Sleep classification in infants by decision tree-based neural networks

Irena Koprinska¹, Gert Pfurtscheller *, Doris Flotzinger

Ludwig Boltzmann Institute of Medical Informatics and Neuroinformatics and Department of Medical Informatics, Institute of Biomedical Engineering, Graz University of Technology, Brockmannagasse 41, A-8010 Graz, Austria

Received 3 July 1995; accepted 7 December 1995

Abstract

This paper presents an AI-based approach to automatic sleep stage scoring. The system TBNN (Tree-Based Neural Network) uses a decision-tree generator to provide knowledge that defines the architecture of a backpropagation neural network, including feature selection and initialisation of the weights. The case study reports a successful application to the data from polygraphic all-night sleep of 8 babies aged 6 months. The teaching input was provided by a medical expert in accordance with the rules of Guilleminault and Souquet. The performance of TBNN is compared with 5 other methods and the results are discussed.

Keywords: Sleep classification; Neural networks; Decision trees; Knowledge-based neural networks

1. Introduction

The classification of sleep stages is an important issue in medical diagnosis because some serious diseases are accompanied by typical sleep disorders. Different physiological signals such as electroencephalogram (EEG), electrooculogram (EOG), electromyogram (EMG), and sometimes also respiration, heart rate, blood pressure, body movement and temperature, etc., have to be recorded during night sleep [10,21,26]. These data are visually scored by an expert and the results displayed in the form of a
hypnogram - a graph describing how the individual sleep stages change throughout the night, e.g., sampled in intervals of 30 s (see Fig. 1). The sleep stage scoring (sleep classification) is based on the rules of Rechtschaffen and Kales [26] for adults and infants over 1 year, and on the rules of Guilleminault and Souquet [7] for younger infants.

Sleep scoring and the construction of a hypnogram is a very laborious and time-consuming task: the paper from an 8 h EEG recording is over 400 m long [29]. It can require several hours of hard work by highly qualified specialists. Moreover, human expertise, which can vary between individual experts, is also mostly inconsistent over such periods of continuous work. Kemp et al. [12] report that the agreement between 6 human experts does not exceed 75%. Hence, a program capable of automatic sleep staging is highly desirable.

Sleep classification in infants is especially difficult because in early infancy EEG, cardiovascular and motor activities change due to the maturation of the nervous system. In addition, transient EEG phenomena, such as sleep spindles and K-complexes in EEG, which are dominant in sleep stage 2, are not found in the first months of life [19]. Reports on automatic sleep scoring on data recorded in the first year of life are, therefore, rare. On the other hand, such classification is very important when studying the Sudden Infant Death Syndrome (SIDS) that has become the major reason for death in early infancy.

Different types of analysis of polygraphic sleep data are possible: spectral analysis of frequency bands [10,14,20], waveform recognition [6,8,30], autoregressive modelling [18,31,32] and model-based sleep analysis [12,13].

Quite recently, investigations of the potential of Neural Networks (NNs) for sleep scoring have been reported. Principe and Tome [23] have tested single- and multi-layer perceptrons (MLP, see [27]). Pfurtscheller et al. [22] and Flotzinger and Pfurtscheller [5] have studied a MLP with one hidden layer and also Learning Vector Quantization (LVQ, see [15]). Schaltenbrand et al. [28] have described an automatic sleep scoring system which also uses MLP.

Even though they have provided encouraging classification results, NNs have several well-known shortcomings:

1. Topology determination, i.e., the decision about the number of code vectors for LVQ, and, in the case of MLP, the decision about the number of hidden units and layers,
The way of connectivity and the initial weights. The initial topology can greatly affect
2. A trained NN is a ‘black box’, i.e., it is very difficult to determine exactly why a NN
makes a particular classification. Because of that, experts are not confident in the
reliability of networks and tend to distrust their classification results.
3. NNs are not able to focus attention on significant features and to discard poor, noisy
and redundant ones. Weak features entail bad performance [33]. On the other hand,
the number of features available for sleep classification can become very high and
this means a high complexity of the respective NN classifier and a lot of computa-
tional time for the NN training. Hence, it is desirable to select those features that
contribute most to classification accuracy, keeping their amount at the minimum (and
also, thus, minimizing the number of measurements on the patient).
4. MLPs suffer from the danger of sticking to local minima and they learn quite slowly
[9,11].
5. Randomly initialised NNs require a lot of training examples. This is proved not only
experimentally but also by the results of the computational learning theory. A
substantial number of training examples is needed for probably approximately correct
(PAC) learning because of the big Vapnik-Chervonenkis dimension (VCD) of MLPs:
[1,17].
These limitations can be overcome if some approximation of the target concept is
available. It could be in the form of rules supplied by a domain expert, or alternatively,
induced by a machine-learning system, such as the induction of Decision Trees (DTs).
This paper describes a Tree-Based Neural Network (TBNN) that maps a decision tree to
a neural net which is then further trained by the backpropagation algorithm. Extensive
experiments show that TBNN is a successful approach for sleep stage scoring. It
overcomes the above mentioned weaknesses and compares favourably to other methods
in terms of classification accuracy, speed and stability.

The next section presents the TBNN algorithm which is then applied to data from 8
infants aged 6 months and compared with other well-known AI systems.

2. The system TBNN

The general outline of TBNN is summarised in Table 1.

2.1. Numerical decision tree generation

For the induction of the decision tree we reprogrammed the famous algorithm ID3
[24]. ID3 creates a decision tree where in each internal node a test of a single feature
leads the path down the tree towards a leaf containing a class label. Tests on numerical
attributes yield an infinite number of possible outcomes. This problem is handled by
partitioning of the attribute’s range of values into 2 subranges and thus treating
numerical attributes as binary variables. To find the best split of the range of attributes
we used the technique suggested by Fayyad and Irani [3]. The result is a binary tree such
as the one depicted in Fig. 2a. In the sequel, the tree-generating part of TBNN will be
Table 1
Overview of the TBNN algorithm

1. Take a subset of the training examples and induce from them a decision tree.
2. Translate the tree into a NN:
   (a) Each class is mapped to one output unit. Each interval imposed on the attribute value by the tests along the path from root to a leaf is mapped to 1 input unit and each path is mapped to 1 hidden unit by connecting the respective input units (regular links).
   (b) Transform the original example description so that the new attributes give the membership of the original attributes into the respective intervals.
   (c) Apart from the regular links determined by the translation process in (a), provide additional links to make the net fully forward-connected.
   (d) Calculate the initial thresholds and weights for the regular links so that the net encodes the tree as closely as possible. Set the weights along the additional links to small random values.
3. Train the network by the backpropagation algorithm using all training examples.

referred to as NID3 (Numerical ID3). However, any other numerical tree-generating system such as C4.5 [25] can be used.

2.2. Network topology

The principle of TBNN's tree-to-net translation is illustrated by the simple example depicted in Fig. 2. As each path from the root to a leaf in the tree corresponds to a classification rule, the tree in Fig. 2a can be rewritten to the 3 mutually exclusive formulae given in Fig. 2b. The formulae in Fig. 2b are disjunctive normal forms, in which atoms are the tests for whether the given attribute value falls into a specific interval of the attribute value range. Denotation of the intervals by lower-case letters as indicated in Fig. 2c facilitates a more concise formulation of the logical expressions, as shown in Fig. 2d. These are then simplified to the form in Fig. 2e and immediately translated into the network topology in Fig. 2f using the correspondence given in Table 2. Regular links between hidden and output units are always positive while the regular links from the input to the hidden layer can be negative or positive, in the case of negated or non-negated intervals, respectively. Dotted lines stand for negative regular connections, solid lines represent positive regular connections. Additional links (the thin lines in the picture) are provided so that each hidden unit is connected to each input and output unit.

2.3. Interval fuzzification

The original attributes are translated into interval membership values. Thus, in the example from Fig. 2, attribute \( a_1 \) from the tree in Fig. 2a is represented by the intervals \( a \) and \( b \) in the net in Fig. 2f. The binary interval-membership function can lead to somehow dogmatic behaviour of the network, a value in the middle of an interval is treated equally to a value close to the boundary. This shortcoming can be alleviated by a simple method for grading the interval membership.

We impose the following constraints on the resulting interval-membership function \( m_i \): the largest value of \( m_i \) is in the middle of the interval and decreases towards the interval boundaries; the more distant the value is from the interval, the smaller the value of \( m_i \).
TBNN first determines the closeness of the attribute value to the interval center:

\[ C_i = \frac{R_i - 2|\mu_i - \chi|}{2R_i} \]  

(1)

where \( \mu_i \) is the middle point of the \( i \)th interval, \( R_i \) is the size of the interval, and \( \chi \) is the actual value of the related attribute. The closeness, \( C_i \), is then subjected to the
Table 2
Correspondence between the rules extracted from the DT and the NN

<table>
<thead>
<tr>
<th>Rules</th>
<th>Neural Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>intervals</td>
<td>input units</td>
</tr>
<tr>
<td>classes</td>
<td>output units</td>
</tr>
<tr>
<td>disjuncts</td>
<td>hidden units</td>
</tr>
<tr>
<td>dependencies</td>
<td>regular links</td>
</tr>
</tbody>
</table>

sigmoid $m_i = 1/(1 + e^{-kC_i})$ of the interval-layer neurons. This mechanism ensures that even if the attribute value $x$ lies behind the interval boundary (e.g., due to noise in the measurements) the example still gets a ‘second chance’ at the next layer.

2.4. Initial weights and biases

Consider the $i$th non-input unit with $n$ positive and $m$ negative regular links (note that for the output layer $m$ is 0). Suppose that $k$ additional links with small random weights from the interval $[-\epsilon, \epsilon]$ have been added to the unit (Fig. 3). Weights along positive regular links to the $i$th hidden unit are set to $w_{i\alpha}$; weights along negative regular links to the $i$th hidden unit are set to $-w_{i\alpha}$; weights to the $i$th output unit are set to $w_{i\beta}$. We denote the thresholds for hidden and output units by $t_{i\alpha}$ and $t_{i\beta}$, respectively.

Note that for finite input values, the sigmoid never reaches 0 or 1. Hence, the notion of the active state (equivalent to output 1 in the case of a step function) must be defined by thresholds. From now on, we will say that the neuron is active if its output is $f(inp) > A$, where $inp$ is the weighted sum of inputs and $A \in (0.5; 1]$ is a user-defined constant. Conversely, the neuron is inactive if $f(inp) < 1 - A$. All other states will be considered undefined.

The neuron will be active if the weighted sum of its inputs exceeds some critical value $inp = \psi$. Solving the equation $A = 1/(1 + e^{-kC_i})$ for $\psi$, we obtain $\psi = \frac{1}{k} \ln \frac{A}{1 - A}$.

Fig. 3. Inputs of a hidden or output neuron in TBNN.
Due to the symmetricity of the sigmoid function, the neuron will be inactive if the weighted sum of its inputs is below $-\psi$.

If the hidden layer is to model the original conjunction of intervals in the presence of additional links, then the following 3 conditions must be satisfied:

(a) The $i$th hidden unit must be active if all positive regular inputs are active (even with the minimum value of $A$) while all negative regular inputs are inactive (even with the maximum value of $1 - A$) and even if all additional links are maximally negative.

(b) The $i$th hidden unit must become inactive if at least one positive regular input is inactive (equal to $1 - A$) and all other positive regular inputs have maximum values (equal to 1), while all negative regular inputs are inactive (equal to 0) even if the additional links provide maximally positive inputs.

(c) The $i$th hidden unit must be inactive if all positive regular inputs are maximally active (equal to 1) and at least one negative regular input is active (equal to $A$), even if all other negative regular inputs are zero and if the additional links provide maximally positive inputs.

These 3 requirements are summarised by the following 3 inequalities:

$$n * A * \omega_{i\alpha} + m * (1 - A) * (-\omega_{i\alpha}) - k * \epsilon - t_{i\alpha} \geq \psi$$

$$1 * (1 - A) * \omega_{i\alpha} + (n - 1) * 1 * \omega_{i\alpha} + m * 0 * (-\omega_{i\epsilon}) + k * \epsilon - t_{i\alpha} \leq -\psi$$

$$n * 1 * \omega_{i\alpha} + 1 * A * (-\omega_{i\alpha}) + (m - 1) * 0 * (-\omega_{i\alpha}) + k * \epsilon - t_{i\alpha} \leq -\psi$$

Turning these inequalities into equations and solving them for $w_{i\alpha}$ and $t_{i\alpha}$, we obtain (note that the last 2 inequalities in Eq. (2) are equivalent):

$$w_{i\alpha} = \frac{2(\psi + k\epsilon)}{A(n + m + 1) - (n + m)}, \quad t_{i\alpha} = \frac{(\psi + k\epsilon)A(n + m + 1) + (n - m)}{A(n + m + 1) - (n + m)}$$

If the output layer is to model the original disjunction of leaves in the presence of additional links, then the following 2 conditions must be satisfied:

(a) The $i$th output unit must become active if at least 1 of the hidden units along the regular links is active (at least equal to $A$), even if all other regular links provide zero input and if the additional links provide maximally negative inputs.

(b) The $i$th output unit must be inactive if all previous units along the regular links are inactive with the maximum value $1 - A$ even if the additional links are maximally positive.

These requirements can be formally described as follows:

$$1 * A * \omega_{i\beta} + (n - 1) * 0 * \omega_{i\beta} - k * \epsilon - t_{i\beta} \geq \psi$$

$$n * (1 - A) * \omega_{i\beta} + k * \epsilon - t_{i\beta} \leq -\psi$$
Turning them into equations and solving them for $w_{i\beta}$ and $t_{i\beta}$, we obtain:

$$w_{i\beta} = \frac{2(\psi + k\epsilon)}{A(n+1) - n}$$

$$t_{i\beta} = (\psi + k\epsilon) \frac{n - A(n-1)}{A(n+1) - n}$$  \hspace{1cm} (5)

If the weights and thresholds are set to the values given by Eqs. (3) and (5), the network will simulate the behaviour of the original DT.

2.5. Backpropagation tuning

TBNN uses the backpropagation algorithm with momentum function as described by Rumelhart and McClelland [27]. The weights are updated according to the following formula:

$$\Delta w_{ij}(t+1) = \eta \cdot \delta_j \cdot o_i + \mu \cdot \Delta w_{ij}(t)$$

where $w_{ij}(t)$ is the weight on a link from unit $i$ to unit $j$ at time $t$, $\delta_j$ is the error of unit $j$, $o_i$ is the activation of unit $i$, $\eta$ is the learning rate and $\mu$ is the momentum constant.

To prevent the network from overtraining, a simple mechanism (suggested by Hecht-Nielsen in [9]) is implemented: the training examples are divided into 2 subsets: 1 for the network training (2/3 of the training examples) and one for on-line testing (1/3 of training examples). The latter set is called training test set. After each run through the training examples (i.e., after each epoch), the system tests the performance on the training test set. The training stops when the classification accuracy on the training test set declines, even if the mean square error on the training set still improves.

3. Application domain

3.1. Subjects and data acquisition

The subjects were 8 babies aged 6 months: BR, KR, RA, KL, BU, PR, GO and FR. For each of them, an 8 h night-sleep recording of 22 biological signals was available. The data were digitised and stored on an optical disk. In addition, different parameters from EEG, ECG and respiratory signals were computed and estimated (details on data acquisition and processing can be found elsewhere [22]).

Based on these pre-processed data, the following 15 parameters were selected for further analysis: respiratory variability from nasal signal (ATM1), respiratory variability from thoracic signal (ATM2), EEG (C3-A1) 1–4 Hz power (EEG11), EOG, heart rate (HR), heart rate variability (HRV), actogram of the left hand (MT1), actogram of the right hand (MT2), integrated EMG, Hjorth parameters (activity, mobility, complexity) from EEG1 (ACT1, MOB1, COM1) and Hjorth parameters from EEG2 (ACT2, MOB2, COM2).

Visual sleep scoring by the expert was based on raw data (read from the optical disk and displayed on a monitor) and included: EEG (2 channels), EOG, EMG, ECG, respiration (nostril and thorax) and actograms from the left and the right arm. As infants
The following 7 classes were distinguished: (1) technical artefact (TA, no possibility to score a sleep stage because of missing signals or unsatisfactory quality); (2) movement (MT); (3) wakefulness (W); (4) REM (rapid eye movement); (5) sleep stage 1; (6) sleep stage 2; (7) sleep stages 3/4.

Therefore, from the AI perspective, the task was to recognize 1 of 7 sleep stages from the examples characterised by 15 numerical attributes. Each data set contains circa 779–960 vectors (examples) normalised into the interval [0; 1]. The distribution of the classes in the individual data sets is shown in Table 3.

### 3.2. Experimental results

This section is subdivided into 2 parts that focus on: (i) the general performance of TBNN for sleep stage scoring and (ii) the classification of the individual stages.

The number of examples is not very high considering how many attributes and classes are involved. Therefore, for the evaluation of the results we used the random subsampling strategy, as suggested, for instance, by Weiss and Kapouleas [33]. Each file was randomly split into 2 non-overlapping subsets, one for training (2/3 of the examples) and one for testing (1/3 of the examples). The experiments were repeated 10 times for different splits and the results were averaged. This methodology is known to provide very good error estimates.

#### 3.2.1. General performance of TBNN

The results are summarised in Table 4. In all experiments with TBNN the learning rate was set to 0.1, the momentum was set to 0.5 and the neuroactivation constant, $A$, was set to 0.92 or 0.94. In addition to MLP, 3 more well-known AI-systems are used as references to which the performance of TBNN is compared; these systems are CN2, C4.5 and LVQ.

#### 3.2.1.1. Systems used for comparison. CN2 [2] is a public domain program that combines the efficiency of ID3 with the flexible search strategy of AQ-family algorithms. The output of the program is an ordered set of if-then rules. CN2 is provided with a special mechanism for the handling of noisy data.
Table 4
Classification accuracy achieved by various systems on sleep stage scoring

<table>
<thead>
<tr>
<th></th>
<th>MLP</th>
<th>CN2</th>
<th>NID3</th>
<th>C4.5</th>
<th>C4.5rules</th>
<th>LVQ</th>
<th>TBNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR</td>
<td>60.0±2.8</td>
<td>72.8±1.7</td>
<td>77.3±3.0</td>
<td>78.1±1.8</td>
<td>71.6±1.7</td>
<td>81.0±1.5</td>
<td>85.5±3.6</td>
</tr>
<tr>
<td>KR</td>
<td>39.7±5.3</td>
<td>56.7±2.6</td>
<td>64.0±1.6</td>
<td>61.7±3.8</td>
<td>62.2±2.3</td>
<td>66.5±2.5</td>
<td>68.3±4.9</td>
</tr>
<tr>
<td>RA</td>
<td>68.3±11.0</td>
<td>75.6±2.4</td>
<td>77.5±1.5</td>
<td>76.8±1.9</td>
<td>75.7±3.0</td>
<td>78.5±2.5</td>
<td>80.8±6.0</td>
</tr>
<tr>
<td>KL</td>
<td>72.6±1.4</td>
<td>70.4±3.3</td>
<td>77.3±1.7</td>
<td>76.6±2.1</td>
<td>75.5±2.6</td>
<td>78.1±1.4</td>
<td>80.2±1.7</td>
</tr>
<tr>
<td>BU</td>
<td>66.4±7.1</td>
<td>63.4±7.9</td>
<td>65.5±4.6</td>
<td>69.7±7.3</td>
<td>69.4±1.8</td>
<td>77.9±1.5</td>
<td>73.1±1.8</td>
</tr>
<tr>
<td>PR</td>
<td>62.8±3.0</td>
<td>61.8±1.7</td>
<td>63.9±2.6</td>
<td>64.2±2.4</td>
<td>63.4±2.7</td>
<td>65.9±2.1</td>
<td>66.8±2.0</td>
</tr>
<tr>
<td>GO</td>
<td>56.8±3.8</td>
<td>54.7±1.8</td>
<td>63.0±2.2</td>
<td>61.1±1.9</td>
<td>62.0±1.7</td>
<td>65.8±2.4</td>
<td>66.3±2.7</td>
</tr>
<tr>
<td>FR</td>
<td>73.4±4.6</td>
<td>74.9±2.0</td>
<td>75.5±1.8</td>
<td>78.8±2.8</td>
<td>78.5±2.0</td>
<td>79.9±1.5</td>
<td>81.3±1.8</td>
</tr>
<tr>
<td>Average</td>
<td>62.5</td>
<td>66.3</td>
<td>70.5</td>
<td>70.9</td>
<td>69.8</td>
<td>73.6</td>
<td>75.3</td>
</tr>
</tbody>
</table>

C4.5 is an advanced version of ID3. In particular, it uses an improved criterion for the best attribute selection and a more sophisticated method of probability estimation - for a detailed description see [25]. The system C4.5 includes an option that turns the tree into rules that are further tuned, which can bring further improvement. The results of this option are shown under the heading C4.5rules.

LVQ is an approach introduced by Kohonen [15]. The essence is to create a few prototypes for each class and then to classify the unseen examples by means of the nearest-neighbour principle. Learning consists of adjustment of the positions of the prototypes. LVQ is widely accepted as a very powerful numeric-data classifier. The results of LVQ were obtained using the public-domain package LVQPack.

In the case of MLP, a randomly initialised fully-connected MLP with a single hidden layer was used. We experimented with various numbers of hidden neurons (including the same number as generated by TBNN) and also with different learning rate and momentum constants. The best results are used for the comparison.

4. Discussion

For a fair comparison, exactly the same 10 randomly selected and different splits of training and testing sets were used for all approaches. Table 4 shows that TBNN provides the best classification accuracy. Note that TBNN and MLP use only 2/3 of the training examples for the backpropagation tuning, because the other 1/3 of the training examples is used as a training test set to prevent overtraining.

The NID3 results mentioned in the table pertain to the situation when the trees were generated from all training examples. Our experiments with NID3 show that the more examples are used for the tree induction, the better is the classification accuracy on the testing set, i.e., NID3 results in the table are always better than those of the NID3-module of TBNN. Hence, Table 4 indicates that TBNN outperforms not only its NID3-initialisation module but also the NID3 algorithm run on all training data. This is achieved by overcoming some shortcomings of DTs:
(i) by the tree-to-net translation the system frees itself from the fixed order of the attribute-value tests;
(ii) using interval fuzzification it escapes from the crisp thresholds imposed in traditional DTs;
(iii) by the linear discriminant function implemented by each node of the network (the opposite to the single attribute test in the tree) and the backpropagation training, the network can find better and simpler decision borders using obliquely separating hyperplanes in contrast with the piecewise linear and parallel-to-the-axes decision boundary created by the DT;
(iv) the network can achieve even more compact representation because it does not cover examples in mutually exclusive fashion and it can learn intermediate concepts and abstractions.

TBNN achieves a classification accuracy that is 4–28% higher than the accuracy of MLPs. An important observation is that TBNN is also faster than MLP (it takes dozens of epochs to train TBNN in comparison with thousands for MLP). This performance is due to the decision tree initialisation which makes it possible to start backpropagation training from a point that is already close to a 'good' solution, i.e., if not close to the global minimum at least close to some acceptable local minimum. This does not only reduce requirements in terms of learning epochs but it also helps avoiding many traps in the form of local minima and saddle points. Moreover, the decision tree initialisation substantially cuts down the number of examples required for successful PAC learning, because DTs appear to have much smaller VCD than randomly initialised MLPs [1]. Last, but not least, as the approach provides straightforward initialisation of the neural net, the user saves a lot of improvisation and parameter settings.
TBNN was not only superior to MLP but also slightly better than all decision tree versions and LVQ. However, in this case, it has to be kept in mind that public domain software packages have been used, apart from TBNN. The slightly better performance of LVQ compared with ID3 and C4.5 was also reported by Kubat et al. [16].

The tree-generating module of TBNN can be viewed as a feature selection mechanism: only those parameters that appear in the tree are mapped in the network. Fig. 4 shows the average values for parameter appearance in the DTs. As can be seen, EOG, ACT2, EEG11, ATM1, ATM2, MOB1, MT1, HR and HRV are found to be the most informative features. It is interesting to mention that similar results have been obtained using genetic algorithms (GAs) as a feature selector [4]. GAs select most often the following 5 parameters: EOG, HR, MT1, MOB1 and ACT2.

The EOG was the most informative parameter for sleep classification, followed by the parameter activity (ACT1 and ACT2) which represents the total power of the EEG. The EMG had only a medium impact on sleep scoring. This can be explained by the fact that good quality EMG recordings in infants are hard to obtain. At the same time, it is of interest to note that the actograms from the hands (MT1 and MT2), which include similar information to the EMG, contributed to the classification process. It is of further interest that, in addition to the conventional signals EOG, EEG and EMG, parameters from the respiratory (ATM1 and ATM2) system and the cardiovascular system (HR, HRV) have an important role for automatic sleep stage scoring in infants of 6 months.

4.1. Classification of the individual stages

A detailed study has shown that some sleep stages are easier to recognize based on the available parameters. For example, Fig. 5 represents the classification of each individual sleep stage for the subject KL. The lowest value of percentage agreement between TBNN and the expert is observed in stage 1. This is quite natural because sleep stage 1 is defined as a transition from wakefulness to the other sleep stages and experts are usually inconsistent in its classification. On the other hand, the highest values were obtained in TA, REM and stage 2. It should be noted, however, that TBNN classification of the individual stages depends on the number of examples within the stages. For instance, actogram (MT) for the babies BR and FR, as well as technical artefacts (TA) for KR and FR are distinguished with a very low accuracy due to the small number of examples (see Table 3).

As one of the important sleep stages in babies is REM (see Introduction), we were especially interested in the ability of our program to correctly classify this particular
stage. Fig. 6 gives the performance of TBNN in the two-class problem; REM versus non-REM stages. The percentage of non-REM patterns in each data set (see column 9 of Table 3) can be viewed as the lower bound of the ability of TBNN to discriminate between REM and non-REM stages. The comparison shows that results are not very promising and indicates that the selected parameters are not sufficient for the correct detection of the REM phase. The results were unsatisfactory and reached about 92% accuracy in only 2 of the subjects (PR and FR).

5. Conclusions

This study demonstrates the promise of a novel AI-based approach to sleep stage scoring. A decision-tree inductive algorithm is used to provide an initial approximation of the solution before the neural network training. This technique makes it possible to start the backpropagation from a 'good' point that reduces the number of training epochs and avoids local minima. Moreover, the decision-tree-growing algorithm prunes out many irrelevant attributes and provides straightforward initialisation of the network.

The method was successfully applied for automated off-line analysis of the all-night sleep in infants aged 6 months. The experiments provide evidence supporting the statement that TBNN offers the following advantages: (i) high classification accuracy with a modest number of training examples and (ii) acceptable computational requirements.

In our future work we intend to devote more attention to the translation of the trained network into symbolic rules that can be understood and analyzed by humans more easily. Another extension includes using expert knowledge for the initialisation of the neural network in addition to the decision-tree induction. We are also currently investigating the possibility of constructing a sleeper-independent classifier.

Acknowledgements

The medical data were recorded, pre-processed and classified by an expert under a grant sponsored by the agency ‘Fonds zur Förderung der wissenschaftlichen Forschung’

![Graph showing performance of TBNN for discrimination of 2 classes: REM versus non-REM.](image)
(Project S49/03). The first author was supported by a fellowship from the Austrian Federal Ministry of Science and Research, 'Büro für Austauschprogramme mit Mittel- und Osteuropa'.

References


