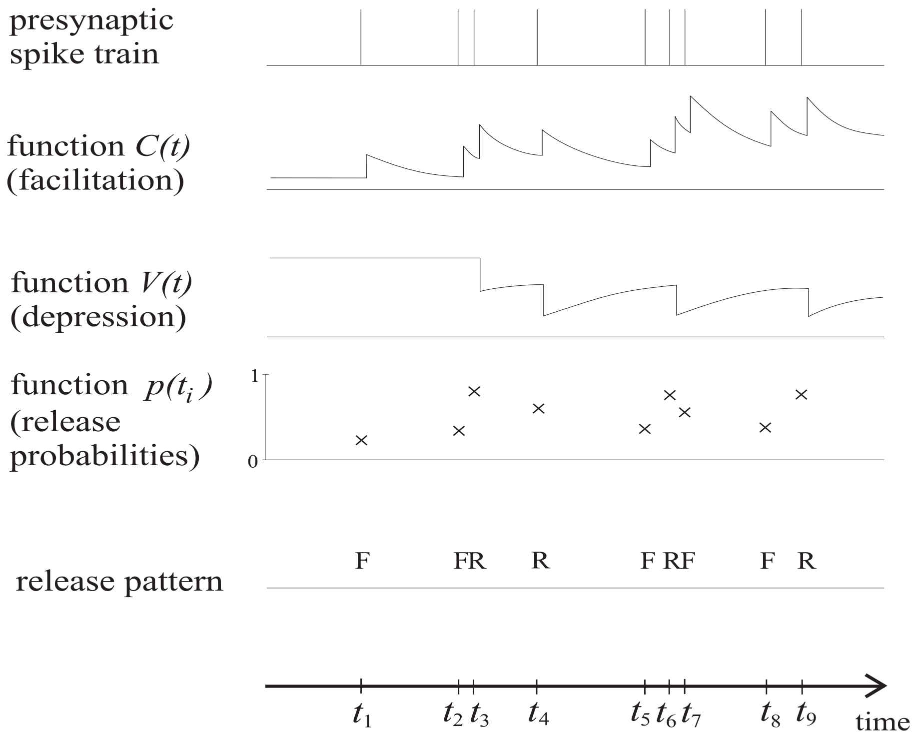

Figure 2.5.: Behavior of the stochastic synapse model, from [Maass and Zador, 1998]

2.3. "Tuning"

In this section I will describe the results when confronting the stochastic synapse with the biological ideal. After a few trials it became obvious that the nontrivial connection of the model's five parameters plus the need for a derivable error function made gradient descent too complex and time consuming to be used for this task. Also the non-deterministic behavior of the DS – after the probability in function 2.6 is calculated, it is determined whether a release occurs by evaluating the probability – forces to build averages over various trials. The process I finally used for the tuning is called *simulated annealing* which is described in detail in Appendix A. The simulations were done with a scientific software package called Matlab, version 5.0. The source codes for the simulations can be found in Appendix B. Biological datas from experiments done by Lynn Dobrunz (The Salk Institute) were used. Some of the datas have already been presented in Figures 2.2 and 2.3.

One of the experiments should explore the dynamic possibilities of the synapse. Therefore two Poisson distributed spike trains with 10 spikes each have been generated and used as input for two different synapses – synapse A and synapse B. The criteria

$$\frac{\sharp Releases\ for\ S_1}{\sharp Spikes\ in\ S_1} - \frac{\sharp Releases\ for\ S_2}{\sharp Spikes\ in\ S_2}$$

that should be maximized has been used in the simulated annealing tuning process. Because the number of spikes is the same in both spike trains the difference becomes maximized when the synapse A releases for all spikes in train "1" and fails for every incoming spike in train "2". After the first tuning process one multiplies the criteria by -1 and trains synapse B.

The results are shown in Figure 2.6 and discussed in detail in [Maass and Zador, 1998]. They were achieved with the help of software developed for this diploma thesis. The left column show the behavior of synapse A and the right one for synapse B. The upper row displays spike train 1 and the lower one spike train 2. One can see that the responses of the synapses A and B are quite different and even the *average* release probabilities of a synapse differ for the two spike trains between 16% and 22%.

Another question that arose was: "Is it possible to get a similar course of release probabilities like a biological synapse for certain spike trains?". The synapse was trained for the datas depicted in Figure 2.2 with the task to minimize the summed square error of the difference between the result of the model and the biological datas. With biological plausible starting values for C_0, V_0, τ_c, α and τ_v, after 10.000 simulation steps the result shown in Figure 2.7 emerged. Note that the datas from the stochastic synapse are averaged over 1000 runs with the winning parameters and that there still could exist a better fit for this model to the biological data.

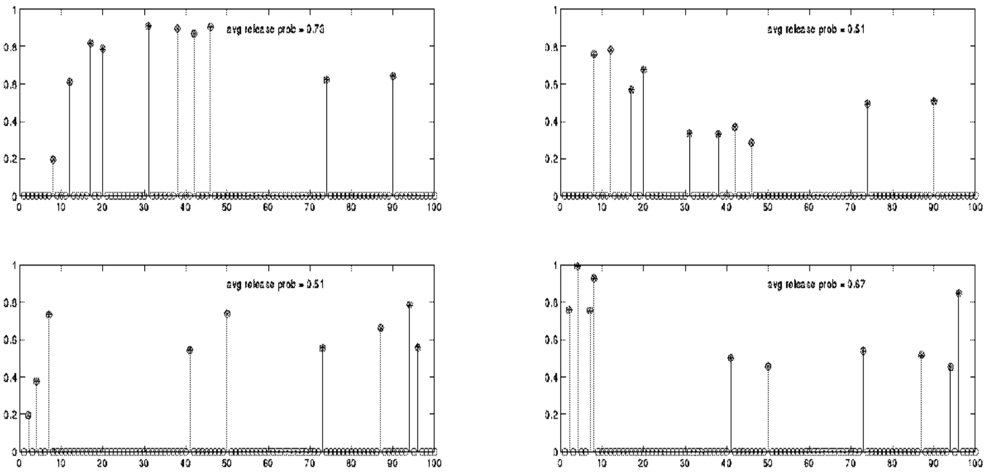

Figure 2.6.: Dynamic behavior of two synapses for two similar spike trains

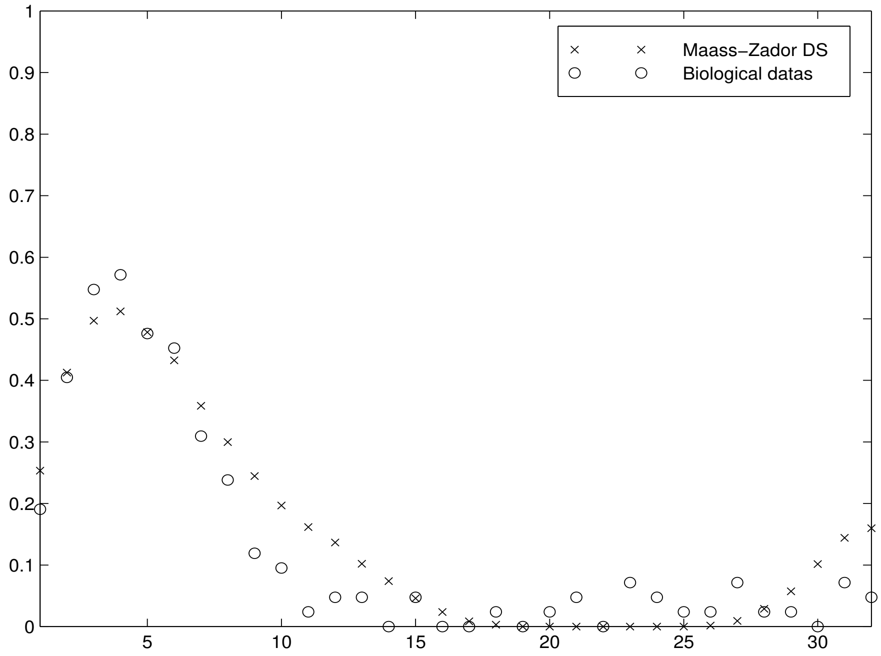

Figure 2.7.: Comparison between the stochastic synapse and biological datas from Lynn Dobrunz

2.4. Limitation of $p(t)$

A limitation of the model can be observed, though. When one considers the amount of change of the release probability between times t_k and t_l out of $S = \{t_1, t_2, \ldots, t_k, t_l, \ldots, t_n\}$ as $|p(t_l) - p(t_k)|$ where

$$\begin{aligned} p(t_l) &= 1 - e^{-C(t_l)\cdot V(t_l)} \\ p(t_k) &= 1 - e^{-C(t_k)\cdot V(t_k)} \end{aligned}$$

one gets

$$\begin{aligned} |p(t_l) - p(t_k)| &= |(1 - e^{-C(t_l)\cdot V(t_l)}) - (1 - e^{-C(t_k)\cdot V(t_k)})| = \\ &= |1 - e^{-C(t_l)\cdot V(t_l)} - 1 + e^{-C(t_k)\cdot V(t_k)}| = \\ &= |e^{-C(t_k)\cdot V(t_k)} - e^{-C(t_l)\cdot V(t_l)}| \end{aligned} \tag{2.7}$$

As $p(t)$ can only take values between 0 and 1 the difference in equation 2.7 will be maximized, when $p(t_l) = 1$ and $p(t_k) = 0$ or the other way around.

Due to the characteristics of Equation 2.6 it is obvious, that if the release probabilities for the first two spikes $\{t_1, t_2\} \subseteq S$ are zero, the release probabilities for all other action potentials $t_i \in S$ are also zero. Two cases have to be considered:

- $V(t) = 0$: If V_0 is zero (otherwise $V(t_1)$ cannot be zero) it will stay at this value, because it can only be *decreased* for incoming spike trains. Hence, it remains zero for the whole simulation.
- $C(t) = 0$: Is would be possible that the resting concentration of Ca^{2+} is zero but not negative. Hence, $C(t_1)$ can be zero. To achieve that $C(t_2)$ is *also* zero, α has to be zero, too. This means that the potential never rises.

$\Rightarrow$ The release probability for one or both of the first two arriving action potential is always greater than zero or it is zero for all spikes.

The stochastic synapse is able to achieve complete *suppression* for all but the first spike. See Figures 2.8 and 2.9 for a demonstration of this ability.

But it can be easily seen that it would be an impossible task for the mathematical model of a dynamic synapse to switch from one probability extremum to the other and back again between three successive incoming spikes. For reasonable values of the parameters the dynamic change of $p(t)$ is limited.

Figure 2.10 helps to figure out how to achieve the biggest possible probability changes. For every incoming spike $C(t)$ becomes increased by a value α. When a PSP is being

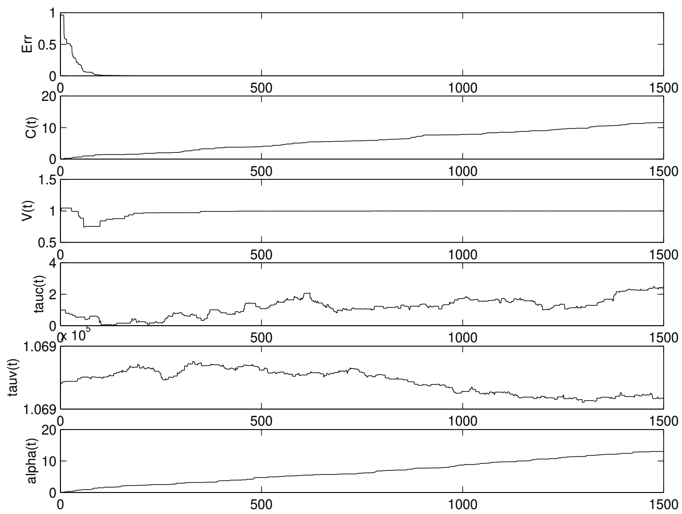

Figure 2.8.: Tuning session for suppressing all but the first incoming spikes

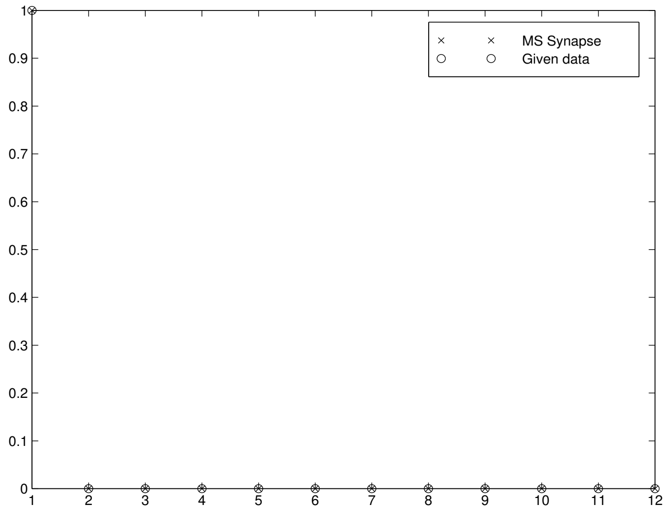

Figure 2.9.: Suppression of all spikes but the first one

released, the amount of readily releasable vesicles is being decreased. Both, $C(t)$ and $V(t)$ drift back to their resting potentials C_0 and V_0 lead by their time constants τ_c and τ_v. As the change of the probability due to $V(t)$ is highest when $V(t)$ does not hit the floor (would not become negative, so the $max()$ function does not get into account) we consider only releases at times t_i with $V(t_i) \geq 1$. C_0 should be as small as possible but still greater than zero because the dynamics of $p(t)$ are highest for small values of $C(t)$ and $V(t)$ (see Figure 2.4).

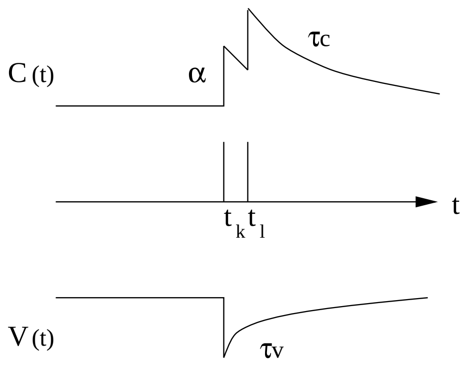

Figure 2.10.: Explanation of probability changes

Two different cases must be considered:

- $p(t_l) > p(t_k)$: For a fixed Δt between two arriving action potentials the release probability from p_k to p_l increases most when

  · V_0 is greater or equal one
  · α is as high as possible
  · τ_c is infinite
  · τ_v is zero
  · no release occurs at time t_k

- $p(t_l) < p(t_k)$: Contradictions arise when one also demands that the probability also *decreases* very rapidly in the same synapse for the same Δt. Then

  · V_0 should still be greater or equal one, **but**
  · α should be equal to zero
  · τ_c should be zero
  · τ_v should be infinite
  · a release must occur at time t_k

For more realistic values of τ_c and τ_v – not equal to zero[3] or infinity – also Δt must be kept very small. With the limitation arising from the above contradictions the model can not implement a "probability unit step or pulse function". This means that it is not capable of guaranteed *suppression* or *emphasization* of certain single spikes. Comparing with biology this is not necessarily a disadvantage. In nature no such computations on spikes trains in neural networks are known today. More than that the datas from Lynn Dobrunz suggest smooth courses of release probability development.

Experiments done in Appendix A with simulated annealing revealed changes of release probability of at most 0.48 between two incoming spikes for a Δt of 100ms and more realistic parameter values.

[3] this would mean that in the model the equations 2.3 and 2.5 are replaced by a Dirac impulse

With the parameter values $C_0 = 0, V_0 = 4, \alpha = 0.09, \tau_c = 20$ and $\tau_v = 11$ that resulted from annealing sessions with biological datas $\Delta p_{max} \approx 0.21$ for values of $\Delta t's$ between 1ms to 1s.

3. Modified stochastic Synapse

3.1. Motivation

The model shown in section 2.2 is simple and powerful for use in a mathematical surrounding. When trying to implement this synapse in aVLSI some problems arise:

- One would have to explicitly evaluate the calculated probability $p(t)$, as in equation 2.6. This means that for every $p(t)$ at times $t \in S$ a decision has to be made whether or not a PSP is released. This leads to a difficult situation in analog electronics. For example, it would be possible to generate a certain amplified noise level and compare it to a changing threshold $p(t)$ but at enormous costs for chip size because this building block would have to be individual for each synapse due to the need for independence of the probability processes.

- Although one transistor [Diorio and Minch, 1996] seems to be too little to get a similar computational power like a biological synapse, this kind of synapse needs more elements than most of any other known synapse models. This drastically limits the number of synapses one can put on a chip or wafer compared to others.

When one considers space code or population coding to represent information in a neural network with dynamic synapses [Maass and Natschläger, 1998], a solution to these problems arises due to a simplification. Information is coded in n pools $U_1, \ldots, U_n$ consisting of N neurons each which are fully connected to another pool V of N neurons like shown in Figure 3.1.

When we assume that all corresponding synapses of the N neurons of each pool U_i have very similar or identical parameters, we can approximate the synaptic behavior of a pool by "averaging" over their activities.

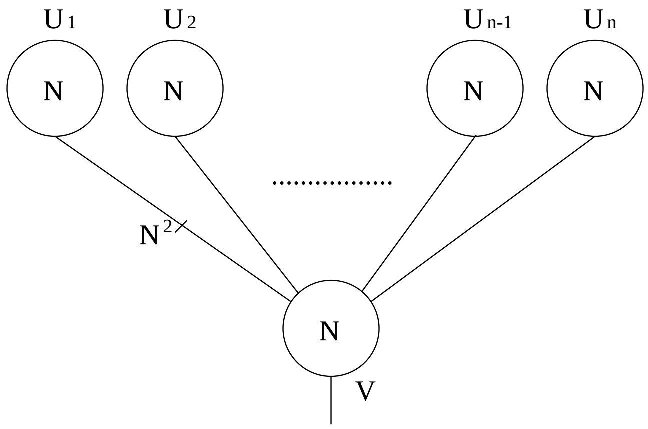

Figure 3.1.: n pools of N neurons each fully connected to a pool V of N neurons

3.2. Forming the model

Let us consider a set M of N dynamic synapses which all have the same parameters C_0, V_0, α, τ_c and τ_v. Every synapse in this set receives the same input spike train and contributes to a common soma. The synapses are all in a relaxed state – no spike has arrived for a long time period.

When the first action potential of the spike train $S = \{t_1, \ldots, t_n\}$ arrives at the presynaptic terminals of synapses M_1 to M_N the concentration of Ca^{2+} ions $C(t)$ increases with α – which is per definition the same for all. So every synapse calculates the release probability like in section 2.2, equation 2.6 and gets the same result.

A PSP is released with the probability $p(t_1)$. When one looks at the behavior of all N synapses of the given set, then for *many* repeated trials or for $N \to \infty$, also $p(t_1)$ synapses will release a vesicle filled with neurotransmitter. Hence, $p(t_1)$ per cent of the synapses decrease $V(t)$ at time t_1 and $(1 - p(t_1))$ per cent do not. At the soma $N \cdot p(t_1)$ PSP's arrive.

To build a bridge to a modified stochastic synapse imagine all N pools of readily releasable vesicles and also the corresponding PSP's united to a "super synapse". Therefore, after a release, $N \cdot p(t_1)$ is being subtracted from this pool of $N \cdot V(t)$ vesicles. After normalization of these former N synapses at the soma with respect to N one can write down a mathematical model:

$$C(t) = C_0 + \sum_{(t_i < t) \wedge (t_i \in S)} c(t - t_i) \tag{3.1}$$

$$c(t - t_i) = \alpha \cdot e^{-\frac{t - t_i}{\tau_c}} \tag{3.2}$$

which is equal to the definition in the original stochastic synapse. The development of readily releasable vesicles is now modeled as

$$V(t) = max(0, V_0 - \sum_{(t_i<t)\wedge(t_i\in R)} p(t_i) \cdot v(t - t_i)) \tag{3.3}$$

and again $v(t - t_i)$ can be a function similar to

$$v(t - t_i) = e^{-\frac{t-t_i}{\tau_v}}$$

When we still assume that the probability, that this synapse releases a vesicle, depends of the states of the Ca^{2+} concentration and the amount of readily releasable vesicles in the presynaptic terminal, then in the new model

$$p(t) = 1 - e^{-V(t)\cdot C(t)} \tag{3.4}$$

While it is easy to prove that this is not only an approximation to a pool of stochastic synapses but an exact simplification for the *first* incoming spike, it is very hard to lead a mathematical proof for a spike train of arbitrary length.

This <u>m</u>odified <u>s</u>tochastic (MS) synapse has – from the electronic engineering point of view – the advantage that it is easier and straight forward to implement. No noise generators or whatever are necessary any more.

The next chapter will illustrate that it still shows a similar behavior like the original model from Maass and Zador for the same synaptic parameters.

4. Simulations with the MS-Synapse

4.1. Fitting with biological data

After forming the model for MS synapse, the question arose: "Can this model behave like a biological synapse – at least when we keep in mind what is known of it today?". Two different results of experiments with synapses from the Salk Institute were given to us.

4.1.1. Linear spaced train

Like in the experiments in Chapter 2.3 the datas consisted of 42 sets of release patterns with a length of 32. The corresponding releases were summed up and divided by the amount of sets to get an "average" of the releases. By using the programs and subroutines listed in Appendix C the modified stochastic synapse was trained to behave like in the given datas.

As the criteria for fitting I used the summed square error[1] over the averaged biological datas (like described above) and the response from the MS synapse. The goal was to minimize this error.

The results of this tuning session – which are shown in Figure 4.1 – emerged after only a few hours of "training" done with simulated annealing. It shows a comparison of the datas from Lynn Dobrunz (marked with an "o") and the MS synapse (marked with "x").

A short section near the end of the annealing process is displayed in Figure 4.2.

The parameters that were finally found are

· $C_0 = 0$
· $V_0 = 3.23969$
· $\tau_c = 24$
· $\tau_v = 10.6897$
· $\alpha = 0.0894924$

[1] This is more a geometrical fit than a logical one. To be exact, one should have used something similar to the likelihood criteria (see Chapter 2.3 and Appendix B).

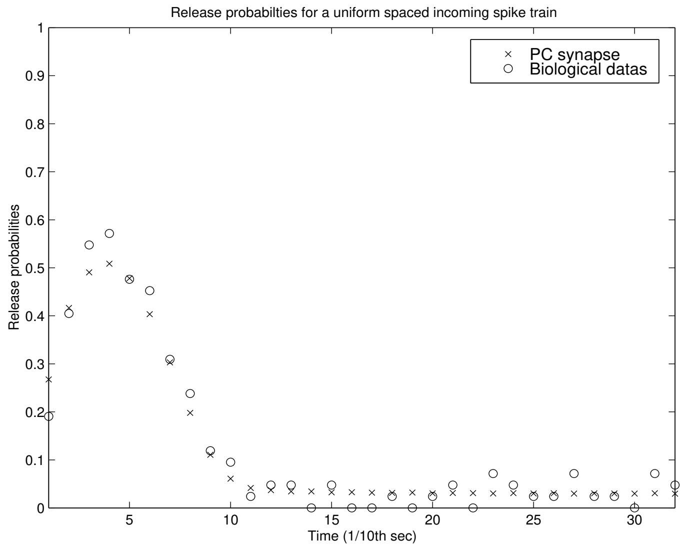

Figure 4.1.: Fitting of the modified stochastic synapse with biological datas

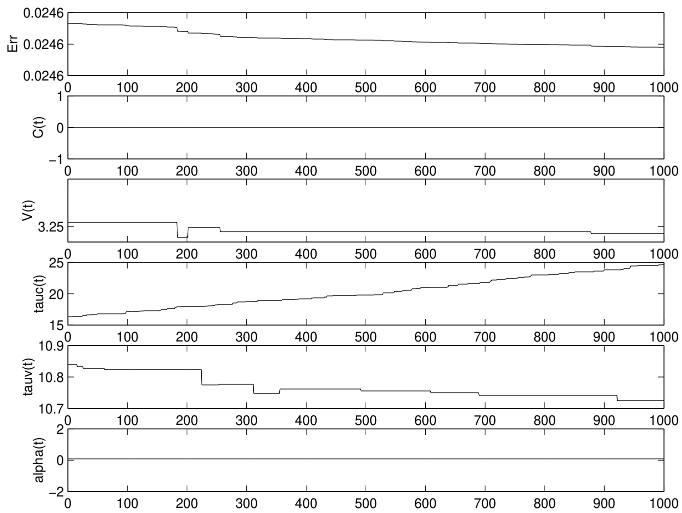

Figure 4.2.: A cut-out of the annealing process

This seems to be a quite plausible result. Experiments from neurobiology suggest that the time constants are in this range of up to 20 seconds. Only the amount of readily releasable vesicles is a bit low. The pool could have a size of about 15 to 20.

On the other side this model shows a more dynamic behavior in terms of Δp compared to the results in Figure 2.7. It keeps the pace of the biological data more accurate. Note that the results from Figure 2.7 had been achieved with the same parameter values.

4.1.2. Getting more stability

When looking at the descriptions of the biological datas I noticed that the several trials with the biological synapse have been made on a single synapse. The interspike intervals are constant 100ms for 32 spikes in a series. This was repeated 42 times every 20 seconds. The next experiment with the tuned model from the last subsection was to examine the behavior for this original stimuli. I used only ten repetitions of the bursts to reduce simulation time. It brought a sobering effect.

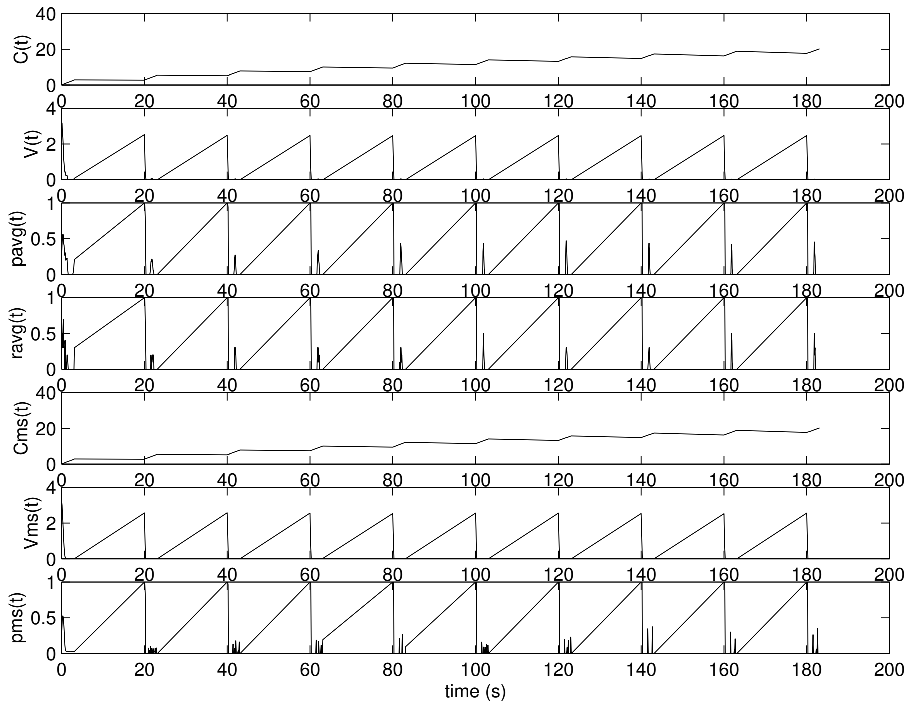

Figure 4.3.: Previously tuned model with original stimuli

It showed that the time constants are far too high for longer experiments. So $V(t)$ and especially $C(t)$ could not reach the starting values again, even after a quite long break of about 20 seconds (see Figure 4.3). This resulted in too high release probabilities for the first spikes of the additional "bursts" while the probabilities for the other action potentials are nearly zero.

The columns from top to bottom display $C(t)$, $V(t)$, the averaged release probabilities $p_{avg}(t)$ and $r_{avg}(t)$, the averaged actual release over 100 trials for the stochastic synapse. Beyond are $C(t)$, $V(t)$ and $p_{ms}(t)$ of the modified stochastic synapse.

Figure 4.4 shows a cut-out from seconds 20 to 23.2. An interesting side effect of this "failed" tuning were the irregularities for the spikes 10 to 32 that look quite similar to the biological datas. This stochastic effect was not reproduceable in tuning sessions, where simulated annealing always found minimas with the best smooth fit.

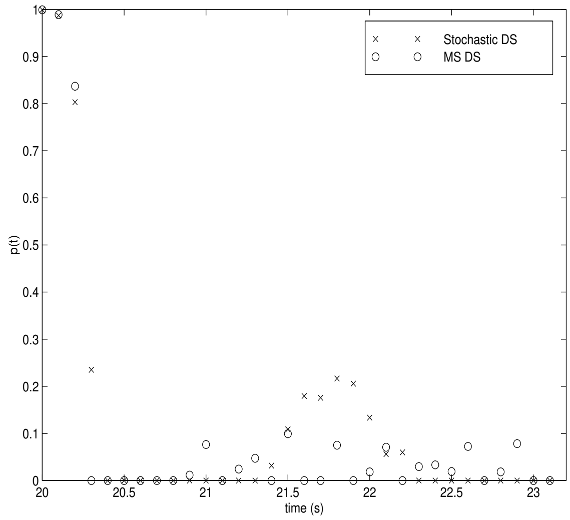

Figure 4.4.: Interesting side effect of the "failed" tuning

Also when applying a *long* spike train with interspike intervals shorter than the break needed by the synapse to get back to it's resting potentials as an input the model shows instable behavior, shown in Figure 4.5.

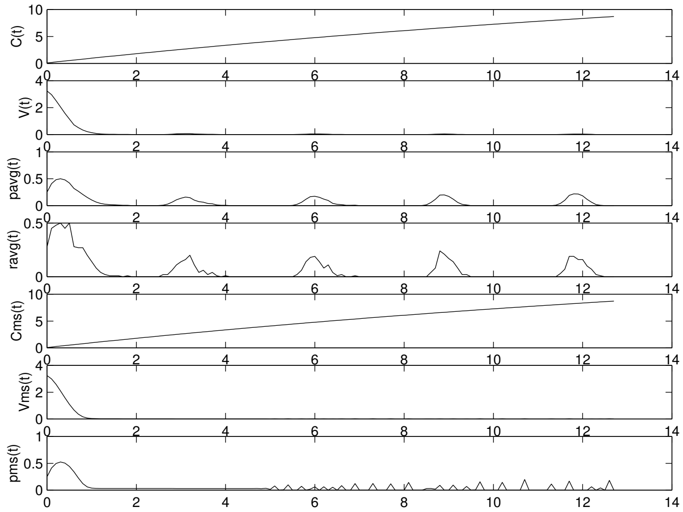

Figure 4.5.: Development of the parameters for a long spike train

A difference between the original stochastic synapse and the modified model is visible in Figures 4.5 and 4.6. The original stochastic synapse locks to a kind of oscillation with a period of about 2.8 seconds while the modified model shows "just stochastic behavior" at a first glance. Again these behaviors are caused by high values of $C(t)$ after about 40 spikes of static low release probabilities (for the modified stochastic synapse).

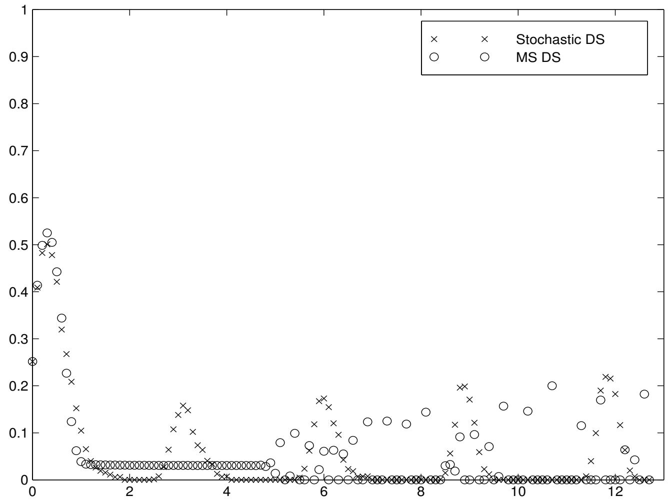

Figure 4.6.: Irregularities for long spike trains

Figure 4.6 shows these behaviors for the two synaptic models with a spike train of 128 spikes with interspike intervals of 100ms.

4.1.3. Retuning

To make the synaptic behavior more stable I retuned the model. This time with ten repetitions of the averaged biological datas copying after each other in time domain at distances of 20 seconds and with the found parameters for C_0, V_0, τ_c, α and τ_v from the last "failed" tuning session as starting values.

Simulated annealing found new parameters astoundingly fast. Note the decrease of error signal from about 30 to 0.35 at the end. Figure 4.7 shows this annealing process.

The outcome was a fit that looks qualitatively very similar to Figure 4.1 and can be seen in Figure 4.8.

The new resulting parameters are

· $C_0 = 0$
· $V_0 = 3.70854$
· $\tau_c = 2.3469$
· $\tau_v = 9.1866$
· $\alpha = 0.0959902$

with the effect, that the response shown in Figure 4.8 is stable for arbitrary repetitions of the 100ms bursts at distances of 20 seconds.

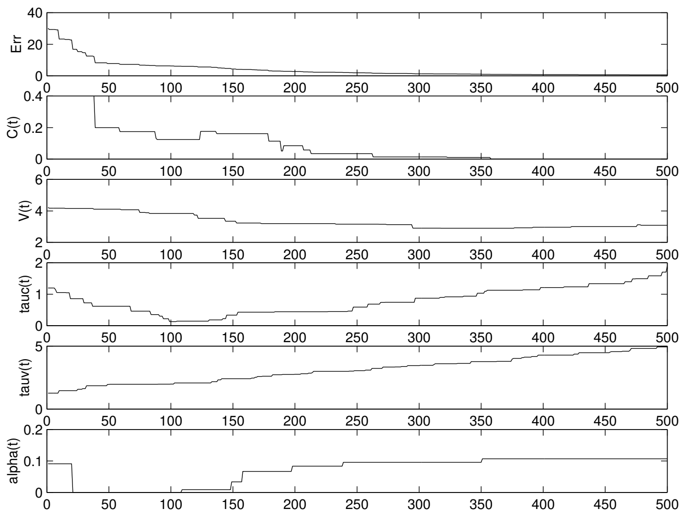

Figure 4.7.: Retuning the modified stochastic synapse

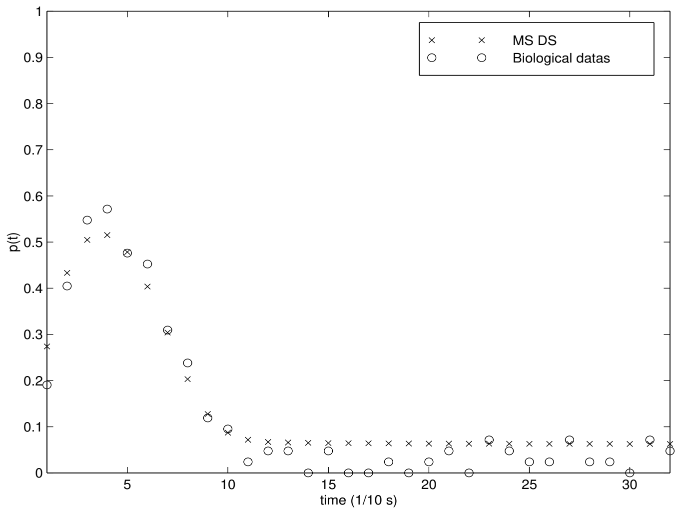

Figure 4.8.: New fitting to the biological data

Also when applying a long spike train like before to the "mistuned" MS synapse, the model now shows stable behavior[2], while the original stochastic synapse locks to oscillations with an even shorter period of about 16 seconds, shown in Figure 4.9. This might be due to the fact that the parameters have been tuned for the answer of the modified synapse and also been taken for the stochastic synapse.

[2]Note that this has been done again with an interspike interval of 100ms and other results *will* occur when e.g. shortening this interval.

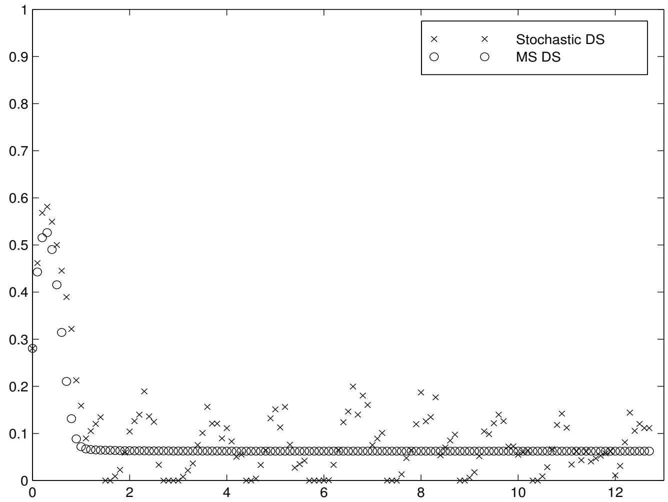

Figure 4.9.: Retrained model's answers to a long spike train

As a conclusion one can see that the selection of the "right" criteria is an important factor to *any* tuning process, maybe especially when looking at stability in the time domain.

4.1.4. Irregular spike train

The second biological datas that have been used during the tuning processes was an irregular spike train, again from Lynn Dobrunz from the Salk Institute. The total length is 420.8 seconds with 128 irregularly distributed action potentials in 15 different trials. Interspike intervals range from a few milli seconds to about 60 seconds. The average over the 15 trials has been used to calculate the release probability.

The preparations for the following experiments showed a few problems with the data:

- First, the low amount of trials lead to rather high steps in the release probability distribution. Also the reliability of the data is not satisfying.
- Second, the fifteen trials had small jitters in the arrival times of the spikes. So an epsilon surrounding of a few milli seconds has been defined to fit the varying points in time into similar time slots, assuming that the resulting error can be neglected.

Figure 4.10 shows the "uniformed" train and Figure 4.11 the corresponding release probabilities. Note that the release probabilities are fairly low. The maximum value at the y-axis is 0.3.

First I used the parameters from the previous tuning sessions to see the response to a different stimuli. The result were rather disappointing. The models – admittingly tuned

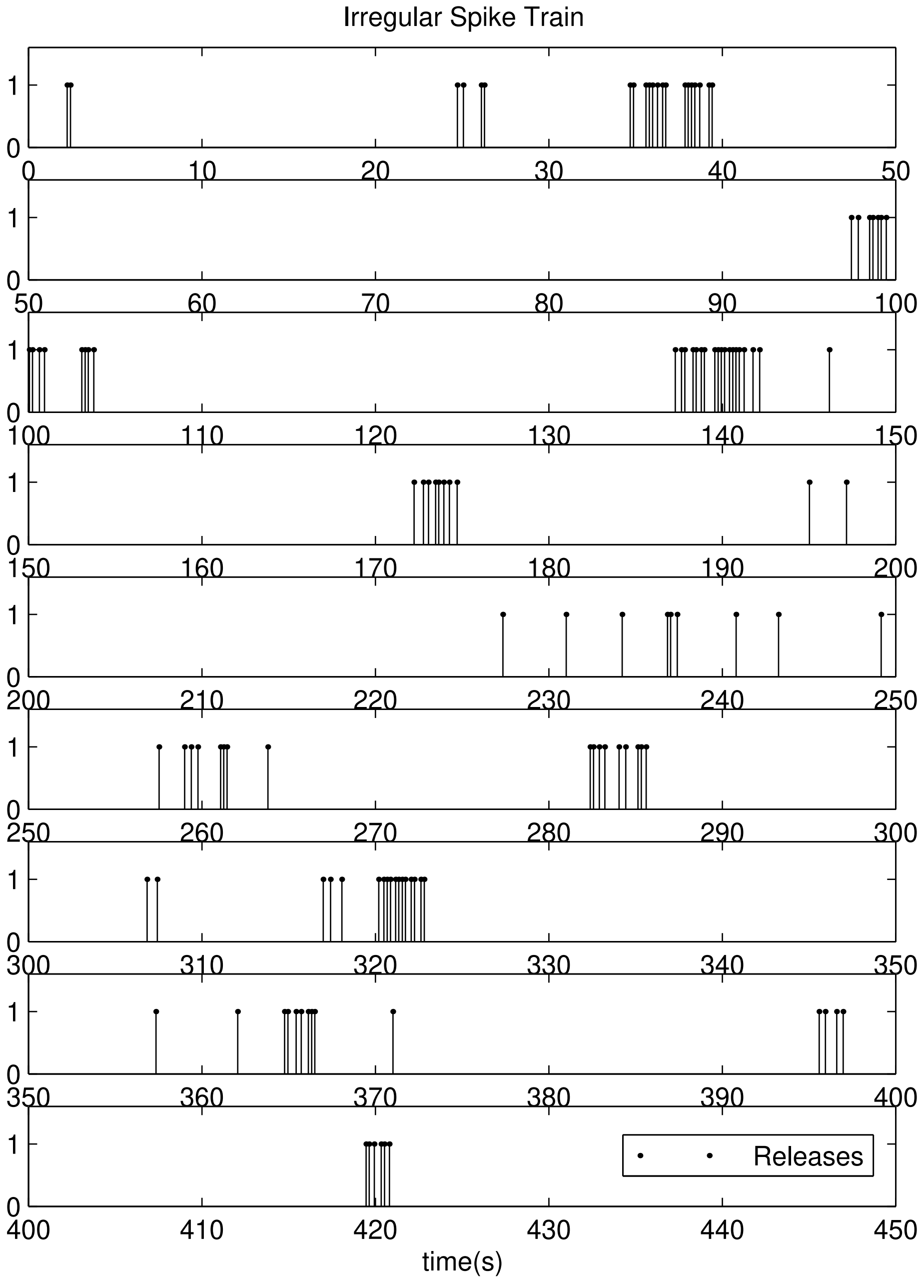

Figure 4.10.: Irregular spike train

a different biological ideal – did not show a similar response like one would eventually expect. The very low release probabilities could not be reproduced. We remember that the probability, that the first spike after a *long* break releases a vesicle, was about 0.3 and the following releases had an even higher probability.

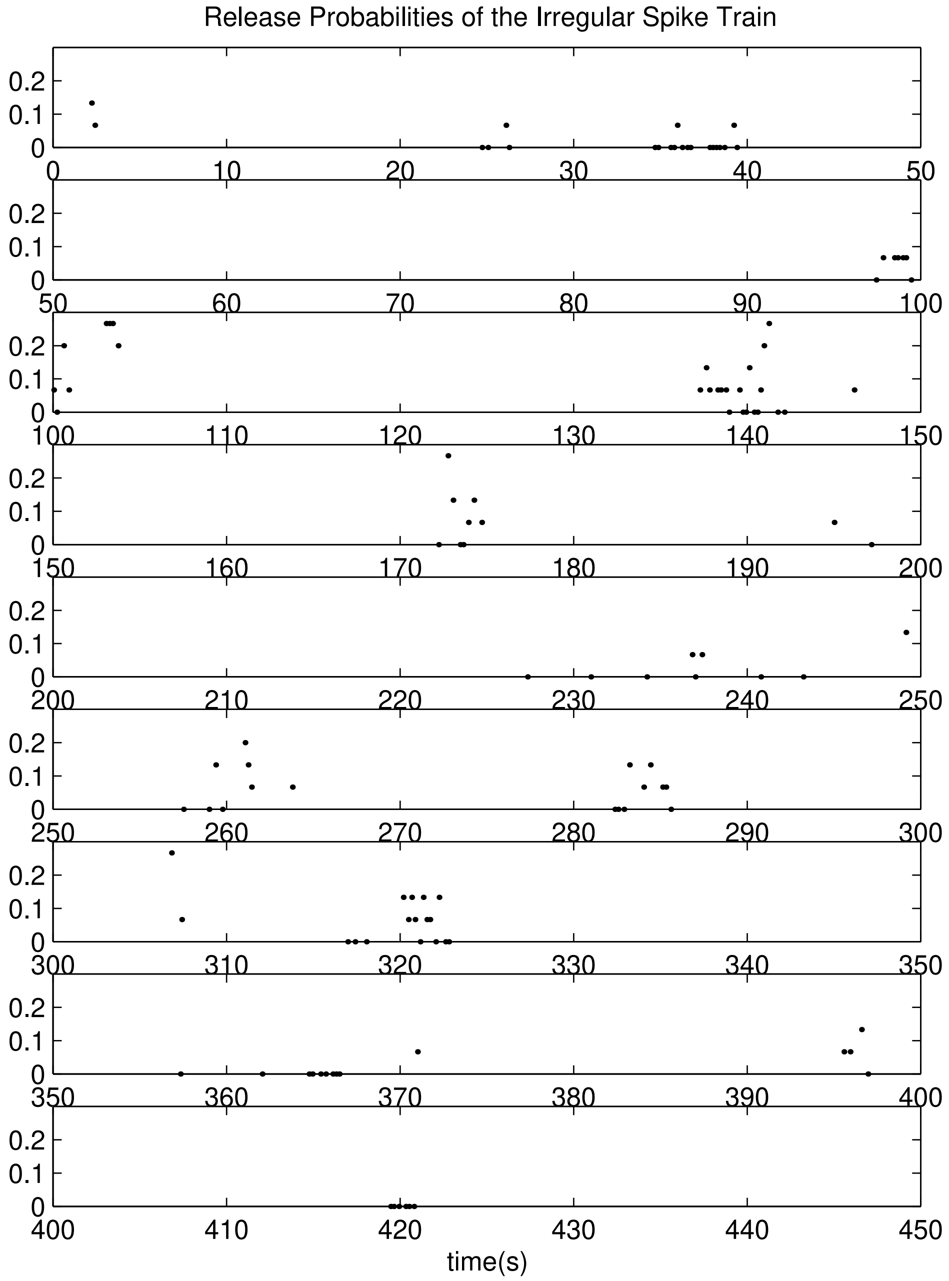

Figure 4.11.: Release probabilities of the irregular spike train

So retuning was necessary. I applied the same procedures as before. The release probabilities were the ideal and the task for simulated annealing was to minimize the summed square error. The annealing found the new parameter relative fast, although a better fit might arise for additional tuning. Figure 4.12 shows this process.

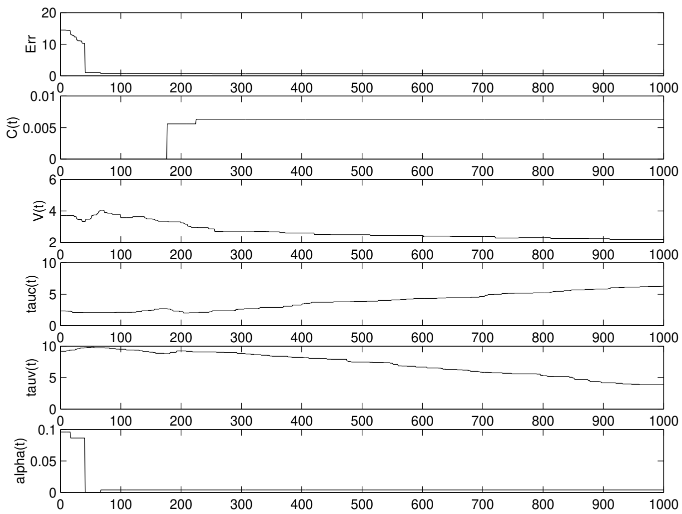

Figure 4.12.: Tuning session with the irregular spike train

The simulated annealing process was "caught in a trap". It lowered α to almost zero and found a low value for C_0 – the error minimized.

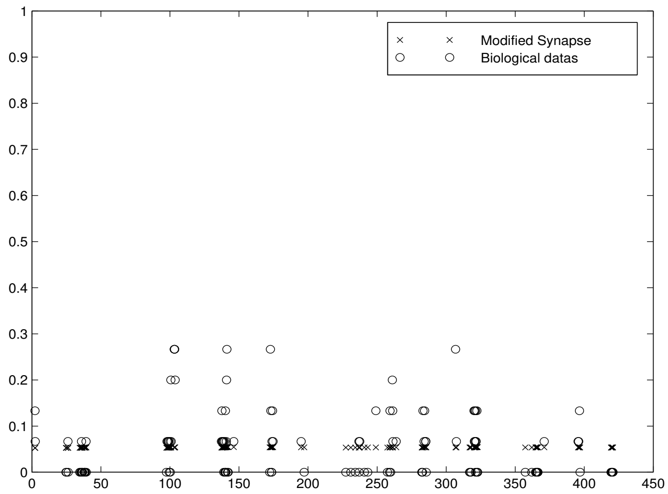

Figure 4.13.: Comparison between synapse model and the biological datas

Figure 4.13 shows the response of the modified synapse to the irregular spike train. At a first glance the solution found looks at this low temporal resolution like a straight line fitted into the data with the minimum distance square error for all targets – a linear regression. Figure 4.14 shows a zoomed-in range of the y-axis or $p(t)$.

Note the small range of different release probability of less than 0.002 during the

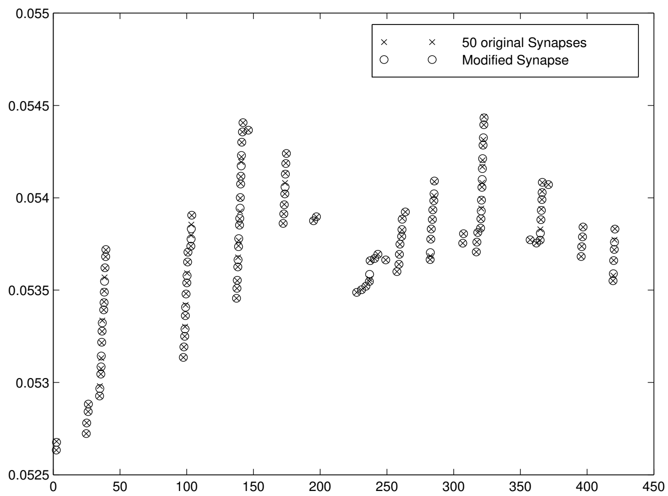

Figure 4.14.: Enlarging of the y-axis to show the development of the probability

whole simulation due to the small α.

The models could not be perfectly fit onto these biological datas, although it must be noticed that the confidence of the datas is not too high. Maybe for a smoother distribution of the release probabilities the fit would become better.

4.2. Analysis in frequency domain

In this section I will focus on the responses of the synapse models to incoming spike train of different frequencies. In the previous chapters a fixed interspike interval of 100ms or $f = \frac{1}{T} = 10$ Hertz has been used throughout the most tuning processes. The irregular spike train consisted of rapid changing and very different intervals between the spikes and was therefore a non-ideal candidate for an analysis of frequency responses.

The testing signals consisted of trains with a fixed length of 100 spikes, regardless of the total time of the simulation. They were applied to a fully rested synapse. This is shown in Figure 4.15.

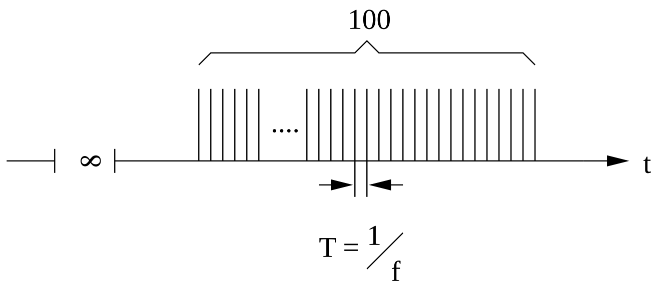

Figure 4.15.: 1^{st} stimuli for frequency analysis

The different responses to frequencies of 0.1 Hz, 0.2 Hz, 0.5 Hz, 1 Hz, 2 Hz, 5 Hz, 10 Hz, 15 Hz, 20 Hz, 50 Hz, 100 Hz and 1 kHz are shown in Figure 4.16. From definition of the models it is clear that the release probability for the first spike is the same for all trials and only depending on the resting values and dynamic parameters of the synapses.

The responses can be very well tuned by the 5 parameters of the model. Intuitively one can say that the probability increase from the first to the second possible release depends strongly on τ_c and τ_v for fixed values of the other parameters. The higher τ_c, the less is the decrease of the Ca^{2+} concentration. The less τ_v, the faster is the recovery – the *reproduction* – of vesicles. Hence, the higher the increase of the release probability.

For constant parameters, if the pool of readily releasable vesicles will not be depleted within the first spike[3], the rise of the slope will be higher for shorter interspike intervals. This phenomenon can be watched in Figure 4.16.

So the response of the synapses, or in other words the probability that a package filled with neurotransmitter is being released into the cleft and a PSP emerges, is the higher the shorter the interval between two arriving spikes is. This behavior fits well into several theories about neural computation, where the strength of a stimuli is coded in the frequency (or the *rate*). In the stochastic synapse and the modified version of it, the probability that a synapse forwards this information depends on it's strength – the frequency of the input signal. But due to the faster depletion of the vesicle pool $V(t)$

[3]this has never been recorded in biological experiments so far

for higher frequencies the maximum achievable probability is limited – in this case to about 0.55.

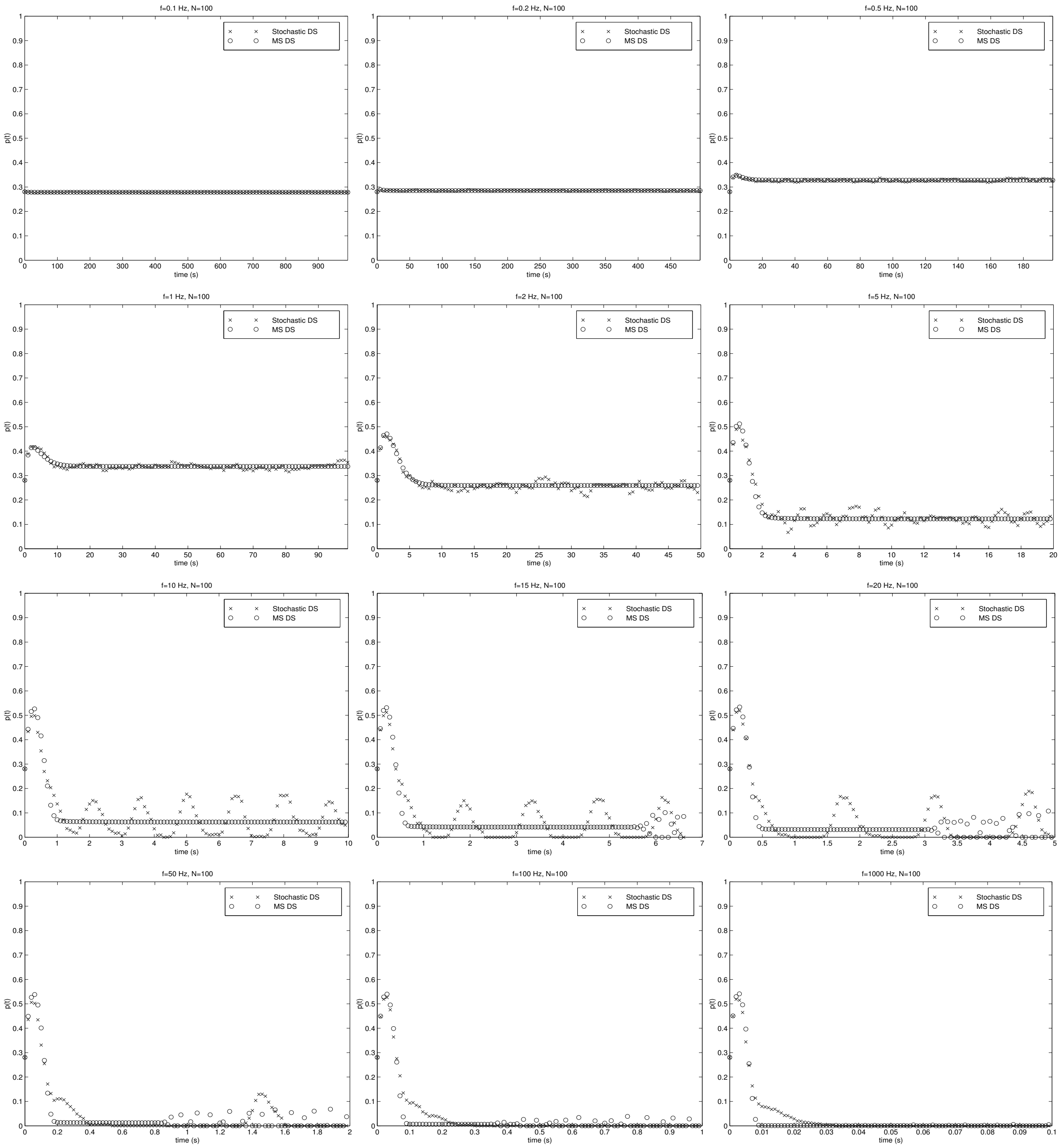

Figure 4.16.: Synaptic responses for different frequencies

How long a synapse can respond to strong inputs at the presynaptic terminal depends on the size of the vesicles pool C_0 and of course on the recovery time τ_v. The depletion can be observed very well in the responses for higher frequencies. For higher frequencies, when the reproduction of vesicles is neglectably low between incoming spikes, the pool size can be estimated by adding up the release probabilities. Again, oscillations and irregularities for the probability can be observed in Figure 4.16 when the pool is depleted and the Ca^{2+} concentration becomes high.

Also a remarkable difference between the two models becomes obvious: The decrease of the release probability is different when it develops towards zero. While the modified synapse shows a steady course to the x-axis, the original model exhibits a slower depletion. This is due to the individualities of the N synapses, where the approximation with the modified "super synapse" becomes deficient. The modified synapse approaches the average over N synapses' parameters when *all* pools of vesicles are well above total depletion. Or in other words, when the nonlinear function $max()$ gets into account the linear approximation falsifies.

To further examine the total activity of a synapse to an incoming spike train a second experiment took place. This time the testing signals consisted of trains with a fixed length of one second, regardless of how many spikes fit into this period. They were applied to a fully rested synapse. This is shown in Figure 4.17.

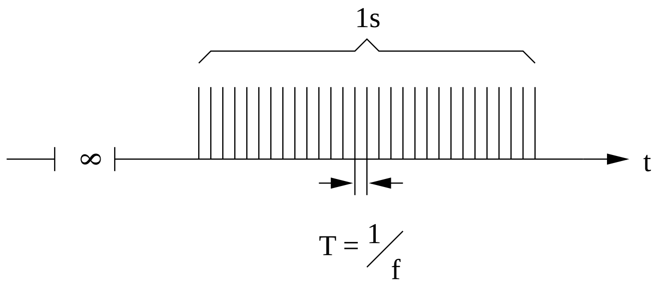

Figure 4.17.: 2^{nd} stimuli for frequency analysis

During the experiment stimuli from 1 Hz to 1 kHz were applied to the synapse and the responses recorded. It proved what was obvious. For Figure 4.18 the release probabilities over f arriving action potentials have been summed up and plotted over the corresponding frequency.

Nearly regardless of the frequency, the summed probability over 1 second remains constant at approximately V_0 plus the amount that can be reproduced in a second. The notch at low frequencies is determined by the low amount of spikes in this second of spike arrivals. Hence, the pool of readily releasable vesicles cannot be fully depleted and the $C(t)$ did not increase too much.

But higher frequencies shift this area of probability in time in direction of the onset of the stimuli. This implicates that at lower frequencies the probability that a release occurs is distributed evenly over time (of course this is limited to times $t_i \in S$) whereas for higher frequencies only the first spikes – with a higher probability – lead to a release and the rest are nearly completely suppressed.

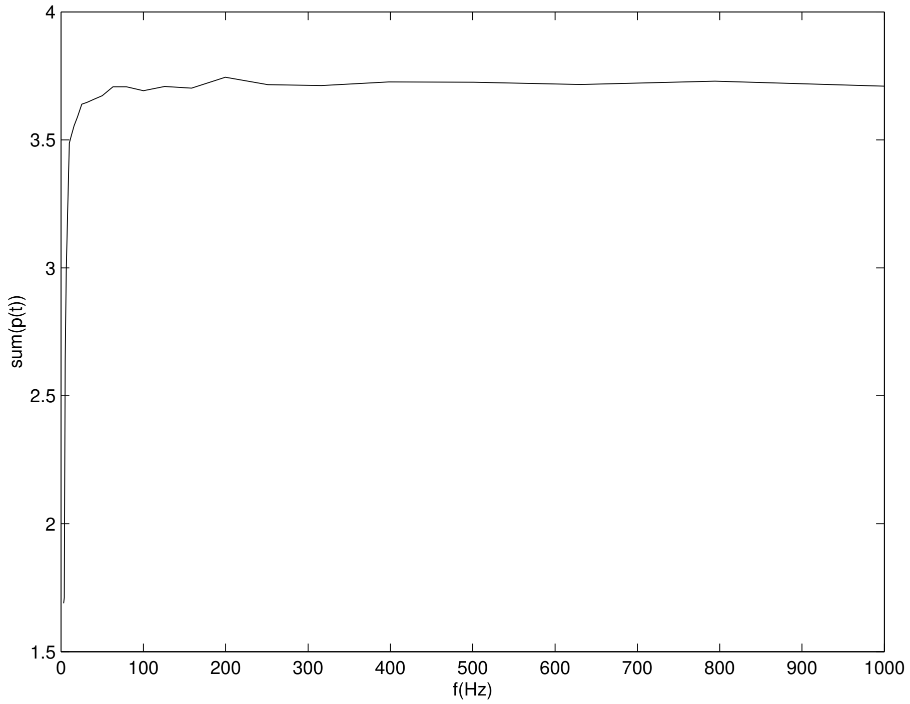

Figure 4.18.: Total release probabilities within one second over frequency

4.3. Varying the pool size N

The release of vesicles in the original stochastic synapse is a stochastic process. Hence, to approximate a pool of N synapses with the least possible error with a single element, probability theory suggests $N \to \infty$. Even in human brain the number of synapses are limited. More than that, experiments from neurobiology revealed in the order of 10 connections from one neuron to the other.

Work and experiments with these two types of synapses and especially comparisons between them showed that the really limiting factor for approximation is *not* the pool size. Even during simulations with $N = 5$ or less to speed up the calculations, the *basic* behaviors remained the same. For datas on this, look at the preceeding experiments, where the number N is stated.

Differences arose when the pool of vesicles became depleted for one or more synapses. That sometimes happened even during the first spikes.

5. Implementation

As mentioned in some chapters before, this modified synapse model is *quite* simple to implement in analog electronics. In the whole process of realization one of the main perspectives was to watch for the further step of integrating the structures on a silicon wafer in analog very large scale integration (aVLSI). All simulations in this chapter have been done with ...

> ... log, a schematic editor, analog and digital simulator, and netlist generator. Log was written by Dave Gillespie; the analog simulation capability was added by John Lazzaro. Log is maintained under Unix by Dave Gillespie and John Lazzaro. Log is distributed under the GNU General Public License.
>
> *from the README file of the v5.40 distribution*

5.1. Basic considerations

Therefore a few side considerations had always to be kept in mind:

- contrary to analog electronics, it is hardly possible to manufacture ohmic resistors with high values in aVLSI when trying to keep the size of a structure small at the same time, because resistors can only be made by building long lines with a "resistive material". A solution would be to use metal oxide semiconductor transistors as a replacement.
- one also has to use area to achieve "big" capacitances since C is proportional to A and due to a given technique of fabrication the material – which could also increase the value – is fixed. Realistic values for *many* capacitors on the same chip are in the range of femto farad (10^{-15}).
- on the other hand side one might need a wide variability of time constants to realize τ_c and τ_v. The time constant for resistive-capacitive network for charging and decharging a capacitor C over a resistor R is defined by

$$\tau = R \cdot C \tag{5.1}$$

This implicates that if R is small one has to use a big capacitance C (or the other way around) to achieve a time constant τ in the range of milli seconds to seconds.

- a special feature of field effect transistors can be exploited to simplify calculation and reduce chip size: subthreshold behavior. In brief this means to work in the nano ampere region, where the transistor shows a behavior where the drain to source current is proportional to exponential of the gate source voltage. This will be explained in more detail in section 5.1.1. [1]

- although material sciences develop new materials and techniques, one of the main problems still is the *analog long-term storage* of parameters for neural computation in analog electronics. See [Elias et al., 1996], [Elias and Northmore, 1995], [Harrison et al., 1998], [Diorio et al., 1995] and [Tran et al., 1996] for details and possible solutions.

In the following sections the very basics of analog electronics will be skipped. For introductions into the techniques of modeling neural networks in analog electronics see [Mead, 1989], [Douglas and Mahowald, 1995] and [Mahowald and Douglas, 1991].

5.1.1. Subthreshold behavior

The types of transistors used in the following sections are so called metal oxide semiconductor field effect transistors – or just MOSFETs. Their schematic symbols are drawn in Figure 5.1. For details about the difference between n- and p-type MOSFETs see [Beuth, 88].

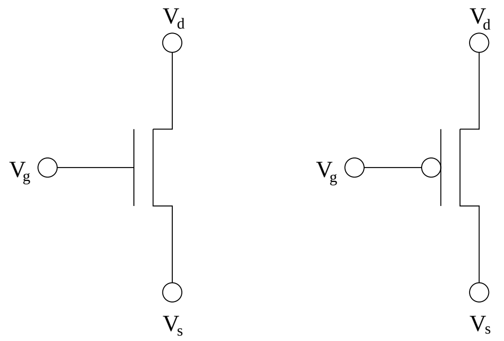

Figure 5.1.: Schematics of n- and p-type MOSFETs

The source-to-drain current for a p-type MOSFET in weak inversion (subthreshold) can be written as

$$I = -q \cdot D \cdot W \cdot \frac{\partial N}{\partial y} = -q \cdot D \cdot W \cdot \frac{N_d - N_s}{L} \tag{5.2}$$

[1] due to the very small currents this can only be used with aVLSI. Otherwise the influences of electro magnetic radiation would falsify the calculations.

where

- q is the signed charge
- D diffusion coefficient
- W is the channel width
- L is the channel length
- N carrier density in the channel

The source and drain carrier densities from the Equation 5.2 are

$$N_s = N_0 \cdot e^{-\frac{\phi_0}{k \cdot T}} \cdot e^{-\frac{q \cdot \psi_s}{k \cdot T}} \cdot e^{\frac{q \cdot V_s}{k \cdot T}} \tag{5.3}$$
$$N_d = N_0 \cdot e^{-\frac{\phi_0}{k \cdot T}} \cdot e^{-\frac{q \cdot \psi_s}{k \cdot T}} \cdot e^{\frac{q \cdot V_d}{k \cdot T}} \tag{5.4}$$

N_0 ... effective number of states per unit area in the channel
ϕ_0 ... the built-in barrier between source and channel
ψ_s ... the surface potential
V_s ... is the source voltage without consideration of the bulk
V_d ... is the drain voltage without consideration of the bulk

With these extensions Equation 5.2 expands to

$$I = q \cdot D \cdot N_0 \cdot \frac{W}{L} \cdot e^{-\frac{\phi_0}{k \cdot T}} \cdot e^{-\frac{q \cdot \psi_s}{k \cdot T}} \left[e^{\frac{q \cdot V_s}{k \cdot T}} - e^{\frac{q \cdot V_d}{k \cdot T}} \right] \tag{5.5}$$

With a linearization for small values around the operating point the surface potential

$$\psi_s = \psi_0 + \kappa V_g \tag{5.6}$$

and κ is the effectiveness of the gate voltage in changing the surface potential[2] ψ_s.

$$\kappa \equiv \frac{\partial \psi_s}{\partial V_g} \tag{5.7}$$
$$\kappa^{-1} = \frac{\partial V_g}{\partial \psi_s} = 1 + \frac{C_{dep}}{C_{ox}} \tag{5.8}$$
$$\Rightarrow \kappa = \frac{C_{ox}}{C_{ox} + C_{dep}} < 1 \tag{5.9}$$

- $\kappa \rightarrow 1$ for light substrate doping
- $\kappa \propto |V_g|$

[2] κ is also called the "backgate coefficient"

Hence, with Equation 5.6 the complete source-to-drain current can be rewritten as

$$I = \underbrace{q \cdot D \cdot N_0 \cdot \frac{W}{L} \cdot e^{-\frac{\phi_0}{k \cdot T}} \cdot e^{-\frac{q \cdot \psi_0}{k \cdot T}}}_{I_0} \cdot e^{-\frac{q \cdot \kappa \cdot V_g}{k \cdot T}} \left[e^{\frac{q \cdot V_s}{k \cdot T}} - e^{\frac{q \cdot V_d}{k \cdot T}} \right] \tag{5.10}$$

$$I = I_0 \cdot e^{-\frac{q \cdot \kappa \cdot V_g}{k \cdot T}} \left[e^{\frac{q \cdot V_s}{k \cdot T}} - e^{\frac{q \cdot V_d}{k \cdot T}} \right] \tag{5.11}$$

When we substitute the unit "Voltage" with $\frac{k \cdot T}{q}$ with $q > 0$ the source-to-drain current can be written for the n- and p-type MOSFET simplified as

$$p - type: \; I = I_0 \cdot e^{-\kappa V_g} \cdot (e^{V_s} - e^{V_d}) = I_0 \cdot e^{V_s - \kappa V g} \cdot (1 - e^{V_{ds}}) \tag{5.12}$$

$$n - type: \; I = I_0 \cdot e^{\kappa V_g} \cdot (e^{-V_s} - e^{-V_d}) = I_0 \cdot e^{\kappa V g - V_s} \cdot (1 - e^{-V_{ds}}) \tag{5.13}$$

These results remain valid for small values of $V_d - V_s$ of about $4 \cdot \frac{k \cdot T}{|q|} \approx 100mV$ up to a $V_{gs} < V_T \approx 1V$. When looking at Equation 5.13 one can see the exponential dependence between I_{ds} and V_{gs}. Exploiting this behavior helps to achieve the calculation of an exponential function with only one transistor. Compared to digital computation this reduces both, the needed chip area and the power consumption.

5.1.2. Above threshold behavior

The "normal" and most common way a MOSFET is used is in the strong inversion regime (above threshold), where larger amounts of charge flow through the channel due to the *drift* of charge carriers.

The charge in the inversion layer of a n-type MOSFET

$$Q_n(y) = -\left(V_g - V(y) - 2 \cdot \psi_B\right) \cdot C_{ox} + \sqrt{2 \cdot \varepsilon_0 \cdot \varepsilon_q \cdot N_A \cdot (V(y) + 2\psi_B)} \tag{5.14}$$

Hence, the source-to-drain current

$$I = W \cdot \mu_n \cdot |Q_n(y)| \cdot \frac{\partial V}{\partial y} \tag{5.15}$$

$$= \frac{W}{L} \cdot \mu_n \cdot C_{ox} \cdot \left[\left(V_g - 2\psi_B - \frac{V_d}{2} \right) \cdot V_d - \frac{2}{3} \cdot \frac{\sqrt{2\varepsilon_0 \varepsilon q N_A}}{C_{ox}} \cdot \left((V_d + 2\psi_B)^{\frac{3}{2}} - (2\psi_B)^{\frac{3}{2}} \right) \right] \tag{5.16}$$

This equation can be splitted into two regions:

- small V_d: linear region, for $V_d \ll V_g - V_T$

$$\begin{aligned} I &= \frac{W}{L} \cdot \mu_n \cdot C_{ox} \cdot \left[(V_g - V_T)V_d - \left(\frac{1}{2} + \frac{\sqrt{\varepsilon_0 \varepsilon q N_A}}{4C_{ox}\sqrt{\psi_B}}\right) \cdot V_d^2\right] \\ &= \frac{W}{L} \cdot \mu_n \cdot C_{ox} \cdot (V_g - V_T)V_d \end{aligned} \tag{5.17}$$

where the threshold voltage $V_T = 2\psi_B + \frac{\sqrt{2\varepsilon_0\varepsilon q N_A 2\psi_B}}{C_{ox}}$

- large V_d: saturation region

$$I = m \cdot \frac{W}{L} \cdot \mu_n \cdot C_{ox} \cdot (V_g - V_T)^2 \tag{5.18}$$

where $m \approx \frac{1}{2}$ for low doping

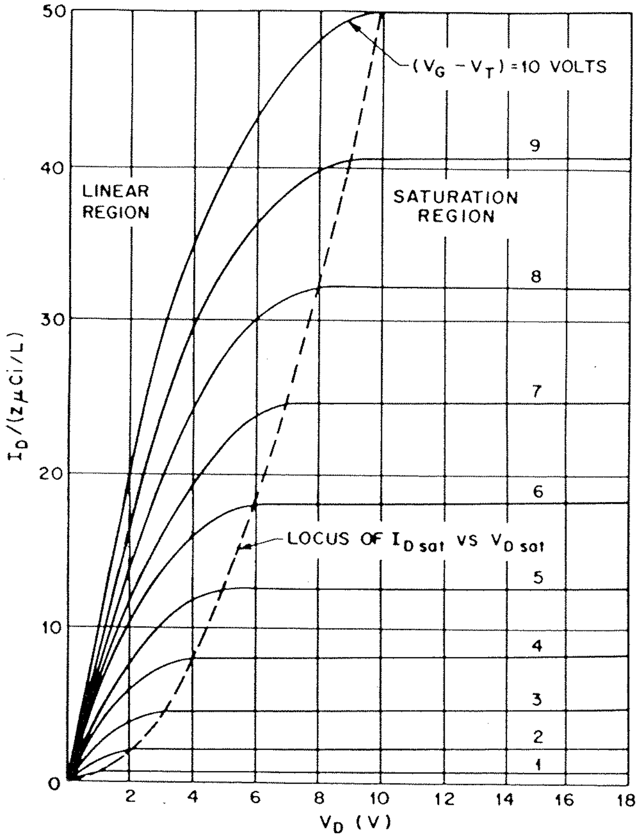

Figure 5.2.: Idealized drain characteristics of a MOSFET

Figure 5.2 illustrates Equations 5.17 and 5.18. The dashed line indicates the locus of the saturation drain voltage. As an approximation and with neglecting the *early effect* the drain current remains constant for $V_d > V_{d,sat}$. In this saturation region – like in Equation 5.18 – the current depends on $(V_g - V_T)^2$ (indicated by the values 1 to 10) and no more on V_d.

5.1.3. MOSFETs as switches

MOSFETs can effectively be used to realize switches. Therefore they are operated either in full saturation (On) or switched off (Off). A n-type MOSFET is most commonly switched on by setting V_{ds} and V_{gs} to 5 Volts and switched off by setting V_{gs} to zero Volts. When considering a circuit with only +5 Volt power supply, the p-Type MOSFET is switched on by V_{sd} =5V and connecting the gate to ground. It is in the Off-state when V_{gs} is also set to 5 Volts.

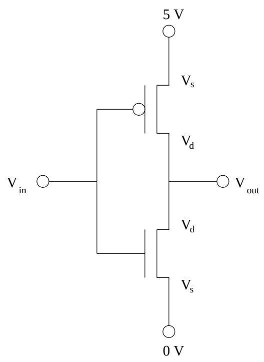

Figure 5.3.: Schematics of a CMOS Inverter

A complementary MOS (CMOS) inverter is shown in Figure 5.3. This simple structure illustrates MOSFETs as switches. V_{out} is the inverted incoming digital signal V_{in}. When reminding the above conditions to switch n- and p-type MOSFETs it becomes obvious that (nearly) no current flows from +5V to ground for steady states of V_{in}. Only one transistor is active at the same time. Idealized: Only during the short transitions of V_{in} (e.g. 2.5 V), when both transistors are partially enabled and for charging and decharging the parasitary output capacitor, power is being consumed.

5.2. Modeling the facilitation

For the implementation of the facilitation of the modified stochastic synapse the previously introduced properties of MOSFETs are used.

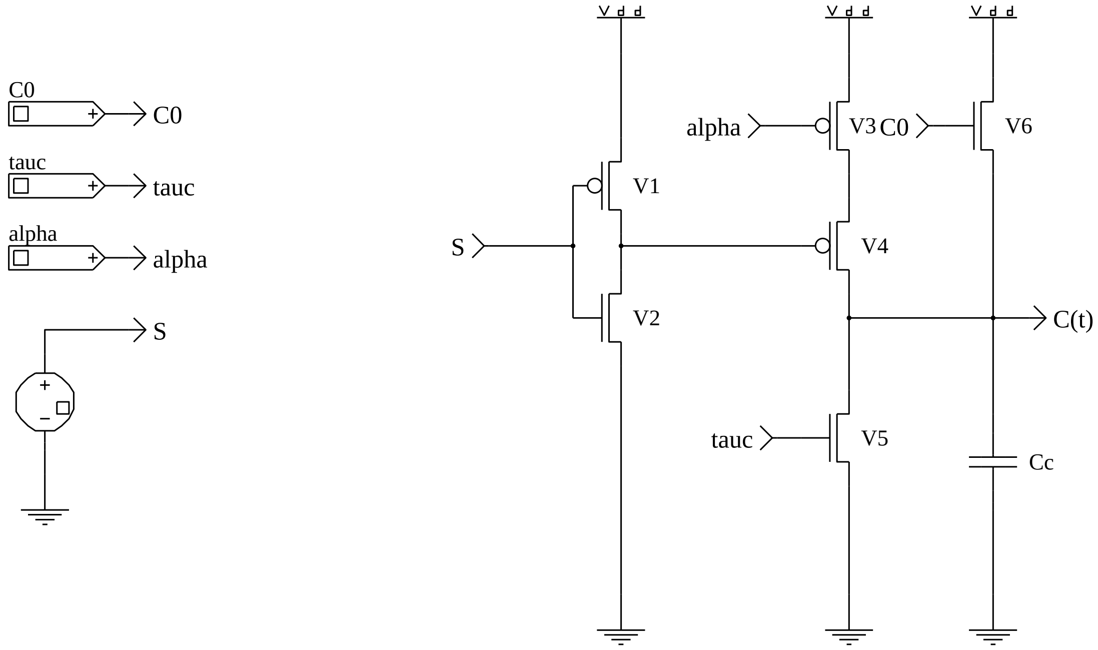

Figure 5.4.: Complete implementation for $C(t)$

Figure 5.4 displays the complete realization of Equation 3.2 in analog electronics. The power supply V_{dd} is +5 Volts. S is similar to the defined spike train from Section 2.2. It is impossible to realize ideal Dirac impulses in analog electronic. Therefore S consists of short +5 Volts voltage pulses. In the simulations for Figure 5.5 pulses of 100 micro seconds in length have been used, although this length could be reduced.

The transistors V_1 and V_2 implement the inverter from Section 5.1.3. It is needed to drive V_4, which operates as a switch, with the right voltages. V_4 should be "On" during the length of the incoming pulses at times $t_i \in S$. Due to it's p-type characteristics we need a resting potential of +5 Volts and pulses to the ground level to switch in on.

The capacitor C_c stores the value of the Ca^{2+} concentration. The MOSFETs V_3, V_5 and V_6 implement the facilitation parameters C_0, τ_c and α. They work as "voltage dependent resistors". The voltage at the gate determines the current, respectively the resistance between source and drain.

V_6 and V_5 work as a voltage divider to generate the resting potential C_0. V_3 determines the current i_3 with which charge is dumped on the capacitor C_c. At the capacitor the charge $Q_c = C_c \cdot V$. In this case V equals $C(t)$.

Also $Q = \int_t i(t)dt$. When V_4 is switched on and the current i_3 flows for a time Δt

onto capacitor C_c the charge $\Delta Q_c = \int_{\Delta t} i_3(t) \cdot dt$ is being added to the existing charge Q_c. C is a constant which is determined at the moment of fabrication.

Hence, when $C(t) = \frac{Q_c}{C_c}$ and the charge ΔQ_c which is determined by α is added to Q_c, the voltage increases with

$$C(t^+) = \frac{Q_c + \Delta Q_c}{C_c} = C(t) + \underbrace{\frac{\Delta Q_c}{C_c}}_{\alpha} = C(t) + \alpha \tag{5.19}$$

The decay of $C(t)$ should be similar to Equation 2.3 but not necessarily exponential decaying. This is implemented with V_5 which works like V_3 but without a switch, so it is active all time. If $C(t)$ is above the resting potential determined by the voltage divider V_6, V_5 the additional charge will flow from the capacitor C_c through V_5 to ground. Simulation results in Figure 5.5 illustrate that this decay is not exponential but much more linear.

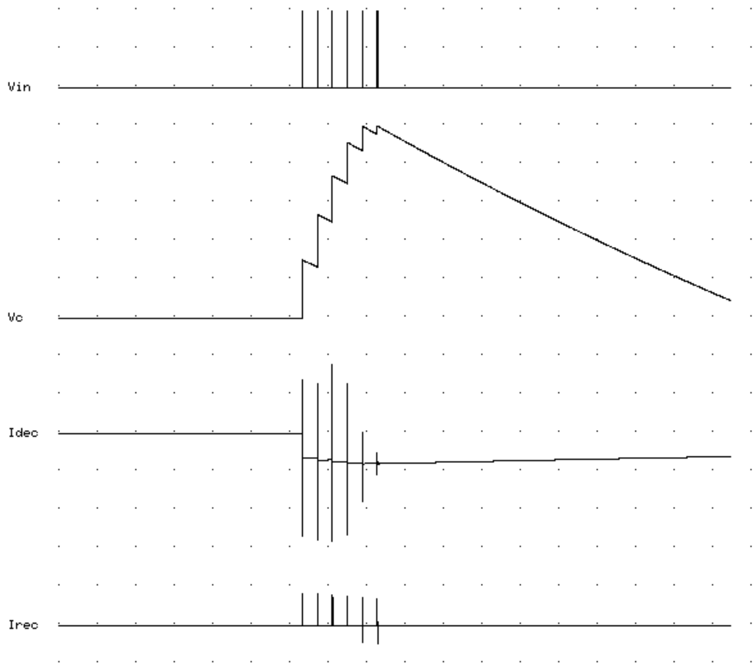

Figure 5.5.: Modeling the Ca^{2+} concentration in analog electronics

V_{in} displays the incoming spike train. I_{dec} and I_{rec} label the currents for decrementing and incrementing the charge respectively the voltage at the capacitor. Another limitation can be observed: although α is constant during the whole simulations, the incremental steps of V_c are different from one spike to the other. The reason is that the current that flows onto a capacitor is also depending on the voltage difference between the actual load of capacitor and the voltage that is being applied to it.

5.3. Modeling the depletion

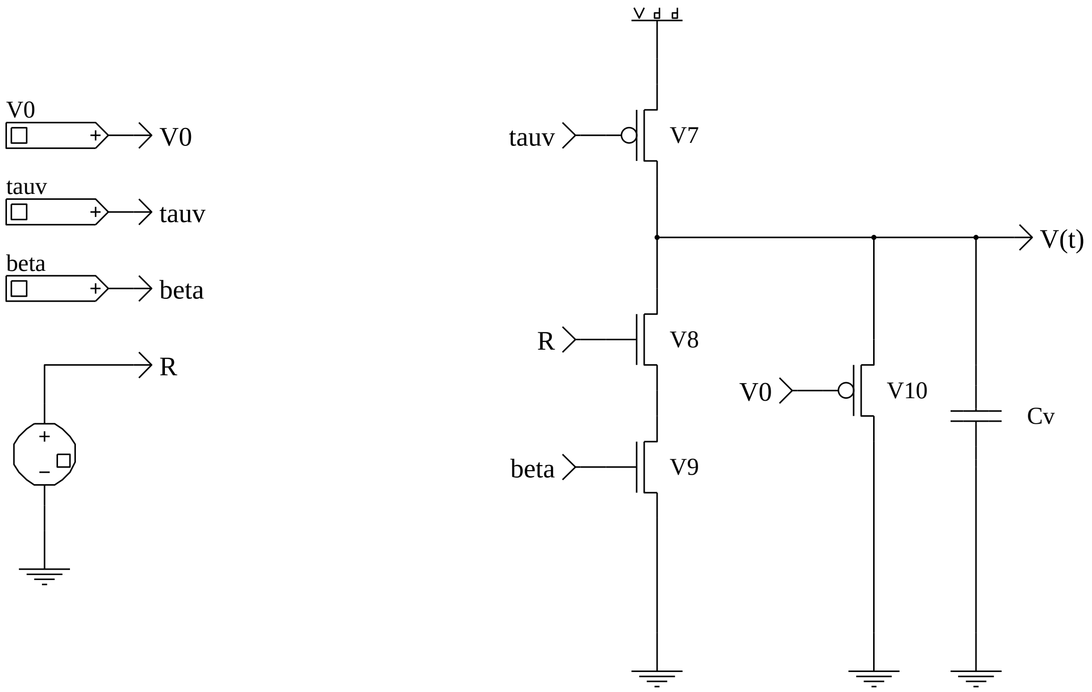

Figure 5.6.: Complete implementation for $V(t)$

The final schematics of the depletion are shown in Figure 5.6. The capacitor C_v stores the value of the amount of readily releasable vesicles. The transistors V_7 and V_{10} implement the depletion parameters V_0 and τ_v.

β models a parameter to determine the decrement of $V(t)$. It can be set to a constant value to implement a decrement of "1". In this case it models Equation 2.4 from the original stochastic synapse from Section 2.2. By keeping $\beta \propto p(t)$ one obtains the development of vesicles of the modified stochastic synapse.

The schematics for modeling the depletion are somehow "inverted" to the $C(t)$ schematics, but the same principles apply. All n-type MOSFETs have been replaced by p-types and the other way around. To keep the functionality also the power supply and ground had to be exchanged. Only the connections of the capacitor remained the same.

V_7 and V_{10} also work as voltage dividers. In this case to generate V_0. When V_8 is switched on, V_9 determines the current i_9 that decrements for a time Δt the charge on capacitor C_v. With the same principles like in Equation 5.19, this time the drain off of charge reduces the voltage.

The speed of regeneration τ_v of missing vesicles after a release is implemented with V_7. If $V(t)$ is beyond the resting amount of readily releasable vesicles V_0 the missing charge will be "refilled" through V_7. This rise of the voltage is also not exponential.

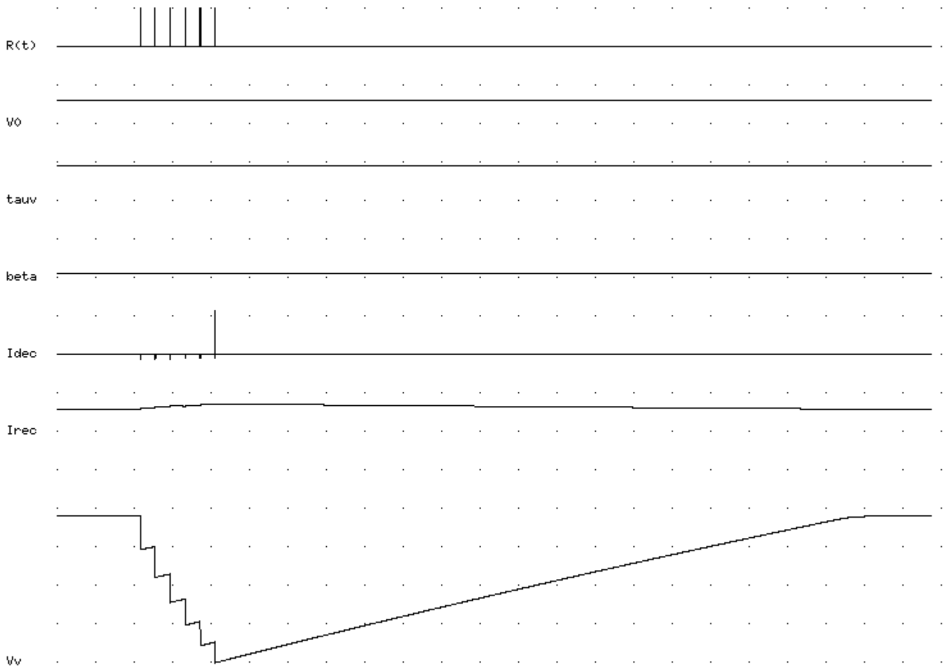

Figure 5.7.: Simulations of the amount of readily releasable vesicles

5.4. Multiplication

To calculate the release probability we need the product $C(t) \cdot V(t)$. Let us look back at Equation 5.17 from the linear range of the above threshold behavior of a MOSFET:

$$I = \frac{W}{L} \cdot \mu_n \cdot C_{ox} \cdot (V_g - V_T) V_d$$

One can see the multiplicative dependence between V_g and V_d. Two constraints are given to use this functionality:

- $V_d \ll V_g - V_T$
- $V_{gs} > V_T \approx 1V$

The disadvantage is the small range for the two variables to multiply. As an approximation this one transistor multiplier can be used though. If one sets $V_{gs} = V(t)$ the 2^{nd} requirement can be partially fulfilled, since $V(t)$ is at least greater than one in most cases. By rescaling the finally calculated probability one can limit the $C(t)$ at this stage, thereby fulfilling the 1^{st} requirement.

For the results of the simulation illustrated in Figure 5.8 I used values of about 1.5 Volts or more for $V(t)$ and limited $C(t)$ between 0 and 0.5 Volts. To use the result in the succeeding stage one has to convert the current to a voltage with a transistor as a resistor according to Ohm's law $V = I \cdot R$.

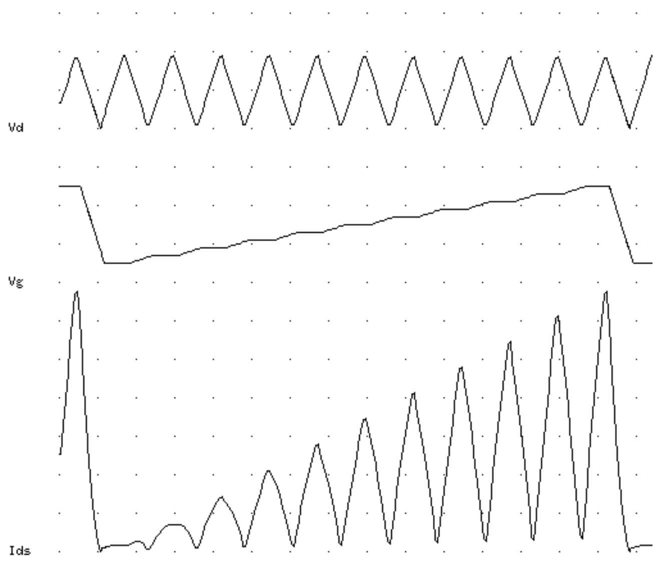

Figure 5.8.: Analog multiplication

5.5. Calculating the release probability

To evaluate the complex Equation 2.6 or 3.4 one can use the physics of computation of a single transistor. By keeping the operating point of this MOSFET in the subthreshold region we can use the functionalities from Section 5.1.1. We remember the source-to-drain current for a n-type MOSFET in subthreshold:

$$I_{ds} = I_0 \cdot e^{\kappa V_g} \cdot \left(e^{-V_s} - e^{-V_d}\right)$$

When one sets V_s to ground, uses V_g to set an operation point and let V_d be equal to $C(t) \cdot V(t)$ we obtain

$$I_{ds} = \underbrace{I_0 \cdot e^{\kappa V_g}}_{I_{p0}} \cdot \left(e^0 - e^{-V_d}\right) \quad (5.20)$$

$$I_{ds} = I_{p0} \cdot \left(1 - e^{-V_d}\right) \quad (5.21)$$

where I_{p0} is a scaling factor and the result I_{ds} is the wanted release probability $p(t)$. By neglecting I_{p0} or setting it to 1 we finally obtain:

$$p(t) = 1 - e^{-C(t) \cdot V(t)}$$

which is equal to Equations 2.6 and 3.4. To stay in subthreshold region we must limit $C(t) \cdot V(t)$ to about 0.5 Volts. A current as output is ideal for easy summing up at a succeeding soma.

5.6. Discussion of the implementation

The example of the preceding implementation of rather complex mathematical functions with a relative small amount of transistors and two capacitors shows the power of calculation with electronics in the analog domain. Not only would the needed chip area be much higher when trying to do the same in digital computation, also the power consumption would rise enormously. Although the achieved accuracy is constant and higher in digital computation, it is *not necessary* in many of the nowadays application, especially when looking at real world sensor data processing e.g. in neuromorphic systems.

One of the constraints from the above implementation that could have a negative consequence for the computation is the limitation of the maximum values of some variables due to the power supply of $V_{dd} = 5V$ or less. For example $V(t)$ can not get above V_{dd}. By scaling the parameters to ranges where the power supply does not get into account and also the physics of computation can be exploited, one can avoid this constraint.

In this design a few problems could not be solved to full satisfaction:

- the different stages are not fully decoupled. This means that e.g. the the current that is needed as input for the multiplication stage must be supplied by the preceding stages (the modeling of $C(t)$ and $V(t)$). A current drawn for a certain time from a capacitor leads to a drop of the voltage and therefore to a falsification of the result. This could be solved by inserting a so called "voltage follower".
- referring to Section 2.2 $p(t)$ is only defined at certain time points $t_i \in S$. Otherwise $p(t)$ is zero. Therefore a pulse with a certain duration has to be generated, e.g. with a simplified non-retriggerable version of a mono-stable multi-vibrator.
- the time course of computation should be watched. Only *after* a probability pulse has been transmitted to the next stage, $V(t)$ may be decremented.

6. Conclusions and Outlooks

In this diploma thesis I worked with new types of synapses called dynamic synapses. After introducing the original stochastic synapse a slight modification lead to a synapse that is able to approximate a pool of N stochastic synapses.

- The experiments showed that it is able to fit several smooth given functions through training of it's five parameters. One of the given function was a realistic course of release probabilities from biological recorded datas from Lynn Dobrunz from the Salk Institute.
- I showed that the models' response $p(t)$ is limited and it can not implement arbitrary functions, especially rapid changes of the release probability like pulse functions. On the other hand this is biologically plausible since no biological synapse has been found that is capable of this.
- Simulations with the models in time domain illustrated that irregularities in the release probability due to a high Ca^{2+} concentration lead to rather stochastic behavior. This behavior also raises the question whether $C(t)$ should be limited. Also in biology there *is* be a mechanism (the reversal potential) to protect the synapse from overcharges for obvious reasons.
- The modified stochastic synapse showed good approximations to pools of synapses. Using one modified stochastic synapse instead of a pool of N original stochastic synapses, would have the advantage that simulations where information is represented in space coding could be shortened by a factor of about $\frac{1}{N}$. Constraints showed up when individual synapses became depleted during simulations and the non-linear function $max()$ that limited $V(t)$ to become negative got into account. This topic could be discussed in more detail in future work.
- Training takes a lot of time because of their complex error functions for both, the original and the modified stochastic synapse. Although both models are derivable, learning rules like back propagation are no good choice because of the time consuming methodology. So a process called *simulated annealing* has been used to achieve the optimization of the parameters. Simulated annealing lead to satisfying results in a relative short time. Still this method might get caught in local minimas of the error function like illustrated in section 4.1.4.

- If an approximation for a pool of N synapses is too deviating the modified stochastic synapse still leads to good results and can be used for *pre-training* of the original model to save time.

- In chapter 5 a possible implementation of the modified stochastic synapse in analog electronic or even aVLSI is presented. Simulations show a similar behavior like the mathematical ideal. Limitations of $C(t)$ and $V(t)$ due to the voltage supply V_{dd} are not necessarily a disadvantage like explained above. The implementation leaves possibilities for scaling the parameters in several ranges to overcome such constraints.

In future work – maybe in cooperation with other universities with experiences and possibilities in fabrication of aVLSI – a chip could be built and applied in real world studies.

A. Simulated Annealing

Simulated annealing is a procedure derived from the physical process of solidification. It is very useful for tuning models depending on multiple parameters, that should be optimized according to a special criteria. The difference to other common procedures is that is unknown for which parameter the model should be tuned. How a “better” result can be achieved is not known, so different settings of parameters are just tried out. One needs at least a *criteria* on which the success can be measured. An expressive but very individually different example to explain *simulated annealing* is:

> One buys an artwork – e.g. a sculpture – and thinks about how to place it in the living room. First you don’t care about a certain place or heading and put it in the middle of the room. You move a step backwards, view it and decide that it doesn’t look too good. Then you move it a bit towards the window and judge again about it’s beauty. You put it away from window again but you turn it a bit to the left instead . . . and so on until you find a position, a heading, a certain height or whatever that looks best on it.

The *parameters* to tune in this example are the degrees of freedom of a three dimensional body in the physical world (and maybe variable surrounding conditions). The *criteria* is one’s individual esthetic view of the world.

Important side conditions are:

- Always change only one parameter at a time.
- The amount of change can be varied during the trials, for example taken from a normal distribution.
- A worse result does not always have to be rejected to give the possibility to escape out of a *local minima*. The chance to keep a worser result decreases with the time (*solidification*). The probability can decay according to functions like $e^{-f(t)\cdot\Delta}$ where $f(t)$ might be $log(t)$ or $t^{\frac{1}{t}}$ or just t and Δ is the difference of the criteria.

Figure A.1 displays one of the first runs of simulated annealing process during a tuning session for the stochastic synapse.

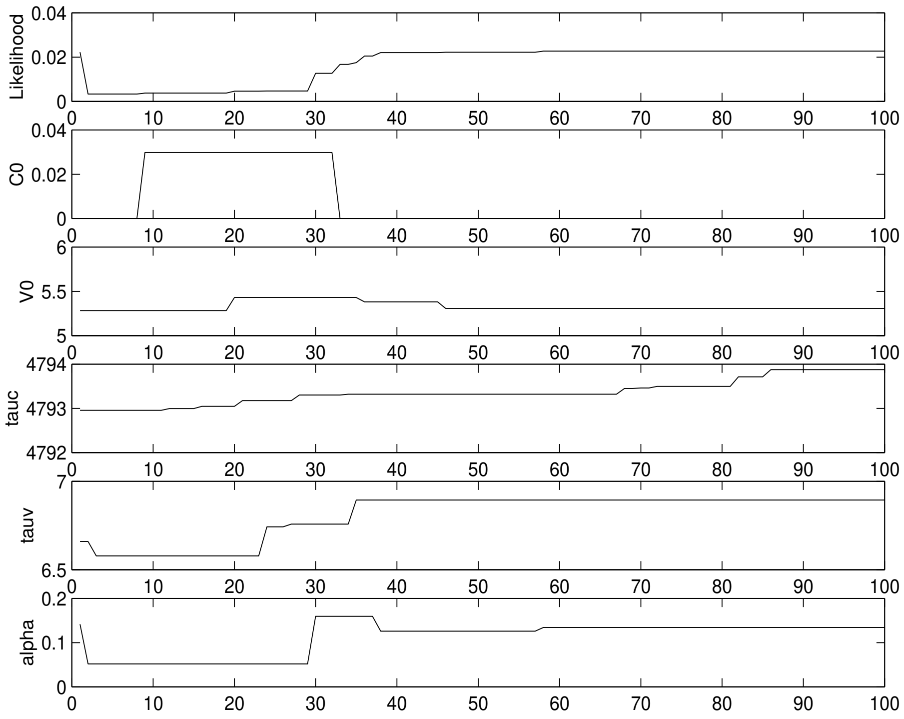

Figure A.1.: Exemplary run of a simulated annealing session

The row labeled "*Likelihood*" is in this case the criteria for the success of the annealing process and says how likely it is that a given release pattern was caused by a dynamic synapse with the according parameters. One can see that it immediately decreases at the beginning of the simulation because the probability that a worser set of parameters is taken is still high.

The choice of a certain criteria decides about the result and depends on what one wants to achieve or to show.

During the tuning of the stochastic synapse various criterias have been chosen:

- To investigate the dynamic characteristics of the synapse – e.g. how different the answer of the synapse might be for two similar spike trains – the criteria might be
$$\frac{\sharp Releases\ for\ S_1}{\sharp Spikes\ in\ S_1} - \frac{\sharp Releases\ for\ S_2}{\sharp Spikes\ in\ S_2}$$
- To compare the synapse with datas from a biological counterpart one might choose the likelihood that a certain release pattern was caused by this synapse as the criteria.
$$\prod_{r_i=1} p_r(t_i) \cdot \prod_{r_i=0} (1 - p_r(t_i))$$
For a more detailed definition see the documentation of the source code in Appendix B. An exemplary run with this criteria can be seen in Figure A.1.

See section 2.3 for results with several criterias during the tuning progress.

The following figures A.2 and A.3 illustrate the behavioral change of the model for different values of C_0. Values called *Trials* (marked with a “x”) show the development of release probability that should be adapted by the model and values labeled *Probability* (marked with an “o”) show the output of the tuned synapse.

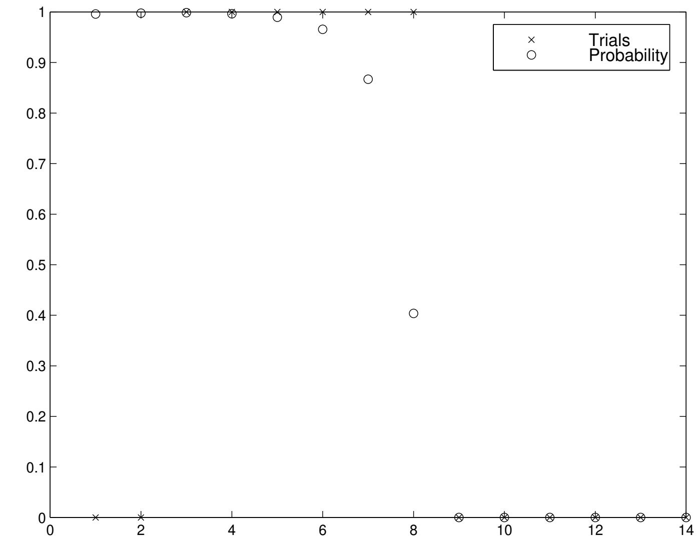

Figure A.2.: Result of the synapse with $C_0 = 1$

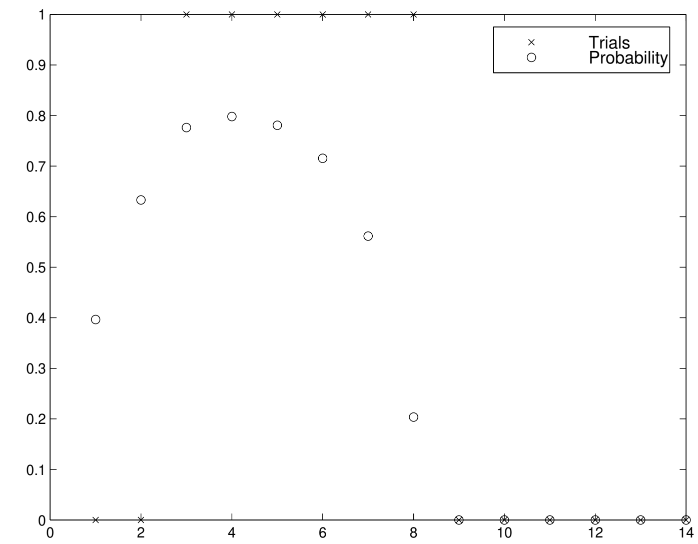

Figure A.3.: Result of the synapse with $C_0 = 0.01$

After 100.000 simulation steps the model yielded the result shown in Figure A.4.

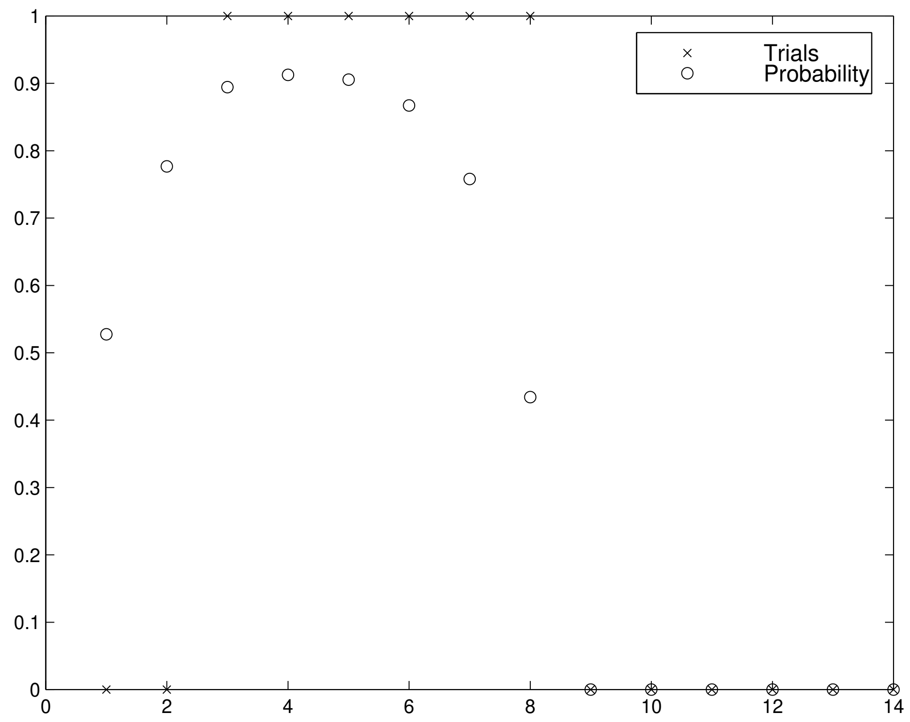

Figure A.4.: Fitting with simulated annealing after 100.000 simulation steps

B. Sources for tuning of the stochastic synapse

This section includes the source codes for the first experiments with the stochastic synapse. All routines have been written for Matlab Version 5.0, running on Linux OS. The programs and subroutines used were:

- **tunelike.m**: Given starting values for C_0, V_0, α, τ_c and τ_v this program tunes autonomously a dynamic synapse with simulated annealing. The interface has been designed to easily reuse returned values for another tuning session.
- **tunelog.m**: The same as "tunelike.m", but this program uses the logarithm to encode the likelihood function instead of pure multiplication. So higher precision can be used without running into rounding errors for very small probabilities like 10^{-90}. The source code as been omitted due to the similarity to "tunelike.m".
- **testexhaustive.m**: This program searches through a given parameter range for the best set of parameters. Here no real tuning is done, but every possible permutation for every parameter between a given starting and ending value with the given step is computed and the best set returned. This is useful for finding a starting point in the vast space of possible parameters for further tuning.
- **likelihood.m**: This function calculates the likelihood that two given release patterns are similar.
- **loglihood.m**: The same as "likelihood.m" but with logarithm. Both subroutines are called by "tunelike.m" or "tunelog.m" respectively.
- **calcp.m**: Implements the stochastic synapse from section 2.2 and calculates the release probability in equation 2.6.
- **prooveb.m**: Similar to "calcp.m" but instead of evaluating the calculated probability to test for a release or failure, it takes a given release pattern. This is used for testing the model against biological datas.
- **plotlike.m**: Output subroutine for development of the parameters C_0, V_0, α, τ_c and τ_v and the "likelihood".

```
function [like,C0_win,V0_win,tauc_win,tauv_win,alpha_win] =
         tunelike ( C0_start, V0_start, tauc_start, tauv_start, alpha_start,
                    ISI, b, sets, k, change_what, am_change, update )
% Tune parameters of a given synapse to fit most likely the given response
%
% Input:  C0_start      resting concentration of calcium
%         V0_start      resting reservoir of vesicles
%         tauc_start    calcium decay constant
%         tauv_start    vesicles decay constant
%         alpha_start   magnitude, respectively, of the response
%         ISI()         interspike intervals of arriving spikes
%         b()()         release patterns which were produced by ISI()
%         sets How many parameter sets should i generate and test?
%         k Parameter for annealing:
% 0    ... c(i)=ln i
% else ... c(i)=i^(1/k)
%         change_what() vector [1,5] that indicates, which of the five
%                       parameters will be tuned during simulated annealing
%                       ALL parameters equal one will be changed!
%         am_change()   vector [1,5] that gives the standard deviation of the
%                       normal distribution with mean zero that is used to
%                       pick the change of the selected parameter.
%         update Update graphics after 'update' sets
%
% Output: like          highest likelihood
%         C0_win        \
%         V0_win         \
%         tauc_win        >  'winning parameters'
%         tauv_win       /
%         alpha_win     /

% Check number of given arguments
if nargin ~= 12, error('Wrong number of arguments.'), end

% determine size of "change"-vector
[xr,xc1] = size(change_what);
if xr~=1, error('change-vector must be vector, not matrix!'), end

% determine size of change amount vector
[xr,xc2] = size(am_change);
if xr~=1, error('am_change must be vector, not matrix!'), end

if xc1~=5, error('change_what must have dimension [1,5]!'), end
if xc2~=5, error('am_change must have dimension [1,5]!'), end

% determine size of first spiketrain
```

```
[xr,xc1] = size(ISI);
if xr~=1, error('Spiketrain must be vector, not matrix!'), end

% determine size of release pattern(s), nr_b is the number of trials
[nr_b,xc2] = size(b);

if xc1~=xc2, error('Spiketrain and b must have same length!'), end

if sets<1, error('sets: Need at least one set ...'), end
if update<1, error('update: Must be greater or equal one'), end


% initialize variables

tmp_like=zeros(1,sets);
C0_run = zeros(1,sets);
V0_run = zeros(1,sets);
tauc_run = zeros(1,sets);
tauv_run = zeros(1,sets);
alpha_run = zeros(1,sets);

C0_run(1) = C0_start;
V0_run(1) = V0_start;
tauc_run(1) = tauc_start;
tauv_run(1) = tauv_start;
alpha_run(1) = alpha_start;

C0_win = C0_run(1);
V0_win = V0_run(1);
tauc_win = tauc_run(1);
tauv_win = tauv_run(1);
alpha_win = alpha_run(1);

like=-1;
p=zeros(nr_b,xc1);
pwin=zeros(1,xc1);

% prepare a list of parameters which could be subject to change
j=1;
for i=1:5
  if change_what(i)==1
    this_one(j)=i;
    j=j+1;
  end
end
[i,nr_para]=size(this_one);
```

```
% loop it!

for i=1:sets

  tmp_like(i)=1;
  for j=1:nr_b

    p(j,:) = prooveb(C0_run(i),V0_run(i),tauc_run(i),tauv_run(i),
                     alpha_run(i),ISI,b(j,:));
    tmp_like(i) = tmp_like(i)*likelihood(p(j,:),b(j,:));

  end

  if tmp_like(i)>like
    like=tmp_like(i);
    C0_win = C0_run(i);
    V0_win = V0_run(i);
    tauc_win = tauc_run(i);
    tauv_win = tauv_run(i);
    alpha_win = alpha_run(i);
    for k=1:xc1
      pwin(k)=1;
      for j=1:nr_b
        pwin(k)=pwin(k)*p(j,k);
      end
    end
  end

% simulate annealing:
  % first: did the result become worse?
  if tmp_like(i)<tmp_like(max(1,i-1))
    % if yes: there is still a chance to keep the changes ...
    if k==0
      verge = log (i);
    else
      verge = i^(1/k);
    end
    pt=(exp(verge*(tmp_like(i)-tmp_like(i-1))));

    fprintf(sprintf('Got worse(%d): %g, %g Pkeep: %g\n',i,tmp_like(i-1),
                     tmp_like(i),pt));
    pause;
    if unifrnd(0,1)>(exp(verge*(tmp_like(i)-tmp_like(i-1))))
%    if unifrnd(0,1)>(exp(-i))
      % no, throw them away
```

```
%       disp('Threw it away ...');
      C0_run(i) = C0_run(i-1);
      V0_run(i) = V0_run(i-1);
      tauc_run(i) = tauc_run(i-1);
      tauv_run(i) = tauv_run(i-1);
      alpha_run(i) = alpha_run(i-1);
      tmp_like(i) = tmp_like(i-1);
%     end
    end
  end

  % first, all parameter for the next run are the same ...
  C0_run(i+1) = C0_run(i);
  V0_run(i+1) = V0_run(i);
  tauc_run(i+1) = tauc_run(i);
  tauv_run(i+1) = tauv_run(i);
  alpha_run(i+1) = alpha_run(i);

  % now look, which parameter will be changed. every number in this_one()
  % will participate in this lottery ...
  % take numbers from 1 to nr_para, which have the same probability to occur
  j=round(unifrnd(0.5,(nr_para+0.49999999)));
  % ok, change the j-th one.
  if this_one(j)==1
    C0_run(i+1)=max(0.000000001,C0_run(i)+normrnd(0,am_change(1)));
  elseif this_one(j)==2
    V0_run(i+1)=max(0.000000001,V0_run(i)+normrnd(0,am_change(2)));
  elseif this_one(j)==3
    tauc_run(i+1)=max(0.000000001,tauc_run(i)+normrnd(0,am_change(3)));
  elseif this_one(j)==4
    tauv_run(i+1)=max(0.000000001,tauv_run(i)+normrnd(0,am_change(4)));
  elseif this_one(j)==5
    alpha_run(i+1)=max(0.000000001,alpha_run(i)+normrnd(0,am_change(5)));
  else
    error('Well, this shouldn''t happen ...');
  end

  if rem(i,update)==0
    plotlike(tmp_like(1:i),C0_run(1:i),V0_run(1:i),tauc_run(1:i),
             tauv_run(1:i),alpha_run(1:i));
  end

end

bav=zeros(1,xc1);
for i=1:xc1
```

```
  bav(i)=sum(b(:,i))/nr_b;
end

%print out winning parameters
nr_b
bav
pwin
plotprob(bav,pwin);
```

```
function [like,C0_win,V0_win,tauc_win,tauv_win,alpha_win] =
         testexhaustive ( C0_range, V0_range, tauc_range, tauv_range,
                          alpha_range, ISI, b, update )
% Exhaustively search through the parameter range of a given synapse
% and report the most probable parameters
%
% Input:  C0_range()    resting concentration of calcium: [begin,end,delta]
%         V0_range()    resting reservoir of vesicles: [begin,end,delta]
%         tauc_range()  calcium decay constant: [begin,end,delta]
%         tauv_range()  vesicles decay constant: [begin,end,delta]
%         alpha_range() magnitude, respectively, of the response: same
%         ISI()         interspike intervals of arriving spikes
%         b()()         release patterns which were produced by ISI()
%         update Update output after 'update' trials
%
% Output: like          highest likelihood
%         C0_win        \
%         V0_win         \
%         tauc_win        >  'winning parameters'
%         tauv_win       /
%         alpha_win     /

% Check number of given arguments
if nargin ~= 8, error('Wrong number of arguments.'), end

% determine size of spiketrain
[xr,xc1] = size(ISI);
if xr~=1, error('Spiketrain must be vector, not matrix!'), end

% determine size of release pattern
[nr_b,xc2] = size(b);

if xc1~=xc2, error('Spiketrain and b must have same length!'), end
if update<1, error('update: Must be greater or equal one'), end

% do some paranoia checking ...
if (C0_range(2)-C0_range(1))<=0, error('illegal range for C0!'), end
if (V0_range(2)-V0_range(1))<=0, error('illegal range for V0!'), end
if (tauc_range(2)-tauc_range(1))<=0, error('illegal range for tauc!'), end
if (tauv_range(2)-tauv_range(1))<=0, error('illegal range for tauv!'), end
if (alpha_range(2)-alpha_range(1))<=0, error('illegal range for alpha!'), end
if (C0_range(3))<=0, error('illegal delta for C0!'), end
if (V0_range(3))<=0, error('illegal delta for V0!'), end
if (tauc_range(3))<=0, error('illegal delta for tauc!'), end
if (tauv_range(3))<=0, error('illegal delta for tauv!'), end
```

```
if (alpha_range(3))<=0, error('illegal delta for alpha!'), end

% initialize variables

i=0;
like=-1;
p=zeros(nr_b,xc1);

% display 'warning' message
no_sets=round(((C0_range(2)-C0_range(1))/C0_range(3)+1)*
              ((V0_range(2)-V0_range(1))/V0_range(3)+1)*
              ((tauc_range(2)-tauc_range(1))/tauc_range(3)+1)*
              ((tauv_range(2)-tauv_range(1))/tauv_range(3)+1)*
              ((alpha_range(2)-alpha_range(1))/alpha_range(3)+1));
fprintf(sprintf('Warning: I will exhaustively search for parameters
                 in the given ranges!\n'));
fprintf(sprintf('         In this case, I will tryout about
                 %d different settings!!!\n',no_sets));

% loop it!

for C0_run=C0_range(1):C0_range(3):C0_range(2)
  for V0_run=V0_range(1):V0_range(3):V0_range(2)
    for tauc_run=tauc_range(1):tauc_range(3):tauc_range(2)
      for tauv_run=tauv_range(1):tauv_range(3):tauv_range(2)
        for alpha_run=alpha_range(1):alpha_range(3):alpha_range(2)

          tmp_like=1;
          for j=1:nr_b
            p(j,:) = prooveb(C0_run,V0_run,tauc_run,tauv_run,alpha_run,
                             ISI,b(j,:));
            tmp_like = tmp_like*likelihood(p(j,:),b(j,:));
          end

          if tmp_like>like
            like=tmp_like;
            C0_win = C0_run;
            V0_win = V0_run;
            tauc_win = tauc_run;
            tauv_win = tauv_run;
            alpha_win = alpha_run;
          end

          i=i+1;
          if rem(i,update)==0
            fprintf(sprintf('Current status:\n'));
```

```
          fprintf(sprintf('    Max. likelihood: %g\n',like));
          fprintf(sprintf('    Best C0        : %g\n',C0_win));
          fprintf(sprintf('    Best V0        : %g\n',V0_win));
          fprintf(sprintf('    Best tauc      : %g\n',tauc_win));
          fprintf(sprintf('    Best tauv      : %g\n',tauv_win));
          fprintf(sprintf('    Best alpha     : %g\n',alpha_win));
          fprintf(sprintf('    Range exhausted: %2.1f%%%%\n\n\n',
                           i*100/no_sets));
        end

      end
    end
  end
 end
end
```

```
function [like] = likelihood ( p, b )
% function [like] = likelihood ( p, b )
%
% Calculates the likelihood, that p corresponds with b
%
% Input:  p()           probability that PSP will be released
%         b()           release pattern which could be produced by p
%
% Output: like          how likely it is, that p corresponds with b

% Check number of given arguments
if nargin < 2, error('Not enough arguments.'), end

% determine size of "p"
[xr,xc1] = size(p);
if xr~=1, error('p must be vector, not matrix!'), end
% determine size of "b"
[xr,xc2] = size(b);
if xr~=1, error('b must be vector, not matrix!'), end
if xc1~=xc2, error('p and b must have same dimension!'), end

% initialize variables

like=1;

% repeat for all members of "p"
for i=1:xc1
  if b(i)==1                       % PSP was released
    like=like*p(i);
  else
    like=like*(1-p(i));            % nope, not this time ...
  end
end
```

```
function [like] = loglihood ( p, b )
% function [like] = loglihood ( p, b )
%
% Calculates the likelihood, that p corresponds with b, but use logarithm
%
% Input:  p()           probability that PSP will be released
%         b()           release pattern which could be produced by p
%
% Output: like          how likely it is, that p corresponds with b

% Check number of given arguments
if nargin < 2, error('Not enough arguments.'), end

% determine size of "p"
[xr,xc1] = size(p);
if xr~=1, error('p must be vector, not matrix!'), end
% determine size of "b"
[xr,xc2] = size(b);
if xr~=1, error('b must be vector, not matrix!'), end
if xc1~=xc2, error('p and b must have same dimension!'), end

% initialize variables

like=0;

% repeat for all members of "p"
for i=1:xc1
  if b(i)==1                          % PSP was released
    like=like+log(p(i));
  else
    like=like+log((1-p(i)));                % nope, not this time ...
  end
end
```

```
function [p, s, c, v] = calcp ( C0, V0, tauc, tauv, alpha, ISI )
% function [p, s, c, v] = calcp ( C0, V0, tauc, tauv, alpha, ISI )
%
% Calculation of the probability that a synapse releases PSPs on an incoming
% spiketrain.
% STOCHASTIC SYNAPSE WITHOUT SPONTANOUS SPIKING !!!
%
% Input:  C0            resting concentration of calcium
%         V0            resting reservoir of vesicles
%         tauc          calcium decay constant
%         tauv          vesicles decay constant
%         alpha         magnitude, respectively, of the response
%         ISI()         interspike intervals of arriving spikes
%
% Output: p()           probability that the spike at time ti triggered the
%                       release of a PSP
%         s()           release pattern: 1 ... released, 0 ... not suppressed
%         c()           "amount" calcium at each spike occurence
%         v()           pool of vesicles       - " -

% Check number of given arguments
if nargin < 6, error('Not enough arguments.'), end

% determine size of ISI
[xr,xc] = size(ISI);

if xr~=1, error('Spiketrain must be vector, not matrix!'), end

if C0   < 0,  error('C0 cannot be negative!'), end
if V0   <= 0, error('V0 cannot be zero or negative!'), end
if tauv <= 0, error('tauv cannot be zero or negative!'), end
if tauc <= 0, error('tauc cannot be zero or negative!'), end
if alpha <= 0, error('Alpha cannot be zero or negative!'), end

% initialize variables

c=zeros(xr,xc);
v=zeros(xr,xc);
p=zeros(xr,xc);
s=zeros(xr,xc);

% do first time step "manually"

c(1)=C0+alpha;              % calculate calcium: spike arrives, add alpha
v(1)=V0;                    % nothing was released yet
```

```
p(1)=1-exp(-c(1)*v(1));      % probability of release
if (unifrnd(0,1)<=p(1))      % test against uniform distribution
  s(1)=1;                    % PSP was released
else
  s(1)=0;                    % nope, not this time ...
end

% now repeat for all members of "ISI"
for i=2:xc
  j=i-1;
  c(i)=C0+alpha;                      % start with C0 and current spike
  v(i)=V0;
  time_passed=0;
  while j>=1
    time_passed=time_passed+ISI(j+1);           % go back in time and add it
    c(i)=c(i)+alpha*exp(-time_passed/tauc);     % add quantil for each spike
    if s(j)==1
      v(i)=max(0,v(i)-exp(-time_passed/tauv));  % subtract vesicle for each
    end                                         % release
    j=j-1;
  end
  p(i)=1-exp(-c(i)*v(i));             % probability of release
  if (unifrnd(0,1)<=p(i))             % test against uniform distribution
    s(i)=1;                           % PSP was released
  else
    s(i)=0;                           % nope, not this time ...
  end
end
```

```
function [p] = prooveb ( C0, V0, tauc, tauv, alpha, ISI, s )
% function [p] = prooveb ( C0, V0, tauc, tauv, alpha, ISI, s )
%
% Calculation of the probability that a synapse releases PSPs on an incoming
% spiketrain.
%
% Input:  C0            resting concentration of calcium
%         V0            resting reservoir of vesicles
%         tauc          calcium decay constant
%         tauv          vesicles decay constant
%         alpha         magnitude, respectively, of the response
%         ISI()         interspike intervals of arriving spikes
%         s()           release pattern which should be similar to the ISIs
%
% Output: p()           probability that the spike at time ti triggered the
%                       release of a PSP in s()

% Check number of given arguments
if nargin < 7, error('Not enough arguments.'), end

% determine size of ISI
[xr,xc1] = size(ISI);
if xr~=1, error('Spiketrain must be vector, not matrix!'), end
% determine size of s
[xr,xc2] = size(s);
if xr~=1, error('Release pattern must be vector, not matrix!'), end
if xc1~=xc2, error('ISI and s must have same size!'), end

if C0   < 0,  error('C0 cannot be negative!'), end
if V0   <= 0, error('V0 cannot be zero or negative!'), end
if tauv <= 0, error('tauv cannot be zero or negative!'), end
if tauc <= 0, error('tauc cannot be zero or negative!'), end
if alpha <= 0, error('Alpha cannot be zero or negative!'), end

% initialize variables

c=zeros(xr,xc1);
v=zeros(xr,xc1);
p=zeros(xr,xc1);

% do first time step "manually"

c(1)=C0+alpha;               % calculate calcium: spike arrives, add alpha
v(1)=V0;                     % nothing was released yet
p(1)=1-exp(-c(1)*v(1));      % probability of release
```

```
% now repeat for all members of "ISI"
for i=2:xc1
  j=i-1;
  c(i)=C0+alpha;                          % start with C0 and current spike
  v(i)=V0;
  time_passed=0;
  while j>=1
    time_passed=time_passed+ISI(j+1);           % go back in time and add it
    c(i)=c(i)+alpha*exp(-time_passed/tauc);     % add quantil for each spike
    if s(j)==1
      v(i)=max(0,v(i)-exp(-time_passed/tauv));  % subtract vesicle for each
    end                                         % release
    j=j-1;
  end
  p(i)=1-exp(-c(i)*v(i));                 % probability of release
end
```

```
function plotlike(like,C0,V0,tauc,tauv,alpha)
% PlotParams ... plot parameters given
%

if nargin > 6,error('Too many arguments !'), end

figure(1);
hold off

  % draw rate
  subplot(nargin,1,1)
  plot(like,'m')
  ylabel('Likelihood')

  % draw V0
  subplot(nargin,1,2)
  plot(C0,'m')
  ylabel('C0')

  % draw V0
  subplot(nargin,1,3)
  plot(V0,'m')
  ylabel('V0')

  % draw tauc
  subplot(nargin,1,4)
  plot(tauc,'m')
  ylabel('tauc')

  % draw tauv
  subplot(nargin,1,5)
  plot(tauv,'m')
  ylabel('tauv')

  if nargin > 5
    % draw alpha
    subplot(nargin,1,6)
    plot(alpha,'m')
    ylabel('alpha')
  end

drawnow;
```

C. Sources for tuning of the modified stochastic synapse

This section includes the source codes for the tuning sessions with the MS synapse and the comparisons with the original stochastic model. The different subroutines for displaying purposes have been omitted. All routines have been written for Matlab Version 5.0, running on Linux OS. The programs and subroutines used were:

- **tuneds.m**: This program tunes autonomously the dynamic synapse with simulated annealing. The starting parameters have to be altered directly in the source code.

- **tradds.m**: This subroutine is called by "tuneds.m" and returns the answer of the stochastic synapse to a spike train.

- **msds.m**: It is called by "tuneds.m" and returns the answer of the MS synapse for a certain spike train.

- **sqrerr.m**: Calculates the square error of two given sets of probabilities – in this case the (different) answers of the two different synapses.

- **gentrain.m**: Generates a set of time points (a spike train like in definitions in Chapter 2 and 3) for a given probability, length and time step.

- **collstat.m**: Collects statistics about release probabilities of certain synapses.

- **collstat2.m**: Similar to "collstat.m".

```
function tuneds()
%TUNETRADDS: Tune a traditional dynamic synapses with minsq. error
%            compared to lynn's data
%

% Harry Fuchs, 1.10.98
% Copyright (c) 1998 by Harry.

%starting parameters ...
c0_start=0;
v0_start=17;
tc_start=20;
tv_start=10.6897;
alp_start=0.0894924;
change_what=[1 0 1 1 1];                     %what params should be tuned for
am_change=[0.1 0.1 0.1 0.1 0.1];             %sigma of normal dist. for each param

s=linspace(0,3.1,32);                        %like in lynn's data
l=length(s);
N=15000;                                     %how many trials
update=100;                                  %when should the output be updated?
M=1000;                                      %build an average over M trials at the end

lynndat=[0.1905 0.4048 0.5476 0.5714 0.4762 0.4524 0.3095 0.2381
         0.1190 0.0952 0.0238 0.0476 0.0476 0      0.0476 0
         0      0.0238 0      0.0238 0.0476 0      0.0714 0.0476
         0.0238 0.0238 0.0714 0.0238 0.0238 0      0.0714 0.0476];

% prepare a list of parameters which could be subject to change
j=1;
for i=1:5
  if change_what(i)==1
    this_one(j)=i;
    j=j+1;
  end
end
[i,nr_para]=size(this_one);

tmp_err=zeros(1,N);
c0_run = zeros(1,N);
v0_run = zeros(1,N);
tc_run = zeros(1,N);
tv_run = zeros(1,N);
alp_run = zeros(1,N);
```

```
c0_run(1) = c0_start;
v0_run(1) = v0_start;
tc_run(1) = tc_start;
tv_run(1) = tv_start;
alp_run(1) = alp_start;

c0_win = c0_run(1);
v0_win = v0_run(1);
tc_win = tc_run(1);
tv_win = tv_run(1);
alp_win = alp_run(1);

err_win=inf;                                   %error is indeed very high ...
p1coll=zeros(M,1);

for i=1:N
  %give a status report ...
  if rem(i,update)==0,fprintf(sprintf('    Iteration: %g/%g\n',i,N));,end

  [c,v,p1,r1]=tradds(s,[c0_run(i) v0_run(i) tc_run(i) tv_run(i) alp_run(i)]);
                           %calculate for trad. ds
  [c,v,p2]=msds(s,[c0_run(i) v0_run(i) tc_run(i) tv_run(i) alp_run(i)]);
                           %calculate for ms ds

  %calculate error
  tmp_err(i)=sqrerr(p2,lynndat);

  if tmp_err(i)<err_win                %save parameters if they're the best so far
    err_win= tmp_err(i);
    c0_win = c0_run(i);
    v0_win = v0_run(i);
    tc_win = tc_run(i);
    tv_win = tv_run(i);
    alp_win = alp_run(i);
  end

% simulate annealing:
  % first: did the result become worse?
  if tmp_err(i)>tmp_err(max(1,i-1))
    %yes, but there's still a chance to keep them!
    if unifrnd(0,1)>(exp(-i))
      %no, throw them away
      %disp('Threw it away ...');
      c0_run(i) = c0_run(max(1,i-1));
      v0_run(i) = v0_run(max(1,i-1));
      tc_run(i) = tc_run(max(1,i-1));
```

```
      tv_run(i) = tv_run(max(1,i-1));
      alp_run(i) = alp_run(max(1,i-1));
      tmp_err(i) = tmp_err(max(1,i-1));
    end
  end

  % second: all parameter for the next run are the same ...
  c0_run(min(N,i+1)) = c0_run(i);
  v0_run(min(N,i+1)) = v0_run(i);
  tc_run(min(N,i+1)) = tc_run(i);
  tv_run(min(N,i+1)) = tv_run(i);
  alp_run(min(N,i+1)) = alp_run(i);

  % now look, which parameter will be changed. every number in this_one()
  % will participate in this lottery ...
  % take numbers from 1 to nr_para, which have the same probability to occur
  j=round(unifrnd(0.5,(nr_para+0.49999999)));
  % ok, change the j-th one.
  if this_one(j)==1
    c0_run(i+1)=max(0.00000000001,c0_run(i)+normrnd(0,am_change(1)));
  elseif this_one(j)==2
    v0_run(i+1)=max(0.00000000001,v0_run(i)+normrnd(0,am_change(2)));
  elseif this_one(j)==3
    tc_run(i+1)=max(0.00000000001,tc_run(i)+normrnd(0,am_change(3)));
  elseif this_one(j)==4
    tv_run(i+1)=max(0.00000000001,tv_run(i)+normrnd(0,am_change(4)));
  elseif this_one(j)==5
    alp_run(i+1)=max(0.00000000001,alp_run(i)+normrnd(0,am_change(5)));
  else
    error('Well, this shouldn''t happen ...');
  end

  if rem(i,update)==0
    plotprob(tmp_err(1:i),c0_run(1:i),v0_run(1:i),tc_run(1:i),tv_run(1:i),alp_run(1:i));
    plotprobs(2,p2,lynndat,'New DS','Lynn''s datas');
  end
end

%
% Now prepare for displaying our results ...
%

fprintf(sprintf('Building average over %g trials ...\n',M));
for i=1:M
  [c,v,p1coll(i,:),r]=tradds(s,[c0_win v0_win tc_win tv_win alp_win]);
end
```

```
p1stat=zeros(1,l);
r1stat=zeros(1,l);
for i=1:l
  p1stat(i)=sum(p1coll(:,i))/M;
end

%calculate again with the winning parameters for the new DS
%(just in case the results got worse during the last run ...)
[c,v,p2]=msds(s,[c0_win v0_win tc_win tv_win alp_win]);

plotprobs(2,p1stat,p2,'Traditional DS','New DS');
plotprobs(3,p1stat,lynndat,'Traditional DS','Lynn''s datas');
plotprobs(4,p2,lynndat,'New DS','Lynn''s datas');
plot3prob(5,lynndat,p1stat,p2);

disp('Winning parameters:');
fprintf(sprintf('    C0   : %g\n',c0_win));
fprintf(sprintf('    V0   : %g\n',v0_win));
fprintf(sprintf('    tauc : %g\n',tc_win));
fprintf(sprintf('    tauv : %g\n',tv_win));
fprintf(sprintf('    alpha: %g\n',alp_win));
fprintf(sprintf('    =>err: %g\n',err_win));
```

```
function [c,v,p,r] = tradds(s,dspar)
%TRADDS: Evaluate a "traditional" dynamic synapse.
%
% TRADDS must be called with both parameters, s and dspar
%
% [c,v,p,r] = TRADDS(s,dspar)
%   s     - Matrix with #synapses rows. Each row is a vector
%   representing the arrival times of incoming spikes.
%   dspar - Vector of dynamic synapse parameters.
%   = [c0, v0, tauc, tauv, alpha]
% Returns
%   c     - Development of calcium concentration in the synapse.
%   v     - Development of vesicles.
%   p     - Course of the probability of releases of the synapse.
%   r     - 1/0 <=> Release/NoRelease.

% Harry Fuchs, 18.9.98
% Copyright (c) 1998 by Harry.

if (nargin~=2), error('Wrong number of input arguments'), end
if size(dspar)~=[1,5], error('dspar has wrong number of parameters'), end;
[nos,len]=size(s);

%predefine array for the results
c=zeros(nos,len);
v=zeros(nos,len);
p=zeros(nos,len);
r=zeros(nos,len);

for i=1:nos
  for j=1:len
    ct=dspar(1);
    vt=dspar(2);
    mytime=s(i,j);
    for k=1:j
      ct=ct+dspar(5)*exp((-mytime+s(i,k))/dspar(3));
      if k>1
        vt=max(0,vt-r(i,k-1)*exp((-mytime+s(i,k-1))/dspar(4)));
      end
    end
    c(i,j)=ct;
    v(i,j)=vt;
    p(i,j)=1.0-exp(-ct*vt);
    if p(i,j)>rand, r(i,j)=1;, end;
  end
end
```

```
function [c,v,p] = msds(s,dspar)
%MSDS: Evaluate an electronically easy to imlement dynamic synapse.
%
% MSDS must be called with both parameters, s and dspar
%
% [c,v,p] = MSDS(s,dspar)
%   s     - Matrix with #synapses rows. Each row is a vector
%   representing the arrival times of incoming spikes.
%   dspar - Vector of dynamic synapse parameters.
%   = [c0, v0, tauc, tauv, alpha]
% Returns
%   c     - Development of calcium concentration in the synapse.
%   v     - Development of vesicles.
%   p     - Course of the probability of releases of the synapse.

% Harry Fuchs, 18.9.98
% Copyright (c) 1998 by Harry.

if (nargin~=2), error('Wrong number of input arguments'), end

if size(dspar)~=[1,5], error('dspar has wrong number of parameters'), end;
[nos,len]=size(s);

%predefine array for the results
c=zeros(nos,len);
v=zeros(nos,len);
p=zeros(nos,len);
r=zeros(nos,len);

for i=1:nos
  for j=1:len
    ct=dspar(1);
    vt=dspar(2);
    mytime=s(i,j);
    for k=1:j
      ct=ct+dspar(5)*exp((-mytime+s(i,k))/dspar(3));
      if k>1
        vt=max(0,vt-p(i,k-1)*exp((-mytime+s(i,k-1))/dspar(4)));
      end
    end
    c(i,j)=ct;
    v(i,j)=vt;
    p(i,j)=1.0-exp(-ct*vt);
    if p(i,j)>rand, r(i,j)=1;, end;
  end
end
```

```
function err = sqrerr(a,b)
%SQRERR: Calculate square error
%
% err = SQRERR(a,b)
%   a,b   - Input vectors
% Returns
%   err   - =sum((a-b)^2)

% Harry Fuchs, 1.10.98
% Copyright (c) 1998 by Harry.

if (nargin~=2),error('Wrong number of input arguments'),end
if size(a)~=size(b),error('Dimesions of vectors do not agree'),end

tmp=a-b;
err=sum(tmp.*tmp);
```

```
function [s] = gentrain(r,length,tau)
%GENTRAIN: Generate a spiketrain with uniform probability.
%
% [s] = GENTRAIN(r,rdev,length,tau)
%   r      - Probability of spike in each time step.
%   length - Length of train (in seconds).
%   tau    - Time step for the generation of the train (in seconds).
% Returns
%   s      - Vector representing the absolute times of spikes.

% Harry Fuchs, 29.9.98
% Copyright (c) 1998 by Harry.

if (nargin~=3),error('Wrong number of input arguments'),end
if (tau>length), error('Tau is greater than length!'),end

i=1;
for mytime=0:tau:length
  if rand<r, s(i)=mytime; i=i+1;, end
end
```

```
function collstat()
%COLLSTAT: Collect statistics about the release probabilities of dynamic synapses.
%

% Harry Fuchs, 30.9.98
% Copyright (c) 1998 by Harry.

c0=0;
v0=3;
tauc=2;
tauv=2;
alp=0.5;
prob=0.02;
p1coll=0;
p2coll=0;
l=length(p1coll);

while (l<10000),                                %limit size ...
  s=gentrain(prob,100,0.005);                   %take different spike trains
  [c,v,p1,r]= tradds(s,[c0 v0 tauc tauv alp]);  %calculate for trad. ds
  [c,v,p2]  = msds  (s,[c0 v0 tauc tauv alp]);  %calculate for ms ds
  if length(p1coll)==1                          %don't add starting value
    p1coll=p1;
    p2coll=p2;
  else
    p1coll=[p1coll p1];
    p2coll=[p2coll p2];
  end
  l=length(p1coll)
end

save 'p1coll.mat' p1coll
save 'p2coll.mat' p2coll

plotstats(p1coll,p2coll);         % plot the probability distribution of p1 and p2
```

```
function collstat2()
%COLLSTAT": Collect statistics about the release probabilities of dynamic synapses.
%           in time domain

% Harry Fuchs, 1.10.98
% Copyright (c) 1998 by Harry.

c0_start=0;
v0_start=3.23969;
tc_start=242;
tv_start=10.6897;
alp_start=0.0894924;

s=linspace(0,3.1,32);                                    %like in lynn's data
l=length(s);
N=50000;                                                %amout of data to sum up
update=100;
p1coll=zeros(N,l);
r1coll=zeros(N,l);

[c,v,p2]=msds(s,[c0_start v0_start tc_start tv_start alp_start]);
  %calculate for ms ds
  %p2 stay's the same all the times, so we only calculate it once ...

for i=1:N
  [c,v,p1coll(i,:),r1coll(i,:)]=
                      tradds(s,[c0_start v0_start tc_start tv_start alp_start]);
                                                         %calculate for trad. ds
  if rem(i,update)==0,fprintf(sprintf('    Iteration: %g/%g\n',i,N));,end
end

%prepare statistics
for i=1:l
  p1stat(i)=sum(p1coll(:,i))/N;
  r1stat(i)=sum(r1coll(:,i))/N;
end

plotprobs(1,p1stat,p2,'p1','p2');
plotprobs(2,p1stat,r1stat,'p1','r1');
plot3prob(3,p1stat,r1stat,p2);
```

Bibliography

[Abbott et al., 1997] Abbot, L.F., Varela, J.A., Sen, K. and Nelson, S.B. (1997). *Synaptic depression and cortical gain control.* Science, 275:220-224.

[Beuth, 88] Beuth, K. (1988). *Elektronik 2: Bauelemente.* Würzburg: Vogel.

[Bialek and Rieke, 1992] Bialek, W. and Rieke, F. (1992). *Reliability and information transmission in spiking neurons.* Trends in Neuroscience, vol. 15, 428-434.

[Borst and Egelhaaf, 93] Borst, A. and Egelhaaf, M. (1993). *Processing of synaptic signals in fly visual interneurons selectively responsible to small moving object.* Brain Theory, Aertsen (editor), pp. 47-66.

[Churchland, P. and Sejnowski, T., 1992] Churchland, P. and Sejnowski, T. (1992). *The Computational Brain.* MIT Press, Cambridge.

[Dinse, 1994] Dinse, H.R. (1994). *A time-based approach towards cortical functions: neural mechanisms underlying dynamic aspects of information processing before and after postontogenetic plastic processes.* Physica D 75, pp. 129-150.

[Diorio and Minch, 1996] Diorio, C. and Minch, B.A. (1996). *A Single-Transistor Silicon Synapse.* IEEE Transactions On Electron Devices, Vol. 43, No. 11.

[Diorio et al., 1995] Diorio, C., Mahajan, S., Hasler, P., Minch, B.A. and Mead, C. (1995 *A high resolution non-volatile analog memory cell.* Proceedings of the International Conference of Circuits and Systems, Seattle, vol. 3, 1995, pp. 2233-2236.

[Douglas and Mahowald, 1995] Douglas, R. and Mahowald, M. (1995). *A Construction Set for Silicon Neurons.* In: *An Introduction to Neural and Electronic Networks*, Academic Press., pp. 277-296.

[Dobrunz and Stevens, 1997] Dobrunz, L. and Stevens, C. (1997). *Heterogeneity of release probability, facilitation and depletion at central synapses.* Neuron, 18:995-1008.

[Borst and Egelhaaf, 1993] Egelhaaf, M. and Borst, A. (1993). *A Look into the Cockpit of the Fly: Visual Orientation, Algorithms, and Identified Neurons.* The Journal of Neuroscience, November 1993, 13(11): 4563-4574.

[Elias et al., 1996] Elias, J.G., Northmore, D.P.M. and Westerman, W. (1996). *An Analog Memory Circuit for Spiking Silicon Neurons.*

[Elias and Northmore, 1995] Elias, J.G. and Northmore, D.P.M. (1995). *Switched-Capacitor Neuromorphs with Wide-Range Variable Dynamics.* IEEE Transactions On Neural Networks.

[Gerstner, 1995] Gerstner, W. (1995). *Time structure of the activity in neural network models.* Phys. Rev. E, Vol. 51, 738-758.

[Gerstner et al., 1993a] Gerstner W., Ritz R. and van Hemmen J.L.. (1993). *A biologically motivated and analytically soluble model of collective oscillations in the cortex: I. Theory of weak locking.* Biol. Cybern. 68:363-374.

[Gerstner et al., 1993b] Gerstner W., Ritz R. and van Hemmen J.L.. (1993). *Why Spikes? Hebbian learning and retrieval of time-resolved excitation patterns.* Biol. Cybern.

[Harrison et al., 1998] Harrison, R.R., Hasler, P. and Minch, B. (1998). *Floating-Gate CMOS Analog Memory Cell Array.* Proceedings of the 1998 IEEE International Symposium on Circuits and Systems, Monterey, CA, 2:204-207.

[Harrison and Koch, 1998] Harrison, R.R. and Koch, C. (1998). *An Analog VLSI Model of the Fly Elementary Motion Detector.* In: Advances in Neural Information Processing Systems 10, Jordan, M.I., Kearns, M.J., Solla, S.A., eds., MIT Press, Cambridge,MA.

[Hopfield, 1995] Hopfield, J.J. (1995). *Pattern recognition computation using action potential timing for stimulus representation.* Nature, vol. 376, pp. 33-36.

[Indiveri et al., 1996] Indiveri, G., Kramer, J. and Koch, C. (1996). *System Implementations of Analog VSLI Velocity Sensors.* IEEE Micro, vol. 16, no. 5, pp. 40-49.

[Katz, 1966] Katz, B. (1966). *Nerve, muscle and synapse.* New York, McGraw-Hill.

[Maass, 1995] Maass, W. (1995). *An Efficient Implementation of Sigmoidal Neural Nets in Temporal Coding with Noisy Spiking Neurons.*

[Maass, 1997] Maass, W. (1997), *On the Relevance of Time in Neural Computation and Learning.* Proceedings of ALT'97, Lecture Notes in Computer Science, Springer, Berlin.

[Maass and Zador, 1998] Maass, W. and Zador, A. (1998). *Dynamic Stochastic Synapses as Computational Units.* Advances of Neural Information Processing Systems, Vol. 10, 194-200.

[Maass and Natschläger, 1998] Maass, W. and Natschläger, T. (1998). *A Model for Fast Analog Computation Based on Unreliable Synapses.* submitted.

[Mahowald and Douglas, 1991] Mahowald, M. and Douglas, R. (1991). *A silicon neuron.* Nature, 354, pp. 515-518.

[McCulloch and Pitts, 1943] McCulloch, W.S. and Pitts, W. (1943). *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics, 5:115-133.

[Mead, 1989] Mead, C.A., (1989). *Analog VLSI and neural systems.* Reading, MA: Addison-Wesley.

[Murray and Tarassenko, 1994] Murray, A. and Tarassenko, L. (1994). *Analogue Neural VLSI: A Pulse Stream Approach.* Chapman & Hall.

[Smith and Hamilton, 1998] Smith, L.S. and Hamilton, A., eds. (1998). *Neuromorphic Systems: Engineering Silicon from Neurobiology.* Singapore, London: World Scientific.

[Thorpe, 1994] Thorpe, S.J. (1994). *Localized Versus Distributed Representation.* In: Arbib, M.A., editor, The Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, pp. 549-552.

[Thorpe and Imbert, 1989] Thorpe, S.J. and Imbert, M. (1989). *Biological constraints on connectionist modelling.* In: Connectionism in Perspective, Pfeifer, R., Schreter, Z., Fogelman-Soulie, F. and Steels, L., eds., Elsevier (North-Holland).

[Tran et al., 1996] Tran, H.V., Blyth, T., Sowards, D., Engh, L. Nataraj, T., Dunne, T., Wong, H., Se rin, V., Lam, T., Hazarian, H. and Hu, G. (1996). *A 2.5V 256-level non-volatile analog storage device using EEPROM technology.* Proceedings of the IEEE International Solid-State Circuits Conference, 1996, pp. 270-271.

[Zador and Dobrunz, 1997] Zador, A. and Dobrunz, L. (1997). *Dynamic synapses in the cortex.* Neuron, 19:1-4.

Index