THE LEARNING PROBLEM FOR DISCRETE-TIME CELLULAR NEURAL NETWORKS AS A COMBINATORIAL OPTIMIZATION PROBLEM

by

Holger Magnussen, Josef A. Nossek, and Leon O. Chua

Memorandum No. UCB/ERL M93/88

17 December 1993

THE LEARNING PROBLEM FOR DISCRETE-TIME CELLULAR NEURAL NETWORKS AS A COMBINATORIAL OPTIMIZATION PROBLEM

by

Holger Magnussen, Josef A. Nossek, and Leon O. Chua

Memorandum No. UCB/ERL M93/88

17 December 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

THE LEARNING PROBLEM FOR DISCRETE-TIME CELLULAR NEURAL NETWORKS AS A COMBINATORIAL OPTIMIZATION PROBLEM

by

Holger Magnussen, Josef A. Nossek, and Leon O. Chua

Memorandum No. UCB/ERL M93/88

17 December 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

This work is supported in part by the Office of Naval Research under grant N00014-89-J-1402 and the National Science Foundation under grant MIP-9001336.

The *Learning* Problem for Discrete-Time Cellular Neural Networks as a Combinatorial Optimization Problem

Holger Magnussen*† and Josef A. Nossek‡ and Leon O. Chua

December 17, 1993

Abstract

"Global" Learning algorithms for Discrete-Time Cellular Neural Networks (DTCNNs) are a class of learning algorithms where the algorithm designs the trajectory of the network. For binary input patterns, the global learning problem for DTCNNs is a combinatorial optimization problem. Properties of the solution space can be deduced using the available theory on Linear Threshold Logic. Results on the required accuracy of the network parameters are given. The problem of deciding whether a given task can be learned for a DTCNN architecture (feasibility problem) is conjectured to be NP-complete. A cost function is defined that is a measure of the errors in the mapping process from a set of input images onto the desired output images. Simulated Annealing methods are used to minimize this cost function. The algorithm is used to find the parameters of a standard DTCNN architecture for a simple pattern recognition task.

1 Introduction

A Discrete-Time Cellular Neural Network (DTCNN) is a spin glass like architecture with parallel, zero-temperature dynamics, translationally invariant weights and only local interconnections. The DTCNN was introduced in [1] as a discrete-time version of the Cellular Neural Network (CNN) [2], and it is related to the work by Walter Little [3].

The operation of the DTCNN is determined by a set of real-valued network parameters. Finding these network parameters for a certain desired task is called learning. Two basically differing approaches can be found in the literature.

In one kind of learning, the detailed operation of the network is designed step by step by the designer. This is usually done by establishing local processing rules at cell level or by designing stable and unstable local subpatterns at each cell. For this reason, this kind of learning will be called *local* learning. Local learning algorithms find the network parameters that implement this

*on leave from: Institute of Network Theory and Circuit Design, Technical University Munich, Munich, Germany
†supported by a grant from the Ernst-von-Siemens Foundation
‡Institute of Network Theory and Circuit Design, Technical University Munich, Munich, Germany

desired, pre-designed behavior of the network, usually by maximizing a measure of robustness. The disadvantage of local learning algorithms is that the complexity of the trajectory depends on the ingenuity of the (human) designer. This might be tolerated if only simple tasks like image preprocessing are executed, where the main objective is to use the projected high processing speed of DTCNN architectures.

This work is entirely devoted to the second kind of learning algorithms, which is referred to as *global* learning. The term "global" is used here, because the task, which has to be learned by the network, is defined by a set of input images (input images of the whole network as opposed to local input images of one cell) and the corresponding desired output images of the network. The global learning algorithm is used to find the network parameters for this task, which implies that the algorithm itself designs the trajectory. Thus much more complicated trajectories are obtainable, and more complicated tasks can be implemented by the network. Unfortunately, global learning algorithms are computationally expensive.

The behavior of the DTCNN depends on the functional mapping, which is implemented by each cell of the network. For binary input images, the size of the solution space, i.e. the number of different mappings, is fixed. Thus the global learning problem can be seen as a combinatorial optimization problem. The theory on Linear Threshold Logic, which has been studied mainly in the mid Sixties, contains some valuable results on the properties and the geometry of the solution space.

Before trying to find an efficient learning algorithm, it makes sense to have a look at the complexity of the problem itself. Especially for problems from the class of NP-complete and NP-hard problems, it has not been possible so far to come up with algorithms which solve the problem in a polynomially bounded time. In this work, it is conjectured that the problem of deciding whether global learning for DTCNNs is feasible or not is NP-complete.

A cost function is introduced, which measures how well the network maps a set of input images onto the desired output images. Learning is thus achieved by minimizing the cost function. Since exact, non-polynomial algorithms cannot be applied for reasons discussed later in this work, one of the few remaining possibilities to solve the learning problem are algorithms of the Simulated Annealing type.

Simulated Annealing was introduced ten years ago as a flexible tool for finding approximate solutions to large combinatorial optimization problems. Since then, it has been used successfully by many researchers.

A Simulated Annealing algorithm is applied to the DTCNN learning problem, and the algorithm was able to come up with relatively good solutions for a simple pattern recognition problem. In this pattern recognition problem, a standard DTCNN is used to distinguish bitmap representations of handwritten "A"s and "B"s. The philosophy behind this application is to show that relatively complicated tasks can be done with an architecture as simple as a standard DTCNN, when the full power of the recurrence of the network is utilized. It should be kept in mind that chips with a sufficient size for the desired applications (e.g. 500×500 pixels for image processing tasks) will only be feasible during the next few years, if the architecture is kept as simple as possible.

Section 2 contains a short introduction to DTCNNs. In Section 3, some results from the theory on Linear Threshold Logic are used for a description of the solution space of the DTCNN learning

problem. The Learning Problem and the Feasibility Problem are formally introduced in Section 4, and the Feasibility Problem for DTCNNs is conjectured to be NP-complete. Section 5 gives a short introduction to Simulated Annealing algorithms, and Section 6 describes the results of applying Simulated Annealing to the DTCNN learning problem. The conclusions of this work are presented in Section 7.

2 Discrete-Time Cellular Neural Networks

The DTCNN is a first-order, discrete-time dynamical system consisting of M identical cells on a (usually) one- or two-dimensional cell grid $\mathcal{CG}$. Its operation is described by a state equation, an output equation, and the initial state:

$$\begin{aligned}
\text{state:} \qquad & x_c(k) = \sum_{d \in \mathcal{N}(c)} a_{d-c} y_d(k) + \sum_{d \in \mathcal{N}(c)} b_{d-c} u_d(k) + i \\
\text{output:} \qquad & y_c(k) = \mathrm{SGN}(x_c(k-1)) \\
& \quad = \begin{cases} 1 & \text{for } x_c(k-1) \geq 0 \\ -1 & \text{for } x_c(k-1) < 0 \end{cases} \\
\text{initial state:} \qquad & y_c(0) = y_{c,0}
\end{aligned} \tag{1}$$

k is a non-negative integer corresponding to the time step. $\mathrm{a} = (a_\nu)$ and $\mathrm{b} = (b_\nu)$ are the *templates* (weighted connections), i is the cell bias. The network parameters a, b and i are translationally invariant, i.e. they are identical for each cell in the network. The "$*_{d-c}$"-notation is used in a symbolical sense, since only the relative position of the two cells c and d with respect to each other, not their absolute location on the cell grid, is important for the value of the template coefficient $*_{d-c}$. $y_c(k)$ is the output of cell c at time step k, $u_c(k)$ is the input signal from the input layer corresponding to cell c at time step k. The input signals are taken from the input grid $\mathcal{IG}$, which has the same dimensions as the cell grid $\mathcal{CG}$.

$\mathcal{N}(c)$ is the *neighborhood* of cell c. For a r-neighborhood each template has the size $(2r+1)\times(2r+1)$. If both the a- and b-template have the same neighborhood size r, each active cell of the network has $N = 2(2r+1)^2$ inputs plus a separate bias input. If both the a- and b-template exist, then N will be even. The input signals are restricted to the interval $u_c(k) \in [-1, 1]$ or are assumed to be binary-valued ($u_c(k) \in \{-1, 1\}$). For some cells in the network, the neighborhood $\mathcal{N}(c)$ would contain cells outside the active cell grid. In order to satisfy the need for these additional cell output values and input values, the grid of active cells is surrounded by a ring of dummy cells, which have a constant output value ε. Let $\mathcal{C}_a$ denote the set of indices of the active cells, and $\mathcal{C}_d$ the set of indices of the dummy cells. Then $y_\mu(k) = \varepsilon$ and $u_\mu(k) = \varepsilon$ for $\mu \in \mathcal{C}_d$ (with $\varepsilon \in [-1, 1]$ or $\varepsilon \in \{-1, 1\}$) for all times k.

It has to be noted that the DTCNN parameters have one degree of freedom: multiplying all template coefficients and the bias with a positive constant will not affect the cell output. For this reason, a norm of all network parameters is usually fixed.

Properties like stability, boundedness of states etc. can be proven by using the existing theory for synchronously clocked Hopfield networks (see also [1]).

For the rest of this work we will make the following assumptions:

- binary and constant input signals, i.e. $u_c(k) = u_c \in \{-1, 1\}$.
- binary and constant dummy cells, i.e. $u_\mu, y_\mu \in \{-1, 1\}$ for $\mu \in C_d$.
- fixed strategy for setting the initial state, i.e. $y_c(0) = u_c$ or $y_c(0) = 1$ or $y_c(0) = -1$.
- fixed set of input patterns (training set).

3 Linear Threshold Logic Background

Under the above assumptions the operation of the network is completely determined by the way in which each cell maps its $N = 2 \cdot (2r+1)^2$ (binary) input signals onto its binary output. The cells are *clocked Linear Threshold Elements* (LTEs). A LTE with N binary inputs implements a *linearly separable Boolean function* $\mathcal{F}_{ls} : \{-1, 1\}^N \rightarrow \{-1, 1\}$. A Boolean function F is called linearly separable, if for N binary input signals $(e_1, \ldots, e_N) \in \{-1, 1\}^N$ there are N parameters $(p_1, \ldots, p_N) \in IR^N$ and a threshold $p_0 \in IR$ such that F can be written as

$$F(e_1, \ldots, e_N) = \begin{cases} +1 & \text{if } p_0 + \sum_{\nu=1}^{N} p_\nu e_\nu \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

Each of the possible 2^N binary input vectors corresponds to a row in a truth table, and as well to an inequality of a form as in (2). Let $y_\nu \in \{-1, 1\}$, $\nu = 1, \ldots, 2^N$ denote the desired output value of the νth row of the truth table.

It is convenient to write the state and the output equation (1) in a vector notation. To get rid of the bias, a constant "+1"-element is prepended as the 0th element to each input vector. Let N be the number of inputs of each cell. Then the indices of the elements of these augmented input vectors $\mathbf{e}$ run from 0 to N. We define the sets

$$\mathcal{E} := \{-1, 1\}^{N+1} \quad \text{with} \quad |\mathcal{E}| = 2^{N+1}$$

$$\mathcal{E}_0 := \{\mathbf{e} \in \mathcal{E} | e_0 = 1\} \quad \text{with} \quad |\mathcal{E}_0| = 2^N$$

Let $\mathbf{p} \in IR$ be a vector, which contains all template coefficients $\mathbf{p}^T = (p_0, \ldots, p_N) = (i, (a_\nu), (b_\nu))$ (in this order). The vector $\mathbf{e}_c^T(k) \in \mathcal{E}_0$ collects a constant term for the bias, the cell output signals, and the input signals (in this order) for cell c of the network at time k, i.e. $\mathbf{e}_c(k) = (1, (y_{\mathcal{N}(c)}(k)), (u_{\mathcal{N}(c)}))$.

Then the cell output of cell c at time $k+1$ can be written as

$$y_c(k+1) = \begin{cases} +1 & \text{if } \mathbf{p}^T \mathbf{e}_c(k) \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

Under the assumptions made at the end of the last section, the output pattern of a DTCNN can now be viewed as a functional of a linearly separable Boolean function F_{ls} of N inputs. Thus the operation of the DTCNN is defined on the set of all linearly separable Boolean functions of N inputs. The theory on Linear Threshold Logic can provide interesting properties of this search space. A good introduction to Linear Threshold Theory are [4], [5], and [6].

3.1 Cardinality of the Search Space

It is important to note here that the number of all possible linearly separable Boolean functions of N inputs can be very large, but it is a finite number. Thus there is only a finite number of different network behaviors, even if the real-valued parameter vector $\mathbf{p}$ suggests something else. The problem of optimizing the output behavior of the DTCNN can therefore be seen as an combinatorial optimization problem. We will elaborate on this fact in Sections 4 and 5.

A very coarse upper bound on the number of all linearly separable Boolean functions is the number of all possible Boolean functions of N inputs, which is 2^{2^N}. This bound is by far too pessimistic, since especially for large N, the number of linear separable Boolean functions is only a small fraction of the number of all possible Boolean functions.

Different upper bounds on R_{ls}^N, the number of linearly separable Boolean functions of N inputs, can be found in the literature. The classical result by Cover [7] on the number of dichotomies of m vectors in IR^N does not apply here, because it requires, that the m vectors are in general position. The tightest bound is due to Winder, and it can be found in the appendix of [4].

$$R_{ls}^N \leq 2 \sum_{k=0}^{N} \binom{2^N - 1}{k} \leq \frac{2^{N^2+1}}{N!}$$

Even for simple problems like DTCNNs with a 1-neighborhood in a- and b-template ($N+1 = 19$ network parameters), the number of linearly separable Boolean functions is approximately $2 \cdot 10^{82}$, and for a 2-neighborhood in a- and b-template ($N+1 = 51$ network parameters), this number is approximately $3 \cdot 10^{688}$.

Hence the cardinality of the problem rises with the order $O(\frac{2^{N^2}}{N!})$, which is worse than the cardinality of many classical combinatorial optimization problems like for example the Traveling Salesman Problem [8].

3.2 Geometric Interpretation

Linearly separable Boolean functions of N inputs have a very figurative geometric interpretation. In the *parameter space* approach, each parameter vector $\mathbf{p}$ corresponds to a point in IR^{N+1}. Consequently, the 2^N equations $\mathbf{e}^T\mathbf{p} = 0$ with $\mathbf{e} \in \mathcal{E}_0$ define 2^N hyperplanes through the origin of IR^{N+1}. Each of the augmented input vectors $\mathbf{e} \in \mathcal{E}_0$ corresponds to the normal vector of a hyperplane. The binary-valued response of the cell to this specific input vector depends on which side of the corresponding hyperplane the point $\mathbf{p}$ is located (either $\mathbf{e}^T\mathbf{p} \geq 0$ or $\mathbf{e}^T\mathbf{p} < 0$).

The space that is dual to the parameter space IR^{N+1} is the *input space* $IR^{N+1,*}$. In this dual space, hyperplanes in the primal space correspond to points in the dual space and vice versa (the null vector is an exception). The 2^N augmented input vectors $\mathbf{e} \in \mathcal{E}_0$ correspond to one half of the vertices of a binary $(N+1)$-cube in IR^{N+1}. The equation $\mathbf{p}^T\mathbf{e} = 0$ defines a hyperplane through the origin of the dual space, and the vector $\mathbf{p}$ is orthogonal to this hyperplane. The hyperplane divides the set of vertices into two disjoint sets, one set, for which $\mathbf{p}^T\mathbf{e} \geq 0$, and a set with $\mathbf{p}^T\mathbf{e} < 0$.

Each of the 2^N hyperplanes through the origin of the parameter space divides IR^{N+1} into two half spaces, both of them convex sets. The intersection of a finite number of (closed) half spaces is called a *convex cone* $\mathcal{C}_\mathbf{p}$:

Definition 3.1 *Let* $\mathbf{p} \in IR^{N+1}$ *so that* $\mathbf{p}^T\mathbf{e} \neq 0 \ \forall \mathbf{e} \in \mathcal{E}$. *Then the* convex cone $\mathcal{C}_\mathbf{p}$ *is defined by*

$$\mathcal{C}_\mathbf{p} = \left\{ x \in IR^{N+1} \mid \mathbf{e}^T\mathbf{p} \cdot \mathbf{e}^T\mathbf{x} > 0 \ \forall \mathbf{e} \in \mathcal{E}_0 \right\}$$

The mapping from the set of all convex cones $\mathcal{C}_\mathbf{p}$ to the set of all possible linearly separable Boolean functions is injective, but the mapping from the parameter space IR^{N+1} onto the set of all possible linearly separable Boolean functions is not, since each linearly separable Boolean function can be realized by infinitely many parameter vectors $\bar{\mathbf{p}} \in \mathcal{C}_\mathbf{p}$.

The following is easy to show:

Lemma 3.1 *Assume* K *points* $\mathbf{x}_k \in \mathcal{C}_\mathbf{p}$. *Then every positive linear combination of the* $\mathbf{x}_k$ *is in* $\mathcal{C}_\mathbf{p}$:

$$\mathbf{x}_k \in \mathcal{C}_\mathbf{p}, \quad (k = 1, 2, \ldots, K) \Rightarrow \left(\sum_{k=1}^{K} \lambda_k \mathbf{x}_k\right) \in \mathcal{C}_\mathbf{p} \quad \forall \ \lambda_k \in IR, \ \lambda_k \geq 0$$

Proof:

$$\mathbf{e}^T\mathbf{p} \cdot \mathbf{e}^T\left(\sum_{k=1}^{K} \lambda_k \mathbf{x}_k\right) = \sum_{k=1}^{K} \underbrace{\lambda_k}_{\geq 0} \underbrace{\mathbf{e}^T\mathbf{p} \cdot \mathbf{e}^T\mathbf{x}_k}_{>0} > 0 \quad \forall \ \mathbf{e} \in \mathcal{E}_0$$

□

Remark: Convexity can be confirmed by choosing $K = 2$ and $\lambda_1 = \gamma, \lambda_2 = (1 - \gamma), \gamma \in [0, 1]$.

From the geometrical interpretation it will be clear that not all of the possible convex cones are limited by the maximum number of 2^N hyperplanes. In other words, only a reduced subset $\mathcal{E}_0^r \subseteq \mathcal{E}_0$ will be necessary to define a Boolean mapping, and thus only a reduced set of inequalities will be needed.

We introduce the reduced subset $\mathcal{E}_0^r$ and its complement $\mathcal{E}_0^{\bar{r}}$, so that $\mathcal{E}_0^r \cap \mathcal{E}_0^{\bar{r}} = \emptyset$ and $\mathcal{E}_0^r \cup \mathcal{E}_0^{\bar{r}} = \mathcal{E}_0$. To simplify the proof for the next theorem, we need the following Lemma:

Lemma 3.2 (Farkas' Lemma): *Let K be a positive integer, and let $\mathbf{c}, \mathbf{y}, \mathbf{a}_1, \ldots, \mathbf{a}_K \in IR^D$, $\mathbf{y} \neq \mathbf{0}$ and $\mathbf{c} \neq \mathbf{0}$, then:*

$$(\mathbf{y}^{\mathrm{T}}\mathbf{a}_i \geq 0 \;\Rightarrow\; \mathbf{y}^{\mathrm{T}}\mathbf{c} \geq 0) \iff \mathbf{c} \in \{\mathbf{x} \in IR^D \mid \mathbf{x} = \sum_i \pi_i \mathbf{a}_i,\; \pi_i \in IR,\; \pi_i \geq 0\}$$

Proof: See for example [9]. □

Now we are ready to establish a result on $\mathbf{e} \in \mathcal{E}_0^{\bar{r}}$: Let as before y_μ be the desired binary output value of a cell, when the input $\mathbf{e}_\mu$ is applied to the cell inputs. Assuming that $\mathbf{e}_\mu^{\mathrm{T}}\mathbf{p} \neq 0$ for all $\mathbf{e}_\mu \in \mathcal{E}_0$, we have $y_\mu \mathbf{e}_\mu^{\mathrm{T}}\mathbf{p} > 0$. Then

Theorem 3.1

$$\mathbf{e}_\nu \in \mathcal{E}_0^{\bar{r}} \iff \exists \lambda_\nu \in IR,\; \lambda_\nu \geq 0 \text{ so that } \mathbf{e}_\nu = \sum_\mu \lambda_\mu y_\nu y_\mu \mathbf{e}_\mu \;\forall \mathbf{e}_\mu \in \mathcal{E}_0 \backslash \{\mathbf{e}_\nu\}$$

Hence a hyperplane defined by its normal vector $\mathbf{e}_\nu$ is redundant, if it can be written as a positive linear combination of the vectors $y_\nu y_\mu \mathbf{e}_\mu$, $\mathbf{e}_\mu \in \mathcal{E}_0 \backslash \{\mathbf{e}_\nu\}$.

Proof: Using Lemma 3.2 and keeping in mind that $\mathbf{p}$ is such that $\mathbf{e}^{\mathrm{T}}\mathbf{p} \neq 0$ for all $\mathbf{e} \in \mathcal{E}_0$ the proof becomes really straightforward by substituting

$$\mathbf{y} \rightarrow \mathbf{p},\; \mathbf{c} \rightarrow y_\nu \mathbf{e}_\nu \text{ with } \mathbf{e}_\nu \in \mathcal{E}_0 \quad \text{and} \quad \mathbf{a}_i \rightarrow y_\mu \mathbf{e}_\mu \text{ with } \mathbf{e}_\mu \in \mathcal{E}_0 \backslash \{\mathbf{e}_\nu\}$$

□

A tight upper bound can be given for the size of the set $\mathcal{E}_0^r$:

Lemma 3.3

$$|\mathcal{E}_0^r| \leq 2^N$$

Proof: Since $\mathcal{E}_0^r \subseteq \mathcal{E}_0$ and $|\mathcal{E}_0| = 2^N$, it is obvious that $|\mathcal{E}_0^r| \not> 2^N$.

Equality will by shown by providing an example for which $\mathcal{E}_0^{\bar{r}} = \emptyset$. Choose for example $\mathbf{p} = (1, 0, \ldots, 0)^{\mathrm{T}}$. In this case, all vectors $y_\mu \mathbf{e}_\mu$ with $\mathbf{e}_\mu \in \mathcal{E}_0$ will have "+1" as their first component. Any positive linear combination of these vectors, i.e.

$$\sum_\mu \pi_\mu y_\mu \mathbf{e}_\mu\,, \quad \pi_\mu \geq 0$$

will give for the first component of the sum

$$\sum_{\mu} \pi_{\mu} = 1$$

Since for any other element but the first element the summation will contain "+1"-entries as well as "−1"-entries, it is impossible to obtain a binary vector as the result of the summation, and thus no element of $\{y_\mu \mathbf{e}_\mu \mid \mathbf{e}_\mu \in \mathcal{E}_0\}$ can be written as a linear combination of the other elements. Thus $|\mathcal{E}_0^{\bar{r}}| = 0$ in this case, and since $\mathcal{E}_0^r$ and $\mathcal{E}_0^{\bar{r}}$ are complementary with respect to $\mathcal{E}_0$, $|\mathcal{E}_0^r| = 2^N$.

It only has to be shown that there are cases in which $\mathcal{E}_0^{\bar{r}} \neq \emptyset$. This will again be done by an example. Choose $\mathbf{p} = (1.01, 1, \ldots, 1)^{\mathrm{T}}$. Then $\{y_\mu \mathbf{e}_\mu\}$ is the set of all binary vectors with more "+1" elements than "−1" elements plus all binary vectors vectors with an equal number of "+1" and "−1" elements, where the first element is "+1". This set of vectors contains for $N \geq 3$ all $N + 1$ vectors with N "+1" elements and one "−1" element. Adding these vectors and dividing the sum by $N - 2$ will result in a vector consisting of only "+1" elements, which belongs to $\{y_\mu \mathbf{e}_\mu\}$ as well. Thus the vector $\mathbf{e} = (1, \ldots, 1)^{\mathrm{T}}$ belongs to the set $\mathcal{E}_0^{\bar{r}}$. It can be shown that $|\mathcal{E}_0^{\bar{r}}|$ is even larger for $N > 3$.

This completes the proof of the theorem. □

Remark: It has been shown that a convex cone $\mathcal{C}_\mathbf{p}$ can be limited by a number of hyperplanes which is exponential in N. It still remains an open problem whether there are cases in which the number of hyperplanes is not exponential in N.

3.3 Accuracy

For any realization of a DTCNN, the actual (physical) network parameters will deviate from the nominal parameters. Let $\mathbf{p}$ denote the nominal parameter vector, $\hat{\mathbf{p}}$ the actual parameter vector, and $\Delta\mathbf{p} = \mathbf{p} - \hat{\mathbf{p}}$ the error. The correct operation of the network can only be guaranteed as long as $\hat{\mathbf{p}} \in \mathcal{C}_\mathbf{p}$, where $\mathcal{C}_\mathbf{p}$ is defined as in Subsection 3.2.

Due to the corrupted parameter vector $\hat{\mathbf{p}}$, the nominal cell state $x_c(k)$ will become $\hat{x}_c(k) = \hat{\mathbf{p}}^{\mathrm{T}} \mathbf{e}_c(k)$. Thus we have

$$\hat{\mathbf{p}} \notin \mathcal{C}_\mathbf{p} \Longleftrightarrow \exists c, k \text{ so that } \hat{x}_c(k) \cdot x_c(k) < 0$$

and in the limiting case

$$\exists c, k \text{ so that } \hat{x}_c(k) = 0$$

3.3.1 Worst-Case Results

Assume that the network parameters $\{p_n\}$ can be realized with a guaranteed error limit $\Delta p_n^{\max}$, where $|\Delta p_n| \leq \Delta p_n^{\max}$. Assume for simplicity that $\Delta p_n^{\max} = \alpha_1 |p_n|$ for all $n = 0, \ldots, N$ and a real-valued, positive *accuracy* α_1.

In the worst-case scenario, the network will still be performing correctly if

$$|\hat{x}_c(k) - x_c(k)| < x_{\min} \quad \forall c, k \tag{4}$$

where

$$x_{\min} := \min_{\mathbf{e} \in \mathcal{E}_0} \left\{ \left| \mathbf{e}^{\mathrm{T}} \mathbf{p} \right| \right\} \tag{5}$$

$x_{\min}$ is the minimum absolute value for any cell state at any time. It is linked to $d_{\min}$, the euclidian distance of the parameter vector $\mathbf{p}$ from the nearest hyperplane in the parameter space, by $x_{\min} = \sqrt{N+1} \cdot d_{\min}$. Thus

$$\begin{aligned} |\hat{x}_c(k) - x_c(k)| &= \left| \sum_{n=1}^{N} e_n(\hat{p}_n - p_n) + e_0(\hat{p}_0 - p_0) \right| \leq \sum_{n=1}^{N} |e_n| \cdot |\hat{p}_n - p_n| + |e_0| \cdot |\hat{p}_0 - p_0| \leq \\ &\leq \max_n \{e_n\} \cdot \sum_{n=0}^{N} |\hat{p}_n - p_n| = \|\mathbf{e}\|_\infty \cdot \sum_{n=0}^{N} \Delta p_n^{\max} = \alpha_1 \cdot \|\mathbf{e}\|_\infty \sum_{n=0}^{N} |p_n| = \\ &= \alpha_1 \cdot \|\mathbf{e}\|_\infty \cdot \|\mathbf{p}\|_1 \end{aligned}$$

Hence a sufficient condition for the accuracy α_1 of the network parameters follows from (4) as

$$\alpha_1 < \frac{x_{\min}}{\|\mathbf{e}\|_\infty \cdot \|\mathbf{p}\|_1} =: r_w(1, \mathbf{p}) \tag{6}$$

Note that in this last expression, $r_w(1, \mathbf{p})$ is exactly the *(relative) robustness in weight space with respect to the 1-norm* as defined by Nachbar in [10]. In the case of binary inputs, $\|\mathbf{e}\|_\infty = 1$.

3.3.2 Accuracy for Network Parameters with a Gaussian Distribution

The above paragraphs gave a worst-case analysis of the error. For a real-world fabrication process, it is more realistic to assume that the network parameters are random variables with a Gaussian probability distribution.

Let $\chi_0, \ldots, \chi_N$ be $N+1$ mutually independent random variables with Gaussian probability distributions characterized by the expected values $E_{\chi_n} = p_n$ (the nominal network parameters) and standard deviations $\sigma_{\chi_n} = \sigma_n$.

Let $p(\chi_0, \ldots, \chi_N)$ be the joint probability density function of the random variables. Due to the mutual independence of the random variables, we can write

$$p(\chi_0, \ldots, \chi_N) = \frac{1}{\sqrt{2\pi}^{N+1} \sigma_0 \cdots \sigma_N} \cdot e^{-\frac{1}{2}\left(\left(\frac{\chi_0 - p_0}{\sigma_0}\right)^2 + \cdots + \left(\frac{\chi_N - p_N}{\sigma_N}\right)^2 \right)} \tag{7}$$

Now let P_{ok} the probability that the network is performing correctly. Then

$$P_{\mathrm{ok}} = \underbrace{\int \cdots \int}_{c_{\mathbf{p}}} p(\chi_0, \ldots, \chi_N) \, d\chi_0 \cdots d\chi_N$$

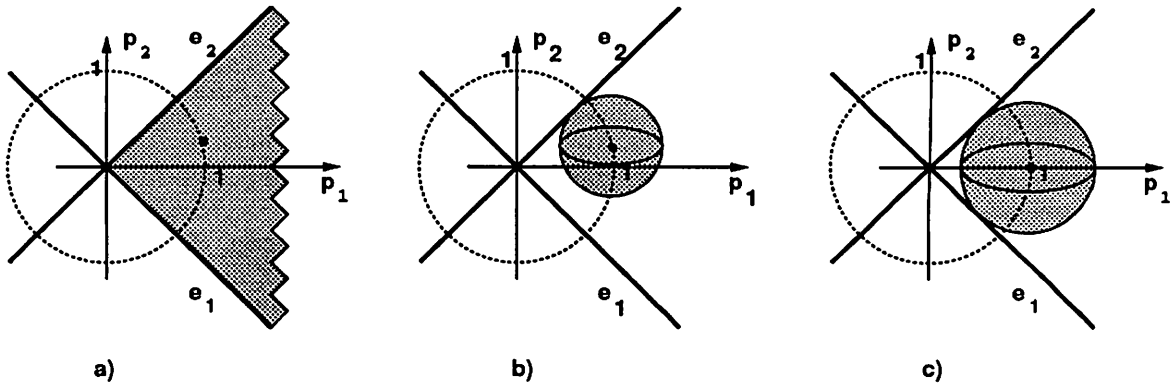

Figure 1: Parameter Space for $N+1=2$: a) convex cone; b) inscribed norm bodies $\mathcal{NB}_1(\mathbf{p},d)$ (shaded circle) and $\mathcal{NB}_2(\mathbf{p},d)$ (shaded ellipsoid); c) optimum inscribed norm bodies

Fig. 1a shows the two-dimensional parameter space for the simple case $N=1$. The shaded area corresponds to the chosen convex cone. In practice, the integral is very difficult to evaluate, because the number of hyperplanes defining the convex cone $\mathcal{C}_\mathbf{p}$ can increase exponentially with N (see Lemma 3.3). Still, it is possible to give a lower bound on P_{ok}. Let $\sigma_{\max} := \max\{\sigma_n\}$ be the maximum standard deviation. We introduce the norm bodies $\mathcal{NB}_1(\mathbf{p},d)$ and $\mathcal{NB}_2(\mathbf{p},d)$

$$\mathcal{NB}_1(\mathbf{p},d) = \left\{ \bar{\mathbf{p}} \in IR^{N+1} \;\middle|\; \sum_{n=0}^{N} (\bar{p}_n - p_n)^2 \le d^2 \right\}$$

$$\mathcal{NB}_2(\mathbf{p},d) = \left\{ \bar{\mathbf{p}} \in IR^{N+1} \;\middle|\; \sum_{n=0}^{N} \frac{(\bar{p}_n - p_n)^2}{\sigma_n^2} \le \frac{d^2}{\sigma_{\max}^2} \right\}$$

$\mathcal{NB}_1(\mathbf{p},d)$ is a hypersphere and $\mathcal{NB}_2(\mathbf{p},d)$ is a hyper-ellipsoid. $\mathcal{NB}_2(\mathbf{p},d)$ has to be introduced, because a closed-form expression can be given for the integration of (7) over $\mathcal{NB}_2(\mathbf{p},d)$. It is obvious that (with $d_{\min} = \sqrt{N+1}^{-1} x_{\min}$, the distance of of the nominal parameter vector $\mathbf{p}$ from the nearest hyperplane)

$$\mathcal{C}_\mathbf{p} \supset \mathcal{NB}_1(\mathbf{p}, d_{\min}) \supseteq \mathcal{NB}_2(\mathbf{p}, d_{\min})$$

Fig. 1b shows both norm bodies for $d = d_{\min}$. Therefore we have

$$P_{\text{ok}} > \underbrace{\int \cdots \int}_{\mathcal{NB}_1(\mathbf{p},d_{\min})} p(\chi_0,\ldots,\chi_N)\, d\chi_0 \cdots d\chi_N \ge \underbrace{\int \cdots \int}_{\mathcal{NB}_2(\mathbf{p},d_{\min})} p(\chi_0,\ldots,\chi_N)\, d\chi_0 \cdots d\chi_N \qquad (8)$$

The right integral in this last expression can be evaluated. The derivation is given in Appendix B. We get

$$P_{\text{ok}} > P_N\left(\frac{d_{\min}}{\sigma_{\max}}\right) := \operatorname{erf}\left(\frac{d_{\min}}{\sqrt{2}\sigma_{\max}}\right) - \sqrt{\frac{2}{\pi}} \cdot e^{-\frac{d_{\min}^2}{2\sigma_{\max}^2}} \sum_{l=0}^{\frac{N}{2}-1} \frac{\left(\frac{d_{\min}}{\sigma_{\max}}\right)^{2l+1}}{1 \cdot 3 \cdot \ldots \cdot (2l+1)} \qquad (9)$$

The "erf(*)"-function is the Gaussian error integral

$$\mathrm{erf}(z) := \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$$

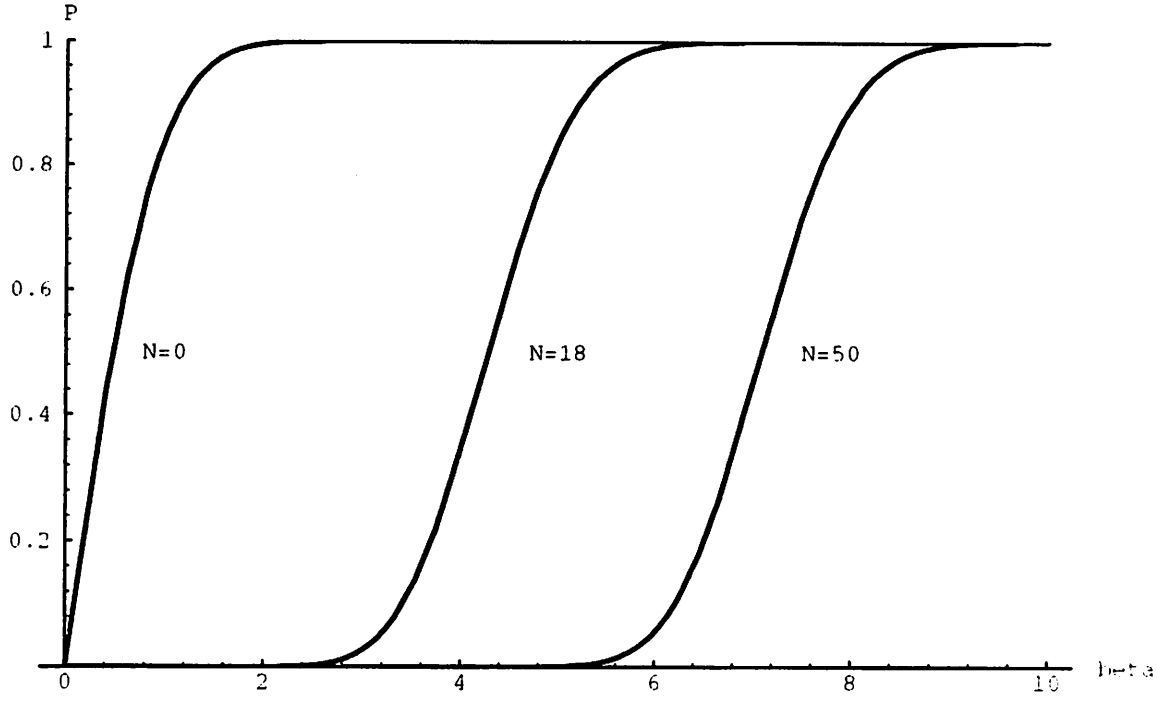

Figure 2: Probability $P_N\left(\frac{d_{\min}}{\sigma_{\max}}\right)$

For $N = 0$, expression (9) reduces to the standard Gaussian error integral. Fig. 2 shows $P_N\left(\frac{d_{\min}}{\sigma_{\max}}\right)$ for values of $N = 0$ (standard Gaussian error integral), $N = 18$ (DTCNN with 1-neighborhood) and $N = 50$ (2-neighborhood). Assuming that $\sigma_n = \alpha_2 \cdot |p_n|$ and thus $\sigma_{\max} = \alpha_2 \cdot \|\mathbf{p}\|_\infty$, we finally get a lower bound on P_{ok}, the probability that the network is working:

$$P_{\mathrm{ok}} > P_N\left(\frac{d_{\min}}{\alpha_2 \cdot \|\mathbf{p}\|_\infty}\right) \tag{10}$$

3.3.3 Finding the Nearest Hyperplane

In the results of the two previous subsections, both (6) and (10) require the knowledge of the distance of the parameter vector $\mathbf{p}$ from the nearest hyperplane. Even the appearently simple problem of determining the nearest hyperplane $\mathbf{e}_{\mathbf{p}}^{\min}$ is difficult:

Theorem 3.2 *The problem of determining whether there is a hyperplane $\mathbf{e}_{\mathbf{p}} \in \mathcal{E}_0$ for a given parameter vector $\mathbf{p}$ with rational elements, which obeys*

$$\left|\mathbf{p}^{\mathrm{T}}\mathbf{e}_{\mathbf{p}}\right| \leq \epsilon$$

where $\epsilon > 0$ is a rational constant, is NP-complete.

For an explanation of the most essential terms regarding complexity theory, refer to Appendix A.

Note that in order to find out whether a certain hyperplane $\mathbf{e}_{\mathbf{p}}^{0} \in \mathcal{E}_0$ is the nearest hyperplane from the parameter vector $\mathbf{p}$, one has to decide whether there is another hyperplane with $\left|\mathbf{p}^{\mathrm{T}}\mathbf{e}_{\mathbf{p}}\right| \leq \left|\mathbf{p}^{\mathrm{T}}\mathbf{e}_{\mathbf{p}}^{0}\right|$.

Proof: see Appendix C.

Note that the polynomial transformation that is given in Appendix C works both ways, i.e. the two problems are polynomially equivalent. Thus it is possible to reduce the problem to an instance of the Value-Independent Knapsack problem.

Even if Knapsack problems are NP-complete, there are pseudo-polynomial algorithms that work quite well in practice, e.g. the family of exact non-polynomial algorithms. Especially Branch-and-Bound methods or Cutting-Plane algorithms (see for example [8] or [11] for an overview) have been used successfully. These algorithms seem to work quite well, if the correlation between the weights $s(u)$ and the values $v(u)$ (see Appendix C) is only weak.

In the case of Theorem 3.2 we are dealing with an instance of the Value-Independent Knapsack problem (or *Stickstacking* Problem). Experimental results [12] suggest, that in this case the performance of traditional approaches to solve the problem breaks down completely. Still, there are attempts to solve this problem. The algorithm in [12] is claimed to reduce the effective problem size from 2^K to $2^{\frac{K}{2}}$.

If the DTCNN is simulated on a computer, it is fairly easy to determine the nearest hyperplane. In this case, the cell states have to be computed anyway, and it is computationally cheap to keep track of the minimum of the absolute values of the cell states.

In a usual setup, not all possible 2^N cell input patterns will occur due to regularities in the training set or the interconnection structure of the DTCNN. Let $\hat{\mathcal{E}}_0 \subseteq \mathcal{E}_0$ denote the set of all cell input patterns $\mathbf{e}$, that occur for any cell at any time k for a specific training set (for the example in Section 6 it turned out that only 7100 out of the $2^N = 262144$ possible input patterns occurred at all). After a simulated run of the network on a computer, both $\hat{\mathcal{E}}_0$ and $x_{\min}$ are known.

3.3.4 A Lower Bound on $d_{\min}$

Using results from Linear Threshold Logic Theory, it is possible to come up with a lower bound on $d_{\min}$ depending on N.

Assume that $x_{\min}$ in (5) is strictly positive. Define

$$\tilde{p}_i := \frac{p_i}{x_{\min}} \qquad i = 0, 1, \ldots, N$$

Then we have

$$\left|\mathbf{e}^{\mathrm{T}}\tilde{\mathbf{p}}\right| \geq 1$$

Using Muroga's terminology, this corresponds to the *normalized system of inequalities in a majority expression* (Definition 3.2.2 in [6]). For any positive, Boolean, linearly separable function of N variables, a bound on the sum of the weights can be given (Theorem 9.3.2.2 in [6]):

$$0 \leq \sum_{i=0}^{N} \tilde{p}_i \leq 2(N+1)\left(\frac{N+1}{4}\right)^{\left(\frac{N+1}{2}\right)}$$

Since any linearly separable Boolean function can be converted into a positive linearly separable Boolean function by just flipping the sign of input variables, the above result is also true for the general case. Thus we can write

$$\sum_{i=0}^{N} |\tilde{p}_i| = \|\tilde{\mathbf{p}}\|_1 \leq 2(N+1)\left(\frac{N+1}{4}\right)^{\left(\frac{N+1}{2}\right)} \tag{11}$$

Plugging these results into (6), we get a lower bound on the relative robustness in weight space

$$r_w(1,\mathbf{p}) = \frac{x_{\min}}{\|\mathbf{e}\|_\infty \cdot \|\mathbf{p}\|_1} = \frac{1}{\|\mathbf{e}\|_\infty \cdot \|\tilde{\mathbf{p}}\|_1} \geq \frac{1}{2(N+1)}\left(\frac{N+1}{4}\right)^{-\left(\frac{N+1}{2}\right)} \tag{12}$$

For all accuracies $\alpha_1 < r_w(1,\mathbf{p})$, the network is guaranteed to perform correctly. Expression (12) confirms that the number of bits required to store the weights is of the order $O(N \log N)$ (see also [13]). For a 1-neighborhood, 27 bits are needed to be able to realize any linearly separable Boolean function (worst-case results!).

Remark: Note that optimum robust solution for a given problem is unique. Therefore, the optimum parameter vector is defined by $N+1$ linearly independent input patterns $\mathbf{e}_\nu \in \mathcal{E}_0$. Even if the Boolean function, which has to be realized by the cell, is not completely specified, the worst-case accuracy requirements will occur, if all these $N+1$ patterns $\mathbf{e}_\nu$ appear at any time step and any cell of the network. This can be achieved for example by setting the input signals u_c and the initial state $y_0(k)$ in a way that the desired patterns occur.

3.3.5 Practical Optimization of the Robustness

In the last three sections it became intuitively clear that the parameter vector $\mathbf{p}_{\text{opt}}$ in a convex cone $\mathcal{C}_\mathbf{p}$ has to be chosen in such a way that the nearest hyperplane is as far away as possible, while a norm of the parameter vector is kept constant. This case is shown in fig. 1c. The problem of finding the $\mathbf{p}_{\text{opt}} \in \mathcal{C}_\mathbf{p}$ that maximizes

$$\max_{\mathbf{p}_{\text{opt}} \in \mathcal{C}_\mathbf{p}} \left\{ \min_{\mathbf{e} \in \mathcal{E}_0} \left\{ |\mathbf{e}^{\mathrm{T}} \mathbf{p}_{\text{opt}}| \right\} \right\} \quad \text{and} \quad \|\mathbf{p}_{\text{opt}}\| = 1 \tag{13}$$

is the well-known *perceptron of optimum stability* problem, and several authors have studied this problem (see for example [14]). Since $\mathbf{p}$ (which defines the convex cone $\mathcal{C}_\mathbf{p}$) is known, the set of vectors $\mathbf{e}_\mu \in \hat{\mathcal{E}}_0$ with desired output responses y_μ must be linearly separable.

Feeding the $\mathbf{e}_\mu \in \hat{\mathcal{E}}_0$ and y_μ into one of the algoritms solving (13), the optimal position of $\mathbf{p}_{\text{opt}} \in \mathcal{C}_\mathbf{p}$ can be computed.

Remark 1: This strategy will only optimize the position of the parameter vector $\mathbf{p}_{\text{opt}}$ inside one given convex cone. It is still possible, that there are other cones with low objective function values, for which the robustness is higher. Due to the complexity of the problem of finding these cones, no easy analytical approach seems possible. One way of incorporating robustness into an objective function (c.f. Section 4.3) would be by adding a term $o_1(\min\{|\mathbf{e}^{\mathrm{T}}\mathbf{p}|\})$ to the objective function, which punishes positions of $\mathbf{p}$ near hyperplanes. As said before, this term can be obtained rather cheaply, if the system is simulated on a computer. The price which has to be paid for this additional term is the introduction of many new local minima into the objective function, which will further complicate the optimization problem.

Remark 2: It has to be mentioned as well that the value of the optimized minimum distance from a hyperplane $d_{\min}^{\text{opt}}$ depends on N as well. In [13], the proposition is proved that for an N input linear threshold element, the number of bits needed to code the weights and the threshold is of the order $O(N \log N)$. This implies that the accuracy requirements for the weights will increase exponentially with N. This is a point where the DTCNN with its few inputs compares favorably to the Hopfield network, where the number of inputs of each cell is at least an order of magnitude larger.

4 Learning for DTCNNs

In this section we will define the Feasibility Problem and the Learning Problem, using the terminology of Judd [15]. Let M be the number of active cells in the network. For the *Supervised Learning Paradigm*, a number of input patterns $\mathbf{u}^{[l]} \in [-1,1]^M$ (*stimuli*) are fed into the network. In some cases, the stimuli are restricted to be binary-valued ($\mathbf{u}^{[l]} \in \{-1,1\}^M$). Each stimulus has exactly one corresponding desired binary-valued output pattern $\mathbf{d}^{[l]} \in \{-1,*,1\}^M$ (*response*), where don't care values ("$*$") are possible. Each stimulus/response pair is called an *item.*

The objective of the network in the learning phase is to associate each stimulus, which is presented during the training phase, with its response. After the training phase, the network ideally responds to each stimulus with the corresponding desired response. A *task* is a set of L items that the network has to learn, where usually $L \ll 2^M$.

Let $\mathcal{A}$ define the architecture consisting of the cell grid of active cells, the dummy cells at the border of the active cell grid, the interconnection structure (neighborhood), and the strategy of setting the initial state.

Let $\mathcal{F}$ define the mapping $\mathcal{F} : [-1,1]^N \longrightarrow \{-1,1\}$ (or $\mathcal{F} : \{-1,1\}^N \longrightarrow \{-1,1\}$ in the binary input case) that is performed by each cell in the network at each time step.

4.1 Feasibility Problem and Learning Problem

Using the above definitions, the mapping $\mathcal{M}_{\mathcal{F}}^{\mathcal{A}} : [-1,1]^M \longrightarrow \{-1,1\}^M$ describes the response of the network. A network output $\mathcal{M}_{\mathcal{F}}^{\mathcal{A}}(\mathbf{u}^{[l]})$ (i.e. the response of the network, if stimulus $\mathbf{u}^{[l]}$ is applied to its input) *agrees* with the response $\mathbf{d}^{[l]}$, if all corresponding elements are the same (or a don't care element appears in the response).

Then the *Feasibility Problem* can be defined as follows:

Definition 4.1 : Feasibility Problem (FP):
INSTANCE: Given is a network architecture $\mathcal{A}$.
QUESTION: Is there a mapping $\mathcal{F}$ so that $\mathcal{M}_{\mathcal{F}}^{\mathcal{A}}(\mathbf{u}^{[l]})$ agrees with $\mathbf{d}^{[l]}$ for all $l = 1, \ldots, L$?

The *Learning Problem* is defined by:

Definition 4.2 : Learning Problem (LP):
INSTANCE: Given is a network architecture $\mathcal{A}$.
QUESTION: For which mapping $\mathcal{F}$ will $\mathcal{M}_{\mathcal{F}}^{\mathcal{A}}(\mathbf{u}^{[l]})$ agree with $\mathbf{d}^{[l]}$ for all $l = 1, \ldots, L$?

Obviously, LP is harder than FP, because each solution of LP will by definition solve FP, but not vice versa.

4.2 Complexity of Learning in Neural Networks

Before trying to find learning algorithms for the DTCNN problem, it makes sense to look at the complexity of the problem. A good introduction to the terminology and related complexity theoretical concepts is the well-written book by Garey and Johnson [16]. The most important terms are explained in Appendix A. The crucial question is, whether it is possible to find an algorithm that will solve the Learning Problem in polynomial time (class P) or not.

Different authors have studied some special neural network architectures, and they were able to come up with proofs of NP-completeness for these types of networks.

The most general result is due to Judd [17], where he claims that FP for the general case is NP-complete. This implies, that, unless P=NP, there is no polynomial-time algorithm that - given *any* network architecture, node function set, and a task - will solve FP in this general case. This is an important, but not very surprising fact. It still has to be studied, though, whether there are networks with restricted architectures or restricted tasks, where FP and LP are solvable in polynomial time. A few results on certain feedforward architectures can be found in the literature:

One of the first references is [18], where it was shown that FP for a multilayer perceptron, whose Boolean mapping $\mathcal{F}$ is restricted to the class of SAFns (Single And Function), is NP-complete. The class of SAFns are those Boolean functions which can be realized by one multi-input AND

gate with optional inverters at its inputs and its output. The proof is done by a reduction of the problem from the SAT (SATISFIABILITY) problem (see [16]).

Judd shows in [15], that FP is NP-complete for *shallow* (feedforward) networks. "Shallow" in this respect means that the (depth of the network (number of layers) / FAN-IN of the cells / realizable Boolean mappings) or a trade-off between these three quantities is bounded, while the width of the network is unbounded.

In [19], the examined architecture consists of a feedforward network with two hidden layer nodes and one output node. The two hidden layer nodes are connected to each of the n inputs, and the output node is connected only to the output of each hidden layer node. Each node implements a Linear Threshold Element (LTE), which realizes a linearly separable Boolean function. It is shown that the FP is NP-complete, even when the two hidden layer nodes have the same weights, but different thresholds.

k-cascade neural nets are examined in [20]. They have n (binary) inputs, and they consist of k LTEs, where the first LTE has n inputs, and the remaining LTEs have $n+1$ inputs. The inputs of the first LTE are connected to the n inputs of the network. The inputs of the second LTE are connected to the n inputs of the network and to the output of the first LTE. The third LTE is connected to the inputs and to the output of the second LTE, and so on. It is shown that the FP for 2-cascade nets is NP-complete, even if the weight vector of the first cell is the negative of the weight vector (the part corresponding to the n inputs) of the second cell. It is conjectured, that FP for the k-cascade neural network architecture is NP-complete.

The result closest to a DTCNN architecture is the *columnar grid* architecture in [17]. It is shown there that loading shallow architectures with grid SCI (Support Cone Interaction) graphs is NP-complete. Note that this architecture consists of locally interconnected cells on a two-dimensional grid, but it is still purely feedforward, and it admits different node functions for each cell.

We will now focus our attention on the DTCNN, and we introduce the *DTCNN Feasibility Problem* (DFP). Let as before N be the number of inputs of each cell, M the number of cells in the network, and L the number of stimulus/response pairs. Then we define:

Definition 4.3 : DTCNN Feasibility Problem (DFP):
INSTANCE: Given is a DTCNN with binary, time-variant input signals.
QUESTION: Are there network parameters $p_0, \ldots, p_N$ *so that the network response* $\mathcal{M}^A_{\mathcal{F}}(\mathbf{u}^{[l]})$ *agrees with the desired response* $\mathbf{d}^{[l]}$ *for all* $l = 1, \ldots, L$ *after not more than* K *time steps, where* K *is bounded by a polynomial in* (L, M, N)*, and the response of the network is stable?*

There are no complexity-related results on FP or LP for binary recurrent architectures like DTCNNs, Hopfield or Little models known to the authors of this report. But still, taking into account the known results mentioned above, we claim

Conjecture 4.1 *The DTCNN Feasibility Problem is NP-complete*

4.3 The Objective Function

At this point it becomes necessary to define the objective function. The objective function is a measure of the errors, which the network makes during the recall phase of the training set. The objective function has to be zero, if the network perfectly fulfills the learning task. The higher the value of the objective function, the more the network deviates from the optimal behavior.

Learning is done by minimizing the objective function. The objective function can be defined as a (multivariate) function of the $N+1$ real-valued network parameters $o : IR^{N+1} \longrightarrow [0,1]$ or more general as a functional $o(\mathcal{F})$ of the Boolean mapping $\mathcal{F}$, which is performed by each cell. We will use the first form, since in this case, the objective function can be written down as a closed-form equation.

A frequently occuring phenomenon in DTCNNs are oscillations, i.e. the network does not reach a fixed point, but it performs stable limit cycles. This behavior is very undesirable, if the network is used in an associative memory type application. For this reason, oscillations have to be punished by high values of the objective function.

Let $y^{[l]}_{c,\mathbf{p}}(\infty) = \mathcal{M}^{\mathcal{A}}_{\mathcal{F}}(\mathbf{u}^{[l]})$ denote the output of cell c, when stimulus $\mathbf{u}^{[l]}$ was fed into the network, and the network has reached a stable fixed point. Then, a distance measure for item l is

$$\Delta^{[l]}_1(\mathbf{p}) = \begin{cases} \frac{1}{4}\sum_{c=1}^{M} \omega_c \cdot (y^{[l]}_{c,\mathbf{p}}(\infty) - d^{[l]}_c)^2 & \text{for stable output patterns} \\ 1 & \text{for stable limit cycles} \end{cases}$$

The $\omega_c \in [0,1]$ are weighting factors, which obey

$$\sum_{c=1}^{M} \omega_c = 1$$

With these weighting factors, different importance can be attached to certain cells in the output image. $\omega_c = 0$ for certain cells means that their final state is ignored (don't care symbols in the response), for example because the respective cell is no output cell. For binary patterns and uniform weighting, the equation for the stable fixed point case corresponds to a normalized Hamming distance. Oscillations are "bad" in terms of the objective function, and thus any optimization algorithm will avoid areas of the parameter space, where oscillations are likely to occur. The objective function $o(\mathbf{p})$ is then defined as the weighted average of the distance measures

$$o_1(\mathbf{p}) = \sum_{l=1}^{L} \Omega_l \Delta^{[l]}_1(\mathbf{p})$$

where again

$$\sum_{l=1}^{L} \Omega_l = 1$$

The $\Omega_l \in [0,1]$ are weighting factors for attaching different importance to item l.

The above objective function can be used for learning algorithms. Its main disadvantage is that the mechanism to avoid oscillations imposes relatively severe restrictions on the values of the network parameters.

A common observation for DTCNNs is that if oscillations occur, usually only a few cells oscillate, while the other cells stay constant [21]. In some applications (like for example the one in Section 6), local oscillations may be tolerated, thereby gaining more flexibility and less restrictions for the template parameters.

Therefore, it makes sense to define a second objective function $o_2(\mathbf{p})$. The idea is to integrate over one period T_0 of the stable limit cycle in the case of nonconstant outputs, while the computation of the distance measure remains the same for the case of stable output patterns.

$$\Delta_2^{[l]}(\mathbf{p}) = \begin{cases} \frac{1}{4}\sum_{c=1}^{M} \omega_c \cdot (y_{c,\mathbf{p}}^{[l]}(\infty) - d_c^{[l]})^2 & \text{for stable output patterns} \\ \frac{1}{4T_0}\sum_{t=T_1}^{T_1+T_0-1}\sum_{c=1}^{M} \omega_c \cdot (y_{c,\mathbf{p}}^{[l]}(t) - d_c^{[l]})^2 & \text{for stable limit cycles} \end{cases}$$

and again

$$o_2(\mathbf{p}) = \sum_{l=1}^{L} \Omega_l \Delta_2^{[l]}(\mathbf{p}) \tag{14}$$

5 Combinatorial Optimization by Simulated Annealing

5.1 Combinatorial Optimization Problems

Combinatorial optimization problems are usually given as pairs (Ω, o). Ω is the finite *solution space*, and $o : \Omega \longrightarrow IR^+$ is a *cost function* (also: *objective function*). An element $s \in \Omega$ from the solution space is usually called *state*. The goal of combinatorial optimization algorithms is to find

$$o^{\min} := \min_{s \in \Omega} \{o(s)\} \ .$$

The state $s^{\text{opt}} \in \Omega$ is the state with the global minimum of the cost function, i.e. $o(s^{\text{opt}}) \leq o(s)\ \forall s \in \Omega$.

Since $|\Omega|$, the size of the solution space, can be very large, but finite, one way to solve combinatorial optimization problems is by *extensive search*, i.e. by just evaluating the cost function for every $s \in \Omega$ and keeping track of the minimum. Unfortunately, $|\Omega|$ usually grows at least exponentially with the problem size. For this reason, extensive search methods become infeasible in most practical cases, even for moderate problem sizes.

For a number of problems (class P), algorithms have been found which can solve the problem in a time, which is bounded by a polynomial in the problem size. A solution like this is usually considered to be satisfactory. Well-known examples for these problems are for example finding the minimum (maximum) of a set of N numbers, determining the shortest path or the shortest spanning tree of weighted complete graphs, deciding whether a graph is planar, the Linear Programming problem, and many more [8].

For other classes of problems ($\notin P$), no algorithms are known which can solve the problem in polynomially bounded time. Using algorithms with exponentially bounded time requirements does not make much sense, because in this case, extensive search methods are preferable.

Some of these difficult problems are "solved" in practice by *exact non-polynomial algorithms*. These algorithms usually work well, but they are not guaranteed to finish in polynomially bounded time. Examples for these algorithms are *Partial Enumeration Schemes* (like *Branch-and-Bound methods*), *Polyhedral methods* (like *Cutting Plane algorithms*) or the famous *Simplex* method for the Linear Programming problem [9], [11], [8]. An essential prerequisite for Branch-and-Bound methods and Cutting Plane methods is that there must exist a relaxation of the original problem, which provides an efficient way of coming up with relatively tight lower bounds $o_i^{\inf}$ on the cost function for certain subsets Ω_i of the solution space, i.e.

$$o_i^{\inf} \leq \inf_{s \in \Omega_i \subset \Omega} \{o(s)\}$$

By using these bounds in an intelligent way, large parts of the solution space can be discarded without explicitly evaluating the cost function for states in these parts of the solution space, thereby significantly reducing the computational burden.

For many applications, it is not absolutely necessary to find the global optimum s^{opt}, but a "good" objective function value will suffice. This is the domain of *non-exact algorithms*, which can find good approximations of the solution for many problems in finite time. These algorithms can be classified into three groups: *Greedy* procedures, *Local Improvement* methods, and *truncated exponential schemes*. Examples for these kind of algorithms are *Steepest Descent* methods, *Monte-Carlo* type methods, or *Simulated Annealing* (see [11], [8]).

5.2 Simulated Annealing Algorithms

Simulated Annealing (SA) methods have been introduced in 1983 in [22], and since then they have been applied quite successfully to many different practical optimization problems in various fields (e.g. [23], [24], [22], [25], [26], [27], [28], [29], [30]). The core of the Simulated Annealing algorithm is based on the Metropolis algorithm [31].

Simulated Annealing is a last alternative for the solution of difficult combinatorial optimization problems, where other algorithms cannot be applied. The algorithm is very easy to implement, and many researchers report that it almost constantly comes up with approximate solutions of good quality. It must be mentioned, though, that if SA algorithms compete directly with other algorithms, they are almost always worse in terms of execution time.

5.2.1 The Algorithm

The SA algorithm explores the solution space by the following strategy: It begins the search from an arbitrary initial state. Then, a new state is generated, and the objective function for this state is evaluated. The new state replaces the current state or is rejected, according to a certain *acceptance rule*, which depends on the objective function values of the current and the new state, and on a

system parameter T, which is called *(system) temperature* in analogy to the cooling of substances in physics. Then a new state is generated from the current state, and the whole process is repeated.

The acceptance function has the property that if the system temperature T is high, then newly generated states, which have a worse objective function than the current state, are accepted with a high probability, while at low values of T, worse states are accepted only with a low probability. States with a lower objective function are always accepted. The idea is now to slowly decrease the system temperature according to a certain schedule (referred to as the *cooling schedule*), thereby forcing the average of the objective function of the visited states to descend to lower values. Since the SA algorithm can accept worse states, the algorithm can escape from local minima under certain conditions. Under certain assumptions on properties of the algorithm and the cooling schedule, it can be shown that the algorithm will reach the global optimum s^{opt} with probability one.

The problem is to set the system parameters and the cooling schedule so that two effects are avoided:

- **adiabatic effect:** The system temperature is lowered too abruptly, so the system ends up in a local minimum too far away from the optimum ("quenching").
- **super-cooling:** The system temperature is lowered too slowly, so that the cooling process takes an unnecessary long time.

The SA algorithm is given in a PIDGIN ALGOL [9] description in fig. 3. The algorithm is specified by three functions (`generate_state`, `accept`, and `update_T`) and two loop termination criteria (`inner-loop criterion` and `outer-loop criterion`). The cooling schedule comprises the accept function and the two loop criteria.

```
begin Simulated_Annealing;
  initialize_state(s_cur);
  o_cur = o(s_cur);
  T = T_0; /* initial temperature */
  do /* outer loop */
    do /* METROPOLIS or inner loop */
      s_new = generate_state(s_cur); /* generate new state */
      o_new = o(s_new);
      if accept(o_new, o_cur, T)
        then s_cur = s_new; /* accept the new state */
             o_cur = o_new;
      endif;
    until(inner-loop criterion false);
    T = update_T; /* set next temperature */
  until(outer-loop criterion false);
end Simulated_Annealing;
```

Figure 3: PIDGIN ALGOL representation of the Simulated Annealing algorithm

5.2.2 Cooling Schedules

Different authors have analyzed the theoretical properties of Simulated Annealing algorithms (e.g. [32], [33], [34], [35], [36], [37], [38]). [39] contains a relatively recent review of SA theory and cooling schedules in terms of homogenous and inhomogenous Markov chains. In a Markov chain, the *next* state only depends on the current state. The transition probabilities from state s' to state s can thus be described by a transition matrix with elements $P_{s's}(T)$.

The following paragraphs describe guidelines for the setting of the three characterizing functions and the two stopping criteria. They basically use material contained in [39] and [30].

functions `generate_state, accept`:
The transition from one state to another state, which was generated by the `generate_state` function, is called a *move*. The *move set* $\mathcal{MS}(s')$ is the set of all states s which can be reached from s' in one move.

Let the Markov chain be homogenous, irreducible ($P_{s's}(T) > 0 \; \forall s' \in \Omega$ and $s \in \mathcal{MS}(s')$) and aperiodic ($\exists s$ so that $P_{ss} \neq 0$). Let $r \in [0,1]$ be a random number with a uniform probability distribution, and let the accept function according to Kirkpatrick [22] as

$$\texttt{accept}(o_{\text{new}}, o_{\text{cur}}, T) = \begin{cases} \text{TRUE} & \text{if } r < e^{\frac{o_{\text{cur}} - o_{\text{new}}}{T}} \\ \text{FALSE} & \text{otherwise} \end{cases} \tag{15}$$

then it can be shown that if an infinite number of steps is executed for a fixed temperature T, the probability distribution of the states reaches an equiblibrium distribution $\pi_s(T)$, which is independent of the initial state, and which is given as a Gibbs distribution

$$\pi_s(T) = \frac{e^{\frac{-o(s)}{T}}}{\sum_{s' \in \Omega} e^{\frac{-o(s')}{T}}}$$

This distribution implies that in the limit $T \to 0$ the probability of the optimum state s^{opt} will approach one.

Unfortunately, it can be shown that if the Markov chain is reversible ($\pi_s(T) P_{ss'}(T) = \pi_{s'}(T) P_{s's}(T)$) then there is no way (apart from degenerate cases) that the Markov chain will reach the equilibrium after a finite number of steps.

In the case of inhomogenous Markov chains (and this is always the case in reality) it has been shown that the SA algorithm will reach the optimum state with probability one under certain stricter conditions on the inhomogenous Markov chain, if a logarithmic cooling schedule $T_n = \gamma / \ln(n + n_0)$ is used, even if only one step per temperature is executed. γ and n_0 are problem dependent parameters, and under certain assumptions this logarithmic cooling schedule can be shown to be a sufficient and necessary condition for convergence to the global optimum. The problem with the logarithmic cooling schedule is that it also needs an infinite amount of time, and thus it is infeasible in practice.

initial temperature:
It has been derived that $T_0 = K\sigma_\infty$ with $K \approx 20$ and σ_∞ the standard deviation of the cost function at $T \to \infty$ (i.e. when each move is accepted) is a reasonable choice.

inner-loop criterion:
Different strategies have been used in the literature, and most of them are based on heuristics instead of a theoretical derivation. Table 1 displays some frequently used strategies (see [30]).

strategy	number of repetitions in inner loop
Single	1
Constant	C (constant)
Geometric	$N(T_{\text{new}}) = \alpha \cdot N(T_{\text{old}})$ with $\alpha > 1$
Energy	repeat, until average cost of accepted states at one temperature varies little
Acceptances	repeat until a certain number of moves have been accepted

Table 1: Different inner-loop criteria

Adaptive strategies with a theoretical foundation are rare for the inner loop criterion. In [23], a statistical model using the shifted γ-function is established, and the inner loop is terminated, if certain system parameters are "close" to the predicted values.

update_T:
Many different strategies are used how to decrease the temperature. For very high values of the temperature, a quasi-equilibrium is reached relatively fast. The idea is to decrease the temperature only so much, that the quasi-equilibrium for the new temperature is obtained quickly, since the quasi-equilibrium at the previous temperature is a good starting point for the new equilibrium. The most popular approaches are summarized in Table 2 (see [30], [39]).

Many researchers using SA methods have reported that the geometric strategy works well in practice, but actually, Hajek has shown that this cooling schedule does not guarantee that the SA process will converge to the global optimum for $T \to 0$ [40].

outer-loop criterion:
Outer-loop stopping criteria are used to avoid unnecessary computations, when a good value of o^{opt} has been reached and the probability that the algorithm will find its way out of this local minimum is very low. Some approaches are given in Table 3 (see [30], [39]).

5.2.3 Aggregate Functions

In their theoretical study of Simulated Annealing algorithms in terms of homogenous Markov chains [32], Otten and van Ginneken introduce three aggregate functions and predict their behavior as a function of the system temperature T (assuming that the system stays near the equilibrium).

strategy	rule for updating T
Constant	$T = T_c$
Arithmetic	$T_{\text{new}} = T_{\text{old}} - C$
Geometric	$T_{\text{new}} = \alpha(T_{\text{old}}) \cdot T_{\text{old}}$ ($\alpha(T)$ usually constant)
Logarithmic	$T_n = \frac{\gamma}{\ln(n_0+n)}$
Aarts & Laarhoven	$T_{\text{new}} = \frac{3\sigma_{\text{old}} T_{\text{old}}}{3\sigma_{\text{old}}+\ln(1+\delta)T_{\text{old}}}$ where δ =const, and σ_{old} is the standard deviation of the objective function at the old temperature T_{old}.
Lam & Delosme	$\frac{1}{T_{\text{new}}} = \frac{1}{T_{\text{old}}} + \frac{\lambda<\delta E^2>T_{\text{old}}^2}{\sigma_{\text{old}}^3}$ where $< \delta E^2 >$ is the second moment of the accepted energy changes
Huang & al.	$T_{\text{new}} = T_{\text{old}} \cdot e^{\frac{-\alpha T_{\text{old}}}{\sigma_{\text{old}}}}$
Otten & v.Ginneken	$T_{\text{new}} = T_{\text{old}} - \frac{C \cdot T_{\text{old}}^3}{\sigma_{\text{old}}^2}$

Table 2: Different temperature updating strategies

strategy	stopping criterion for outer loop
Iterations	fixed number of iterations
Temperature	$T < T_{\text{end}}$
Energy	$<E>$ changes little in several iterations where $<E>$ is the first moment of the cost
Acceptances	number of accepted moves below a limit
Romeo & al	compare $\lvert o^{\max}(T) - o^{\min}(T)\rvert$ with $\Delta o(T)$ (in one move) for accepted moves only; if equal, then stop and perform one inner loop with $T = 0$.
Otten & v.Ginneken	$\frac{\sigma_{\text{old}}^2}{T(E_\infty - E)} < \varepsilon$ where E_∞ is the average cost at $T \to \infty$

Table 3: Different outer-loop criteria

These functions are the *approximated expected value* of the cost function $\bar{E}(T)$, the *approximated standard deviation* of the cost $\bar{\sigma}(T)$, and the *accessability* $\bar{H}(T)$.

$$\bar{E}(T) = \frac{1}{i(T)} \cdot \sum_{i=1}^{i(T)} o_i \qquad \text{and} \qquad \bar{\sigma}(T) = \sqrt{\frac{1}{i(T)-1} \cdot \sum_{i=1}^{i(T)} (o_i - \bar{E}(T))^2}$$

The summation is over all $i(T)$ accepted states $o_i = o(s_i)$ at one temperature T.

The accessibility is given by

$$\begin{aligned} \bar{H}(T_{\text{new}}) &= \bar{H}(T_{\text{old}}) - \frac{\bar{E}(T_{\text{old}}) - \bar{E}(T_{\text{new}})}{T_{\text{new}}} \\ \bar{H}(\infty) &= \ln |\Omega| \end{aligned}$$

where $|\Omega|$ is the cardinality of the solution space.

Otten and van Ginneken identifiy a *critical temperature* T_{crit}, a *region of weak control* (for $T > T_{\text{crit}}$), and a *region of strong control* (for $0 < T < T_{\text{crit}}$). Table 4 summarizes the results in [32] regarding the predicted behavior of the aggregate functions.

function	$0 < T < T_{\text{crit}}$ (weak control)	$T > T_{\text{crit}}$ (strong control)
$\bar{E}(T)$	$\bar{E}(\infty) + \frac{\bar{\sigma}^2(\infty)}{T_{\text{crit}}} \left(\frac{T}{T_{\text{crit}}} - 2 \right)$	$\bar{E}(\infty) - \frac{\bar{\sigma}^2(\infty)}{T}$
$\bar{\sigma}(T)$	$\bar{\sigma}(\infty) \cdot \frac{T}{T_{\text{crit}}}$	$\bar{\sigma}(\infty)$
$\bar{H}(T)$	$\bar{H}(\infty) + \frac{\bar{\sigma}^2(\infty)}{T_{\text{crit}}^2} \left(\ln \left(\frac{T}{T_{\text{crit}}} \right) - \frac{1}{2} \right)$	$\bar{H}(\infty) - \frac{\bar{\sigma}^2(\infty)}{2T^2}$

Table 4: Predicted behavior of the aggregate functions

6 Experiments

Since DTCNNs have a restricted architecture, it can be expected that DTCNNs will not be as powerful and the areas of application not as wide as for a fully-connected architecture with translationally variant weights (e.g. the standard Hopfield network). Especially the fact that the DTCNN only has a local interconnection structure suggests, that the input data should contain at least some information contained in the 1D or 2D neighborhood of each cell. Image processing tasks seem to be promising from this point of view. Actually, several authors have used DTCNNs and

CNNs for basic image processing tasks (see [41], [1], [29], [21], [42]). In all these cases, the network parameters were determined by the use of *local* learning (design) algorithms. It has been reported, that these tasks could also be learned by global learning algorithms [43].

A more demanding task are pattern recognition problems, which are too involved for local learning algorithms. So far, no attempts have been made to do pattern recognition with an architecture consisting exclusively of a standard DTCNNs (in [44], CNNs are used to preprocess the input data, while the actual classification is done by conventional digital logic).

Fully-connected, synchronous Hopfield networks are often used as an auto-associative or hetero-associative memory. In these applications, patterns are "stored" in the network as stable fixed points. Input patterns are applied to the network, and they relax to the nearest stable state, thereby mapping input patterns onto output patterns. Theoretical considerations suggest that the number of patterns, which can be stored in the network, is linked to the number of free (independent) network parameters [45], [46], [47], [48], [49].

A fully-connected M-cell Hopfield network with symmetric weights has exactly $0.5 \cdot (M^2 - M)$ independent weights. Due to its local interconnection structure and the translational invariance of the weights, a DTCNN has much fewer independent weights compared to the Hopfield network, typically only 19 (1-neighborhood) or 51 (2-neighborhood) weights. This suggests that it might not be possible to store many different patterns in a DTCNN. It is very difficult to stabilize a large number of output patterns, which are not correlated with the input patterns.

6.1 Description of the Problem

The problem treated here is a very simple one. The task of the network is to distinguish bitmap representations of the letters "A" and "B". The input data is given as 30 instances of 10×10 binary bitmaps of the "A"s and "B"s each. Black pixels are represented by a $+1$ value, white pixels by a -1 value. The input patterns are surrounded by a ring of white dummy cells. Fig. 4 shows all 60 bitmaps.

The network is a standard DTCNN with a 1-neighborhood in **a**- and **b**-template, and a bias i.

The task, which the network has to perform, is very easy: The desired output patterns for the "A"s are completely black ("+1") output patterns, and the desired output patterns for the "B"s are completely white ("−1"). This easy strategy reflects the fact, that the number of patterns, which can be stored in the network, is relatively small. At the same time, the translational invariance of the desired output patterns is a logical consequence from the translational invariance of the network parameters.

Since the objective function o_2 (see (14)) imposes fewer restrictions, it is used instead of o_1.

Remark 1: The idea that not a detailed binary output image, but only the Hamming distance between the network output image and an all black or an all white image constitute the output of the network, leads to the introduction of the **O***utput* **M***ajority* **D***ecision mode* (OMD) for

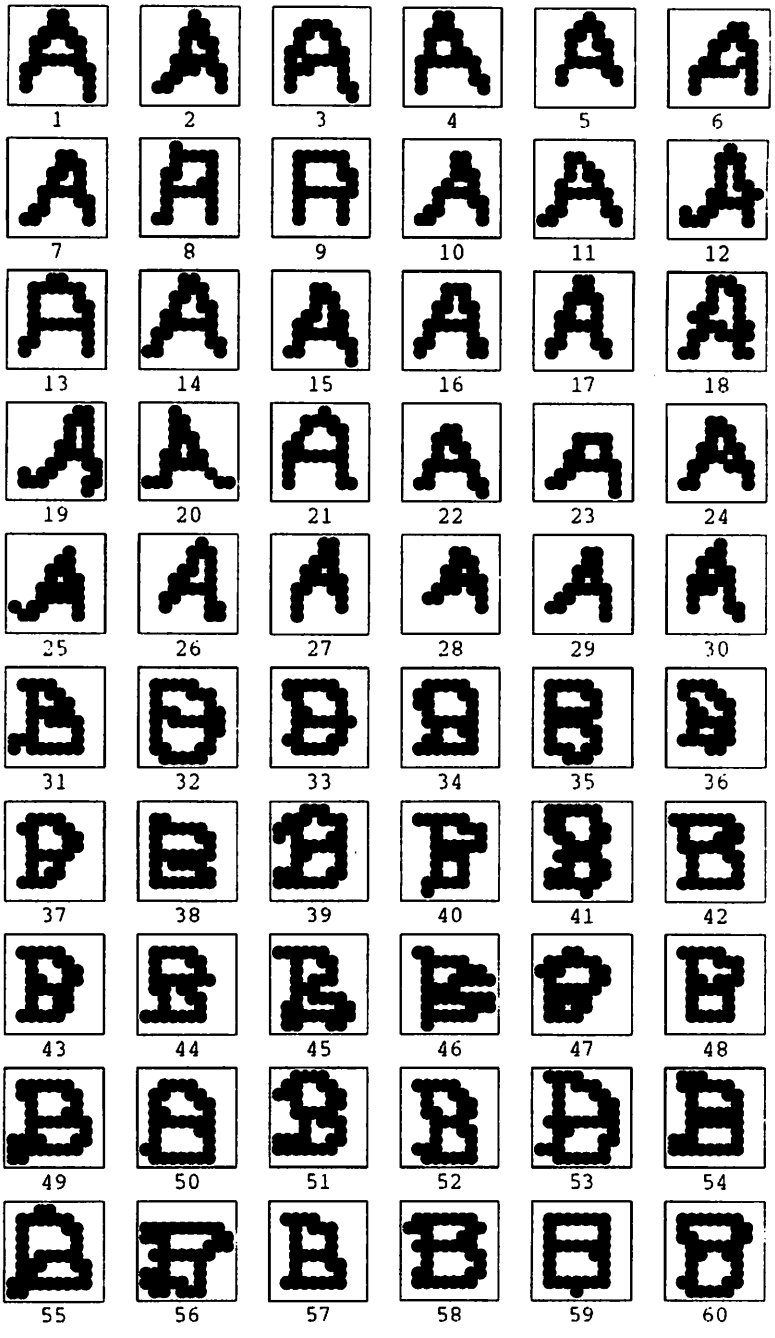

Figure 4: Training Set

DTCNNs. In the OMD mode, the output signal y^{OMD} is a scalar, which can only take on the disrete values

$$y^{\mathrm{OMD}} \in \left\{-1, -\frac{M-2}{M}, -\frac{M-4}{M}, \ldots, \frac{M-4}{M}, \frac{M-2}{M}, 1\right\}$$

The signal y^{OMD} originates from summing up all binary cell outputs y_c and dividing it by the number of cells M. It can be obtained relatively easy in a hardware realization, since only one additional global summing wire is necessary.

DTCNNs can operate at a high clock frequency. Results obtained from an experimental DTCNN chip indicate that clock frequencies of 10 MHz are easily feasible [50]. Thus, once the data is on the chip, the processing can be done relatively fast. The bottleneck of the whole system will be the pattern data I/O, since for reasonable cell grid sizes, data can only be transferred sequentially (or row-wise). Using the signal y^{OMD} instead of the whole output pattern will get rid of the output data readout phase, and at the same time support the idea of translational invariance.

If the input signals are stored on-chip, then it is straightforward to implement a mechanism which can reset the initial state of the system, either to the input image or to an all black or an all white pattern. Thus it is possible to apply different sets of templates to the same input image, without reloading the input images onto the chip each time.

Remark 2: It has to be added that it is also possible to use continuously-valued input signals $u_c(k) \in [-1, 1]$. For the learning algorithm, it will be no difference whether it processes binary-valued or continuously-valued input signals.

6.2 The Simulated Annealing Algorithm for the DTCNN Learning Problem

In Subsection 4.2 it was conjectured that the Feasibility Problem for DTCNNs (DFP) is NP-complete. Since the LP is even harder, it does not make sense to look for an algorithm, that solves the problems in polynomial time.

Unfortunately, the functional relationship between the system state and the objective function as given in (14) is rather complicated. To evaluate the objective function at one state in the solution space, the whole network actually has to be evaluated for each item of the training set. Since the number of possible cell input patterns $\mathbf{e}$ grows exponentially with N, the number of cell inputs, there does not seem to be a way of saving computational effort by careful bookkeeping. In addition, no way of finding cheap and reasonable lower bounds on the objective function for subsets of the solution space Ω_i seems obvious. For this reason, Branch-and-Bound methods and Cutting Plane algorithms are not applicable to the problem. Hence, Simulated Annealing seems to be the best alternative.

For the accept function, the standard form (15) is chosen. The choice of a suited move set is a difficult problem. In the DTCNN case, each of the convex cones in the parameter space (see Section 3.2) corresponds to a state s in the solution space Ω. Thus each state s corresponds to a linearly separable Boolean function F.

Unfortunately, there is no easy way to parametrize linear separable Boolean functions. One possiblilty is to use the $N + 1$ network parameters from the parameter vector $\mathbf{p}$ to describe a linear separable Boolean function. This approach makes sense, because the network parameters are needed for the operation of the DTCNN to evaluate the objective function. The mapping from $\mathbf{p}$ onto the system states (the set of all possible linearly separable Boolean functions) is not injective. Taking $\mathbf{p}^{opt}$, the optimally robust parameter vector in a convex cone $\mathcal{C}_{\mathbf{p}}$, would change this, but there is no cheap way to obtain $\mathbf{p}^{opt}$ from $\mathbf{p}$, since a convex cone can be defined by exponentially many hyperplanes (see Lemma 3.3).

Linearly separable Boolean functions of N inputs can also be uniquely characterized by their $N+1$ *Dertouzos parameters* or by their $N+1$ *Chow parameters* [5], both of which are $N+1$ integer numbers, but there is no easy way to compute the network parameters $\mathbf{p}$ from these characterizing parameters.

The first approach was used for the parametrization. The following move set was chosen: Let $\mathbf{p}^{\text{rand}}$ be a $N+1$-dimensional random vector, whose elements $p_n^{\text{rand}} \in [-1,1]$ are mutually independent and uniformly distributed. Then

$$\mathcal{MS}(\mathbf{p}) = \left\{ \bar{\mathbf{p}} \in IR^{N+1} \mid \|\bar{\mathbf{p}} - \mathbf{p}\|_\infty \leq \rho,\ \rho \in IR^+ \right\}$$

A new move is generated from $\mathbf{p}$ by $\bar{\mathbf{p}} = \mathbf{p} + \rho \cdot \mathbf{p}^{\text{rand}}$. The constant ρ is clearly depending on the problem. Experimental results indicated that $\rho \approx 0.05$ is a good value for a DTCNN with a 1-neighborhood ($N = 19$). Choosing ρ too large will result in a mere random search without any neighborhood properties, and a too small value will result in $\bar{\mathbf{p}} \in \mathcal{C}_{\mathbf{p}}$ too many times.

6.3 Experimental Results

The Simulated Annealing algorithm was tested with different annealing schedules. In almost all runs, the algorithm found states with good objective function values. It turned out that the quality of the final approximation is not extremely sensitive with respect to the cooling schedule that is used.

Fig. 5 shows the approximated expected value $\bar{E}$ of the objective function depending on the temperature. Each data point corresponds the average of $1000 \ldots 2000$ evaluations of the objective function at the corresponding temperature. The temperature is displayed on the horizontal axis, and it is scaled by a factor $1E4$. The region of strong control and the region of weak control can be clearly identified. The critical Temperature T_{crit} is at about $200/1E4 = 0.02$. The linear behavior in the weak control region and the hyperbolic behavior in the strong control region can clearly be recognized. The final solutions lie in the interval $[0.02, 0.08]$ for low temperature values.

In fig. 6 the approximate standard deviation of the objective function is displayed. Since the data is very noisy, it has been smoothed using

$$\sigma_{\text{new}} = 0.9 \cdot \sigma_{\text{old}} + 0.1 \cdot \sigma_{\text{new}}$$

These data points reflect the predicted behavior of the aggregate function only poorly. The predicted constant behavior in the strong control region can be observed only for very high temperatures ($T > 2000/1E4 = 0.2$). The value of σ for $T \to \infty$ is at around 0.07. There is a broad transition region between the strong control and the weak control regions. In the weak control region, the predicted linear behavior can be clearly observed. Interestingly, there seem to be two different slopes. The lower "path" comes exclusively from runs using the Otten/van Ginneken temperature update law. Three simulations were run with 1000 and 2000 iterations per temperature each, and the results were similar every time. The upper slope stems from all other cooling schedules that were tried out.

Fig. 7 displays the dependence of the acceptance rate (i.e. the number of accepted moves divided by the number of all moves) on the temperature T. The acceptance rate seems to be closely correlated

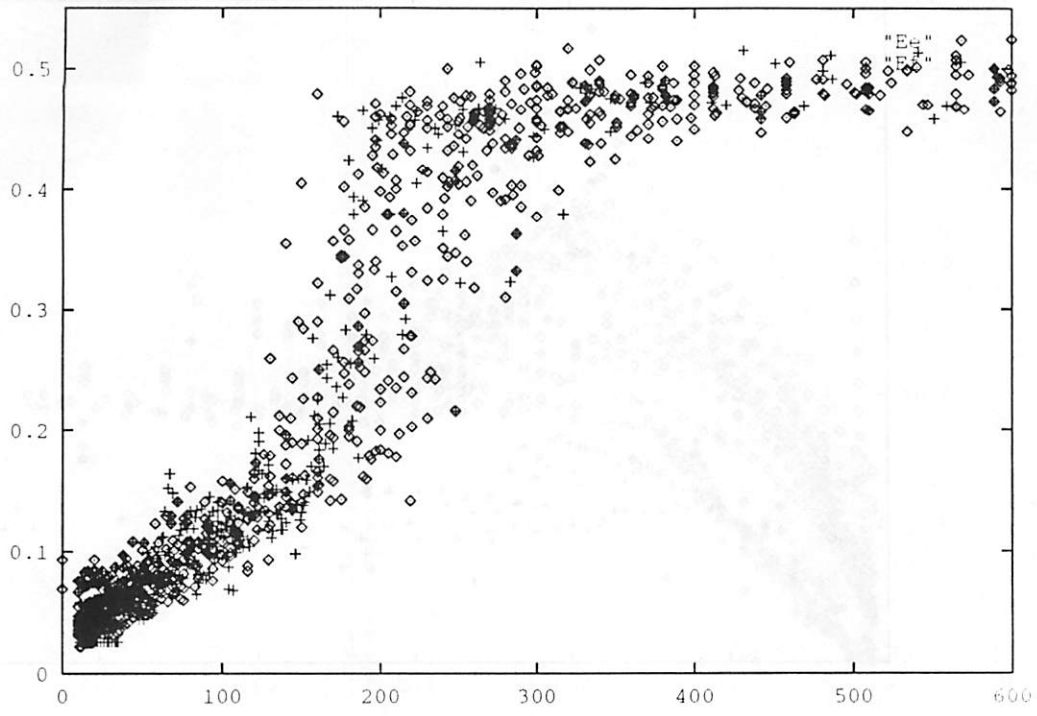

Figure 5: Average $E(T)$ of the objective function; T is scaled by factor $1E4$

with T. At high temperatures, close to 100% of the moves are accepted. At a slightly higher temperature before the approximated expected value drops to lower values, the acceptance rate decreases. For low temperatures the acceptance rate goes assymptotically against zero, thereby signifying that most of the moves are in vain.

The initial temperature was first set according to the $T_0 = K\sigma_\infty$ rule. It turned out that choosing $K \approx 20$ is too high. The algorithms performed better, when $T_0 \approx 2 \ldots 5$, because in this case more time is spend at lower temperature values.

The following temperature update rules were tried: (Otten/van Ginneken), (Aarts/Laarhoven), (Huang/Romeo), the geometric schedule, the arithmetic schedule, and a hand-designed schedule, where many iterations were executed in that region where the slope of $E(T)$ was steepest (this was done to perform a particularly accurate search in the region, where the global sorting takes place). It turned out that, as far as the final values of the objective function, i.e. the quality of the solution, are concerned, there are no significant differences between these algorithms, when the total number of iterations used for each algorithm was roughly the same. Since the geometric schedule is the easiest to implement, it should be preferred. It turned out that the total number of iterations seems to have a more important influence on the quality of the solution than the temperature update law.

In almost all experiments, the inner loop was executed for either 1000 or 2000 steps. While this does not seem to matter so much as far as the quality of the final result is concerned, the aggregate functions will become less smooth, if the inner loop is executed only 1000 times.

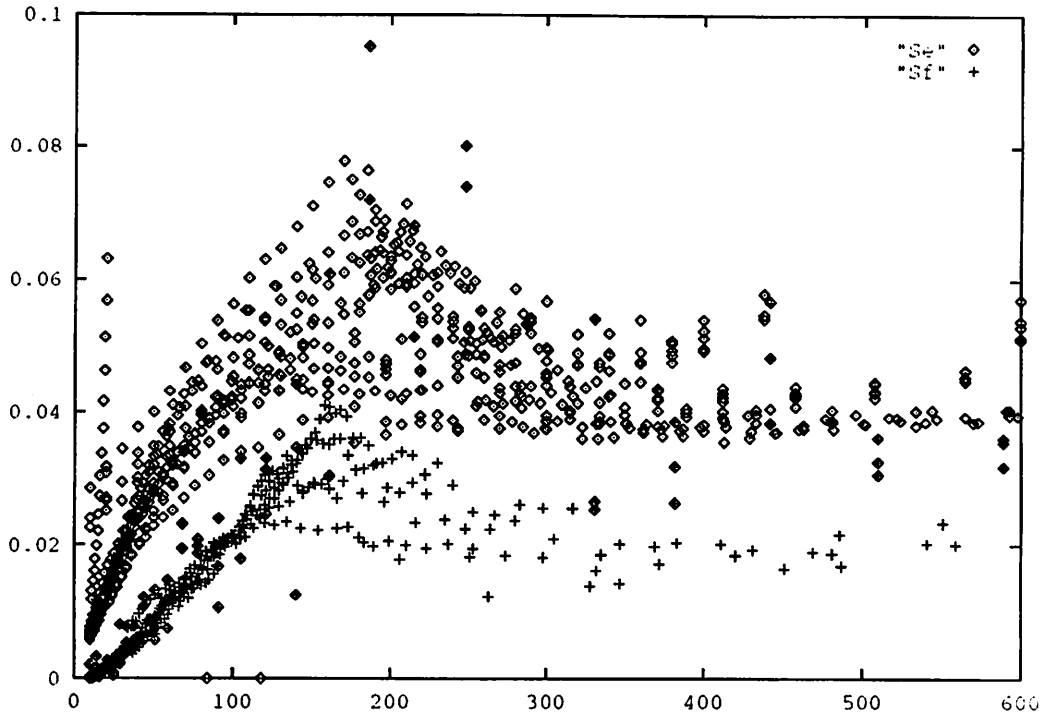

Figure 6: Standard deviation$\sigma(T)$ of the objective function; T is scaled by factor 1E4

Three experiment were performed with a geometric cooling schedule, where the inner Metropolis loop was executed exactly once for each temperature. In these cases, the algorithm performed relatively bad.

Sometimes, the average of the cost function at $T \rightarrow 0$ is worse than the objective function of a state, that was visited during the annealing algorithm. For this reason, it makes sense to check after each evaluation of the objective function whether the new value is lower than the previous optimum, and to store this value, if it is better.

It should be mentioned that due to the long running times (roughly 2 days on a workstation), the number of inner loop steps is limited. In these experiments, the algorithm was never run for more than $1E5$ steps. This is little compared to the number of iterations that other authors (e.g. $200E6$ steps in [24]) use. The results of the experiments suggest that by using more iterations, the quality of the solutions could be improved further.

Simulated Annealing methods did not find a solution to the problem that was better than the best solution found by the HYBRID algorithm [21], but they found good solutions more reliably than the HYBRID algorithm and are therefore a more robust tool.

It is outside the scope of this work to study the behavior of the DTCNN when applied to the pattern recognition task. [21] contains a more detailed description of solution strategies used by the network.

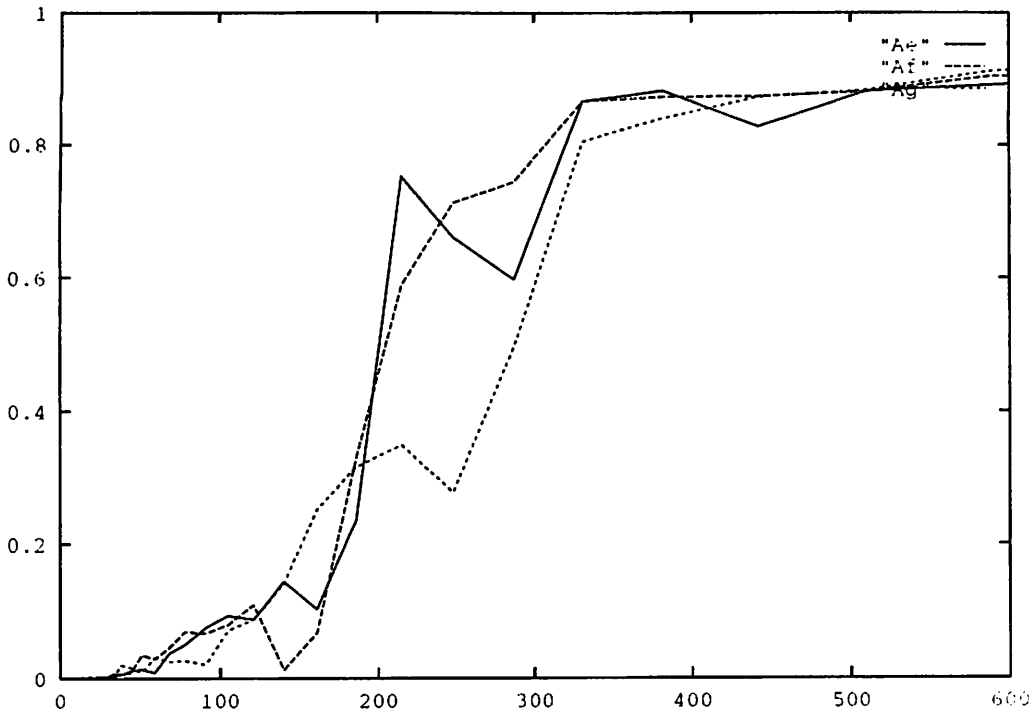

Figure 7: Acceptance rate; T is scaled by factor 1E4

7 Conclusions

Assuming binary input patterns, fixed and binary-valued dummy cells and a fixed strategy to set the initial state, the behavior of DTCNNs depends only on the linearly separable Boolean mapping, which is performed by each cell of the network at each time step. Using results from Linear Threshold Logic theory, the cardinality of the solution space is shown to be of the order of $O(2^{N^2}/N!)$, where N is the number of inputs of each cell of the network. The parameter space is divided by 2^N hyperplanes through the origin into a large number of convex cones. Each of these cones corresponds to a linearly separable Boolean function, and it can be defined by at most 2^N hyperplanes.

It is shown that the reliability of the network (probability of errors in a hardware realization) depends on the distance of the network parameter vector from the nearest hyperplane. Even the simple problem of determining the nearest hyperplane is NP-complete. A worst-case bound on the required accuracy of the network parameters is given. In a simulation of a DTCNN on a computer, the nearest hyperplane can be obtained during a regular run of the network. The problem of finding the optimum position of the parameter vector in a convex cone is a standard "perceptron of optimal stability" problem.

The Feasibilty Problem and the Learning Problem for DTCNNs are formally defined. Results of other authors on the complexity of the Learning Problem for certain network architectures are

reviewed. The Feasibility Problem for DTCNNs is conjectured to be NP-complete, where it is required that the network has to reach the solution after K time steps.

An objective function is defined, which attributes a cost to each state of the solution space. Learning is achieved by minimizing this objective function as a function of the template parameters.

A short overview over Simulated Annealing algorithms is given, and different cooling schedules are reviewed. Simulated Annealing is then applied to the problem of minimizing the objective function, thereby finding an approximate solution to the learning problem. A standard DTCNN architecture is used for a simple pattern recognition problem, where the DTCNN has to distinguish bitmap representations of "A"s and "B"s. The Simulated Annealing algorithms constantly finds relatively good approximate solutions. The quality of the solutions produced by the algorithms is robust with respect to details of the cooling schedule.

Solving the Learning Problem for DTCNNs by Simulated Annealing is a computationally expensive method, but due to the difficulty of the problem, it seems to be the only feasible method.

8 Acknowledgements

The first author would like to thank Professor Chua for his hospitality and all the people in Prof. Chua's lab for their help and comments, especially his Ph.D. students Ken Crounse, Chai Wah Wu, Kevin Halle, and Bert Shi.

A Introduction to Terminology of Complexity Theory

This section gives a short introduction to the terminology of complexity theory. For further information, the book by Garey and Johnson [16], from which most of the definitions and problems in this section are taken, is highly recommendable.

A *problem* is a general question to be answered. Usually, a problem has several parameters or variables. A problem is given by a general description of the variables and the conditions, which the solution of the problem has to satisfy.

An *instance* of the problem is obtained by setting the variables of the problem to specific values.

An *algorithm* is a general, step-by-step procedure for solving a problem. An algorithm *solves* a problem, if it finds a solution for *any* instance of a problem.

The cost of an algorithm is usually measured in terms of execution time or in terms of required storage space. Both quantities depend on the size of the instance of the problem, i.e. the number of variables of the instance of the problem.

A very famous example is the *Traveling Salesman Problem*, where a set of N cities $c_1, c_2, \ldots, c_N$

is given, and distances $d(c_i, c_j)$ between all cities. The problem is to find the shortest tour for a salesman, who starts from one city, visits each city exactly once, and finally returns to the starting point. In this example, the size of an instance of the problem is N, the number of cities.

Most practical problems have a corresponding *decision problem.* A decision problem is a problem where the answer is either "yes" or "no". In the case of the Traveling Salesman Problem, the corresponding decision problem would be: "Is there a tour with a tour length shorter than a given total distance d?". Decision problems are usually easier than the corresponding "full" problems.

The interesting question is how the execution time of an algorithm solving a problem scales with the size of the instance of the problem. In general, it is desirable that the execution time is bounded by a polynomial in the size of the instance. All those problems for which algorithms exist, which can solve the problem in polynomial time (i.e. time bounded by a polynomial in the size of the instance) are said to belong to *class P.*

In other cases, the execution time can only be bounded by a function which depends exponentially on the size of the instance. This case is highly undesirable, since in this case, only small instances of the problem can be solved in practice due to the fast growth of execution time. Problems, for which no polynomial time algorithm exists, are called *intractable.*

The *class NP* contains all those problems that can be solved in polynomial time by a nondeterministic algorithm ("NP" stands for "nondeterministic polynomial"). These algorithms consist of a guessing stage (this stage has built-in intuition, i.e. the capability of an oracle) and a checking stage. If a problem is in NP, then this implies that, given a correct solution by an oracle, it can be figured out in polynomial time whether the solution is indeed correct or not.

The relationship between classes P and NP is still not completely determined. For sure, $P \subseteq NP$, but it is still an unresolved problem whether $P=NP$ or not. It is strongly assumed, but still an unproved conjecture, that $P \neq NP$.

Based on this conjecture, the class of *NP-complete* problems is introduced. This is an equivalence class of problems, for which, assuming the validity of the $P \neq NP$ conjecture, no polynomial time algorithm can be found. Membership in this class is shown by *polynomial transformation.* A polynomial transformation of problem 1 to problem 2 is a polynomial time algorithm, which transforms any instance of problem 1 to an instance of problem 2.

To show that a certain problem belongs to the class of NP-complete problems, two steps have to be taken:

- Show that the problem belongs to the class NP, i.e. that a correct solution can be veryfied in polynomial time.
- Show that a polynomial transformation from a problem, which is known to belong to the class of NP-complete problems, to the problem in question exists. This implies that, assuming that a polynomial time algorithm for the desired problem exists, then any instance of the known NP-complete problem could be solved in polynomial time as well, using this polynomial transformation. This is a contradiction, and thus the problem is equivalent to the known NP-complete problem.

The Traveling Salesman problem is one of the most famous NP-complete problems.

In addition, another class of problems is introduced, i.e. the class of *NP-hard* problems. A problem belongs to this class, if it is not in NP and there exists a polynomial transformation from a problem known to be NP-complete. The class is called "NP-hard", because problems belonging to this class are at least as hard as NP-complete problems.

The foundations for the theory of NP-completeness were laid by Stephen Cook in [51].

B Evaluation of the Error Integral

The right integral in expression (8) can be evaluated. We introduce the $(N+1)$-dimensional elliptical coordinates

$$\begin{array}{rcccccccc}
\chi_0 & = & \rho \cdot \sigma_0 & \cos\phi_1 & & & & & \\
\chi_1 & = & \rho \cdot \sigma_1 & \sin\phi_1 & \cos\phi_2 & & & & \\
\chi_2 & = & \rho \cdot \sigma_2 & \sin\phi_1 & \sin\phi_2 & \cos\phi_3 & & & \\
\vdots & & \vdots & \vdots & & & & & \\
\chi_{N-1} & = & \rho \cdot \sigma_{N-1} & \sin\phi_1 & \sin\phi_2 & \cdots & \sin\phi_{N-1} & \cos\phi_N \\
\chi_N & = & \rho \cdot \sigma_N & \sin\phi_1 & \sin\phi_2 & \cdots & \sin\phi_{N-1} & \sin\phi_N
\end{array}$$

where $\rho \geq 0$, $\phi_N \in [0, 2\pi]$, and $\phi_n \in [0, \frac{\pi}{2}]$ for $n = 1, \ldots, N-1$. Evaluating the expression

$$d\chi_0 \cdots d\chi_N = \det\left(\frac{D(\chi_0, \ldots, \chi_N)}{D(\rho, \phi_1, \ldots, \phi_N)}\right) d\rho \; d\phi_1 \cdots d\phi_N$$

leads to

$$d\chi_0 \cdots d\chi_N = \left|\left(\rho^N \cdot \sigma_0 \cdots \sigma_N\right) \cdot \left(\sin^{N-1}\phi_1 \cdot \sin^{N-2}\phi_2 \cdots \sin\phi_{N-1}\right)\right| \cdot d\rho \; d\phi_1 \cdots d\phi_N \qquad (16)$$

The expression $\frac{D(\chi_0,\ldots,\chi_N)}{D(\rho,\phi_1,\ldots,\phi_N)}$ is the Jacobian determinant of the functions $(\chi_0, \ldots, \chi_N)$. Since

$$\sum_{n=0}^{N} \left(\frac{\chi_n}{\sigma_n}\right)^2 = \rho^2$$

we get from combining (7), (8), (16), and using $\beta := d_{\min} \cdot \sigma_{\max}^{-1}$:

$$P_{\text{ok}} > \int_{\phi_N=0}^{2\pi} \underbrace{\int_0^{\pi} \cdots \int_0^{\pi}}_{\phi_{N-1}\cdots\phi_1} \int_{\rho=0}^{\beta} \frac{\rho^N \cdot e^{-\frac{\rho^2}{2}}}{\sqrt{2\pi}^{N+1}} \cdot \left(\sin^{N-1}\phi_1 \cdot \sin^{N-2}\phi_2 \cdots \sin\phi_{N-1}\right) \left(d\rho \; d\phi_1 \cdots d\phi_N\right) \qquad (17)$$

All these integrals can now be evaluated independently from each other. The definite integrals over the trigonometric functions can be found in a mathematical reference book [52]:

$$\int_0^{\pi} \sin^{2m} x dx = \pi \cdot \frac{1 \cdot 3 \cdot \ldots \cdot (2m-1)}{2 \cdot 4 \cdot \ldots \cdot (2m)}$$

$$\int_0^{\pi} \sin^{2m+1} x dx = 2 \cdot \frac{2 \cdot 4 \cdot \ldots \cdot (2m)}{1 \cdot 3 \cdot \ldots \cdot (2m+1)}$$

Combining two terms gives

$$\int_0^{\pi} \int_0^{\pi} \sin^{2m} x_1 \cdot \sin^{2m+1} x_2 \; dx_1 dx_2 = \frac{2\pi}{2m+1}$$

Since we know that N is even (c.f. Section 2), we can combine the above terms and get

$$\int_0^{\pi} \sin^{N-1} \phi_1 \; d\phi_1 \cdot \ldots \cdot \int_0^{\pi} \sin \phi_{N-1} \; d\phi_{N-1} \cdot \int_0^{2\pi} d\phi_N = 2 \cdot \frac{\sqrt{2\pi}^N}{(N-1) \cdot \ldots \cdot 3 \cdot 1} \tag{18}$$

For the term depending on ρ we use the recursion

$$\int_0^{\beta} \rho^N \cdot e^{-\frac{\rho^2}{2}} d\rho = (N-1) \int_{\rho=0}^{\beta} \rho^{N-2} \cdot e^{-\frac{\rho^2}{2}} d\rho - \beta^{N-1} e^{-\frac{\beta^2}{2}} \tag{19}$$

Since N is even, we can reduce the integration involving ρ to an expression containing the Gaussian error integral plus a sum of other terms. Applying (19) $\frac{N}{2}$ times, we finally arrive at the expression

$$\int_0^{\beta} \rho^N \cdot e^{-\frac{\rho^2}{2}} d\rho = 1 \cdot 3 \cdot \ldots \cdot (N-1) \cdot \left[\int_0^{\beta} e^{-\frac{\rho^2}{2}} d\rho - e^{-\frac{1}{2}\beta^2} \sum_{l=0}^{\frac{N}{2}-1} \frac{\beta^{2l+1}}{1 \cdot 3 \cdot \ldots \cdot (2l+1)} \right] \tag{20}$$

Since the sum is finite, the expression will converge for all β due to the influence of the exponential function. Putting together (17), (18), and (20), we get

$$P_{\text{ok}} > P_N(\beta) := \text{erf}\left(\frac{\beta}{\sqrt{2}}\right) - \sqrt{\frac{2}{\pi}} \cdot e^{-\frac{1}{2}\beta^2} \sum_{l=0}^{\frac{N}{2}-1} \frac{\beta^{2l+1}}{1 \cdot 3 \cdot \ldots \cdot (2l+1)}$$

This is equal to (9). The "erf(*)"-function is the Gaussian error integral

$$\text{erf}(z) := \frac{2}{\sqrt{\pi}} \int_0^{z} e^{-t^2} dt$$

C Proof of Theorem 3.2

In order to prove Theorem 3.2, we first have to introduce the *Knapsack* Problem (also called *0-1 Knapsack* problem, see for example the Appendix of [16]):

Definition C.1 (Knapsack):
INSTANCE: Finite set U, for each $u \in U$ a size $s(u) \in Z^+$, a value $v(u) \in Z^+$, and positive integers B and K.
QUESTION: Is there a subset $U' \subseteq U$ such that

$$\sum_{u \in U'} s(u) \leq B \qquad \text{and} \qquad \sum_{u \in U'} v(u) \geq K \quad ?$$

The Knapsack problem is NP-complete [16], even if $s(u) = v(u)$ for all $u \in U$ (*Value-Independent Knapsack* problem). It can be solved in pseudo-polynomial time.

Proof of Theorem 3.2: It is obvious that the problem in Theorem 3.2 is in NP, since if we are given a correct guess $\mathbf{e}_{\mathbf{p}}^{\text{cor}}$ by an oracle machine, we can figure out in $O(N)$ by just evaluating the scalar product, that $|\mathbf{p}^{\mathrm{T}}\mathbf{e}_{\mathbf{p}}^{\text{cor}}|$ is indeed smaller or equal than ϵ.

Secondly, we will give a polynomial reduction from the Value-Independent Knapsack problem. Let $N = |U|$. Introduce the binary variables $e_\nu \in \{-1, 1\}$ with $\nu = 0, 1, \ldots, N$. Let $e_0 = 1$ and

$$e_\nu = \begin{cases} +1 & \text{if } u_\nu \in U' \\ -1 & \text{otherwise} \end{cases} \qquad \nu = 1, 2, \ldots, N$$

Let $p_\nu = s(u_\nu)$ for $\nu = 1, 2, \ldots, N$, let $\epsilon = (B - K)$, and

$$p_0 = \sum_{\nu=1}^{N} s(u_\nu) - K - B$$

This transformation is polynomial in N. We now claim that the Value-Independent Knapsack problem has a solution if and only if the problem of finding $e_\nu \in \{-1, 1\}$ with $\nu = 1, 2, \ldots, N$ so that

$$|\mathbf{e}^{\mathrm{T}}\mathbf{p}| = \left| \sum_{\nu=0}^{N} e_\nu p_\nu \right| \leq \epsilon$$

has a solution. This is our original problem from Theorem 3.2. The fact that the p_ν in the original problem are rational is not a restriction, since it it possible to multiply all p_ν with one positive integer, so that then all p_ν become integers.

Using

$$\sum_{u \in U'} s(u) = \frac{1}{2} \sum_{\nu=1}^{N} (e_\nu + 1) \cdot s(u_\nu)$$

and thus

$$
\begin{aligned}
& 2K \leq 2 \sum_{u \in U'} s(u) \leq 2B \\
\iff \quad & 2K - B - K \leq \sum_{\nu=1}^{N} (e_\nu + 1) \cdot s(u_\nu) - B - K \leq 2B - B - K \\
\iff \quad & -(B-K) \leq \sum_{\nu=1}^{N} e_\nu \cdot s(u_\nu) + 1 \cdot \sum_{\nu=1}^{N} s(u_\nu) - B - K \leq (B-K) \\
\iff \quad & -\epsilon \leq \sum_{\nu=1}^{N} e_\nu \cdot p_\nu + e_0 \cdot p_0 \leq \epsilon \\
\iff \quad & \left| \sum_{\nu=0}^{N} e_\nu \cdot p_\nu \right| \leq \epsilon
\end{aligned}
$$

This proves the claim, and thus we have polynomially reduced the Knapsack problem to the problem of Theorem 3.2. This completes the proof. □

References

[1] H. Harrer and J. A. Nossek, "Discrete-time cellular neural networks," Tech. Rep. TUM-LNS-TR-91–7, Institute for Network Theory and Circuit Design, Technical University Munich, Germany, 1991.

[2] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Trans. on Circuits and Systems*, vol. Vol. 35, pp. 1257–1272, Oct. 1988.

[3] W. A. Little, "The existence of persistent states in the brain," *Math. Biosciences 19*, pp. 102–120, 1974.

[4] P. M. Lewis and C. L. Coates, *Threshold Logic.* Wiley, New York, 1967.

[5] C. Sheng, *Threshold Logic.* Toronto, Ryerson Press; London, New York, Academic Press, 1969.

[6] S. Muroga, *Threshold Logic and its Applications.* Wiley-Interscience, 1971.

[7] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with application in pattern recognition," *IEEE Transactions on Electronic Computers*, vol. EC 14, pp. 326–334, 1965.

[8] N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, eds., *Combinatorial Optimization.* Wiley, 1979.

[9] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, 1. ed., 1982.

[10] P. Nachbar, J. A. Nossek, and J. Strobl, "The generalized adatron algorithm," in *Proc. of the International Symposium on Circuits and Systems*, vol. Vol. 4, (Chicago, Ill., USA), pp. 2152–2156, 1993.

[11] R. G. Parker and R. L. Rardin, *Discrete Optimization.* Academic Press, 1 ed., 1988.

[12] J. H. Ahrends and G. Finke, "Merging and sorting applied to the zero-one knapsack problem," *Operations Research*, vol. Vol. 23, pp. 1099–1109, Nov. 1975.

[13] P. Raghavan, "Learning in threshold networks," in *Proc 1988 Workshop on Computational Learning Theory: COLT'88*, (MIT), Aug. 1988.

[14] E. Gardner, "The space of interactions in neural network models," *Journal of Physics A (Mathematical and General)*, vol. Vol. 21, no. 1, pp. 257–270, 1988.

[15] S. Judd, "On the complexity of loading shallow neural networks," *Journal of Complexity*, vol. Vol. 4, pp. 177–192, 1988.

[16] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness.* W. h. Freeman, 1979.

[17] J. S. Judd, *Neural Network Design and the Complexity of Learning.* MIT Press, 1990.

[18] S. Judd, "Learning in networks is hard," in *IEEE First International Conference on Neural Networks*, vol. Vol. 2, pp. 685–692, 1987.

[19] A. Blum and R. R. Rivest, "Training a 3-node neural network is np-complete," in *1988 Workshop on Computational Complexity Theory COLT'88*, pp. 9–18, 1988.

[20] J.-H. Lin and J. S. Vitter, "Complexity results on learning by neural nets," *Machine Learning*, vol. Vol. 6, pp. 211–230, May 1991.

[21] H. Magnussen and J. A. Nossek, "A way of learning for discrete-time cellular neural networks: The hybrid algorithm," Tech. Rep. TUM-LNS-TR-93–1, Institute for Network Theory and Circuit Design, Technical University Munich, Germany, 1993.

[22] S. Kirkpatrick, C. D. Gelatt, and M. V. Vecchi, "Optimization by simulated annealing," *Science*, vol. Vol. 220, pp. 671–680, May 1983.

[23] F. I. Romeo, "Simulated annealing: Theory and applications to layout problems," *Ph.D. Thesis, University of California, Berkeley*, May 1989.

[24] S. Patarnello and P. Carnevali, "Learning networks of neurons with boolean logic," *Europhysics Letters*, vol. Vol. 4, pp. 503–508, Aug. 1987.

[25] D. E. van den Bout and I. Thomas K. Miller, "Graph partitioning using annealed neural networks," *IEEE Transactions on Neural Networks*, vol. Vol. 1, pp. 192–203, June 1993.

[26] J. Lam and J.-M. Delosme, "Simulated annealing: A fast heuristic for some generic layout problems," in *IEEE International Conference on Computer-Aided Design, ICCAD-88*, (Santa Clara, CA), pp. 510–513, Nov. 1988.

[27] T. M. Kwon, "Threshold net synthesis using simulated annealing," in *IEEE SOUTHEASTCON'91*, vol. Vol. 2, (Williamsburg, VA), pp. 1069–1073, Apr. 1991.

[28] H. Fleisher, J. Giraldi, D. B. Martin, and R. L. Phoenix, "Simulated annealing as a tool for logic optimization in a cad environment," in *IEEE International Conference on Computer-Aided Design: ICCAD-85*, (Santa Clara, CA, USA), pp. 203–205, Nov. 1985.

[29] S. Kollias, "A study of neural network applications to signal processing," in *EURASIP Workshop 1990*, (Sesimbra, Portugal), pp. 233–242, Feb. 1990.

[30] M. E. Johnson, ed., *Simulated annealing (SA) and optimization : modern algorithms with VLSI, optical design, and missle defense applications.* American Sciences Press, 1988.

[31] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculation by fast computing machines," *J.Chem.Phys*, vol. Vol. 21, pp. 1087–?, 1953.

[32] R. H. J. M. Otten and L. P. P. P. van Ginneken, *The Annealing Algorithm.* Kluwer Academic Publishers, 1989.

[33] M. D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *IEEE International Conference on Computer-Aided Design: ICCAD-86*, (Santa Clara, CA), pp. 381–384, Nov. 1986.

[34] G. B. Sorkin, "Efficient simulated annealing on fractal energy landscapes," *Algorithmica*, vol. Vol. 6, no. 3, pp. 367–418, 1991.

[35] P. N. Strenski and S. Kirkpatrick, "Finite lenght annealing schedules," *Algorithmica*, vol. Vol. 6, no. 3, pp. 346–366, 1991.

[36] E. H. L. Aarts and P. J. M. van Laarhoven, "Statistical cooling: A general approach to combinatorial optimization problems," *Philips Journal of Research*, vol. Vol. 40, no. 4, pp. 193–226, 1985.

[37] R. Holley and D. Stroock, "Simulated annealing via sobolev inequalities," *Communications in Mathematical Physics*, vol. Vol. 115, no. 4, pp. 553–569, 1988.

[38] B. Gidas, "Nonstationary markov chains and convergence of the annealing algorithm," *Journal of Statistical Physics*, vol. Vol. 39, no. 1–2, pp. 77–131, 1985.

[39] F. Romeo and A. Sangiovanni-Vincentelli, "A theoretical framework for simulated annealing," *Algorithmica*, vol. Vol. 6, no. 3, pp. 302–345, 1991.

[40] B. Hajek, "Cooling schedules for optimal annealing," *Mathematics of Operations Research*, vol. Vol. 13, no. 2, pp. 311–329, 1988.

[41] L. O. Chua and L. Yang, "Cellular neural networks: Applications," *IEEE Trans. on Circuits and Systems*, vol. Vol. 35, pp. 1273–1290, Oct. 1988.

[42] G. Seiler, "Small object counting with cellular neural networks," in *Proc. International Workshop on Cellular Neural Networks and their Applications CNNA-90*, (Budapest, Hungary), pp. 114–123, Dec. 1990.

[43] H. Magnussen and J. A. Nossek, "Towards a learning algorithm for discrete-time cellular neural networks," in *Proc. Second International Workshop on Cellular Neural Networks and their Applications CNNA'92*, (Munich, Germany), pp. 80–85, Oct. 1992.

[44] T. Sziranyi and J. Csicsvari, "High-speed character recognition using a dual cellular neural network architecture (cnnd)," *IEEE Transactions on Circuits and Systems, II: Analog and Digital Signal Processing*, vol. Vol. 40, pp. 223–231, Mar. 1993.

[45] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA, Biophysics*, vol. Vol. 79, pp. 2554–2558, Apr. 1982.

[46] R. J. McElice, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The capacity of the hopfield associative memory," *IEEE Transactions on Information Theory*, vol. Vol. IT-33, pp. 461–482, July 1987.

[47] S. V. B. Aiyer, M. Niranjan, and F. Fallside, "A theoretical investigation into the performance of the hopfield model," *IEEE Transactions on Neural Networks*, vol. Vol. 1, pp. 204–215, June 1990.

[48] A. N. Michel, J. A. Farrell, and H.-F. Sun, "Analysis and synthesis techniques for hopfield type synchronous discrete time neural networks with application to associative memory," *IEEE Trans. on Circuits and Systems*, vol. Vol. 37, pp. 1356–1366, Nov. 1990.

[49] C. Y. Ho, I. Sasase, and S. Mori, "On the capacity of the hopfield associative memory," in *IJCNN International Joint Conference on Neural Networks*, vol. Vol. 2, (Baltimore,MD), pp. 196–201, June 1992.

[50] H. Harrer and J. A. Nossek, "New test results of a 4 by 4 discrete-time cellular neural network chip," in *Proc. Second International Workshop on Cellular Neural Networks and their Applications CNNA'92*, (Munich, Germany), pp. 163–168, Oct. 1992.

[51] S. A. Cook, "The complexity of theorem-proving procedures," in *Third Annual ACM Symposium on Theory of Computing*, (Association for Computing Machinery, New York), pp. 151–158, 1971.

[52] I. S. Gradshteyn and I. M. Ryzhik, *Table of Integrals, Series, and Products.* Academic Press, 1 ed., 1980.