Design and Optimization of Neuro-Fuzzy-Based Recognition of Musical Rhythm Patterns

Tillman Weyde

Research Department of Music and Media Technology

University of Osnabrück

Osnabrück, Germany

tweyde@uos.de

Klaus Dalinghaus

Institute of Cognitive Science

University of Osnabrück

Osnabrück, Germany

kdaling@uos.de

Abstract   The task of recognizing patterns and assigning rhythmic structure to unquantized musical input is a fundamental one for interactive musical systems and for searching musical databases since melody is based on rhythm. We use a combination of combinatorial pattern matching and structural interpretation with a match quality rating by a neuro-fuzzy system that incorporates musical knowledge and operates on perceptually relevant features extracted from the input data. It can learn from relatively few expert examples by using iterative training by relative samples. It shows good recognition results and the used methods of pre-filtering and optimization facilitate efficient computation. The system is modular, so feature extraction, rules, and perceptual constraints can be changed to adapt it to other areas of application.

Key words: rhythm, neural nets, fuzzy logic, segmentation, similarity, optimization

INTRODUCTION

Finding musically meaningful units and determining their relations is crucial for musically sensible user interaction. Cooper and Meyer already stated: "To experience rhythm is to group separate sounds into structured patterns."[1] Music theory and music psychology have determined features that are of importance in the perception and cognition of music. Although there is a large body of research on the properties of auditory temporal pattern perception (see e.g. [2], [3], and [4]), a coherent paradigm or theoretical framework to support computer models and applications has not yet been established.

Musical Pattern Processing

There are two main aspects of rhythmic structure which need to be considered for recognition and analysis: *segmentation* and *similarity*. The segmentation process divides a sequence of events into groups or –musically speaking– motifs. The similarity of groups determines the internal structure of a sequence or the relation to another sequence, e.g. a given rhythm and its performance by a user or different variations of a rhythm.

Existing theories of musical similarity determine similarity ratings for motifs (e.g. [5]), but they do not name the individual differences of notes which is important for interactive or analytical applications. Approaches like musical string-matching (e.g. [6]) count differences on a per-note basis, but they do not take into account gradual differences of individual notes. An integrated model which accounts for gradual and structural similarities on a note-by-note basis is needed. Musically meaningful structures are constrained by the auditory perception of music, which should be taken into account.

Integrating Knowledge and Learning

A way of modelling uncertain and incomplete knowledge is needed for musical computation, since our knowledge of musical structures and processes shows more and more to be incomplete and vague as we try to solve musical tasks with computers. Besides using musical knowledge it should also be possible to let a model learn from data, since human experts can perform musical tasks well and the implicit knowledge in their actions should be used.

There have been some approaches to interpret rhythm with neural networks by Linster [7] and by Roberts [8]. They had limited success for reasons of representation of temporal data to the network and of inflexible and hardly extensible systems due to a 'black-box' approach to neural nets. On the other hand purely algorithmic approaches (e.g. [9], [10],

and [11]) show a lack of flexibility and robustness.

In this paper an algorithmic framework, a learning method and experimental results are presented. It integrates existing knowledge from music psychology as well as music theory and learning from data with interpretable results and a musically meaningful and differentiated system output.

SYSTEM ARCHITECTURE

The initial motivation for this work was to bring more musical intelligence into a music tutorial application where the users should try to play back rhythms presented to them. Our aim is to give differentiated and musically meaningful responses to the users about how their input differs from the presented rhythm and what should be improved.

Our system currently supports three modes: segmentation only, matching patterns and structural comparison. The segmentation mode emulates segmentation by a listener. The pattern matching mode simulates the recognition of a single rhythmic group. It assigns the notes in the matched patterns and yields thus detailed information on the differences of the patterns. The interpretation mode combines and extends the former two. It assigns groups and allows a detection and description of structural changes like omissions, insertions or changed order of groups as well as differences on the note level.

The general processing scheme used in this system starts with the combinatorial generation of segmentations, assignments of groups, and assignments of notes within assigned groups. The combination of segmentations and assignments of groups and notes in the groups is called an *interpretation*. Features motivated by music theory and music psychology are extracted from interpretations and a rating based on these features is calculated by a neuro-fuzzy system. The interpretation receiving the best rating is used for determining system output.

The combinatorial generation of interpretations causes a problem, since the number of interpretations grows exponentially with sequence length. One way to reduce complexity is filtering interpretations to remove perceptually implausible interpretations at each processing stage. We also use a Branch-and-Bound optimization to further increase efficiency. The system architecture is shown schematically in figure 1.

Musical Rhythm Patterns

Input data   The Input data represent musical notes based on the MIDI data (Musical Instrument Digital Interface). Three values of the input events are used: onset time, duration, and key velocity (loudness). If there is a task sequence to which the input is to be compared, the task is encoded in the same way.

Segmentation and group assignments   There are perceptual constraints on the length (number of notes) and the temporal duration of groups. The number of events in a group is restricted as well known since Miller's 'magical seven'[12]. The maximum group length is an adjustable parameter in our system and a setting of 4 or 5 has shown to be adequate which agrees with the literature (see [13] and [14]).

Empirical evidence suggests that durations of perceptual groups lie in a range of approximately 0.5 to 2 seconds ([15]). It is also known that temporal proximity of events is an important factor in segmentation, i.e. relatively long distances between events tend to end a group (see [16]). Since grouping by temporal proximity is dominant over accent grouping (see [2]) we can filter out segmentations that grossly contradict grouping by proximity. We also assume that groups containing only one element should not occur unless they have considerable distance to the neighbor notes (see [10]), else they are filtered out. We model these constraints by generating all segmentations within the given range of group lengths and testing the generated interpretations, if they do not meet the constraints.

On the interpretation level input groups are assigned to task groups based on input and task segmentation. All possible assignments for all combinations of segmentations are calculated. Group assignments are filtered out, if the difference of group length exceeds a limit. We tentatively set this limit to 2, which ensures with a maximum group length of 5 that at least half of the notes are assigned.

Note assignments and tempo variants   In a match of input and task groups the notes

of the input group are assigned to notes of the task group. Notes that are not assigned are marked as additional (leaving the temporal relations of the other notes unchanged) or inserted (the rest of the group being moved by the amount of time occupied by note). Task notes are marked as assigned, subtracted, and removed respectively.

Tempo variants are calculated for every assigned input group based on pairs of assigned notes as anchor points. Different possibilities for the choice of the two anchor notes yield different tempo variants. For every tempo variant transformations by translations, dilatations, additions, insertions, subtractions and deletions are performed until the best match is found. The transformations determine feature values like tempo relation, relative position, and structural measures for extra and missing notes.

The transformations are shown schematically in figure 2. The result allows measuring the deviation of the group from the expected position, the tempo deviation, and deviations of the individual notes concerning timing, loudness and length. Only the tempo variant which produces the best similarity rating is used for further calculation in order to reduce calculation time. These features are used as input for the rating system.

Feature Extraction

On both, the group level and structure level, similarity and segmentation features are extracted that form the basis for interpretation rating. For segmentations we calculate ratings based on the groups' lengths and number of notes. For duration we use four input nodes which reach their maximum activation for group durations spaced approximately even on a log scale (0.4, 0.7, 1.2, and 2.0 seconds). This design allows good fitting to a preference curve on the considered range of group lengths. The same design is used for the number of notes in a group. We have five different input nodes corresponding to group lengths from one to five. The regularity of group lengths as well as of group intervals (distance between the beginning of groups) is also rated by calculating their variance. For each group we have a node representing whether the inter-onset-interval (IOI) between the last note and the next group is larger than IOIs within the group, and whether the first note is relatively loud, which perceptually indicates the beginning of a group (see [3]).

For similarity on the group and interpretation level, nodes representing different kinds of imprecision and incorrectness are used as input for the Fuzzy-Prolog program. On the group level we use nodes for early notes, late notes, too loud notes, too soft notes, too long notes, too short notes, which contribute to precision rating. Added notes, inserted notes, subtracted notes, deleted notes are features contributing to a rating of correctness. Group position and tempo stability are calculated relative to the previous group and tempo plausibility is based on spontaneous and preferred tempo studies ([17], [18]), more details on the calculations can be found in [19].

A feature specific for the interpretation level is a rating for the order of groups. This is calculated by dividing the actual number of intersections in the group assignment graph by the number of maximally possible intersections:

$$\frac{i}{n \cdot (n - 1)/2} \tag{1}$$

where i is the number of the intersections in the assignment graph and n the number of group assignments (see figure 3). The other nodes on the interpretation level combine the corresponding group-level nodes.

Combinatorial Considerations

The generation of interpretations causes a problem, since the number of interpretations grows exponentially with sequence length. We reduce complexity by filtering interpretations according to perceptual constraints at each processing stage using Branch-and-Bound optimization to further increase efficiency.

Since we do not allow overlapping segments there are 2^{n-1} possibilities of segmenting a sequence of length n. For given upper and lower bounds of group length the possible number of segmentations can be calculated recursively. Let S be a note sequence of length n. Given minimal length min and maximal length max the number of segmentations seg

can be computed recursively in the following way:

$$seg(n) \;=\; \begin{cases} \sum\limits_{i=n-max}^{n-min} seg(i) & \text{if } n > max \\[2mm] 1 + \sum\limits_{i=1}^{n-min} seg(i) & \text{if } min \le n \le max \\[2mm] 0 & \text{if } n < min \end{cases} \qquad (2)$$

$$\text{where } n, min, max \in \mathbb{N}_+$$

$seg(n)$ grows exponentially with n. min and max have also considerable influence, since they determine the number terms in the sum for each n. The influence of min is greater than that of max, since the upper bound of the summation index i is more important because of the growth of $seg(n)$. The values for sequence length and values of min and max are shown in table .

For the number of possible assignments na of two sequences I, T with given segmentations Sg_i, Sg_t containing $i = |Sg_i|, t = |Sg_t|$ groups the following holds:

$$na(t, i) = (t + 1)^i \qquad (3)$$

The basis is $t + 1$ because every input group can be assigned to a task group or nothing. So the number $int(T, S)$ of all possible interpretations for sequences T and I can be calculated as:

$$int(T, S) = \sum_{j=1}^{seg(|T|)} \sum_{k=1}^{seg(|I|)} na(|Sg_j(T)|, |Sg_k(I)|), \qquad (4)$$

where $Sg_j(T)$ and $Sg_k(I)$ iterate through alle possible segmentations of T and I. If we set $|T| = |I| = n$ and assess $|Sg_j|$ and $|Sg_k|$ as $n/(MinGrpLen + 1)$, we get a rough approximation:

$$int(T, S) \approx seg(n)^2 \cdot \left(\frac{n}{MinGrpLen + 1} + 1 \right)^{\frac{n}{MinGrpLen + 1}}. \qquad (5)$$

This yields with $min = 1$ for $l = 5$ ca. $6{,}000$ and for $l = 10$ already approximately

$1{,}600{,}000{,}000$ interpretations. The number of interpretations to be rated grows fast with l and for each interpretation all note assignments and tempo variants need to be calculated. Therefore, without filtering and optimization of the selection process a interactive application in near real-time is not possible.

EXTENDED FUZZY-PROLOG

Fuzzy-Prolog

The module for rating segmentation quality and group similarity is based on an extension of *Fuzzy-Prolog* as described by Nauck, Klawonn, and Kruse [20]. Fuzzy-Prolog is a programming language based on *Prolog* which operates on truth values in the interval $[0, 1]$. A Prolog program consists of a set of rules and facts in the following forms:

$$\text{conclusion} \;\leftarrow\; \text{premise}_1 \wedge \ldots \wedge \text{premise}_n.$$
$$or \qquad\qquad (6)$$
$$\text{conclusion} \;\leftarrow\; .$$

In a Fuzzy-Prolog program a truth value is assigned to every rule and every proposition (premise or conclusion). If the truth values $[\![\varphi]\!]$ and $[\![\psi]\!]$ of two propositions φ and ψ are given, the truth value of the conjunction of these propositions can be evaluated as

$$[\![\varphi \wedge \psi]\!] = f_\wedge([\![\varphi]\!], [\![\psi]\!]) \qquad (7)$$

using the evaluation function $f_\wedge$ of a fuzzy operator with values in the interval $[0, 1]$. For the evaluation of implications the *Goguen-Implication* is being used:

$$[\![\varphi \to \psi]\!] = \begin{cases} \dfrac{[\![\psi]\!]}{[\![\varphi]\!]} & \text{if } \psi < \varphi, \\[3mm] 1 & \text{else}. \end{cases} \qquad (8)$$

A Fuzzy-Prolog program is a set of rules of the form $\varphi \to \psi$ and facts of the form $\to \psi$

that have a non-zero truth value assigned. These values should not be interpreted as the actual truth values of the propositions, but should be regarded as their lower bounds. This is because a higher truth value of a proposition may be derived by using the *modus ponens* generalized for Fuzzy-Prolog:

$$\frac{\varphi, \quad \varphi \rightarrow \psi \quad (\alpha, \beta)}{\psi \qquad (\alpha \cdot \beta)} \tag{9}$$

where α and β denote the truth values of φ and ψ.

Operators

Fuzzy-Prolog can be used with various fuzzy-logical operators which are defined by their evaluation function. Usually these functions are defined as t-norms and t-conorms (see e.g. [21]). Yet t-norms are not differentiable, and we want to allow compensation between the operands, which is not possible with t-norms and t-conorms. Since we need an operator that is differentiable for backpropagation and we want the results of the operator to be independent of the number of operands, we defined an operator we call the q-operator which has an adjustable parameter q and is related to the class of operators Yager presented in [22]:

$$\mu_{\otimes,q}(\alpha_1, \ldots, \alpha_n) = \left(\frac{1}{n}\sum_{i=1}^{n}\alpha_i^q\right)^{\frac{1}{q}} \qquad q > 0 \tag{10}$$

For $q = 1$ it calculates the average value of the operands. In the limit of $q \rightarrow 0$ the operator converges to the minimum operator $\top_{min}$. In the limit of $q \rightarrow \infty$ it approximates the maximum operator $\bot_{min}$. Thus we can define the following fuzzy logical q-operators for conjunction and disjunction with $q \geq 1$:

$$\begin{aligned} \mu_{\wedge,q} &= \mu_{\otimes,\frac{1}{q}} \\ \mu_{\vee,q} &= \mu_{\otimes,q} \end{aligned} \tag{11}$$

These operators allow for a certain amount of compensation between the operands that can be adjusted by the q parameter. For our system we use a q-value of 2. For illustration figure shows graphs of the q-operators for conjunction and disjunction. Like most compensatory operators the q-operator is not associative, but we do not make use of associativity, so this is not a problem.

Fuzzy-Prolog and Neural Nets

When trying to model how the rules contribute to the output of a fuzzy system, the rules must be assigned truth values individually. These values can be ad hoc estimates, but they have to be adjusted to achieve good performance. The idea is now to automate this process by optimizing the truth values using examples. It has been shown by Nauck and colleagues [20] that Fuzzy-Prolog programs are equivalent to feed-forward neural nets. If a Fuzzy-Prolog program matches certain requirements, it can be transformed into a feed-forward neural net. Neural net learning algorithms can then be used to optimize the fuzzy truth values.

For this purpose every proposition is assigned to a neuron and every rule is realized by connecting every neuron representing a premise to the neuron representing the conclusion. This entails several issues concerning the appropriate net structure, which can be resolved by inserting additional nodes (see [20]). A net generated from fuzzy-logical rules is called a *fuzzy-logical neural net* (FLNN).

Extensions

If we require that every neuron has only one outgoing connection, we can simplify the generated nets, by shifting the weight of a rule from the conclusion's incoming to the outgoing connections. The rule

$$\Phi : X_1 \otimes \ldots \otimes X_n \rightarrow X_{out} \tag{12}$$

can be transformed into the structure shown in figure 5. FLNNs generated by this *simplified transformation* have an individual weight for every system input, which is useful

especially for interpreting learning results, and would need extra rules without simplified generation.

Apart from using the simplified transformation we extend Fuzzy-Prolog in two ways. One is that we allow the use of the disjunction in rules. With the simplified transformation, the same effect can be obtained by using multiple rules, it is just a more convenient notation. The other is the integration of a list processing feature into the Fuzzy-Prolog system. This is necessary, since we do not know in advance how many groups there will be in an interpretation. The number of groups varies for different segmentations of the same input. We call these rules with variable numbers of premises multi-rules resp. multi-neurons with variable numbers of connections. For learning the connections to a multi-neuron share a single weight which corresponds to the truth value of the multi-rule.

INTERPRETATION RATINGS

Fuzzy Rules

Fuzzy rules allow to express musical knowledge in a way similar to natural language. Input features extracted from the interpretations are combined and weighted on several levels by using fuzzy rules to produce an overall rating of the interpretation. The rules used represent knowledge from music theory and music psychology about the segmentation and similarity of musical rhythms. These are all the rules currently used. Rules that operate on the group level have an index attached, indicating that this rule is applied once for every group.

(Output neurons for different modes:)

$$CInputSegmentQual \leftarrow CInputSegmentation \tag{13}$$
$$GNoteQual_i \leftarrow GGroupQual_i \tag{14}$$
$$CCategoryQual \leftarrow CGroupQual \tag{15}$$

(Overall rating and similarity:)

$$CIntrQual \leftarrow CGroupQual \wedge COrder \wedge CTaskSegmentation \wedge \\ CInputSegmentation \tag{16}$$
$$CGroupQual \leftarrow GGroupQual_1 \wedge \ldots \wedge GGroupQual_n \tag{17}$$
$$GGroupQual_i \leftarrow GTempo_i \wedge GPrscn_i \wedge GCorrect_i \wedge GRelPos_i \tag{18}$$
$$GTempo_i \leftarrow GTpoStbl_i \wedge GTpoPlsbl_i \tag{19}$$
$$GTpoStbl_i \leftarrow aglb_i \vee agl2b_i \vee agl3b_i \vee aglb2_i \vee aglb3_i \tag{20}$$
$$GPrscn_i \leftarrow GTooLate_i \wedge GTooEarly_i \wedge GTooLong_i \wedge \\ GTooShort_i \wedge GTooLoud_i \wedge GTooSoft_i \tag{21}$$
$$GCorrect_i \leftarrow GAddition_i \wedge GInsertion_i \wedge GSubtraction_i \wedge GDeletion_i \tag{22}$$

(Segmentation:)

$$CInputSegmentation \leftarrow CEqualLen \wedge CEqualNum \wedge CInputSeg \tag{23}$$
$$CInputSeg \leftarrow GInputSeg_1 \wedge \ldots \wedge GInputSeg_n \tag{24}$$
$$GInputSeg_i \leftarrow GInputDuration_i \wedge GInputLength_i \wedge GInputBeginLoud_i \wedge \\ GInputEndLong_i \wedge GInputEndRest_i \tag{25}$$
$$GInputDuration_i \leftarrow GInputDur04_i \vee GInputDur08_i \vee \\ GInputDur13_i \vee GInputDur20_i \tag{26}$$
$$GInputLength_i \leftarrow GInputLen1_i \vee GInputLen2_i \vee GInputLen3_i \vee \\ GInputLen4_i \vee GInputLen5_i \tag{27}$$
$$CTaskSegmentation \qquad < \textbf{analogous to } CInputSegmentation > \tag{28}$$

Not all the rules can be discussed in detail here, but the following remarks can give an idea which considerations guided the system design. The rules for similarity on the structure level are based among others on the segmentation rating for input and task. So the proposition *CInputSegmentation* represents the quality of an input segmentation based on rules and facts on group and interpretation level. The truth values of propositions of the form GInputDur*XX* (rule 26) are the inputs features mentioned earlier, which return values close to one if the length of the respective group is near to 0.4, 0.8, 1.3 or 2.0 seconds.

Similarly the rules GInputLen*X* (rule 27) represent group lengths of 1, 2, 3, 4, and 5. Relatively loud events tend to mark a group beginning and long events tend to end a group, which is reflected by propositions on the group level (rule 25)

Rule 24 is a multi-rule where the transition from the interpretation level to the group level takes place and a variable number of inputs is combined to one node. The output node *CIntrQual* returns the rating for a whole interpretation that comprises segmentation, similarity of groups, and assignment of groups. The extra output neurons are needed for implementation reasons, the weights of their connections is fixed to 1. The stucture of the net generated from the whole Fuzzy-Prolog program is shown in figures 6, 7, and 8.

Learning

After converting the Fuzzy-Prolog program into a fuzzy-logical neural net we can start the training process. For this purpose some modifications of the backpropagation algorithm are needed to adapt it to the evaluation functions of the fuzzy logical operators. We replace the weighted sum as net input function net of every neuron by the evaluation function for the operator that is used in the corresponding rule. These functions get already weighted inputs as arguments. Therefore we need the derivative of the used evaluation operators. For the q-operators we can calculate

$$\frac{\partial net_j^{(p)}}{\partial w_{ij}} = o_i^{(p)} \cdot drvt_{ij}^{(p)} \qquad (29)$$

where w_{ij} is the weight of the connection from neuron i to j, o_i is the output value of neuron i, and $drvt_{ij}^{(p)}$ is defined as the outer derivative:

$$drvt_{ij}^{(p)} = \left(\frac{1}{n}\right)^{\frac{1}{q}} (o_i^{(p)} w_{ij})^{q-1} \left(\sum_{r=1}^{n} (o_{k_r}^{(p)} w_{k_r j})^q\right)^{\frac{1}{q}-1} \qquad (30)$$

The resultant modified backpropagation rule can be written as

$$\triangle w_{ij} = \eta \sum_{p \in P} o_i^{(p)} \cdot drvt_{ij}^{(p)} \cdot \delta_j^{(p)} \qquad (31)$$

with

$$\delta_j^{(p)} = \begin{cases} f'(net_j^{(p)})(t_j^{(p)} - o_j^{(p)}) & \text{if } j \text{ is output neuron,} \\ f'(net_j^{(p)}) \sum_{s=1}^{m} \delta_{k_s}^{(p)} \cdot w_{jk_s} \cdot drvt_{jk_s}^{(p)} & \text{else} \end{cases} \qquad (32)$$

For a multi-neuron getting input from several instances of a neuron with different values, a weight update for every instance is calculated and the sum is applied to the shared weight.

Iterative Training by Relative Ratings

The straightforward approach to learning ratings by example is to collect example ratings, but this would need large numbers of ratings the expert. Since there is a large number of possible interpretations it is in general not feasible to rate a substantial number of interpretations for a given input. The system is therefore trained with relative ratings following the method by Braun, Feulner and Ulrich [23]. This approach is useful for networks calculating a rating value and has been used for learning game strategies. For this purposes the absolute values are not relevant. The relation between the rating of two different reactions to a situation is important for the decision which move to take. Similarly it is not important which ratings the interpretations actually get, it is sufficient that the desired interpretation is rated best.

The network architecture for training by relative ratings is shown in figure 9. The neural net is duplicated with shared weights and the output neurons are connected with a comparator neuron. We can assume without loss of generality that interpretation rated better by the expert is given as *input 1*. The output of the comparator neuron is then calculated as $\max(n_2 - n_1, 0)$. Therefore the target value for the comparator neuron is 0. If the result is not 0, it can be used as error value for backpropagation. This means that pairs of interpretations are given and one of each pair is marked as better. After successful learning the system assigns higher ratings to the better interpretations.

To further reduce the number of ratings needed from the expert we use *iterative training*. We take the interpretation preferred by the expert and dynamically generate relative

examples by comparing the expert interpretations to those preferred by the system. If they differ a relative rating sample is generated. Normally, after training the relation of ratings is correct for the given relative samples but other interpretations are rated better than the ones provided by the expert. So the process of generating sample pairs and training the network is iterated until for all samples the expert interpretation is rated best by the system or a maximum number of iterations is reached.

The depth of the net makes standard backpropagation slow since the size of the gradient gets very small on the lower layers of the net. To overcome this problem we use the RPROP [24] algorithm for training the network with which our net converges dramatically faster than with other learning methods (in figure learning curves are shown for standard backpropagation with momentum and for RPROP). Usually generalization suffers from fast learning, but tests showed that it does not happen in this case. This is probably because there is no distributed representation in the network unless it is explicitly encoded. So overlearning does not take place and the network is extended only when needed. This capacity of the network is limited compared to a MLP, but it allows to interpret training results and can lead to good generalization, if the structure defined by the fuzzy rules is appropriate for the learning task.

APPLICATION

Implementation

We have implemented an experimental application of the system described in this paper called *RhythmScan* which allows to generate samples, to train and test the system. It is a Java application with a Graphical User Interface and the Fuzzy-Prolog interpreter written in C connected via JNI and uses XML data format for storage to facilitate data reuse. The system can be interactively used and trained by giving keyboard input and preferred interpretations. User input can be provided via a MIDI or via the computer keyboard, in the latter case without velocity information. The preferred segmentations and assignments can be defined interactively by a user (an expert) on a graphical user interface. Differing segmentations and assignments by system and user can be seen in figure which shows the expert view of the user interface.

After training the system adequately to most segmentation tasks, e.g. modelling subjective rhythmization and performs structural comparison of user input for a given task like in interactive music tutorials. The system is able to detect delayed, interrupted, and repeated input as well as tempo and timing deviations.

Optimization

It is the central idea of Branch-and-Bound to avoid calculations that cannot improve the result when searching an optimal path in a graph. To do this, one tries to find bounds on the maximal rating when incrementally calculating partial solutions. The maximal rating of solutions containing the partial solution is compared to a value that has already been reached or predetermined as a minimum. If the maximal rating is lower, it is not necessary to complete and rate the solutions containing this partial solution.

For applying Branch-and-Bound we organize the calculation of the interpretations corresponding to a tree structure. Every node represents a group assignment and every edge represents a different way of combining them. The fist level below the root presents the assignment of the first input group, then follow the branches to the assignment of the second group, then the third group etc. In this tree we can determine for each node the maximal rating that can be achieved with maximal rating of the not yet assigned groups. If this maximal rating is better that another one already achieved, it is not necessary to follow this branch any further. This is illustrated in figure . With i levels and a branching degree of t, corresponding to i and $t - 1$ groups in input and task, the complexity is reduced from $O(t^i)$ to $O(t \cdot i)$ in the optimal case.

The increase of efficiency that is actually achieved depends mainly on the heuristic that determines the order in which the interpretations are evaluated. The earlier good ratings are found, the more branches can be skipped. We use the segmentation ratings, which can be computed in advance and sort the interpretations accordingly before rating them.

Effect of Filtering and Branch-and-Bound

Influential factors for calculation cost of the selection algorithm are the length of the sequences, filtering and the Branch-and-Bound optimization. The result of a test with 20 samples with different lengths, each with and without Branch-and-Bound and filtering, are listed in table 2. It can be seen clearly that both optimizations have more effect on longer sequences. But even for short sequences the additional computation for the optimizations has no negative effects. Filtering only has more effect than Branch-and-Bound only, both methods combined have the best effect and reduce the computation time significantly: for the examples used by a factor of more than 3000. Optimization has the strongest effect for long sequences. This behavior is useful for interactive applications, where maximal answer times should be kept low.

Training Results

For testing the system in comparison mode we used 100 samples, consisting of 50 sequences as performed by students in original form and with gaussian noise added ($\sigma = 5$ MIDI units for loudness and $\sigma = 80$ msec for timing). The interpretations were defined by graduate students. The data are not representative or suitable for drawing conclusions on rhythm perception in general, but they are examples of the desired output in a tutorial situation.

The fuzzy-logical net described before was trained with this data. For comparison a linear net and a standard neural network (multi-layer perceptron, MLP) with one hidden layer of four neurons were also trained. The training results are shown in Table 3. While the results of the linear net are much worse than the two others, the results of the FLNN and the MLP are similar. We reach a percentage of 88% resp. 90% identical to the expert interpretations on the training set and 80% on the test set. There are often many acceptable interpretation possibilities and the interpretations that differ from those of the experts are mostly musically acceptable. For both the FLNN and the MLP, the number of musically implausible samples was 3 on the test set and training set combined, i.e. 3%.

The weights of the trained FLNN were mostly in the ranges which could be expected from the music theory and psychology. One unexpected tendency was, that the regularity of group length and duration received almost zero weights. This may be due to the design of the measures or because of the rather unmetrical input sequences.

CONCLUSIONS

Modelling musical knowledge in combination with machine learning enables us to solve musical tasks without a complete model or exact knowledge of perceptual and other relevant processes. The system can be easily modified or extended by adding, changing or removing rules or input features.

Overall 97% of the interpretations chosen by the system are musically acceptable, which is a good result. Although the standard neural net gave similar results, the FLNN offers the advantages of being interpretable, easily extendable, and more efficiently computable because of fewer connections in the network and better effect of optimizations. The weights in the trained system give insight to the relevance of rules, although the sample sets are not yet representative and too small to draw any generalized conclusions.

The results encourage to further explore this method of pattern recognition and analysis. Next challenges are the integration of pitch and metrical information into the system, which has already been begun for pitch with first promising results. The extension to streaming mode and further optimization of the computational efficiency are important for practical applications. It is planned to use the segmentation and similarity measures described here for intelligent search in musical databases.

REFERENCES

[1] Grosvenor W. Cooper and Leonard B. Meyer. *The Rhythmic Structure of Music*. University of Chicago, Chicago, 1960.

[2] Diana Deutsch. Auditory pattern recognition. In K. R. Boff, L. Kaufman, and J. P.

Thomas, editors, *Handbook of Perception and Human Performance: Cognitive Processes and Performance*, volume 2, chapter 32, pages 32–1–49. John Wiley and Sons, New York, 1986.

[3] S. Handel. *Listening: An Introduction to the Perception of Auditory Events*. MIT Press, Cambridge, Massachusetts, 1989.

[4] Peter Desain and Luke Windsor, editors. *Rhythm Perception and Production*. Swets and Zeitlinger, Lisse, 2000.

[5] Guerino Mazzola and Oliver Zahorka. Topologien gestalteter motive in kompositionen. In Wolfgang Auhagen, Bram Gätjen, and Klaus Wolfgang Niemöller, editors, *Systemische Musikwissenschaft - Festschrift Jobst Peter Fricke zum 65. Geburtstag*. Universität Köln, Köln, 1995.

[6] Tim Crawford, Costas S. Iliopoulos, and Rajeev Raman. String-matching techniques for musical similarity and melodic recognition. In Walter B. Hewlett and Eleanor Selfridge-Field, editors, *Melodic Similarity*, volume 11 of *Computing in Musicology*, chapter 3, pages 73–99. The MIT Press, Cambridge, Mass., 1998.

[7] Christiane Linster. Get rhythm: A musical application for neural networks. Arbeitspapiere der GMD 365, Institut für Angewandte Informationstechnik, 1989.

[8] Simon Roberts. Interpreting rhythmic structure using artificial neural networks. Diss., University of Wales, 1996.

[9] James Tenney and Larry Polansky. Temporal gestalt perception in music. *Journal of Music Theory*, 24(2):205–41, 1980.

[10] Fred Lerdahl and Ray Jackendoff. *A Generative Theory of Tonal Music*. The MIT Press, Cambridge, Mass., 1983.

[11] Emilios Cambouropoulos. Musical parallelism and melodic segmentation. In *Proceedings of the XII Colloquium of Musical Informatics*, Gorizia, Italy, 1998.

[12] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, 1956.

[13] Stephen Handel and Peyton Todd. Segmentation of sequential patterns. *Journal of Experimental Psychology: Human Perception and Performance*, 7(1):41–55, 1981.

[14] J. P. Swain. The need for limits in hierarchichal theories of music. *Music Perception*, 4(1):121–148, 1986.

[15] Uwe Seifert, Fabian Olk, and Albrecht Schneider. On rhythm perception: Theoretical issues, empirical findings. *Journal of New Music Research*, 24(2):164–95, 1995.

[16] Stephen Handel. Temporal segmentation of repeating auditory patterns. *Journal of Experimental Psychology*, 101:46–54, 1973.

[17] Paul Fraisse. Rhythm and tempo. In Diana Deutsch, editor, *The Psychology of Music*, chapter 6, pages 149–180. Academic Press, New York, 1982.

[18] Richard Parncutt. A perceptual model of pulse salience and metrical accent in musical rhythms. *Music Perception*, 11(4):409–64, 1994.

[19] Bernd Enders and Tillman Weyde. Automatische Rhythmuserkennung und -vergleich mit Hilfe von Fuzzy-Logik. *Systematische Musikwissenschaft*, IV(1-2):101–113, 1996.

[20] Detlef Nauck, Frank Klawonn, and Rudolf Kruse. *Neuronale Netze und Fuzzy-Systeme*. Computational Intelligence. Vieweg, Braunschweig, 2 edition, 1996.

[21] Rudolf Kruse, Frank Klawonn, and Jörg Gebhard. *Fuzzy-Systeme*. Leitfäden und Monographien der Informatik. Teubner, Stuttgart, 1993.

[22] Ronald R. Yager. On a general class of fuzzy connectives. *Fuzzy Sets and Systems*, 4:235–242, 1980.

[23] Heinrich Braun, Johannes Feulner, and V. Ulrich. Learning strategies for solving the planning problem using backpropagation. In *Proceedings of NEURO-Nimes 91, 4th International Conference on Neural Networks and their Applications*, 1991.

[24] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 586–591, San Francisco, CA, 1993.
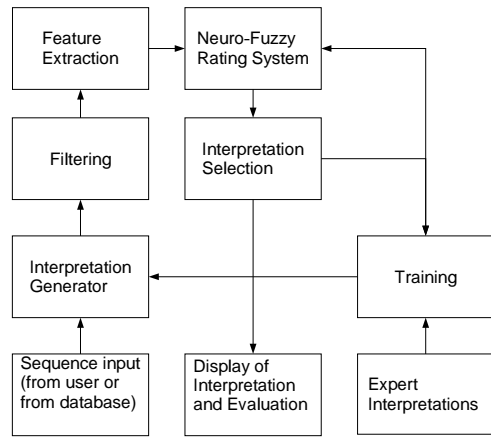
FIGURES
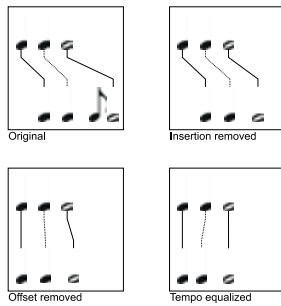

Figure 1: System Modules.
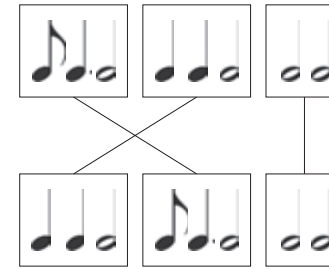

Figure 2: Calculation of tempo variants.

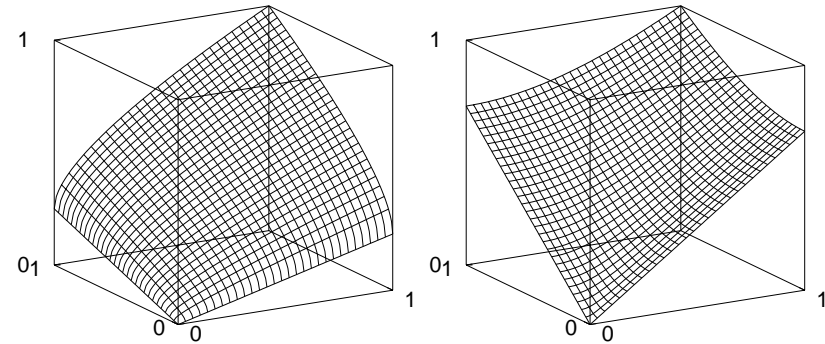Figure 3: Group assignments for a rhythmic interpretation.

Figure 4: Graphs of the q-operator for conjunction and disjunction with $q = 2$.
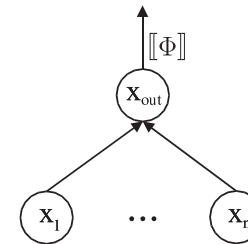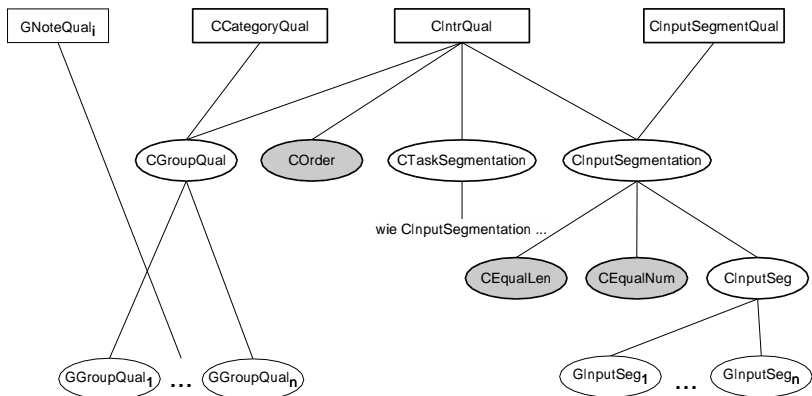

Figure 5: Simplified net structure.

Figure 6: Structure of the neuro-fuzzy program on the structure level: Nodes on the top are output nodes (rectangles), nodes on the bottom (gray) are net inputs (Fuzzy-Prolog facts). Nodes with bold border operate on the structure level, all other nodes operate on the group level. Connections to Nodes marked with '···' are transitions from structure to group level.
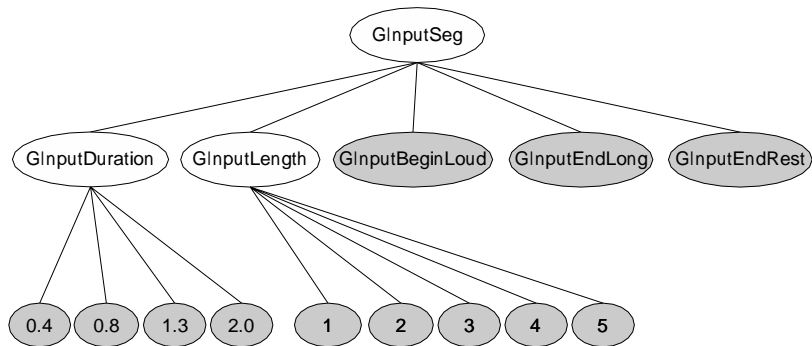

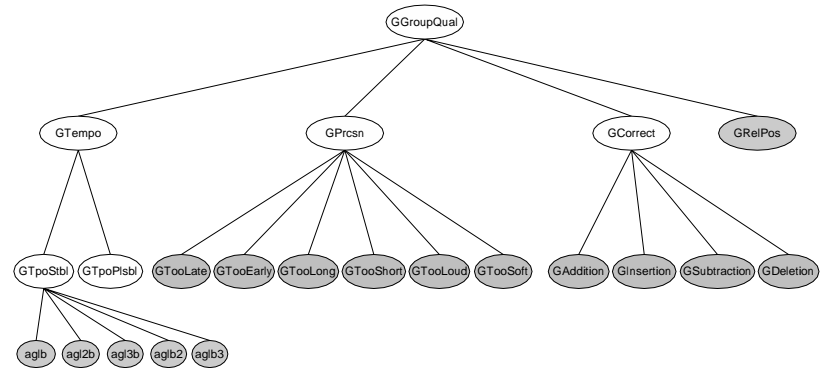Figure 7: Structure of the neuro-fuzzy program on the group level for segmentation.

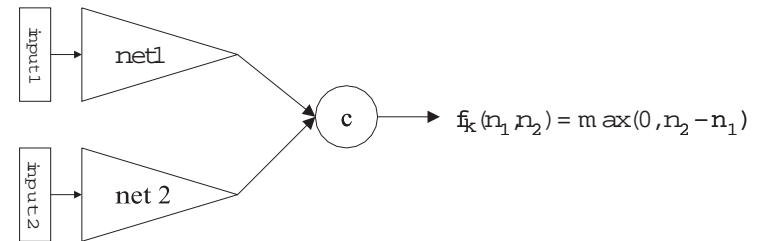Figure 8: Structure of the neuro-fuzzy program on the group level for group similarity.

Figure 9: Scheme for relative training
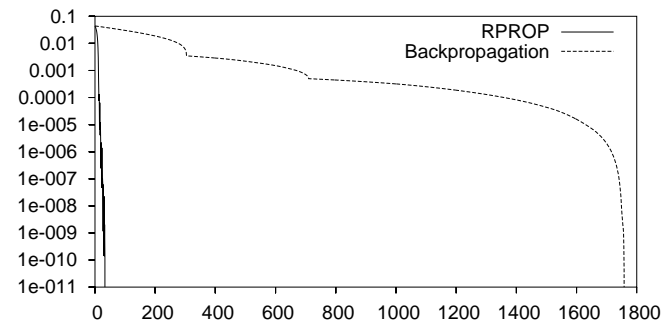
$$f_k(n_1, n_2) = \max(0, n_2 - n_1)$$

Figure 10: Learning curves for Standard Backpropagation with Momentum and RPROP. Leaps in the curve show generation of new training samples between training iterations.
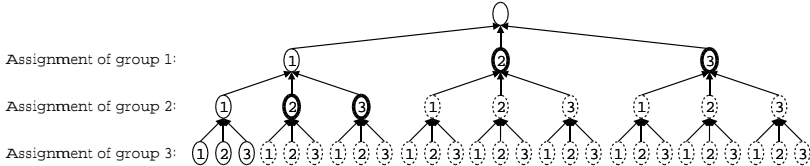
Figure 11: Branch-and-Bound scheme: representation of group assignment as a tree. Using a depth-first search starting from the left a upper bound can be determined at the bold nodes. In the optimal case the ratings for the dotted nodes do not need to be calculated, which reduces the computational effort from 27 to 9 ratings in this example.

Figure 12: *RhythmScan* user interface: black lines show segmentation and assignments calculated by the system, yellow lines show input by an expert user

TABLES

n	$seg(n)$ for $[min, max]$					
	$[2, 5]$	$[2, 6]$	$[2, 7]$	$[1, 5]$	$[1, 6]$	$[1, 7]$
1	0	0	0	1	1	1
2	1	1	1	2	2	2
3	1	1	1	4	4	4
4	2	2	2	8	8	8
5	3	3	3	16	16	16
6	4	5	5	31	32	32
7	7	7	8	61	63	64
8	10	12	12	120	125	127
10	24	29	31	464	492	504
12	57	71	79	1793	1936	2000
14	134	175	200	6930	7617	7936

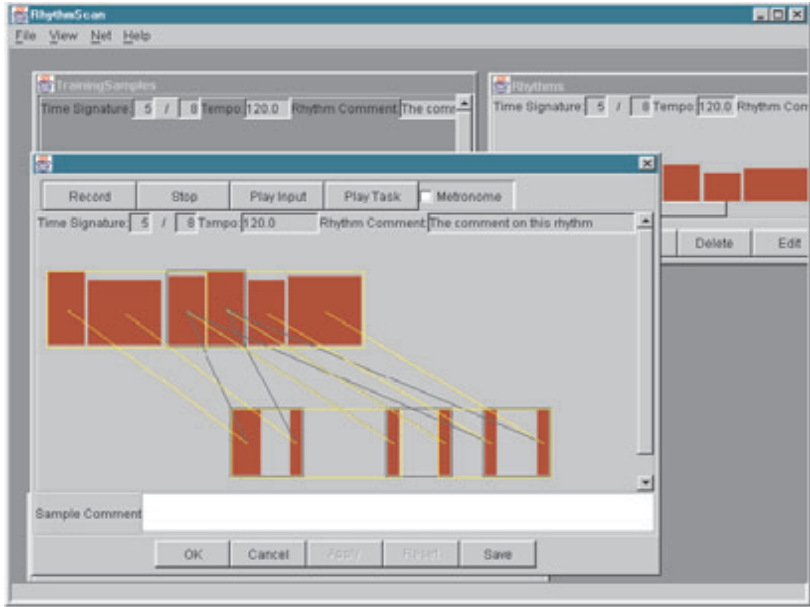Table 1: Number of possible segmentations depending on sequence length and permitted group lengths.

Summed sequence-length	Filters and Branch-and-Bound	Only Filters	Only Branch-and-Bound	No Optimization
4	0,03	0,03	0,04	0,05
6	0,06	0,07	0,19	0,32
6	0,14	0,24	0,19	0,30
6	0,03	0,03	0,20	0,32
8	0,13	0,33	0,77	5,35
8	0,20	0,54	1,60	5,34
8	0,15	0,22	0,77	5,33
9	0,21	0,34	3,36	25,79
10	0,41	1,07	19,23	128,37
10	0,19	0,22	15,80	163,25
10	0,36	1,07	25,91	163,67
10	0,35	0,79	20,46	163,60
10	0,19	0,48	3,67	162,96
10	0,36	1,08	15,68	163,25
11	0,50	2,56	6,30	1069,03
11	0,28	1,62	5,74	1070,62
12	0,62	4,43	11,69	4322,32
12	0,68	4,43	11,25	4323,07
12	0,70	4,43	11,29	4325,34
12	0,80	4,25	12,69	4323,47
Mean	0,32	1,41	8,34	1021,09
Median	0,25	0,67	6,02	163,11
Minimum	0,03	0,03	0,04	0,05
Maximum	0,80	4,43	25,91	4325,34

Table 2: Calculation time of interpretation selection in seconds. The first column shows the sum of the lengths of input and task, which were equal in most samples.

	LN	FLNN	MLP
Deviating interpretations on training set	6 (12%)	6 (12%)	5 (10%)
Error value (SSE)	3.95e-7	3.94e-7	2.40e-7
Generated relative samples	267	230	250
Error on relative samples	97 (36%)	26 (11%)	26 (10%)
Deviating interpretations on test set	36 (72%)	10 (20%)	10 (20%)

Table 3: Training results of different network types: LN: Linear net, FLNN: fuzzy-logical neural net, MLP: multi layer perceptron, one hidden layer.