

# Analyzing EEG Signals Using the Probability Estimating Guarded Neural Classifier

Torsten Felzer and Bernd Freisleben, *Member, IEEE*

**Abstract**—This paper introduces a neural network architecture for classifying feature vectors symbolizing portions (or segments) of an electroencephalogram (EEG) trace of a human subject. This classification task is the one that is typically required when developing a so-called brain–computer interface (BCI), which analyzes the EEG signals of a subject in order to “understand” the subject’s thoughts. However, instead of merely saying which “category of thoughts” (i.e., which class) the respective input feature vector belongs to, the network described here estimates the probabilities of an EEG segment being associated with each individual class. The network, which is called PeGNC (for probability estimating guarded neural classifier), is tested with two kinds of experiments. In the first experiment, the  $\alpha$ -rhythm associated with a human subject closing the eyes is detected online with the help of a frequency-based representation. Since the EEG signal is, in general, always a mixture of numerous action potentials generated simultaneously and it is, thus, very likely that mental activities result in overlapping classes, it is reasonable to believe that the PeGNC network—which does not select any one single class, but determines probability values for each mental category—is particularly suitable for this kind of EEG analysis. The second experiment deals with this issue on the basis of an offline analysis of simulated data.

**Index Terms**—Brain–computer interface (BCI), counterpropagation, electroencephalogram (EEG) analysis, human–computer interaction.

## I. INTRODUCTION

INSTEAD of the standard communication channel between a human being and a computer involving keyboard and mouse, physically disabled individuals who are unable to reliably use their hands depend on alternative solutions. One possibility is given by systems that are based on the analysis of signals related to muscular activity (see, e.g., [1] or [2]). With the newest progress in modern computer technology, the attempt to devise a direct link between the human brain and a computer—which was first mentioned in [3]—became more and more popular over the last 10–15 years. A brain–computer interface (BCI) is a system that analyzes the brain–electrical activity of a subject [e.g., represented by the corresponding electroencephalogram (EEG) signal] and tries to generate appropriate feedback actions, thus enabling the user to communicate with a computer by wilfully altering his or her brainwaves [4]–[8].

In most cases, analyzing the brain–electrical activity typically means translating consecutive segments of the EEG signal into meaningful feature vectors and trying to *categorize* or *classify*

them, often with the help of a neural network classifier. Therefore, the goal is to identify and to recognize recurrent patterns in the EEG signal and to associate those with a limited number of categories or classes.

In general, artificial neural networks (ANNs) may be seen as an implementation of a mapping  $f$  from an  $n$ -dimensional input vector space  $A$  onto an  $N$ -dimensional output space  $B$ . Due to the distributed structure of ANNs, the images  $f(i_1)$  and  $f(i_2)$  are similar (i.e., not too far apart in Euclidean space), if the input patterns  $i_1$  and  $i_2$  themselves are similar.

For classification tasks, it is very common to encode the recognized class in a so-called “1-of- $N$  vector,” which means that one component of the output vector is set to 1 (sometimes, that output component contains a real value close to 1.0), while all other components yield 0 (or values close to 0.0). The interpretation of this is pretty obvious: let there be  $N$  “output classes;” then the output vector marks which one of these the input vector belongs to.

The internals of the mapping  $f$  are “programmed” in an initial *training phase*. The network is trained with a finite number of known example patterns, and the actual operation of the network consists of applying it to a possibly infinite number of unknown input patterns in the *recall phase*.

It is the aforementioned property of preserving similarities that allows the ANN to produce “useful” output for unknown input vectors by generalizing from the patterns in the *training set*, and thus makes ANNs ideal for pattern recognition tasks.

Suppose, for example, the task is to recognize handwritten digits. In this case, the input space  $A$  corresponds to a set of points in the two-dimensional (2-D) Euclidean vector space,  $N = 10$  is equal to the total number of digits and  $B$  is the set of all 1-of-10 vectors.

Let  $i_1$  and  $i_2$  be the two patterns depicted in Fig. 1. The two patterns seem to be sufficiently similar, so that the images  $f(i_1)$  and  $f(i_2)$  should be similar as well (or even identical): in both cases, the digit “9” should be recognized. However, when looking at the patterns in Fig. 2, the situation is not that clear. The pattern in Fig. 2(b) is definitely an “8,” but the one in Fig. 2(a) is either an “8” with a much too short lower left arc or an unfortunate “9” [compare to Fig. 1(b)].

This shows that it is sometimes not enough to only have a 1-of-10 vector indicating “it is a 9” or “it is an 8.” Instead, the output vector should indicate how probable it is that the respective input pattern is an “8” and—simultaneously—how probable that it is a “9.” The architecture introduced here is designed to produce exactly this kind of output vectors.

Since brainwaves consist of a mixture of numerous individual signals, it is very probable that EEG analysis based on the classi-

Manuscript received September 7, 2001.

The authors are with the Department of Mathematics and Computer Science, University of Marburg, D-35032 Marburg, Germany (e-mail: freisleb@informatik.uni-marburg.de).

Digital Object Identifier 10.1109/TNSRE.2003.819785

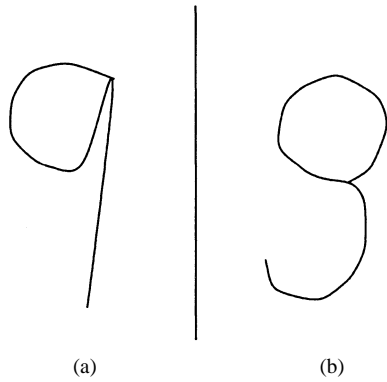


Fig. 1. Two nines?

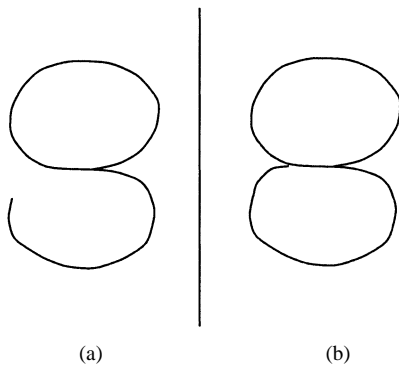


Fig. 2. Two eights?

fication of EEG patterns in a way belongs to the same type of pattern recognition tasks as the example illustrated previously. This means that some particular attributes are always present—only more or less dominant (see also [9]). The input signal would, thus, contain elements of multiple patterns, and rigorously selecting one might be misleading. Besides, that property might even be independent of the input representation, i.e., the way in which the EEG signal is presented to the network.

For instance, one way to represent the EEG signal is to transform the curve into the frequency domain and to take the amplitudes at certain frequencies as components of the input feature vector. The relevant range of brain-electrical activity is divided into  $\alpha$  waves (8–13 Hz),  $\beta$  waves (14–30 Hz),  $\theta$  waves (4–7 Hz), and  $\delta$  waves (0.5–3 Hz) (e.g., [10]). The EEG activity of an awake, healthy subject exhibits a predominance of  $\alpha$  and  $\beta$  waves, and in the sleep, longer waves ( $\theta$  and  $\delta$ ) become more dominant. However,  $\alpha$  and  $\beta$  activity is not completely absent when sleeping—the contribution of this activity is merely reduced.

In this paper, we introduce a novel neural network architecture for analyzing brainwaves, which is called probability estimating guarded neural classifier (PeGNC). It is an extension of the GNC network described in [11] and [12]. The GNC network is basically a forward-only counterpropagation network [13] together with two special neurons eliminating two very common problems of ANN classifiers, but still producing discrete yes/no answers. The PeGNC contains modified output units with additional counters, while the remaining architecture is identical to the GNC network. Therefore, it benefits from the same advantages, but is additionally able to yield much more

precise output values based on the relative frequencies within the training set. Experimental results on EEG classification tasks will be presented to demonstrate the performance of our proposed approach.

This paper is organized as follows. Section II presents previous work on neural classification of EEG signals. In Section III, the architecture of the PeGNC network is described in detail. Experimental results on EEG classification tasks are presented in Section IV. Finally, Section V concludes the paper and outlines areas for further research.

## II. RELATED WORK

Different network architectures and input representations for the automatic recognition of EEG patterns have been developed. For example, Jandó *et al.* [14] examined the use of backpropagation networks for the detection of certain elements (which are fairly easy to recognize visually) in the EEG traces of rats. In their approach, the raw EEG data (sampled at a rate of 100 Hz) is presented to three-layer networks of different sizes, trained with the backpropagation learning algorithm of Rumelhart *et al.* [15]. In addition, the fast Fourier transformation (FFT, e.g., [16]) is used to examine the frequency content of the EEG data.

The best results were obtained with networks containing a single output cell, which is expected to yield the output value “1” if (and only if) the network “thinks” that the input vector contains the requested EEG element (and “0” otherwise). In that case, the classification results were subject to two different kinds of errors— independent of application field, network size, and input representation. On the one hand, the network might detect something, when there really is nothing (also called a false positive). On the other hand, the network might miss the occurrence of the requested pattern (denoted false negative).

The most effective configurations produced merely 1%–7% false negatives and only between 18% and 40% false positives, thus showing the applicability of backpropagation classifiers to the problem of EEG analysis.

Anderson *et al.* [17] use a backpropagation network as part of an entire BCI system to associate the brainwaves of a subject with one of a limited number of mental states. Using a frequency-based input representation, their network was able to accurately classify 73% of untrained samples.

Yet another backpropagation approach is the one of Steuer *et al.* [18], where the network is used to classify different cognitive processes—e.g., related to conceptual and pictorial thinking—with feature vectors based on the activity in the  $\beta$  band and an accuracy of up to about 95%.

A completely different network based on a statistical model is used by Tsuji *et al.* [19] to examine differences in brain-electrical activity caused by opening and closing the eyes and by stimulating the subjects’ eyes with a flash light. The achieved accuracy varied between about 60% and 90% and largely depended on the respective subject.

Pfurtscheller *et al.* [20] used a so-called learning vector quantizer (LVQ) to classify EEG patterns accompanying left and right hand movement (see also [21]). Their network is based on the Kohonen [22] algorithm, and represents a particular kind of a nearest-neighbor classifier: the network contains a number

of adjustable “codebook” vectors, and the output depends on those codebook vectors that are the nearest (in Euclidean sense) to a frequency-based input vector (the basic training and recall mechanisms are very similar to those of the FCPN network described below). The classification accuracy of their system reached 89%–100% for the greater part of the subjects participating in their experiments.

In [23], a distinction sensitive LVQ (DSLVO) is applied to the same kind of movement-related data. The innovative feature of that network is an adjustable weight vector which ultimately emphasizes the more important components of the input feature vector. The network is, thus, able to identify optimum frequency components and electrode positions for the classification task.

The asynchronous signal detector, described in [24] and [25], is designed to distinguish between idle brain states and voluntary control signals—i.e., the goal is to facilitate the use of a BCI aimed at control applications by removing the necessity of constantly producing conscious “do nothing” signals, when the operator really does not want to issue a mental command (which probably is the case most of the time). The heart of the system is again a nearest-neighbor classifier based on vector quantization.

### III. NETWORK ARCHITECTURE

As already mentioned, the PeGNC introduced in this paper is an extension of the GNC network described in [11], [12]. The GNC itself is an extension of a counterpropagation network (CPN) originally developed by Hecht-Nielsen [13].

In addition to the mapping  $f$  from input onto output space mentioned in the introduction, the original CPN approach of Hecht-Nielsen also involves the inverse mapping  $f^{-1}$ . However, since the inverse mapping is of no relevance for either the GNC network or the PeGNC network, this paper will exclusively concentrate on the forward-only variant of the paradigm of counterpropagation networks, so-called FCPNs.

The theoretical foundations, the learning rules, and the mathematical formulae describing the output calculation of the PeGNC network and its predecessors are discussed in the following.

#### A. FCPN

An FCPN is a network comprising three layers of artificial neurons: an input layer (with size  $n$ ); a classification layer (with size  $k$ ); and an output layer (with size  $N$ ) (see Fig. 3).

An artificial neuron is a simple computational unit with several inputs and one output. The unit calculates the output depending on a weighted sum of its inputs, and the output of each unit in one layer serves as input for each unit in the next.

As already mentioned previously, an FCPN may be seen—like any neural network—as an implementation of a mapping from an input space onto an output space. In this case here, it works as follows. Each classification unit  $C_i$  calculates an individual activation  $a_i$  by summing all  $n$  components of the input vector  $\vec{x}$  (which is normalized to length 1.0 in Euclidean space)—each multiplied with a corresponding input weight  $w_{ij}$ , ( $j = 1, \dots, n$ )

$$a_i = \sum_{j=1}^n w_{ij} \cdot x_j, \quad i = 1, \dots, k.$$

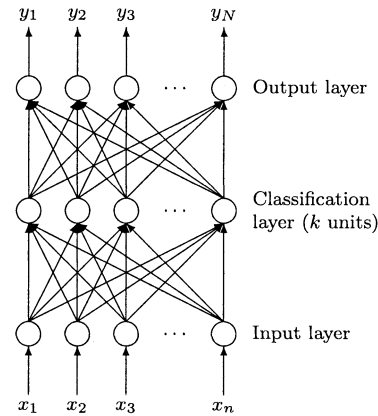


Fig. 3. FCPN.

The classification units then enter a sort of competition as to which one may determine the output of the network. The winner of this competition is the classification unit  $C_{i_0}$  with the highest activation, and the output (or state)  $c_{i_0}$  of that unit is set to “1,” that of all others to “0”

$$c_i = \begin{cases} 1, & \text{if } a_i \geq a_j \forall j \neq i \\ 0, & \text{else} \end{cases} \quad i = 1, \dots, k.$$

The component  $y_j$  of the overall output vector  $\vec{y}$  is simply set to a weighted sum of the inputs of the corresponding output unit, so it is only dependent on the output weight  $q_{ji_0}$  belonging to the winner  $C_{i_0}$

$$y_j = \sum_{i=1}^k q_{ji} \cdot c_i = q_{ji_0}, \quad j = 1, \dots, N.$$

Another interpretation is the following. The vector

$$\vec{w}_i = (w_{i1}, w_{i2}, w_{i3}, \dots, w_{in})$$

combining the weights between a classification unit  $C_i$  and the  $n$  input units will be called input weight vector. Likewise, the vectors

$$\vec{q}_i = (q_{1i}, q_{2i}, q_{3i}, \dots, q_{Ni}), \quad i = 1, \dots, k$$

containing the weights between the classification layer and the output layer will be denoted output weight vectors. In other words, each classification unit  $C_i$  is associated with an  $n$ -dimensional input weight (or position) vector  $\vec{w}_i$  and an  $N$ -dimensional output weight (or simply output) vector  $\vec{q}_i$ . Like the input vector  $\vec{x}$ , the vectors  $\vec{w}_i$ , ( $i = 1, \dots, k$ ) are (initially) normalized to length 1.

Then, the computation of the activation  $a_i$  is simply the calculation of the dot product between the input vector

$$\vec{x} = (x_1, x_2, x_3, \dots, x_n)$$

and the position vector  $\vec{w}_i$ . Since both of these vectors are normalized to length 1.0 in Euclidean space<sup>1</sup>, the dot product is equal to the cosine of the enclosed angle. Therefore, the maximum possible activation is 1.0 (the maximum of the cosine function), and the activation may be seen as a measure of how close the vectors are (where the maximum is only reached in the

<sup>1</sup>Since the weights are adjusted during training without further normalization, this is not totally correct. However, the  $\vec{w}_i$ 's always stay within the unit sphere, and the deviation from normalization is thus negligible.

case of identical vectors—with an enclosed angle of  $0^\circ$ ). If  $i_0$  is the index of the classification unit with the largest activation, then the overall output vector

$$\vec{y} = (y_1, y_2, y_3, \dots, y_N)$$

is set equal to the output weight vector  $\vec{q}_{i_0}$ .

In short, the classification unit whose position vector is *closest to the input vector* “has the right to” determine the output of the network, which is set to the output weight vector of that classification unit. The goal of the initial training phase already mentioned in the introduction is to adjust the weight vectors, so that the mapping outlined previously yields acceptable results.

The training consists of initializing the components of the weight vectors with random numbers, normalizing the  $\vec{w}_i$ s, cyclically presenting pairs of possible input and corresponding target output vectors—comprising the training set—to the network, and adjusting the weights after each presentation.

The weight adjustment involves two steps. First, the input weights are adjusted in an unsupervised, competitive way (see, e.g., [22]) employing the winner-take-all principle. Let  $(\vec{x}_0$  and  $\vec{t}_0)$  be the pair of input and target output vectors presented to the network at any point  $P$  and  $i_0$  be the index of the classification unit whose position vector  $\vec{w}_{i_0}$  is closest to  $\vec{x}_0$  (at time  $P$ ), then  $\vec{w}_{i_0}$  is adjusted according to (1), while all other input weights remain unchanged

$$\vec{w}_{i_0}(P+1) = \vec{w}_{i_0}(P) + \eta \cdot (\vec{x}_0 - \vec{w}_{i_0}(P)) \quad (1)$$

with an appropriate learning rate  $\eta$  (with  $0 < \eta \leq 1$ ).

The geometrical effect of this learning rule is that the “position of the winning classification unit” is moved toward the input vector, so that the activation of the winner has the tendency to be even larger for the pair  $(\vec{x}_0, \vec{t}_0)$  during the next cycle through the training set.

Although the origin of the rule belonging to the second weight adjustment step is Grossberg’s outstar structure [26]—i.e., is based on a supervised learning technique—its structure and its geometrical interpretation are completely analogous. The rule is shown in (2)

$$\vec{q}_{i_0}(P+1) = \vec{q}_{i_0}(P) + \zeta \cdot (\vec{t}_0 - \vec{q}_{i_0}(P)) \quad (2)$$

with an appropriate learning rate  $\zeta$  (with  $0 < \zeta \leq 1$ ).

According to this rule, the output weight vector  $\vec{q}_{i_0}$  of the winning classification unit is moved in the direction of the target output  $\vec{t}_0$ . As the term “winner-take-all” already indicates, the weight vectors belonging to the other classification units remain (again) unchanged.

The magnitude of the “movement” of the weight vectors solely depends on the learning rates  $\eta$  and  $\zeta$ , and since the network’s “expertise” in remembering the associations in the training set is expected to grow as training goes on, the changes should become smaller and smaller. Therefore, the learning rates are initialized at the beginning of the training and decreased after any cycle  $C$  through the training set according to (3)

$$\eta(C+1) = \frac{\eta(C)}{1+\xi}, \quad \zeta(C+1) = \frac{\zeta(C)}{1+\xi} \quad (3)$$

with some constant  $\xi > 0$ .

Training is stopped—i.e., weights are not changed any more—after a preset number of cycles, or when the weight changes become too small (i.e., after the learning rates have dropped below a certain threshold), or when the network memorizes the associations in the training set reasonably well (with only a small deviation of the actual network output from the target output for each training pattern, which means that a certain convergence has been reached).

The outcome of the learning rules is that—after several cycles through the training set—the input space is partitioned into subspaces with the position vectors representing the average (in Euclidean sense) of all trained input vectors belonging to one of those subspaces, and the corresponding output vector being the mean output vector among the associated target outputs. New untrained input vectors activate the classification unit belonging to the subspace they fall into, and the associated output vector is expected to represent the most reasonable classification result.

### B. Introducing Guards

The standard FCPN as described previously has to face two serious problems, one of them referring to the training phase, the other one affecting the recall phase.

First, suppose there are two training pairs  $(\vec{x}_a, \vec{t}_a)$  and  $(\vec{x}_b, \vec{t}_b)$ . If the two input vectors  $\vec{x}_a$  and  $\vec{x}_b$  are sufficiently close to each other—in Euclidean sense—and, thus, belong to the same cluster (or subspace) of the input space, the presentation of either one of those vectors will result in the selection of the same winning classification unit. This causes problems if  $\vec{t}_a$  and  $\vec{t}_b$  are too far apart, because the learning rule for the output weights (2) then moves the corresponding output vector in completely different directions during the same training cycle.

This behavior disturbs the training process considerably, and weight convergence becomes almost impossible. In the general case, the training set contains more than two pairs spanning any one cluster, and most of the target output vectors associated with input vectors belonging to the same cluster are not too distant (which is why generalization works in the first place). A pair containing a target output vector that is very far away from all other output vectors within the respective cluster is called outlier, and the entire phenomenon described here is often referred to as the outlier problem.

The second drawback of the FCPN has to do with misclassifications (i.e., wrong output vectors) due to incomplete and inappropriate training sets. As already mentioned, the effect of the training phase is the clustering of the input space. Unfortunately, the network “assumes” the input space to be spanned by the training set. Therefore, if—in the recall phase—an input vector, which is very far away from all training vectors, is presented to the network, the classification result will most probably be incorrect.

Of course, the network does return an output vector—the one associated with the winning classification unit. However, the activation of each one of the classification units will be very small, and merely selecting the unit with the maximum activation does in this case not lead to a classification that would be based on a certain (i.e., sure) decision. Consequently, the term *uncertainty problem* shall denote this kind of misclassifications.

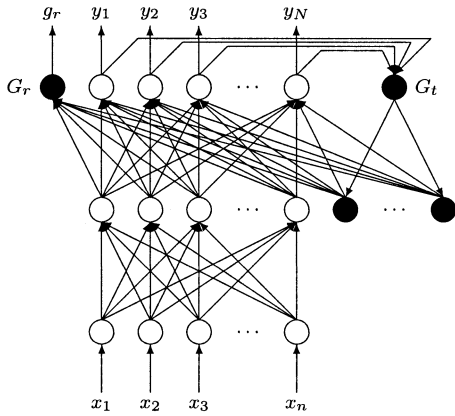


Fig. 4. GNC.

The outlier problem as well as the uncertainty problem are well-known in the ANN community, and the strategy adopted to overcome these drawbacks often decides on the usefulness of a neural network classifier. The GNC network is basically a FCPN with two additional security output neurons or guards—named  $G_t$  and  $G_r$ —which take care of outliers in the training phase and uncertain decisions in the recall phase. Furthermore, the classification layer of the GNC network contains a pool of initially unused (i.e., initially not taking part in the training and competition process), extra neurons (see Fig. 4, additional units drawn in black).

In order to be able to treat outliers, the  $G_t$  guard has to detect them. This detection involves the computation of the error—i.e., the difference between the actual (i.e.,  $\vec{y}_0$ ) and the target output vector for a particular training pair ( $\vec{x}_0, \vec{t}_0$ ). For simplicity, the squared error is used, which is equal to the dot product of the vector ( $\vec{y}_0 - \vec{t}_0$ ) with itself

$$E = (\vec{y}_0 - \vec{t}_0) \cdot (\vec{y}_0 - \vec{t}_0). \quad (4)$$

The definition of an outlier says that the corresponding target output vector is very different from all those belonging to the same input cluster. Therefore, the  $G_t$  neuron “complains” (i.e., outputs a “1”) if the error takes on a value larger than a threshold  $E_{\max}$  during training:

$$g_t = \begin{cases} 1, & \text{if } E > E_{\max} \\ 0, & \text{else.} \end{cases} \quad (5)$$

In such a case, the detected outlier is associated with its “own” classification unit by “opening” a new, previously unused one, whose position vector is set equal to the current input vector and whose output vector gets assigned the target output vector of the current training pair. From this time on, the corresponding new classification unit will be treated exactly as the other (“used”) ones—in particular, it will compete with all other winner-take-all units.

As a consequence, the outlier does not disturb the learning process concerning the respective input cluster any longer, since the new classification neuron will win the competition, when the outlier is presented during the next training cycle. This is because the corresponding activation yields the maximum value of 1.0 (which is the dot product of any unit vector with itself).

The described mechanism is not applied during the first few training cycles, since the weight vectors are initialized with random values, so the error  $E$  will be relatively large for every single training pair (at the beginning), which means that all of them would be regarded as outliers. In order to give the network a chance to sort of “self-initialize,” the state  $g_t$  of the  $G_t$  neuron will not be considered before the completion of a small number of training cycles (on the order of 3–5).

The  $G_r$  guard is responsible for the uncertainty problem in the recall phase. As already stated previously, uncertain decisions are accompanied by relatively small activations of the winning classification unit. In order to be able to assess the magnitude of the activations, the minimum winning activation of all classification units is determined in one additional sweep through the entire training set  $T$  after training has been stopped. In other words, the minimum  $\theta_i$  among the activations  $a_i = \vec{x}_t \cdot \vec{w}_i$  for all  $\vec{x}_t \in T$  that caused classification unit  $C_i$  to be the winner is computed

$$\theta_i = \min_{\vec{x}_t \in T} \{ \vec{x}_t \cdot \vec{w}_i \mid \vec{x}_t \cdot \vec{w}_i \geq \vec{x}_t \cdot \vec{w}_j, j \neq i \}. \quad (6)$$

The computation is done for all  $i \in \{1, 2, 3, \dots, K\}$ , where  $K$  is the total number of used classification neurons (at the end of the training).

The value  $\theta_{i_0}$  then leads to a criterion as to what activations have to be considered as being too small for a winning classification unit  $C_{i_0}$ . The state  $g_r$  of the recall guard will be set to “1” (informing the user that the classification result is not reliable), if the corresponding activation  $a_{i_0}$  is less than (or equal to)  $\theta_{i_0}$  reduced by some tolerance factor  $\tau < 1$  (typically  $\tau = 0.9$ )

$$g_r = \begin{cases} 1, & \text{if } (\vec{x}_r \cdot \vec{w}_{i_0} \geq \vec{x}_r \cdot \vec{w}_i, i \neq i_0) \\ & \wedge (\vec{x}_r \cdot \vec{w}_{i_0} \leq \tau \cdot \theta_{i_0}) \\ 0, & \text{else} \end{cases} \quad (7)$$

where  $\vec{x}_r$  is the input vector presented to the network.

The  $\theta$ -values for (used but) “unidentified” classification units—i.e., those that have never won during the final sweep through the training set—are assigned  $1/\tau$ . This has the effect that  $G_r$  will always consider a decision based on the win of a neuron  $C_{i_1}$  belonging to an “unknown” cluster as “uncertain” (since  $\tau \cdot \theta_{i_1}$  then equals 1.0—which is the maximum possible activation— $g_r$  will thus always yield 1).

### C. Introducing Counters

This subsection presents a variant of the GNC network—the so-called PeGNC network—that is more suitable for classification tasks involving overlapping output classes, such as the analysis of EEG data.

The  $G_r$  guard of the GNC produces an additional binary value that says whether or not the network output is based on a sure decision. Instead, it would be desirable to have a value between 0 and 1, a probability, estimating (or “assessing”) the coherence of the actual output vector and the current input vector, following from the associations in the training set.

Besides, the GNC has a problem when outliers are too “similar” to other patterns. Let there be four “similar” input vectors  $\vec{x}_a, \vec{x}_b, \vec{x}_c$ , and  $\vec{x}_d$  in the training set (see Fig. 5, a schematic sketch projected into 2-D space), which all activate the same classification neuron  $C_{i_0}$ , before the  $G_t$  mechanism is started.

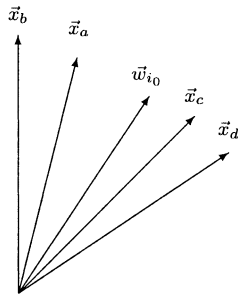


Fig. 5. “Similar” vectors.

The target output vectors associated with the last three will also be “similar”

$$\vec{t}_b \approx \vec{t}_c \approx \vec{t}_d$$

but the target output of the first one  $\vec{t}_a$  is completely dissimilar, so that the corresponding training pattern will be treated as an outlier, once  $G_t$  enters the scene.

Let Fig. 5 depict the situation just before  $G_t$  is started. The vector  $\vec{x}_b$  is further away from the input weight vector  $\vec{w}_{i_0}$  than from  $\vec{x}_a$ .

After the initiation of the  $G_t$ -process, pattern  $(\vec{x}_a, \vec{t}_a)$  is assigned to its own classification neuron with input weight vector  $\vec{x}_a$  and output weight vector  $\vec{t}_a$ . When after that the input vector  $\vec{x}_b$  is presented to the network, this new classification neuron (and *not* the “common” one with index  $i_0$ ) will win the competition, but since

$$\vec{t}_a \not\approx \vec{t}_b$$

as already stated, the guard  $G_t$  will also regard the pattern  $(\vec{x}_b, \vec{t}_b)$  as an “outlier” and “open” a new classification neuron (although  $\vec{x}_b$  really belongs to the same input cluster as  $\vec{x}_c$  and  $\vec{x}_d$ ). The situation might get even worse, if the training set contained a fifth pattern  $(\vec{x}_e, \vec{t}_e)$  with

$$\vec{x}_e \approx \vec{x}_{a,b,c,d}$$

and

$$\vec{t}_a \approx \vec{t}_e \not\approx \vec{t}_{b,c,d}.$$

Admittedly, this example seems to be a bit constructed, and it cannot be experienced in practice too often, but nevertheless, it is imaginable theoretically. Fortunately, the cure for the second problem (too similar outliers) almost naturally leads to a solution for the first (the desired probability values): the trick is to examine not one single winner, but an entire set of winners.

The winner in the GNC network depends on the activations  $a_i, i = 1, \dots, K$ , and a set of winners in the PeGNC network should, thus, do so as well. The winner set  $S$  might be defined as follows. Suppose  $C_{i_1}$  is the classification unit with the largest activation  $a_{i_1}$

$$a_{i_1} \geq a_i \quad \forall i \neq i_1$$

then  $S$  contains all those units with an activation above  $a_{i_1} - \rho$

$$S = \{C_i \mid a_i \geq a_{i_1} - \rho\}$$

with an appropriate precision parameter  $\rho$ .

Instead of classifying a pattern as an outlier if the classification unit with the largest activation has a “wrong” output vector, one now could check all units in the corresponding winner set one by one, starting with the one having the largest activation—in decreasing order. Once a unit with appropriate output vector is found, this one will be declared to be the (“ultimate”) winner (and the weight update equations will only be applied to the weight vectors of that unit). The guard  $G_t$  then merely complains, if neither unit in the winner set has a “correct” output vector.

Employing this procedure,  $(\vec{x}_a, \vec{t}_a)$  will still be treated as an outlier in the example introduced earlier, but the subsequent presentation of  $\vec{x}_b$  will yield the expected behavior, i.e., the classification neuron  $C_{i_0}$  will be declared as the winner, and the pattern  $(\vec{x}_b, \vec{t}_b)$  will, thus, be *no* outlier. Therefore, using a *set* of winners largely reduces the risk of unnecessarily opening new classification units—and it will become clear in the following that the generation of probability values is only a logical extension of that idea.

The further addition to the standard FCPN that is discussed below requires limiting the output vectors to the set  $O$  of 1-of- $N$  vectors

$$O = \{(o_1, o_2, \dots, o_N) \mid \exists! i : o_i = 1 \wedge o_j = 0, \forall j \neq i\}$$

i.e., the set of  $N$ -dimensional vectors with exactly one component equal to 1, whereas all other components are equal to 0—the output vectors thus mark one of  $N$  output classes using the obvious encoding.

The limitation to the set  $O$  (which in practice is actually no limitation, since, for classification tasks, the underlying encoding represents the method of choice anyway) facilitates the training of the output weight vectors considerably. It is now possible to initialize the  $\vec{q}_i$ s with all 0s, i.e.,

$$\forall i : \vec{q}_i(0) = \vec{0}$$

and, since it makes no sense to compute some “mean output vector” when using the encoding underlying  $O$ , (2) may be changed to

$$\vec{q}_{i_0}(P+1) = \begin{cases} \vec{t}_0 \in O, & \text{if } \vec{q}_{i_0}(P) = \vec{0} \\ \vec{q}_{i_0}(P), & \text{else} \end{cases} \quad (2')$$

where  $i_0$  is the index of the winning classification unit.

This makes the parameter  $\zeta$  obsolete, and the training guard  $G_t$  does not need to compute the error any longer: it suffices to just check whether certain vectors are equal. If all elements of the winner set have output weight vectors unequal to the target output vector, then the current pattern will be treated as an outlier (unless, of course, there is an element in the winner set with a “virgin”  $\vec{0}$  output weight vector<sup>2</sup>). The state  $g_t$  of the training guard  $G_t$  (which decides on the question whether or not a formerly unused unit will be opened at point  $P+1$ ) can, therefore, be determined as follows:

$$g_t = \begin{cases} 1, & \text{if } \forall C_i \in S : (\vec{q}_i(P) \neq \vec{t}_0) \wedge (\vec{q}_i(P) \neq \vec{0}) \\ 0, & \text{else.} \end{cases} \quad (5')$$

Furthermore, the discrete output set  $O$  makes it possible to associate an “identified” classification unit  $C_{i_1}$ —i.e.,  $\vec{q}_{i_1} \neq \vec{0}$ —with

<sup>2</sup>In that case, this “unidentified” unit will be declared to be the (“ultimate”) winner.

one particular of a total of  $N$  output classes, so the vector  $\vec{q}_{i_1}$  can simply be represented by an integer number  $q_{i_1}$  between 1 and  $N$  ( $q_{i_1}$  denoting the index of the only component equal to 1 in  $\vec{q}_{i_1}$ ).

Finally, the modified “learning” principle as far as the output weight vectors are concerned makes yet another modification indispensable. The total error will already be 0 after only two or three cycles through the training set, because the input patterns will have already found “their” respective classification neurons, and the output weights will be *equal* to the desired outputs. Therefore, the error dropping below a threshold cannot be used as a convergence criterion any more. Training has to go on, since in general the partitioning of the input space is not finished yet.

The goal of the PeGNC network is to return  $N$  probability values

$$y'_j \quad (j = 1, \dots, N)$$

one for each output class. The computation of those values can be done with the help of counters  $n_i$  ( $i = 1, \dots, K$ ) associated with the used classification units at the end of the training phase. The counter values are determined during the final sweep through the training set  $T$  for the computation of the  $\theta$ -values. The value  $n_{i_0}$  indicates how often the classification unit  $C_{i_0}$  has (“ultimately”) won the competition (with  $0 \leq n_{i_0} \leq |T|$ ).

The relative counter frequency of a class within the winner set ( $S$ , as defined previously) can be used as an estimate for the desired probability value. To compute this frequency, one has to divide the sum of the counter values of the units resulting in the output class in question by the total sum of the counter values of all units in the winner set

$$y'_j = \frac{\sum_{C_i \in S \wedge q_i = j} n_i}{\sum_{C_i \in S} n_i}.$$

The recall guard  $G_r$  (which is still present in the PeGNC network) complains, either if one of the units in the winner set is “unidentified” or not sufficiently activated or if the maximum probability  $y'_{j_0}$  stays below a certain threshold  $\phi$ , or formally

$$g_r = \begin{cases} 1, & \text{if } (\exists C_i \in S : a_i \leq \tau \cdot \theta_i) \\ & \vee (y'_{j_0} \leq \phi) \\ 0, & \text{else.} \end{cases} \quad (7')$$

After determining the maximum output component  $y'_{j_0}$ , one can turn the PeGNC network into a “usual” classification network (though with more detailed output information than when using the “plain” GNC net) by additionally computing the output vector  $\vec{y} \in O$  according to

$$y_{j_0} = 1 \wedge y_j = 0 \forall j \neq j_0.$$

#### IV. EXPERIMENTAL RESULTS

In order to demonstrate that the described classifier is suitable for the analysis of EEG signals, a simple experiment was carried out. The network was trained with frequency-based EEG data originating from one 30-year-old male subject recorded over the occipital lobe—at electrode sites  $O_1$  and  $O_2$  according to the international 10–20 system (see [27])—and sampled at a rate of 256 Hz.

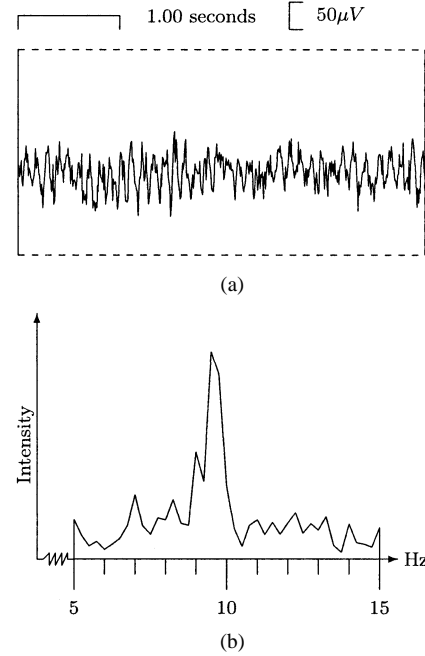


Fig. 6. (a) EEG reading with closed eyes. (b) Spectrum belonging to (a).

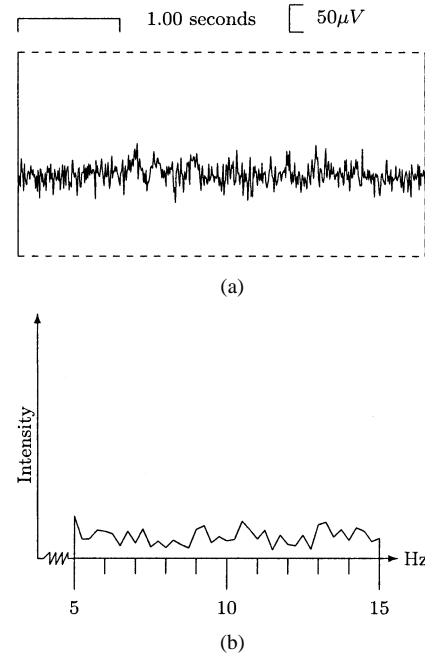


Fig. 7. (a) EEG reading with open eyes. (b) Spectrum belonging to (b).

Half-second portions were transformed into the frequency domain with the help of a fast Fourier transformation, and the Fourier coefficients belonging to the range 5–15 Hz were used to compute the feature vectors. The total number of training samples was 50, and the subject was asked to keep his eyes closed in about half of those samples and to leave them open in the rest—the sequence of “closed samples” and “open samples” was chosen at random. The network’s task was, thus, to distinguish between “closed” and “open”, i.e., the classifier learned to recognize whether or not the subject closed the eyes.

As can be seen in Figs. 6(a) and (b) and 7(a) and (b), the frequency spectrum for “closed samples” clearly differs from

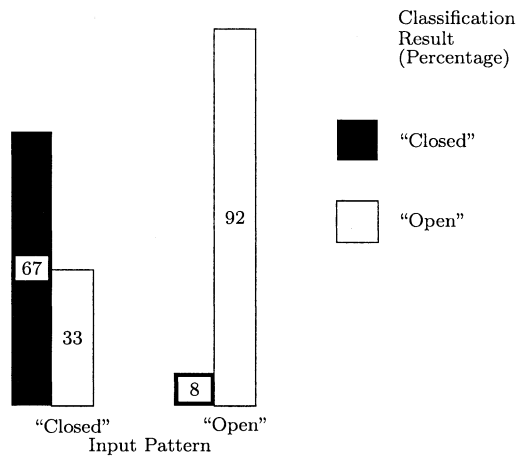


Fig. 8. "Eye-closure" experiment.

that for "open samples," since it reveals a distinct peak at about 10 Hz. The cursor control device described in [28] used this EEG-based signal to switch between two operation modes.

Given the discriminating nature of the training data, it was not surprising that the PeGNC network was indeed able to learn the projected task. To evaluate the performance of the network, new untrained samples were generated "on-line." For this, the spontaneous EEG of the subject was recorded, and the data corresponding to a *sliding window* of length 0.5 s was continuously classified. The feature vectors belonging to 100 randomly selected EEG segments (50 segments recorded while the subject kept his eyes closed and 50 with open eyes) were taken as test samples.

Fig. 8 shows the average percentages returned by the trained network tested with these new samples. The network's output for the "closed" class averaged over 50 test samples recorded with closed eyes was 67%. For "open samples," it was even better: the network returned 92% for the "open" class—again averaged over 50 corresponding test samples.

The superiority of the result for "open samples" is due to the fact that the corresponding feature vectors do not really have a clear orientation (as opposed to the "closed vectors"), so the resulting subspace of the input space representing "open samples" is much larger.

By the way, the network is sure in only about 14% of all 100 trials. The reason for this is the stringent behavior of the  $G_r$  neuron: the recall guard labels a decision as "uncertain" if any *one* member of the winner set is either "unidentified" or not activated large enough. The situation would change a lot (i.e., the "sureness rates" would go up), if  $G_r$  would complain only if all members (or, say, the majority) of the winner set hinted at an uncertain decision.

The values 67% and 92% must not be mistaken for the common true positives (TP) performance measure. The result for the closed samples, for example, does not mean that the network returned the correct class in only 67% (of 50 samples). One has to consider that the PeGNC network does not return just one class as network output. Rather, the network returned probabilities for each output class in every single trial, and the average probability for all 50 "closed samples" was 67% for the "closed" class.

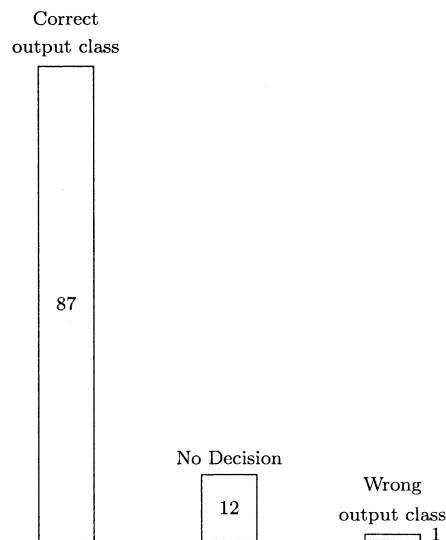


Fig. 9. "Thresholded" result.

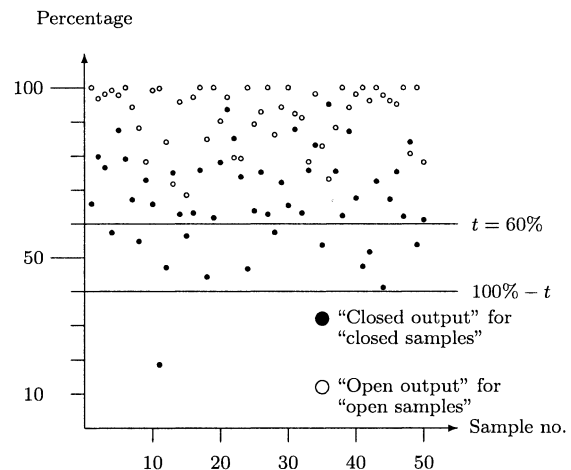


Fig. 10. Distribution of all test samples.

The PeGNC output may be compared to that of "usual" (e.g., nearest-neighbor) classifiers by ignoring the  $G_r$  guard and assigning a threshold of e.g.,  $t = 60\%$ —meaning that an output class would be declared to be the sole classification result, if the PeGNC network returned a probability of 60% or more for that class. As illustrated in Fig. 9, this would result in only 1% misclassifications and merely 12% unclassifiable samples in the experiment illustrated above (distribution see Fig. 10).

The figures outlined previously exemplify a good result, not a revolutionary one. Any decent nearest-neighbor classifier would most probably yield equally good results (especially the discriminating nature of the input data justifies such a statement). The true power and superiority of the PeGNC network is revealed in experiments involving more than two, overlapping classes, where it does not suffice to only pick one single class as classification result.

One example for that (second) type of experiments can be derived as an extension from the eye-closure experiment. The feature vectors underlying the rather elementary first experiment for two typical example patterns (one for each output class) are

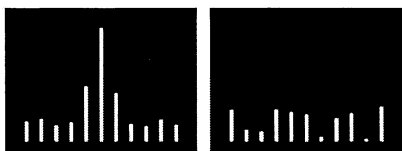


Fig. 11. Two feature vectors.

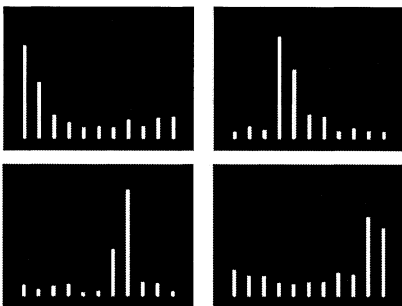


Fig. 12. Four “hills.”

shown in Fig. 11. After a certain preprocessing and data compression, those 11-dimensional vectors contain the frequency contributions between 5 and 15 Hz.

If we define a collection of two or three adjacent vector components with much larger amplitudes than its environment as a hill and the rest as valley, we can say that the left pattern of Fig. 11 contains one “hill” around the center and “valleys” anywhere else. Extending the idea of “hills” and “valleys” might lead to the four patterns depicted in Fig. 12, each having its “hill” at a different place.

Now, let us assume that these four patterns represent four different classes, and that it were relatively easy to produce a larger set of training patterns for those classes. One possible interpretation might be the realization of a 2-D cursor control device (see also [29]) where classes 1 and 2 are responsible for horizontal and classes 3 and 4 for vertical cursor movement.

Of course, supporting a binary information—like in the eye-closure experiment—suffices for a BCI, but since translating complex relationships into sequences of yes/no decisions is awfully slow, that solution represents merely the minimum requirement. Therefore, the goal in BCI research should always be to find multidimensional communication channels, like in the “four-hills problem” described earlier. It is exactly here where it becomes obvious why using the PeGNC network is necessary or at least advantageous.

Theoretically spoken,<sup>3</sup> presenting a neural network classifier trained with a set of training patterns describing the “four-hills problem” (with each pattern belonging to one of four classes) with a feature vector like the one shown in Fig. 13 at recall time raises interesting questions about the possible network output.

A “usual” neural network classifier, e.g., an instance of a FCPN classifier, tries to associate the input feature vector with

<sup>3</sup>The considerations concerning the “four-hills problem” are entirely hypothetical, because the mental activities allowing to produce that kind of clear and distinct class-specific patterns, together with an independent control over multiple dimensions, have yet to be found.

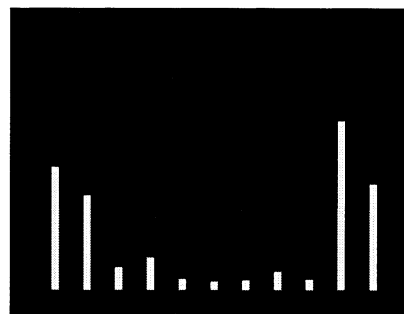


Fig. 13. “Overlapping pattern.”

the class that belongs to the subspace of the input space that fits best. So the output will only be one class, at best class 1 or 4 in the example above. Since Fig. 13 is by definition totally different from all training patterns “seen” before, it is even possible that the network is totally confused and returns one of the other two classes as classification output (all this depends on the concrete training phase).

The GNC network would probably notice that the pattern in Fig. 13 is too dissimilar from all training patterns, so its  $G_r$  guard would label the network output as “uncertain.” Discarding those input patterns—which would be the consequence of such an uncertain decision—is definitely better than simply returning one (possibly wrong) class. However, this does not make too much sense when the quota of patterns to be discarded is too high, which probably is the case for EEG-dependent data.

On the other hand, it is not too difficult to give the pattern in Fig. 13 a meaningful interpretation. The pattern somehow “overlaps” those for classes 1 and 4, so the result should be a combination of both, i.e., (in the words of a cursor control device) the cursor should move vertically and horizontally at the same time. And since both “hills” are almost equally high, the cursor movement should be diagonal with a slope of about 45°.

All this can be realized with the help of the PeGNC network, which is designed to return a probability of about 50% for the two classes 1 and 4, respectively. In fact, an implementation of the PeGNC network actually returned the output vector

$$(0.48, 0.0, 0.0, 0.52)$$

in a test run based on computer-generated data.

In this second experiment, the trained network was tested with a set of 200 patterns: 50 patterns containing one “hill” and 150 patterns containing two “hills” (like the one shown in Fig. 13). However, the training set consisted of 200 patterns with only one “hill” each. In order to illustrate the multidimensional result, the 200 “trials”—i.e., pairs of input and resulting actual output vectors—were visually inspected and subjectively classified into “acceptable” and “not acceptable” trials. The network performance is summarized in Fig. 14.

The input patterns with only one “hill”—which were new to the network but which corresponded to the training patterns concerning general structure—yielded acceptable output vectors without any exception (i.e., in all 50 cases). This perfect result was accompanied by the fact that the network was very sure about its decisions—in 94% of the “one-hilled” trials. Evidently, accuracy rates of that kind cannot be expected, when the

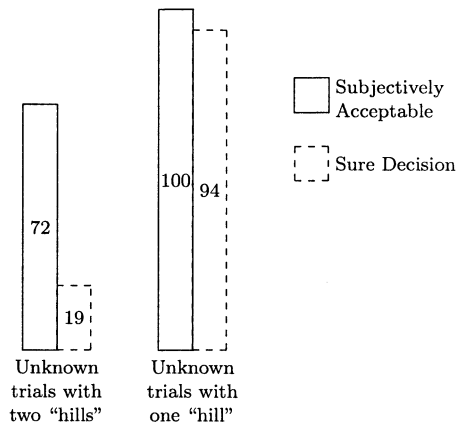


Fig. 14. Network performance in experiment #2.

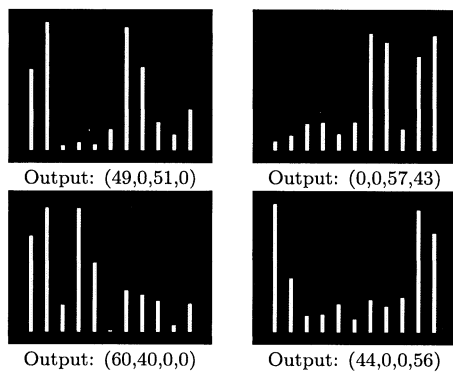


Fig. 15. "Acceptable" trials.

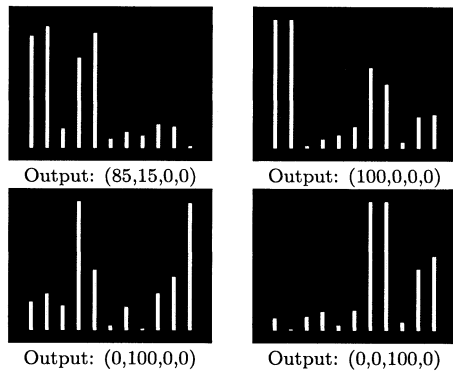


Fig. 16. Problem-causing patterns.

network is confronted with input patterns that differ in structure from all training patterns—it is no surprise that the network did not mark its decisions as uncertain in only 19% of all trials with two "hills." Although not all output vectors in those trials were perfect, it can be noted that, in total, the PeGNC returns much better results than "usual" classifiers. The output vectors in 72% of the 150 "two-hilled" trials were subjectively acceptable (examples shown in Fig. 15).

Trials in which the trained network detects merely one of two "hills" or largely overestimates one of two approximately equally high "hills" are not acceptable, and Fig. 16 presents some of those 28% problem causing patterns together with their respective output vectors.

## V. CONCLUSION

A neural network classification architecture for EEG analysis has been presented. The network estimates how probable it is that a certain input feature vector belongs to a particular output class, and since its structure is based on the GNC network, it is called PeGNC.

The PeGNC contains a set of input weight vectors, each representing a particular subspace of the input space. Every single subspace coincides with (exactly) one output class, while the number of subspaces is, in general, greater than the number of output classes. This means that each output class may be associated with several subspace representatives.

Partitioning the input space is done by adjusting the weight vectors in an initial training phase, which consists of repetitive sweeps through a set of example patterns, the training set.

The probability estimations depend on the relative frequencies following from the training set. After the iterative weight adjustment process, it is counted how many input patterns in the training set belong to each subspace. In the recall phase—involving the presentation of new unknown input vectors to the network—a set of subspaces (also called winner set), which most probably cover the input vector, is determined by means of looking at the respective distances between the input vector and the subspace representatives.

Dividing the counter sum of those subspaces associated with one particular output class by the counter sum of all subspaces in the winner set yields the aforementioned relative frequency for the corresponding output class. The frequency values are solely dependent on the counters, which themselves depend on the partitioning of the input space. Therefore, the quality of the network output follows directly from how well the input space is covered by the training set.

Since the network does not return one single class as the classification result, but probability values relating the input vector to every single class, it is ideal for tasks involving overlapping classes, i.e., tasks where classifying the input vector as belonging to only one class would lead to unreliable results.

It is very probable that the classification of EEG segments belongs to this latter type of tasks, regardless of the input representation, because the EEG signal is always a mixture of numerous individual signals—variations in the feature vectors, thus, originate from the fact that certain elements are sometimes more, sometimes less dominant. Therefore, the PeGNC is suitable for using it in a BCI, which is based on the analysis and the classification of EEG patterns.

The network was evaluated with the help of two simple experiments. The first experiment was based on real-world EEG data, and the classifier had to distinguish between only two classes describing whether or not a subject kept his eyes closed. A thresholded implementation of the network yielded 87% correct classifications with only 12% unclassifiable samples and 1% misclassifications. The second experiment was an extension of the first one based on simulated (or computer-generated) data with four classes. The results of this second experiment demonstrated the superiority of the PeGNC network (over "usual" approaches) for tasks with overlapping classes.

There are two open problems to be solved in the future. The immediate goal is related to integrating the PeGNC network

into a complete BCI system, which really means building a BCI “around” the net, since the classification of the EEG patterns is the “heart” of any brainwave-based BCI (a preliminary solution is described in [30]). The second objective refers to employing this newly built system in a “trial-and-error” fashion in order to identify mental activities, which produce “good” classification results, while still allowing comfortable human-computer communication—possibly yielding patterns with properties as ideal as in the hypothetical example of the “four-hills problem.” This particularly includes the search for ways to fully benefit from the superiority of the PeGNC network as far as a certain type of tasks is concerned.

REFERENCES

[1] T. Felzer and B. Freisleben, “Controlling a computer using signals originating from muscle contractions,” in *METMBS’02—Proceedings of the 2002 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*. Las Vegas, NV: CSREA, 2002, vol. 2, pp. 336–342.

[2] —, “HaWCoS: The “hands-free” wheelchair control system,” in *ASSETS 2002—Proceedings of the Fifth International ACM SIGCAPH Conference on Assistive Technologies*. Edinburgh, Scotland, U.K.: ACM, 2002, pp. 127–134.

[3] J. Vidal, “Toward direct brain-computer communication,” *Annu. Rev. Biophys. Bioeng.*, pp. 157–180, 1973.

[4] M. Polak and A. Kostov, “Development of brain-computer interface: Preliminary results,” in *Proc. 19th Annu. Int. Conf. IEEE Engineering in Medicine and Biology Soc.*, 1997, pp. 1543–1546.

[5] J. R. Wolpaw, N. Birbaumer, W. J. Heetderks, D. J. McFarland, P. H. Peckham, G. Schalk, E. Donchin, L. A. Quatrano, C. J. Robinson, and T. M. Vaughan, “Brain-computer interface technology: A review of the first international meeting,” *IEEE Trans. Rehab. Eng.*, vol. 8, pp. 164–173, June 2000.

[6] G. Pfurtscheller, C. Neuper, C. Guger, W. Harkam, H. Ramoser, A. Schlögl, B. Obermaier, and M. Pregenzer, “Current trends in Graz brain-computer interface (BCI) research,” *IEEE Trans. Rehab. Eng.*, vol. 8, pp. 216–219, June 2000.

[7] W. D. Penny, S. J. Roberts, E. A. Curran, and M. J. Stokes, “EEG-based communication: A pattern recognition approach,” *IEEE Trans. Rehab. Eng.*, vol. 8, pp. 214–215, June 2000.

[8] T. Felzer, “Verwendung verschiedener Biosignale zur Bedienung computergesteuerter Systeme,” Ph.D. dissertation, Dept. Comput. Sci., Darmstadt Univ. Technol., Darmstadt, Germany, 2002.

[9] J. d. R. Millán, J. Mouriño, M. G. Marciani, F. Babiloni, F. Topani, I. Canale, J. Heikkonen, and K. Kaski, “Adaptive brain interfaces for physically-disabled people,” in *Proc. 20th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, 1998, pp. 2008–2011.

[10] R. Klinke and S. Silbernagl, *Lehrbuch der Physiologie*, 2nd ed. Stuttgart, Germany: Georg Thieme Verlag, 1996.

[11] B. Freisleben, M. Hoof, and R. Patsch, “Using counterpropagation neural networks for partial discharge diagnosis,” *Neural Comput. Applicat.*, vol. 7, no. 4, pp. 318–333, 1998.

[12] M. Hoof, “Pulse-sequence-analysis: A new method of partial discharge diagnosis,” Ph.D. dissertation (in German), Dept. Elect. Eng. Comput. Sci., Univ. Siegen, Siegen, Germany, 1997.

[13] R. Hecht-Nielsen, “Counterpropagation networks,” *Appl. Opt.*, vol. 26, pp. 4979–4984, 1987.

[14] G. Jandó, R. M. Siegel, Z. Horváth, and G. Buzsáki, “Pattern recognition of the electroencephalogram by artificial neural networks,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 86, pp. 100–109, 1993.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, vol. 1, pp. 318–362.

[16] W. Törmig and P. Spellucci, *Numerische Mathematik für Ingenieure und Physiker*, 2nd ed. Berlin, Germany: Springer-Verlag, 1990, vol. 2, Numerische Methoden der Analysis, ch. 11, Interpolation und Approximation.

[17] C. W. Anderson, S. V. Devulapalli, and E. A. Stolz, “EEG as a means of communication: Preliminary experiments in EEG analysis using neural networks,” in *Proc. 1st Annu. ACM Conf. Assistive Technologies (ASSETS)*, Marina del Ray, CA, 1994, pp. 141–147.

[18] D. Steuer, B. Schack, G. Grieszbach, and W. Krause, “Classification of human cognitive processes by the use of an improved neural backpropagation network,” in *Proc. IEEE Int. Conf. Neural Networks*, vol. 5, 1995, pp. 2495–2500.

[19] T. Tsuji, O. Fukuda, H. Ichinobe, and M. Kaneko, “A log-linearized Gaussian mixture network and its application to EEG pattern classification,” *IEEE Trans. Syst., Man, Cybern. C, Applicat. Rev.*, vol. 29, pp. 60–72, Feb. 1999.

[20] G. Pfurtscheller, J. Kalcher, C. Neuper, D. Flotzinger, and M. Pregenzer, “On-line EEG classification during externally-paced hand movements using a neural network-based classifier,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 99, pp. 416–425, 1996.

[21] J. Kalcher, D. Flotzinger, S. Göllly, C. Neuper, and G. Pfurtscheller, “Graz brain-computer interface (BCI) II,” in *Computers for Handicapped Persons*, W. L. Zagler, G. Busby, and R. R. Wagner, Eds. Berlin, Germany: Springer-Verlag, 1994, ICCHP’94, pp. 170–176.

[22] T. Kohonen, *Self-Organization and Associative Memory*. Berlin, Germany: Springer-Verlag, 1989.

[23] M. Pregenzer and G. Pfurtscheller, “Frequency component selection for an EEG-based brain to computer interface,” *IEEE Trans. Rehab. Eng.*, vol. 7, pp. 413–419, Dec. 1999.

[24] G. E. Birch and S. G. Mason, “Brain-computer interface research at the Neil Squire foundation,” *IEEE Trans. Rehab. Eng.*, vol. 8, pp. 193–195, June 2000.

[25] S. G. Mason and G. E. Birch, “A brain-controlled switch for asynchronous control applications,” *IEEE Trans. Biomed. Eng.*, vol. 47, pp. 1297–1307, Oct. 2000.

[26] S. Grossberg, “Embedding fields: A theory of learning with physiological implications,” *Math. Psych.*, vol. 6, pp. 209–239, 1969.

[27] H. Jasper, “The ten twenty electrode system of the international federation,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 10, pp. 371–375, 1958.

[28] D. W. Patmore, W. L. Putnam, and R. B. Knapp, Assistive cursor control for a PC window environment: Electromyogram and electroencephalogram based control. presented at *Proc. Center on Disabilities Virtual Reality Conf.* [Online]http://www.csun.edu/cod/conf/1994/proceedings/Wec-1.htm

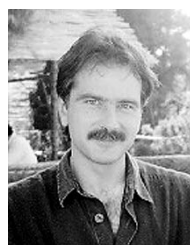
[29] J. R. Wolpaw and D. J. McFarland, “Multichannel EEG-based brain-computer communication,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 90, pp. 444–449, 1994.

[30] T. Felzer and B. Freisleben, “BRAINLINK: A software tool supporting the development of an EEG-based brain-computer interface,” in *METMBS’02—Proceedings of the 2002 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*. Las Vegas, NV: CSREA, 2002, vol. 2, pp. 329–335.



**Torsten Felzer** received the M.Sc. degree in computer science from the University of Colorado at Boulder in 1994, and the Dipl. degree in computer science and the Ph.D. degree from Darmstadt University of Technology, Darmstadt, Germany, in 1996 and 2002, respectively. His dissertation was on the use of biosignals for control applications.

His research interests include neural networks, EEG analysis, BCI design, and human-computer interaction.



**Bernd Freisleben** (M’98) received the M.S. degree in computer science from The Pennsylvania State University, State College, in 1981, and the Ph.D. degree in computer science from Darmstadt University of Technology, Darmstadt, Germany, in 1985.

He is currently a Full Professor of computer science with the Department of Mathematics and Computer Science, University of Marburg, Marburg, Germany. His research interests include computational intelligence, biomedical signal processing, and scientific computing.