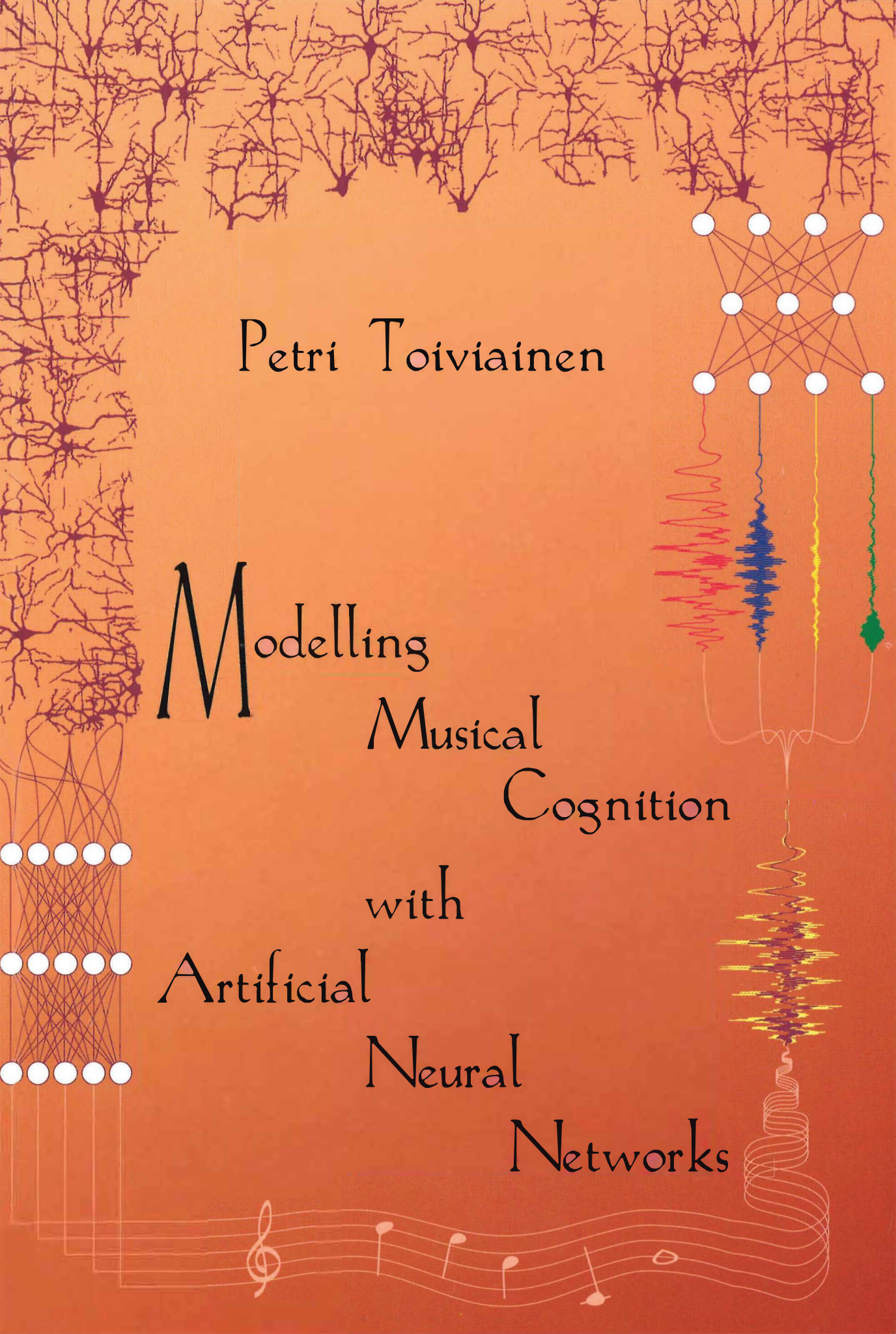
Petri Toiviainen

Modelling Musical Cognition with Artificial Neural Networks



JYVÄSKYLÄ STUDIES IN THE ARTS 51

Petri Toiviainen

Modelling Musical Cognition with Artificial Neural Networks

Esitetään Jyväskylän yliopiston humanistisen tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston vanhassa juhlasalissa (S212)
helmikuun 23. päivänä 1996 klo 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Humanities of the University of Jyväskylä,
in Auditorium S212, on February 23, 1996 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 1996

Modelling Musical Cognition with Artificial Neural Networks

JYVÄSKYLÄ STUDIES IN THE ARTS 51

Petri Toiviainen

Modelling Musical Cognition with Artificial Neural Networks

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 1996

Editor
Erkki Pekkilä
Department of Musicology
University of Jyväskylä

URN:ISBN:978-951-39-7890-7
ISBN 978-951-39-7890-7 (PDF)
ISSN 0075-4633

ISBN 951-34-0655-5
ISSN 0075-4633

Cover
Design: Petri Toiviainen

Gurnmerus Kirjapaino Oy
Saarijärvi 1996

This work is dedicated to my parents, Anne and Jaakko Toiviainen.

Abstract

Toiviainen, Petri
Modelling musical cognition with artificial neural networks
Jyväskylä: University of Jyväskylä, 1996. 40 p.
(Jyväskylä Studies in the Arts,
ISSN 0075-4633; 51)
ISBN 951-34-0655-5
Diss.

As a highly abstract form of human activity, music is a challenging realm to study. During the last ten years, the connectionist paradigm has provided insights into many domains of human behaviour, including musical activity and experience. Artificial neural networks, or connectionist systems, can be characterized as strongly idealized models of networks formed by biological neurons: consisting of a bulk of simple interconnected processing units, they employ parallel distributed processing and are capable of learning and self-organizing.

The present study focuses on aspects of musical cognition such as perceptual learning, self-organization, feature extraction, sequential processing, autoassociative recall, and short-term memory. More specifically, processes related to the classification and recognition of musical timbre and the learning and generation of melodies are modelled using artificial neural networks.

The results support the view that the connectionist paradigm provides a plausible alternative for modelling the dynamics of certain music-related cognitive processes. Being inherently capable of generalizing, associating on the basis of content, and tolerating noisy or distorted input, artificial neural networks exhibit functions characteristic of the human way of perceiving, thinking, and acting.

Keywords: music, cognition, artificial neural networks, self-organization, sequential processing, musical timbre, improvisation

FOREWORD

"That [jazz] music gets on my nerves!"
Aarni Toiviainen (1987–)

This dissertation is a collection of my attempts to understand what happens when a human interacts with his musical environment. In this challenging task I have leaned mainly on my education in physics and my practical experience as a jazz musician and lecturer in music.

Most of this work was carried out at the Department of Musicology, University of Jyväskylä during the period 1990–1995, as part of two research projects, "Receiving, learning and producing music as a cognitive process" and "Physicalism, connectionism, and representation", both funded by the Academy of Finland.

Three of the publications on which this thesis is based are joint articles. Publication II was co-written by Mauri Kaipainen. He wrote the first half of the Introduction and the Discussion, while I wrote the rest. Study V was performed together with Topi Järvinen. Apart from the sections that dealt with our own previous research, both of us contributed significantly to each part of the study. The actual writing was, however, divided, so that Järvinen wrote the sections on the results, while I was responsible for the mathematical discussions. My contribution to study VI consists of the design and implementation of the preprocessing stage. In the publication concerned I have written the sections "Turning musical flow into pitch vectors by autocorrelation" and "Turning pitch vectors into short-to-long-term inhibitory memory pools".

Since computer simulations were an essential part of my studies, I have included source codes (in C language) of the two most important neural network simulators I have used: the Kohonen map simulator used

in studies II and III (Appendix 1) and the jazz improvisation model used in studies IV and V (Appendix 2).

This work was made possible by a great deal of help and support from a number of people. First of all, I would like to express my deepest gratitude to Professor Jukka Louhivuori, the director of the two aforementioned research projects. I think he demonstrated considerable liberality by asking me, a novice in cognitive musicology, to join his research projects. His continuous support and enthusiasm for my work as well as his far-sighted and creative attitude have been crucial during the course of this project.

I am much indebted to Dr. Marc Leman who has in the course of the entire research process offered many valuable critical remarks on my work. In autumn 1993, I had the opportunity to work as a visiting researcher at his institute (IPEM, University of Ghent, Belgium). This period turned out to be very important for this project.

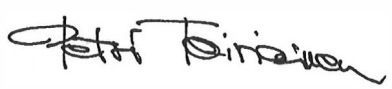
I am grateful to Professor Matti Vainio for his sympathetic attitude towards my work and also for backing my applications in various committees. Professor Jussi Timonen was the supervisor of my master's thesis in theoretical physics (1987). My gratitude to him for leading me into the world of non-linear dynamical systems.

Mauri Kaipainen and Topi Järvinen have on several occasions offered many valuable comments on drafts of my papers and on my work in general. Moreover, I wish to thank them for their smooth collaboration when preparing our joint articles. My gratitude also to John Richardson for proof-reading a number of my articles and guiding me in the subtleties of the English language.

This work would not have been possible without the financial support given by the Academy of Finland. Moreover, I am indebted to the University of Jyväskylä for providing me with research facilities and financing some conference trips I have made. The financial support given by Ellen ja Artturi Nyyssösen säätiö is also gratefully acknowledged.

Finally, I would like to express my gratitude to my wife Tuija and my sons Maunu, Aarni, and Leo for their love, support, and patience.

Jyväskylä, 2 January, 1996



LIST OF ORIGINAL PUBLICATIONS

This thesis is based on the following publications, which will be referred to in the text by their Roman numerals:

I Toiviainen, P. 1992. The organisation of timbres: a two-stage neural network model. *Workshop Notes of the ECAI 92 Workshop on Artificial Intelligence and Music.* Vienna: ECCAI.

II Toiviainen, P., Kaipainen, M. & Louhivuori, J. 1995. Musical timbre: similarity ratings correlate with computational feature space distances. *Journal of New Music Research,* 24(3), 282–298.

III Toiviainen, P. 1995. Optimizing auditory images and distance metrics for self-organizing timbre maps. *Journal of New Music Research,* in press.

IV Toiviainen, P. 1995. Modeling the target-note technique of bebop-style jazz improvisation: an artificial neural network approach. *Music Perception,* 12(4), 399–413.

V Järvinen, T. & Toiviainen, P. 1995. Connectionist jazz and tonal hierarchy: a statistical multilevel analysis. (Submitted)

VI Kaipainen, M., Toiviainen, P. & Louhivuori, J. 1995. A self-organizing map that recognizes and generates melodies. In Pylkkänen, P. & Pylkkö, P. (Eds.), *New directions in cognitive science.* Publications of the Finnish Artificial Intelligence Society, 286–315.

APPENDIX 1: C source code of the Kohonen map simulator used in studies II and III

APPENDIX 2: C source code of the jazz improvisation model used in studies IV and V

CONTENTS

1 INTRODUCTION

Artificial neural networks, also referred to as connectionist or parallel distributed processing (PDP) systems[1], can be characterized as strongly idealized models of networks formed by biological neurons: although the basic principles of their mechanisms and structures have been adopted from biological neural networks, they are generally not intended to model the physiological processes in the neural tissue. Technically, they are nonlinear dynamic systems consisting of a multitude of simple, interconnected processing units, frequently referred to as (artificial) neurons. They have three important properties. First, they are parallel, i.e., the processing units interact simultaneously and independent of each other[2]. Second, they are distributed, i.e., their knowledge resides in the strengths of interneuronal connections; and the data manipulated by them is represented as patterns of neuronal activation. Third, they are adaptive, i.e., when exposed to data from the environment, they are capable of learning by adjusting the strengths of their interneuronal connections.

A variety of types of artificial neural networks have been developed for a wide range of purposes. They can be categorized, for instance, on the basis of their architecture, dynamics, the type of data they process, or learning algorithm. Two important partitions will be briefly discussed here,

---

1 The terms 'artificial neural network' and 'connectionist system' will subsequently be used as synonyms.

2 Artifial neural networks are mostly simulated on traditional serial computers. Consequently, the processing is in fact serial — on the lowest level of description. On a higher level of description it is, however, parallel.

viz., dynamic vs. static networks and supervised vs. unsupervised learning.

The behaviour of static networks is characterized by equations that are memoryless. Their output is a function of the current input only, not of past inputs or outputs. This category embraces, a.o., the perceptron (Rosenblatt, 1958) and the multilayer perceptron (Rumelhart, Hinton & Williams, 1986) network. The latter, also referred to as the back-propagation network, is perhaps the most used and studied type of artificial neural network. In such a network, the neurons are organized in layers — an input layer, one or more hidden layers, and an output layer — and have feedforward connections from each layer to the next one. Such a network is usually trained by the back-propagation of error algorithm (Rumelhart, Hinton & Williams, 1986), in order to produce a desired mapping from the input space to the output space. This algorithm is actually an optimization procedure: the goal is to minimize the error between desired and obtained outputs by means of the gradient descent method. Applications of multilayer perceptron networks include pattern recognition and functional approximation.

Dynamic networks are systems with memory. Their dynamical behaviour is described by differential or difference equations. Dynamic networks can be further divided into two subcategories: networks with output feedback and those with state feedback.

Networks with output feedback can be used for processing sequential data. Under this category belongs the network architecture proposed by Jordan (1986): a multilayer perceptron network which has feedback connections from the output layer to the input layer. Variants of this architecture have been used for musical applications (see below).

Networks with state feedback are typically single-layer networks with feedback connections between nodes. In the most extreme case the neurons are completely interconnected, i.e., every neuron is connected to every other. Networks with state feedback can be used, for instance, for constraint satisfaction, i.e., for solving problems where one has to take into account many concurrent local constraints. Such networks can find a solution to a given problem which in the best possible way takes into account the given constraints. They are set in operation by presenting them with a pattern of neuronal activations which represents the problem being solved. This disturbs the equilibrium state of the network and starts a flow of activation between neurons — a process referred to as relaxation. The flow continues until the network settles to a stable state. This equilibrium can be conceived of as a solution to the problem in question. The state of the network can be characterized using a global function which describes how well the network satisfies the given constraints. This function is referred to as the energy function (Hopfield, 1984), harmony function (Smolensky, 1986), or goodness function (McClelland & Rumelhart, 1988). Solving problems is equivalent to minimizing the energy function or maximizing the harmony or goodness function. Besides problem solving, networks with state feedback can be utilized for pattern completion through auto-association: each pattern of activation the network has memorized corresponds to a minimum of energy in its state space. When

being presented with a noisy or distorted version of a memorized pattern, these networks are capable of recollecting the original pattern through relaxation. Examples of networks with state feedback are the interactive activation model (Rumelhart, Hinton & McClelland, 1986), Hopfield network (Hopfield, 1984), Boltzmann machine (Hinton & Sejnowski, 1986), and harmony theory network (Smolensky, 1986).

Artificial neural networks can be further categorized, on the basis of the type of learning they employ, into supervised and unsupervised learning networks. The networks discussed so far belong under the former category: they are intended to produce a desired output from a given input. The latter category, unsupervised learning networks, is probably the most interesting class of artificial neural networks, because their way of extracting knowledge about the environment resembles that of biological systems. Such networks are presented with only input samples; these are grouped into classes which are self-similar through a process referred to as self-organization. Examples of self-organizing networks are the Adaptive Resonance Theory (ART) networks — ART 2 (Carpenter & Grossberg, 1987) and ART 3 (Carpenter & Grossberg, 1990) — and Kohonen's self-organizing feature map (Kohonen, 1989); all of these belong under the category of static networks. Whereas the ART networks perform an automatic categorization of the input data set, the Kohonen network maps the input vectors onto a two-dimensional surface while retaining their topological relationships.

Connectionist models have been successfully applied to a wide range of domains, ranging from physics, mathematics, computer science, and engineering to the biological, medical and cognitive sciences, and artificial intelligence. Processing vectors, they have showed themselves to be particularly fit for handling ill-defined, continuous-valued target domains. These include, e.g., optimization, control, prediction, pattern recognition, and diagnosis.

Until now, a serious shortcoming of cognitive modelling has been its narrow focusing on explicit knowledge and symbolic processing. While the existence of a symbol-based mode of thinking in certain cognitive processes probably cannot be totally disputed, a great proportion of mental phenomena, for instance, creativity, intuition, and humour, cannot be accounted for on the basis of such rational, atomistic, and explicit reasoning. Moreover, there is evidence that cognitive processes such as perception, memory, thinking, judgement, and problem-solving can all occur implicitly, i.e., outside conscious awareness (Valentine 1995). While it seems unlikely that models based on the orthodox AI could explain such processes, the connectionist paradigm, where mental activity is seen as a holistic process and information is represented implicitly, may provide means for elucidating processes behind these mental phenomena. Besides exhibiting behaviour similar to that observed in human information processing, connectionist systems are appealing because of their neural realism: they can be regarded as more than merely functional models of mind. While plausible connectionist models of certain aspects of human mental action, in particular those related to memory and perception, have been proposed, some aspects may prove to be difficult to model within this paradigm; the

latter include, for instance, those related to consciousness, volition, and subjective experience.

The strengths of connectionist systems, from the point of view of cognitive modelling, are summarized below (McClelland, Rumelhart & Hinton, 1986; Rumelhart & McClelland, 1986; Gutknecht, 1992):

- Learning capability. Connectionist systems are usually not programmed but trained by presenting them with examples. On the basis of those examples, they adapt to their environment. In supervised learning, the network is trained to produce a desired output for each input vector. In unsupervised learning, the network produces, through self-organization, a categorical or topographical representation of the input.
- Generalization capability. Connectionist systems are capable of extracting significant features from the training set and using them for processing a novel input pattern. Mathematically, the network is capable of successfully extrapolating an intended function from its training sample.
- Content-addressability. When presented with a part of a learned input pattern, the network can retrieve, or associate, the whole pattern. Any learned pattern can thus be addressed by its content rather than its location in the memory.
- Noise tolerance. Connectionist systems are robust against noisy or incomplete input patterns.
- Tolerance towards overloading of information. Connectionist systems do not have a fixed storage capacity: when a network is overloaded with input data, similar components of information tend to blend together, resulting in generalization of features.

The aforementioned strong points of connectionist systems make them well suited for modelling sectors of human cognition such as perceptual learning and information processing at a subconscious level.

On the other hand, connectionist systems have been purported to lack some capabilities, mostly related to high level reasoning and problem solving, which traditional symbol-based models have (Fodor & Pylyshyn, 1988). According to Gutknecht (1992), limitations of connectionist models include:

- Poor explanation capabilities. Attempts have been made to explain the behaviour of connectionist networks by, for instance, analyzing the structure of the connection strength matrix learned. These explanations are, however, at the level of primitive features of the network, such as activation functions and energy landscapes. Explanations on a higher level of knowledge are difficult to achieve.
- Difficulties with structured representation. Structured knowledge, such as concept hierarchies or inference nets, is difficult to represent in connectionist systems, contrary to traditional AI models.
- Lack of compositionality. Connectionist systems are usually designed for one particular kind of applications and consist of one

'monolithic' network. As a consequence, it is more difficult than in traditional systems to reuse and recombine pieces of their knowledge.

While these claims certainly are not trumped-up, one can question whether the listed weaknesses are inherent properties of connectionist systems or caused by methodological problems. The ability of a neural network to represent and deal with structured knowledge depends to a great extent on the degree of hierarchy and modularity of the network's architecture; the same applies to compositionality. On the other hand, study VI shows that even a relatively simple connectionist system can satisfactorily process temporally structured data, if the latter has been preprocessed so that it is hierarchical itself. Furthermore, the explanation capability varies between different types of networks: whereas a multi-layer perceptron network, for instance, contains a great deal of implicit knowledge which is often difficult to explain, a Kohonen network is by its nature much more explicit.

An essential aspect which affects the performance of an artificial neural network is how the input data is represented. According to a general view, a distributed representation seems to better exploit the strengths of those networks than a local one. A distributed representation can be defined from several points of view. For instance, it may imply representing the data as a vector, each component of which stands for a specific microfeature of the represented domain. Or, it may denote representing many items at once over the same set of processing units or connection weights. It must be noted that there is no clear distinction between local and distributed representation, but rather there exists a continuum between those two extremes. If an item is represented as a collection of microfeatures, each of those features can again be represented either in a local or a distributed manner. For example, the most local way of representing chords in a connectionist system is to designate one neuron for representing each chord. A more distributed representation is obtained by representing each chord as the tones it is composed of. The degree of distribution can be further increased by representing each of those tones as a harmonic or subharmonic series of frequencies.

Local and distributed representations are frequently referred to as symbolic and subsymbolic, respectively. A distributed representation may, however, be composed of a collection of symbolic microfeatures; calling such a representation subsymbolic is somewhat inconsistent. According to Leman (1993), a genuine subsymbolic representation of sound should be based on the way the brain codes information received by the auditory system. Yet, as he further points out, it is mostly necessary to use simplified forms of representations, i.e. hybrid ones, because of the extreme complexity of the representations the brain uses.

The use of the connectionist paradigm for modelling musical cognition can be justified from several points of view. To construct a rule-based expert system, an expert is needed who can verbalize the rules which define the solutions of the problem in question. These rules must be correct and consistent. A large part of our musical activities is, however, not verbalizable. The ability to play music, for instance, is learned to a great extent

by example or through mimicking other players, rather than through memorizing explicit rules concerning the musical style in question. Furthermore, musicians themselves often find it difficult or even impossible to analyze their own performances. Connectionist models, being capable of extracting implicit knowledge from examples, offer a more plausible alternative for modelling these kinds of cognitive processes.

Many tasks of musical cognition involve low-level processing of often noisy or distorted sound data. These include, for instance, the perception of pitch, the recognition of timbre (i.e., sound colour), and the localization and segregation of sound sources. Attempts to model these kinds of processes with traditional AI systems may prove to be intricate: in order to be capable of properly dealing with noisy or distorted input data, such models should probably be equipped with a multitude of rules. Even relatively simple connectionist systems, on the other hand, have been shown to tolerate noise and distortions present in such low-level tasks of musical cognition.

2 CONNECTIONIST MODELLING OF MUSIC: A REVIEW OF LITERATURE

2.1. Overviews

General directions of connectionist modelling of music have been outlined at least by Bharucha (1988), Camurri, Haus, and Zaccaria (1986), Leman (1988, 1989), Lischka (1991), Loy (1991), and Marsden and Pople (1989a, 1989b). Bharucha (1988) discusses several models of music cognition, including a constraint satisfaction network for Western harmony, an auto-associative network simulating cross-cultural differences in tonal implications, and back-propagation networks that learn sequential musical schemata and specific musical sequences. Camurri et al. (1986) outline how musical processes can be described and performed by means of Petri nets. Leman deals with the question how sequential musical information can be stored and processed in a connectionist network (1988), and outlines a general background for the application of connectionist systems to music (1989), as well as presenting examples of musical applications of spreading activation networks, constraint-satisfaction networks, supervised learning networks, and self-organizing networks. A survey of current research paradigms in cognitive musicology is provided by Lischka (1991), followed by a critique of their basic assumptions and a suggestion for an alternative, more biologically-based, approach. Loy (1991) presents an informal overview of some of the traditional interests and problems of music research in the computer music community and describes the influence of connectionist theories on them. A distributed framework of rule-based

systems for modelling musical behaviour is proposed by Marsden and Pople (1989a, 1989b).

As stated above, connectionist systems lend themselves especially well to modelling low-level cognitive processes such as perceptual learning and subconscious processing of information. Attempts to model such musical processes have been carried out by a number of researchers. These studies cover such fields of musical cognition as the perception of pitch, harmony, tonality, timbre, melody, rhythm, metre, and musical sequences.

2.2. Pitch

The perception of pitch has been modelled within the connectionist paradigm by Sano and Jenkins (1989), Laden (1994), and Taylor and Greenhough (1994). Sano and Jenkins (1989) use a feedforward network to reduce the level of information from a spectral (cochlear) representation through a semitone bucket representation to a unified pitch representation, consisting of the octave and the normalized pitch. Their model works only with single complex tones. In a later article, they (Jenkins, 1991) suggest how their model could be extended to process multiple tone inputs. Laden (1994) describes a parallel learning algorithm based on the notion that a stimulus does not need to be physically present for a response to be learned. In the algorithm, exposing the network to a stimulus at one pitch only is enough to specify the harmonic template; the rest of the pitches are learned by a simple copy-and-translate procedure. Taylor and Greenhough (1994) utilize a self-organizing network architecture called ARTMAP, which is based on adaptive resonance theory (ART) networks. According to them, their model is capable of developing a great insensitivity to phase, timbre, and loudness when classifying pitch.

2.3. Harmony, key, and tonality

Connectionist studies on the perception of harmony, key, or tonality have been carried out by a number of researchers (Bharucha, 1987, 1991, 1994; Bharucha & Todd, 1989; Scarborough, Miller & Jones, 1989; Laden & Keefe, 1989; Beyls, 1990; Gjerdingen, 1989, 1990, 1992; Leman, 1990, 1991, 1992a, 1992b; Griffith, 1993, 1994). Bharucha (1987, 1991) presents a constraint satisfaction network called MUSACT, which is capable of extracting chords from tones and keys from chords. He also (Bharucha, 1991) discusses different possibilities for the representation of pitch as well as their strengths and weaknesses. Bharucha and Todd (1989) explore how connectionist systems can be employed to model the acquisition of tonal expectations through passive exposure (see also Bharucha, 1994). Using a back-propagation network with a decaying memory and a cascading algorithm (McClelland & Rumelhart, 1988), they found that their model learned to absorb cultural schematic expectancies, even though it was trained to produce only specific veridical expectancies.

Scarborough et al. (1989) present a feedforward network which identifies tonality from temporally integrated input data, represented as notes of

the chromatic scale. They also suggest a network architecture to map pitch nodes into scale degrees. Laden and Keefe (1989) discuss alternatives for representing pitch in a neural net model for pitch classification, including pitch-class, harmonic complex, and subharmonic complex representations. Beyls (1990) explores how complex dynamical systems and self-organizing systems can be used for modelling the emergence of musical morphologies.

Gjerdingen (1989, 1990) describes a self-organizing network architecture, based on ART 2 networks (Carpenter & Grossberg, 1987), intended to extract high-level concepts from a stream of musical data. When exposed to some early works of Mozart, the network is capable of achieving a musically sound material categorization of chords and chord pairs. In an attempt to address the so-called 'temporal chunking problem', Gjerdingen (1992) employs a masking field (Cohen & Grossberg, 1987) embedded in an ART 3 architecture. After having been trained on a sample of 535 chords from solo sonatas by Handel, the network was found to be capable of extracting the varying-scale embedded temporal patterns of chords that characterize harmonic syntax.

Leman (1990, 1991) outlines a model for the study of the ontogenesis of tonal functions. He uses a distributed representation of chords, based on Terhadt's psychoacoustical theory of tone perception (Terhardt, Stoll & Seewann, 1982), and Kohonen's self-organizing neural network (Kohonen, 1989). The network is found to organize basically in terms of the circle of fifths. Furthermore, when the statistical distribution of the chords is set to correspond to that in the music of Beethoven, Schubert, and Brahms, the network is found to give the most stable response, in terms of the error between the input vector and the synaptic vector of its characteristic neuron, to those chords that have the highest frequency of occurrence. While chords in the aforementioned studies are considered as static, time-independent objects, in later studies Leman (1992a, 1992b) adopts a dynamic approach: from a stream of input data, he creates a tone context by temporal integration. When trained on the thus obtained input vectors, the Kohonen network is found to develop a response structure which correlates strongly with Krumhansl's (1990) psychological data.

Griffith (1993,1994) aims to model how people establish a sense of tonality and encode pitch invariance. Comprising modular combinations of various forms of shunting, adding, and tracking memory, as well as ART 2 networks, Kohonen feature maps, and feedforward nets, his model self-organizes to categorize both stable tonal centres and scale degrees.

2.4. Timbre

For timbre recognition and classification, connectionist systems have been designed by Bertelli et al. (1991), Feiten, Frank, and Ungvary (1991), De Poli, Prandoni, and Tonella (1993), Cosi, De Poli, and Lauzzana (1994), Feiten and Günzel (1994), and Casey (1994). Bertelli et al. (1991) present a model for recognizing sound sources, based on FFT, a simple ear model, and the Kohonen feature map. Feiten et al. (1991) used the spectra of 102 synthetic random sounds of six different categories as input to a Kohonen

feature map. They found that the network was able to map the sounds in agreement with the predefined categories. They also describe a time delayed neural net for recognizing sonic oppositions as verbal attributes. De Poli et al. (1993) used a three-dimensional version of the Kohonen feature map. As input they used the sound stimuli of Grey (1975), in order to reconstruct Grey's timbre space. According to them, the analogies with Grey's results were encouraging. Cosi et al. (1994) used an auditory model and the Kohonen network to map the sounds of 12 acoustic instruments in both clean and noisy conditions. The obtained map showed a topological organization which was found to agree with subjective classification of those sounds. Moreover, the Kohonen network was able to recognize noisy versions of the sounds. Employing two hierarchical Kohonen networks, Feiten and Günzel (1994) treated dynamic sounds as sequences of steady-state components. The first Kohonen network mapped the steady-state spectra; the trajectories obtained were then used as input to the second Kohonen network. Casey (1994) makes use of feedforward networks in an approach to parameter estimation for physical models of sound-generating systems. He shows that such network models are appropriate for learning to map sounds to parametric representations.

2.5. Rhythm and metre

The processing of temporal aspects such as rhythm and metre plays an important role in music. However, the mechanisms behind it are not yet well understood. Maybe for this reason the connectionist studies dealing with these aspects are not as numerous as those related to pitch, harmony, and tonality. Attempts to model the perception of rhythm and metre (Desain & Honing, 1989; McGraw, Montante & Chalmers, 1991; Scarborough, Miller & Jones, 1992; Large & Kolen, 1994; McAuley, 1994) relate to broader topics such as musical expectancy (Bharucha & Todd, 1989) and time series prediction (Dirst & Weigend, 1993). Desain and Honing (1991) have developed a connectionist quantizer which is capable of inferring the metre from input data containing timing variations. Their model works to adjust perceived inter-onset intervals so that every pair of those intervals is adjusted toward an integer ratio, if it is already close to one. McGraw et al. (1991) trained various recurrent networks as beat detectors; they found that a network trained on one melody at different tempi may not correctly respond to the same melody played at a tempo not included in the training set. Scarborough et al. (1992) applied the parallel constraint satisfaction paradigm in their model of metre perception called BeatNet. It consists of an array of idealized low-frequency oscillators with different periods that operate to synchronize their output with event onsets, producing a metrical grid of the style suggested by Lerdahl and Jackendoff (1983). Large and Kolen (1994) introduce a novel connectionist unit capable of phase- and frequency-locking to periodic components of incoming rhythmic patterns. Networks of these units can self-organize temporally structured responses to rhythmic patterns. McAuley (1994) uses a network of integrate-and-fire oscillators capable of synchronizing with incoming rhythmic data.

2.6. Perception and generation of musical sequences

While the previously mentioned studies can be regarded as attempts to model musical perception and cognition, many of the connectionist approaches to the generation of musical sequences do not emphasize this aspect, but rather aim to exploit the strengths of connectionist systems for more straightforward musical applications, such as algorithmic composition.

Connectionist models of algorithmic composition have been proposed by Todd (1989, 1991), Lewis (1991), Mozer (1991, 1994), Stevens and Wiles (1993), and Bellgard and Tsang (1994). Todd's (1989) approach is based on Jordan's (1986) sequential network architecture: a three-layer back-propagation network processing one note at a time, with feedback connections from the output layer to the input layer. Temporal context is provided by integrating the activation values of the input units. The compositions produced by this network suffer from lack of global structure; to overcome that, Todd (1991) suggests an architecture with two hierarchically connected sequential networks. A similar note-by-note technique has been used by Stevens and Wiles (1993). Lewis (1991) presents a connectionist approach to algorithmic composition which he calls Creation by Refinement. The network is trained by a supervised gradient descent algorithm (back-propagation of error) to be a "music critic". To compose melodies, the input units are first randomly initialized. The input vector is then refined, using a second gradient descent search, so that the output vector obtained will satisfy the desired criteria. Bellgard and Tsang (1994) demonstrate how Boltzmann machine networks (Hinton & Sejnowski, 1986) can be used for harmonizing chorales.

Mozer (1991, 1994) utilizes a recurrent network trained by a variation of the back-propagation of error algorithm, referred to as the unfolding of time procedure (Rumelhart, Hinton & Williams, 1986), for note-by-note composition. In his model, he uses the distributed representation of pitch suggested by Shepard (1982): pitch is represented in a five-dimensional space. In this space, each pitch is specified by the pitch height as well as points on the chroma circle and the circle of fifths. Owing to this more advanced representation his model seems to perform better than that of Todd (1989), but the melodies produced still lack coherence and structure.

The aforementioned studies employ networks which are trained using variants of the back-propagation of error algorithm. While having manifested its power in a number of connectionist applications, this algorithm might be regarded as farfetched from a cognitive point of view: it is difficult to imagine how such a supervised learning procedure would correspond to the way humans adapt to the environment. More plausible models of the perception and generation of musical sequences have been proposed by Page (1994) and Kaipainen (1994); both are based on self-organization. Page (1994) criticizes previous connectionist approaches to algorithmic composition for being inappropriate as models of perception. His criticism is based on the following arguments: the networks are not self-organizing; they do not allow incremental learning; they require the training set to be presented to them hundreds of times before learning is

achieved; they are not able to perform melody recognition; they do not lend themselves to the formation of network hierarchies; and the back-propagation of error learning algorithm lacks biological plausibility. His own approach is based on hierarchical ART 2 networks (Carpenter & Grossberg, 1987) furnished with masking fields (Cohen & Grossberg, 1987). Having been trained on simple nursery-rhyme melodies, the network was probed with short musical sequences; the elicited musical expectations were found to correspond strongly with those suggested by the training set.

Kaipainen (1994) utilizes his self-organizing model, MuSeq, for demonstrating his dynamic theory of musical knowledge ecology. He postulates two directions of interaction, i.e., knowledge-acquisition and knowledge-use, and two kinds of knowledge, i.e., "knowing-what" for the recognition of the current musical situation and "knowing-how" for the determination of the consequences that actualize music. By varying the degree of dominance of the knowing-how, he is able simulate a continuum of musical behaviours: from an "autistic" repetitor at one extreme to a passive pattern-recognizer at the other.

2.7. Performance

Connectionist attempts to model musical performance have been carried out by Baggi (1992), Battel et al. (1993), Bresin et al. (1991), and Sayegh (1989). With his NeurSwing network, Baggi (1992) aims to investigate the constituents of swing in jazz music. Generating piano, bass, and drum output through MIDI in real time, his network consists of a harmonic net which produces chord substitutions for a given harmonic structure, and a stylistic net through which the degree of intensity, consonance, and interaction of the virtual rhythm section can be controlled. Battel et al. (1993) and Bresin et al. (1991) have carried out experiments on a back-propagation network trained to perform musical scores. In a listening test, the network was found to produce deviations in loudness, duration, and timbre which led to meaningful interpretations of the given compositions. Sayegh (1989) formulates the problem of fingering for string instruments as an optimization problem. He uses a connectionist system for minimizing a given cost function in order to find the best fingering for a given melodic phrase.

3 AIM OF THE STUDIES

The studies on which this thesis is based aim to explore certain music-related cognitive processes, using connectionist modelling. These processes include the classification and recognition of musical timbre (I, II, III) and pitch (VI), as well as the learning and generation of melodies (IV, V, VI). In more general terms, the studies attempt to investigate such processes as perceptual learning (I, II, III, VI), self-organization (I, II, III, VI), feature extraction (I, II, III, VI), sequential processing (I, IV, V, VI), autoassociative recall (IV, V), and short-term memory (I, IV, V, VI).

In connectionist modelling of timbre perception, the evaluation of the suggested models has until now remained half-finished; it has been mostly based on the researchers' intuitive interpretations of the responses of the networks. Studies II and III attempt to address this problem by comparing the timbre maps obtained with similarity ratings concerning the same set of stimuli. While the spectral content of a tone certainly has a significant role in timbre perception, there is evidence that other factors, such as attack transients, spectral gradients, and frequency and amplitude modulations, may be important as well. Until now, these factors have been neglected in connectionist timbre research. Study III endeavours to investigate the significance of these elements in timbre perception.

Within the rule-based paradigm, there have been several studies on improvisation (e.g., Fry, 1980; Ames and Domino, 1992; Bel & Kippen, 1992). While the models proposed in these studies perform rather well, they have a fundamental problem: their function is totally dependent on the personal views of the modeller about what are the essential aspects of improvisation. Study IV attempts to approach this modelling problem from a connectionist perspective. There the process of improvisation is described as a structured sequence of temporal associations; the latter are

produced by an auto-associative neural network. The network learns how to improvise in a given style by being exposed to excerpts of improvisations in that style.

Since music is fundamentally a temporal phenomenon, the question of how time is represented in a model of musical processing is essential. Within the connectionist paradigm, where all data is represented as vectors, the most straightforward way is to transform time into a spatial dimension: the temporal development of a quantity is represented by one large vector consisting of several subvectors, each of which represents the state of that quantity within a short period of time. Representations of this type have been used at least by Cosi et al. (1994), De Poli et al. (1993), and Feiten et al. (1991); this approach has also been adopted in studies II, III, IV, V. Such a representation of time can certainly be criticized for being rigid and farfetched from a neurophysiological point of view. Compromising the biological faithfulness of some aspects of the model, however, often allows the modeller to focus on other, more relevant ones without making the model too complicated. Another, often used (e.g., Gjerdingen, 1989, 1990, 1992; Scarborough et al., 1992; Todd, 1989) method of providing temporal context to sequential musical data is that of using leaky integrators: a given portion of the previous input vector is added to the current one. While this approach seems to correspond better to the way biological systems deal with temporal data, it still has some drawbacks, which are discussed more closely in article VI. In that article, we suggest a method of implementing short-term memory in a connectionist system which, we believe, combines the good temporal resolution of the time-window approach with the elasticity and biological plausibility of the simple leaky integrator method.

Due to its distributed nature, the behaviour of a connectionist system is often difficult to interpret. Perhaps for this reason many of the reports about connectionist studies concentrate on describing the details of the network used, while neglecting the analysis of the response of the system almost totally. The output of a connectionist model should, however, be compared with that of the system it aims to model in order to evaluate its performance. Accordingly, we have compared the responses of self-organizing networks on tone stimuli with those of human subjects on the same stimuli (II, III); a further way of evaluating the model would be to localize the responses on the auditory cortex to the stimuli used (III). Additionally, we have compared certain statistical properties of the output generated by a sequential autoassociative network with those of the material it was trained on (V).

4 MATERIALS, METHODS, AND RESULTS

The materials, methods, and results are described in detail in the original publications I–VI.

4.1. Self-organization of musical timbre (I-III)

The studies on musical timbre (I-III) are based on the Kohonen self-organizing map (KSOM) (Kohonen, 1989). Based on the assumption that lateral inhibition and redistribution of synaptic resources are responsible for self-organization in biological systems, the KSOM is capable of identifying the most salient features of the set of input vectors it has been exposed to, and mapping them onto a two-dimensional space while retaining the topological relationships of the vectors.

Study I introduces a hierarchical architecture of KSOMs, where aspects related to time and frequency are processed on different levels. It attempts to provide a neurobiologically more plausible, dynamic, alternative to the common, static way of processing the input vectors.[3] In the simulations, the network was found to be capable of mapping the 128 FM synthetic sounds of the training set in a way which mostly corresponded to the perceived similarities of the sounds.

While the acoustic preprocessing in study I was based on the Fast Fourier Transform, in studies II and III an advanced auditory model, designed by Van Immerseel and Martens (1992) and modified by Leman (1994), was used. In these studies, the set of input stimuli consisted of 27 sounds produced using additive synthesis. The responses of the KSOM to

---

3 A similar method has subsequently been used by Feiten and Günzel (1994).

these sounds were compared with similarity rating (SR) data, obtained using the same set of stimuli, by calculating Pearson's correlations. The correlation between the SR and KSOM data was of the same order of magnitude as the inter-subject correlations, supporting the hypothesis that it is appropriate to describe the timbre similarity rating behaviour in terms of a metric, analogous to that of the KSOM (II).

The main concern of study III was to explore to what degree certain dynamic aspects of sound, such as transients during the onset period, spectral gradients, and frequency and amplitude modulations, contribute to the perception of timbre. The method used was that of (1) constructing gradient images, which were supposed to qualitatively represent responses of auditory neurons sensitive to frequency and amplitude modulation; (2) varying the degree of emphasis on the onset of tones in the auditory images; and (3) varying the distance metric used in training the KSOM. The matrices of interstimulus distances, obtained both from the auditory images and the responses on the KSOM, were compared with the SR data by calculating Pearson's correlations. Using the methods mentioned above, a significant increase in correlation was achieved both for the auditory images and the responses of the KSOM; the main contributor to this was found to be the emphasizing of onset. The correlations obtained from the responses on the KSOM were found to be lower than the respective ones obtained from the auditory images; this would imply that it is not possible to project timbre onto two dimensions without distorting the metrical relationships between stimuli.

4.2. Sequential processing of melodies (IV-VI)

A sequential connectionist network which models the target-note technique of bebop jazz is presented in study IV. The model is based on a recurrent autoassociative network which receives external activation from context modules representing the harmonic context. It is taught by presenting it with melodic patterns together with the harmonic context. Having learned improvised jazz solos played by the trumpet player Clifford Brown, the network was found to be capable of applying its knowledge to a new harmonic context and producing stylistically fairly consistent melodies on the micro level. It was, however, unable to cope with larger structures such as melodic phrases. To overcome this, a hierarchical network architecture has been suggested elsewhere (Toiviainen, 1993). It comprises (1) a higher-level network, which would learn and produce the target notes of a longer harmonic progression; and (2) a short-term memory network capable of storing information about the melody produced earlier.

While the analysis of the output of the network in study IV was based mainly on the author's intuitive views and experience of bebop jazz, a more objective analysis was carried out in study V. The objective was to examine to what degree the network is able reproduce the tonal hierarchy which can be found in the improvisation it was trained on. The method of analysis, suggested by Järvinen (1995), is based on calculating the statistical distribution of the twelve tones in the chromatic scale on several metrical

levels. The tone profiles of the input and output of the network, obtained using this method, were compared by examining the frequencies of individual tones on each metrical level. The input and output were compared also on a more general level using the direction cosine as the measure of similarity. It was found that while the network successfully reproduced the local tonal characteristics of the training set, it was to a large extent unable to extract the information about the global tonal tendencies.

Study VI introduces a sequential self-organizing model which is capable of recognizing and generating melodies. The input data to the model is extracted from an acoustical stream of live music. The author's contribution to this study consists of the design and implementation of the preprocessing stage. In the first stage, the instantaneous pitch of the sound signal used as input is extracted by means of a method based on running autocorrelation. In the second stage, the stream of pitch vectors obtained is converted into another stream of vectors, where each vector concatenates an array of several subvectors, corresponding to a queue of memories with different time spans; the contents of these memories — referred to as short-to-long-term inhibitory memory pools (SLIMP) — are generated by temporal integration and an excitation-inhibition mechanism. We believe that the SLIMP method provides a neurally more plausible way of building a hierarchical memory in connectionist systems than the traditional time windowing praxis. The SLIMP preprocessing allows any number of memory pools to be chosen; the relative weights of them can be controlled parametrically. After exposure to ten test melodies, the model was found to be capable of both identifying the melodies, and making generalizations about classes of musical situations. The SLIMP preprocessing was found to provide, on one hand, a better temporal resolution than is obtained using the simple leaky integrator method, and, on the other hand, a better robustness with respect to minor tempo variations than is obtained using sharp time windowing.

5 CONCLUSIONS

The goal of the studies on which this thesis is based was to model music-related cognitive processes such as (1) perceptual learning and feature extraction in the classification of musical timbre; and (2) sequential processing, autoassociative recall, and memory in the recognition and generation of melodies. An essential part of the studies was the evaluation of the models; this was carried out by comparing the output of the networks either with the training set (V) or corresponding psychological data (III, IV).

The results of the studies support the view that the connectionist paradigm provides a plausible alternative for modelling the dynamics of certain cognitive processes, especially those involving perceptual learning by self-organization. Being inherently capable of generalizing, associating on the basis of content, and tolerating noisy or distorted input, artificial neural networks exhibit functions characteristic of the human way of perceiving, thinking, and acting.

The results of the experiments suggest that in constructing connectionist models which aim to extract knowledge from an acoustical input, one should pay a good deal of attention to the design of the preprocessing stage. Using the auditory model designed by Van Immerseel and Martens (1992), for instance, resulted in timbre maps which correlated with the similarity ratings much stronger than those obtained by representing the stimuli as physical amplitudes of the partials (II). Emphasizing the onsets of the stimuli and adding gradient images resulted in a further significant increase of that correlation (III). The SLIMP preprocessing stage used in study VI was also found to ameliorate the performance of the model notably, as compared to using simple leaky integrators.

While artificial neural networks fluently process static data, dealing with sequential input and output is a more arduous task. When choosing

between different representations of time, one often has to compromise between the performance of the network and the biological plausibility of the representation. For instance, in studies IV-V the former has been stressed at the expense of the latter. A crucial problem in artificial neural networks applied to music is how high-level features can be extracted from time-varying input data. According to our view, the SLIMP preprocessing method presented in study VI provides a means of constructing a hierarchical memory from a stream of acoustical data which combines good performance with neural plausibility.

The task of constructing exhaustive models of certain high-level cognitive processes of music, using the connectionist framework exclusively, may prove to be a challenging one. The improvisation of melodies, for instance, involves simultaneous processing on several hierarchical levels, such as melodic patterns, phrases and choruses. One possible way of addressing this type of problem would be to use hybrid models, or mixed connectionist-symbolic models (see, e.g., Clark, 1989; Hendler, 1989; Minsky, 1991; Gutknecht, 1992). While this approach seems to exploit the strengths of both paradigms, it may, however, bring about philosophical problems. A large part of human cognition can probably not be explained as based on the manipulation of symbols. Consequently, in any modelling approach striving for an increased psychological and biological plausibility, the justification of applying the rule-based paradigm should be carefully pondered. Research on modelling high-level cognitive processes with modular and hierarchical artificial neural networks is still in its infancy. This approach would deserve more attention in the future.

REFERENCES

Ames, C. & Domino, M. 1992. Cybernetic composer: an overview. In M. Balaban, K. Ebcioglu & O. Laske (Eds.), *Understanding music with AI: perspectives in music cognition.* Cambridge, MA: MIT Press.

Baggi, D. L. 1992. NeurSwing: an intelligent workbench for the investigation of swing in jazz. In D. L. Baggi (Ed.), *Readings in computer generated music.* Los Alamitos, CA: IEEE Computer Society Press, 79–94.

Battel, G. U., Bresin, R., De Poli, G. & Vidolin, A. 1993. Automatic performance of musical scores by means of artificial neural networks: evaluation with listening tests. In G. Haus & I. Pighi (Eds.), *Proc. of X Colloquium on Musical Informatics.* Milan: AIMI.

Bel, B. & Kippen, J. 1992. Bol processor grammars. In M. Balaban, K. Ebcioglu & O. Laske (Eds.), *Understanding music with AI: perspectives in music cognition.* Cambridge, MA: MIT Press.

Bellgard, M. I. & Tsang, C. P. 1994. Harmonizing music the Boltzmann way. *Connection Science,* 6(2-3), 281–298.

Bertelli, U., Bima, C., Camurri, A., Cattaneo, L., Jacono, P., Podestà, P. & Zaccaria, R. 1991. SOUL: un sensore acustico adattivo per il riconoscimento di sorgenti sonore. In A. Camurri & C. Canepa (Eds.), *Proc. of IX Colloquium on Musical Informatics.* Genoa: AIMI/DIST, 201–207.

Beyls, P. 1990. Musical morphologies from self-organizing systems. *Interface,* 19(2–3), 205–218.

Bharucha, J. J. 1987. Music cognition and perceptual facilitation: a connectionist framework. *Music Perception,* 5, 1–30.

Bharucha, J. J. 1988. Neural net modeling of music. *Proceedings of the first workshop on AI and music.* Minneapolis/St. Paul: AAAI-88, 173–182.

Bharucha, J. J. 1991. Pitch, harmony and neural nets: a psychological perspective. In P. M. Todd & D. G. Loy (Eds.), *Music and Connectionism.* Cambridge, MA: MIT Press, 84–99.

Bharucha, J. J. 1994. Tonality and expectation. In R. Aiello (Ed.), *Musical perceptions.* New York: Oxford University Press, 213–239.

Bharucha, J. J. & Todd, P. M. 1989. Modeling the perception of tonal structure with neural nets. *Computer Music Journal,* 13(4), 44–53. Reprinted in P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism.* Cambridge, MA: MIT Press, 128–137.

Bresin, R, De Poli, G. & Vidolin, A. 1991. Reti neurali per il controllo delle deviazioni temporali nell'escuzione musicale. In A. Camurri & C. Canepa (Eds.), *Proc. of IX Colloquium on Musical Informatics.* Genoa: AIMI/DIST, 201–207.

Camurri, A., Haus, G. & Zaccaria, R. 1986. Describing and performing musical processes by means of Petri nets. *Interface,* 15(1), 1–23.

Carpenter, G. A. & Grossberg, S. 1987. ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics,* 26, 4919–4930.

Carpenter, G. A. & Grossberg, S. 1990. ART 3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks,* 3, 129–152.

Casey, M. A. 1994. Understanding musical sound with forward models and physical models. *Connection Science,* 6(2–3), 355–371.

Cohen, M. A. & Grossberg, S. 1987. Masking fields: a massively parallel neural architecture for learning, recognizing, and predicting multiple groupings of patterned data. *Applied Optics,* 26, 1866–1891.

Clark, A. 1989. *Microcognition. Philosophy, cognitive science, and parallel distributed processing.* Cambridge, MA: MIT Press.

Cosi, P., De Poli, G. & Lauzzana, G. (1994). Auditory modelling and self-organizing neural networks for timbre classification. *Journal of New Music Research,* 23, 71–98.

De Poli, G., Prandoni, P. & Tonella, P. (1993). Timbre clustering by self-organizing neural networks. In G. Haus & I. Pighi (Eds.), *Proc. of X Colloquium on Musical Informatics.* Milan: AIMI.

Desain, P. & Honing, H. 1989. The quantization of musical time: a connectionist approach. *Computer Music Journal,* 13(3), 56–66. Reprinted in P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism.* Cambridge, MA: MIT Press, 150–160.

Dirst, M. & Weigend, A. S. 1993. Baroque forecasting: on completing J. S. Bach's last fugue. In A. Weigend & N. Gerschenfeld (Eds.), *Predicting the future and understanding the past.* Reading, MA: Addison-Wesley, 151–172.

Feiten, B., Frank, R. & Ungvary, T. (1991). Organisation of sounds with neural nets. *Proc. of 1991 ICMC.* San Francisco: ICMA.

Feiten, B. & Günzel, S. (1994). Automatic Indexing of a sound database using self-organizing neural nets. *Computer Music Journal,* 18(3), 53–65.

Fodor, J. & Pylyshyn, Z. 1988. Connectionism and cognitive architecture: a critical analysis. *Cognition,* 28, 3–71.

Fry, C. 1980. Computer improvisation. *Computer Music Journal*, 4(3), 48–55.

Gjerdingen, R. O. 1989. Using connectionist models to explore complex musical patterns. *Computer music journal*, 13(3), 67–75. Reprinted in P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism*. Cambridge, MA: MIT Press, 138–146.

Gjerdingen, R. O. 1990. Categorization of musical patterns by self-organizing neuronlike networks. *Music Perception*, 7(4), 339–370.

Gjerdingen, R. O. 1992. Learning syntactically significant temporal patterns of chords: a masking field embedded in an ART 3 architecture. *Neural Networks*, 5, 551-564.

Grey, J. M. 1975. *An exploration of musical timbre*. Ph. D. dissertation, Stanford University, Stanford, CA.

Griffith, N. 1993. Representing the tonality of musical sequences using neural nets. In J. Louhivuori & J. Laaksamo (Eds.), *Proceedings of the First International Conference on Cognitive Musicology*, Jyväskylä, Finland, 109–132.

Griffith, N. 1994. Development of tonal centres and abstract pitch as categorizations of pitch use. *Connection Science*, 6(2-3), 155–176.

Gutknecht, M. 1992. The 'postmodern mind': hybrid models of cognition. Connection Science, 4(3–4), 339–364.

Hendler, J. A. 1989. Marker-passing over microfeatures: towards a hybrid symbolic/connectionist model. *Cognitive Science*, 13, 79–106.

Hinton, G. E. & Sejnowski, T. J. 1986. Learning and relearning in Boltzmann machines. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press, 282–317.

Hopfield, J. J. 1984. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci. USA*, 81, 3088–3092.

Järvinen, T. 1995. Tonal hierarchies in jazz improvisation. *Music Perception*, 12(4), 415–438.

Jenkins, B. J. 1991. Addendum to: A neural network model for pitch perception. In P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism*. Cambridge, MA: MIT Press, 50–53.

Jordan, M. I. 1986. *Serial order: a parallel distributed processing approach.* Technical report ICS–8604. La Jolla: University of California, Institute for Cognitive Science.

Kaipainen, M. 1994. *Dynamics of musical knowledge ecology. Knowing-what and knowing-how in the world of sounds.* Ph. D. dissertation, University of Helsinki, Finland.

Katz, B. F. 1994. An ear for melody. *Connection Science*, 6(2-3), 299–324.

Kohonen, T. 1989. *Self-organization and associative memory*. (2nd Ed.) Berlin: Springer-Verlag.

Krumhansl, C. 1990. *Cognitive foundations of musical pitch*. New York: Oxford University Press.

Laden, B. 1994. A parallel learning model for pitch perception. *Journal of New Music Research*, 23(2), 133–144.

Laden, B. & Keefe, D. H. 1989. The representation of pitch in a neural net model of chord classification. *Computer Music Journal,* 13(4), 12–26. Reprinted in P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism.* Cambridge, MA: MIT Press, 64–78.

Large, E. W. & Kolen, J. F. 1994. Resonance and the perception of musical meter. *Connection Science,* 6(2-3), 177–208.

Leman, M. 1988. Sequential (musical) information processing with PDP-networks. *Proceedings of the first workshop on AI and music.* Minneapolis/St. Paul: AAAI-88, 163–172.

Leman, M. 1989. *Artificial neural networks in music research.* Reports from the seminar of musicology — Institute for psychoacoustics and electronic music. University of Ghent.

Leman, M. 1990. Emergent properties of tonality functions by self-organization. *Interface,* 19(2–3), 85–106.

Leman, M. 1991. The ontogenesis of tonal semantics: results of a computer study. In P. M. Todd & D. G. Loy (Eds.), *Music and Connectionism.* Cambridge, MA: MIT Press, 100–127.

Leman, M. 1992a. Tone context by pattern integration over time. In D. L. Baggi (Ed.): *Readings in computer generated music.* Los Alamitos, CA: IEEE Computer Society Press, 117–138.

Leman, M. 1992b. The theory of tone semantics: concept, foundation, and application. *Minds and Machines,* 2, 345–363.

Leman, M. 1993. Symbolic and subsymbolic description of music. In G. Haus (Ed.), *Music processing.* Madison, WI: A-R Editions.

Leman, M. 1994. Schema-based tone center recognition of musical signals. *Journal of New Music Research,* 23, 169–204.

Lerdahl, F. & Jackendoff, R. 1983. *A generative theory of tonal music.* Cambridge, MA: MIT Press.

Lewis, J. P. 1991. Creation by refinement and the problem of algorithmic music composition. In P. M. Todd & D. G. Loy (Eds.), *Music and Connectionism.* Cambridge, MA: MIT Press, 212–228.

Lischka, C. 1991. Understanding music cognition: a connectionist view. In G. De Poli, A. Piccialli & C. Roads (Eds.), *Representations of musical signals.* Cambridge, MA: MIT Press, 417–446.

Loy, D. G. 1991. Connectionism and musiconomy. In P. M. Todd & D. G. Loy (Eds.), *Music and Connectionism.* Cambridge, MA: MIT Press, 20–36.

Marsden, A. & Pople, A. 1989a. Modeling musical cognition as a community of experts. *Contemporary Music Review,* 3(1), 29–42.

Marsden, A. & Pople, A. 1989b. Towards a connected distributed model of musical listening. *Inteface,* 18(1–2), 61–72.

McClelland, J. L. & Rumelhart, D. E. 1988. *Explorations in parallel distributed processing.* Cambridge, MA: MIT Press.

McClelland, J. L., Rumelhart, D. E. & Hinton, G. E. 1986. The appeal of parallel distributed processing. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press, 3–44.

McGraw, G., Montante, R. & Chalmers, D. (1991) *Rap-master network: exploring temporal pattern recognition with recurrent networks.* Technical report no. 336. Computer Science Department, Indiana University.

Minsky, M. 1991. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine, 12,* 34–51.

Mozer, M. C. 1991. Connectionist music composition based on melodic, stylistic and psychophysical constraints. In P. M. Todd & D. G. Loy (Eds.), *Music and Connectionism.* Cambridge, MA: MIT Press, 195–212.

Mozer, M. C. 1994. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science,* 6(2-3), 247–280.

Page, M. P. A. 1994. Modelling the perception of musical sequences with self-organizing neural networks. *Connection Science,* 6(2-3), 223–246.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review,* 65, 386–408.

Rumelhart, D. E., Hinton, G. E. & McClelland, J. L. 1986. A general framework for parallel distributed processing. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press, 45–76.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. 1986. Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press, 318–362.

Rumelhart, D. E. & McClelland, J. L. 1986. PDP models and general issues in cognitive science. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press, 110–146.

Sano, H. & Jenkins, B. K. 1989. A neural net model for pitch perception. *Computer Music Journal,* 13(3), 41–48. Reprinted in P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism.* Cambridge, MA: MIT Press, 42–49.

Sayegh, S. 1989. Fingering for string instruments with the optimum path paradigm. *Computer Music Journal,* 13(3), 76–84. Reprinted in P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism.* Cambridge, MA: MIT Press, 243–251.

Scarborough, D. L., Miller, B. O. & Jones, J. A. 1989. Connectionist models for tonal analysis. *Computer Music Journal,* 13(3), 49–55. Reprinted in P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism.* Cambridge, MA: MIT Press, 54–60.

Scarborough, D. L., Miller, B. O. & Jones, J. A. 1992. On the perception of meter. In M. Balaban, K. Ebcioglu & O. Laske (Eds.), *Understanding music with AI: perspectives in music cognition.* Cambridge, MA: MIT Press, 427–447.

Shepard, R. N. (1982). Geometrical approximations to the structure of musical pitch. *Psychological Review,* 89, 305–333.

Smolensky, P. 1986. Information processing in dynamical systems: foundations of harmony theory. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press, 194–281.

Stevens, C. & Wiles, J. 1993. Representations of tonal music: a case study in the development of temporal relationships. In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. E. Elman & A. S. Weigend (Eds.), *Proceedings of the 1993 Connectionist Models Summer School.* Hillsdale, NJ: Erlbaum Associates, 228–235.

Taylor, I. & Greenhough, M. 1994. Modelling pitch perception with adaptive resonance theory artificial neural networks. *Connection Science,* 6(2-3), 135–154.

Terhardt, E., Stoll, G. & Seewann, M. 1982. Algorithm for extraction of pitch and pitch salience from complex tonal signals. *Journal of the Acoustical Society of America,* 71(3), 679–688.

Todd, P. M. 1989. A connectionistic approach to algorithmic composition. *Computer Music Journal,* 13(4), 27–43. Reprinted in P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism.* Cambridge, MA: MIT Press, 173–189.

Todd, P. M. 1991. Addendum to: A connectionist approach to algorithmic composition. In P. M. Todd & D. G. Loy (Eds.) 1991, *Music and Connectionism.* Cambridge, MA: MIT Press, 190–194.

Toiviainen, P. 1993. An artificial neural network approach for modelling harmony-based jazz improvisation. In H. Katayose (Ed.), *Proceedings of IAKTA/LIST international workshop on knowledge technology in the arts.* Osaka: IAKTA/LIST, 79–88.

Valentine, E. 1995. Deconstructing cognition: towards a framework for exploring non-conceptualised experience. In Pylkkänen, P. & Pylkkö, P. (Eds.), *New directions in cognitive science.* Publications of the Finnish Artificial Intelligence Society, 1–9.

Van Immerseel, L. M. & Martens, J.-P. 1992. Pitch and voiced/unvoiced determination with an auditory model. *Journal of the Acoustical Society of America,* 91(6), 3511–3526.

YHTEENVETO

Musiikin kognition mallintaminen keinotekoisilla hermoverkoilla

Keinotekoiset hermoverkot eli konnektionistiset systeemit ovat biologisten hermosolujen muodostamien verkkojen vahvasti idealisoituja malleja: niiden rakenteen ja toiminnan perusperiaatteet on lainattu biologisista hermoverkoista, mutta ne eivät yleensä pyri mallintamaan hermokudoksen fysiologisia prosesseja. Ne koostuvat useista toisiinsa kytketyistä yksinkertaista prosessointiyksiköistä l. keinotekoisista neuroneista. Keinotekoiset hermoverkot ovat rinnakkaisia: prosessointiyksiköt vuorovaikuttavat yhtäaikaisesti ja toisistaan riippumattomasti. Ne ovat hajautettuja: niiden tietämys sijaitsee neuronien välisissä kytkentävoimakkuuksissa, ja niiden käsittelemä tieto esitetään neuronien aktivaatiokuvioina. Ne ovat myös adaptiivisia: ne pystyvät oppimaan muuttamalla neuronien välisten kytkentöjen voimakkuuksia.

Kognition mallintamisessa on perinteisesti käytetty symboli- ja sääntöpohjaisia tekoälyjärjestelmiä. Suurta osaa kognitiosta on kuitenkin vaikea selittää rationaaliseksi, atomistiseksi ja eksplisiittiseksi järkeilyksi. On havaittu, että sellaiset kognitiiviset prosessit kuten havaitseminen, muisti, ajattelu, arvostelu ja ongelmanratkaisu voivat kaikki tapahtua implisiittisesti, tiedostamatta. Tällaisten ilmiöiden mallintaminen perinteisen tekoälyn keinoin voi olla ongelmallista. Keinotekoisia hermoverkkoja käyttävä mallintaminen, joissa mielen toiminnot nähdään holistisena prosessina ja tieto esitetään implisiittisessä muodossa, voi valottaa paremmin näiden mielen ilmiöiden takana olevia prosesseja.

Keinotekoiset hermoverkot luontuvat hyvin monien musiikin kognitiivisten prosessien tutkimiseen. Suuri osa musiikillisesta toiminnastamme ei ole kielellisesti kuvattavissa. Esimerkiksi soittamaan opimme enimmäkseen esimerkin avulla tai matkimalla sen sijaan että opettelisimme suuren joukon kyseiseen musiikkityyliin liittyviä eksplisiittisiä sääntöjä. Muusikoiden on usein myös vaikeaa tai jopa mahdotonta analysoida omia esityksiään. Monissa musiikin kognitiivisissa toiminnoissa käsitellään äänitietoa, joka usein sisältää kohinaa tai on muuten vääristynyttä. Musiikin kognitiota on tutkittu keinotekoisten hermoverkkojen avulla monesta näkökulmasta: verkoilla on mallinnettu mm. äänenkorkeuden, äänenvärin, rytmin ja metrin havaitsemista, tonaliteetin kehkeytymistä sekä musiikillisten sekvenssien havaitsemista ja tuottamista.

Tämä työ perustuu kuuteen erilliseen julkaisuun. Niissä on tutkittu keinotekoisten hermoverkkomallien avulla useita musiikin kognitioon liittyviä prosesseja: äänenvärin ja sävelkorkeuden tunnistusta sekä melodioiden oppimista, tunnistusta ja tuottamista. Yleisemmin sanoen työssä on tutkittu mm. aistinvaraista oppimista, itsejärjestäytymistä, piirreirroitusta, sarjallista prosessointia, autoassosiatiivista muistamista ja lyhytkestoista muistia.

Äänenvärin tunnistukseen liittyvät tutkimukset (I-III) perustuvat Kohosen itsejärjestäytyvään piirrekarttaan. Tämä perustuu sille oletukselle, että itsejärjestäytyminen biologisissa systeemeissä on lateraalisen inhibition ja synaptisten resurssien uudelleenjakautumisen tulos. Kohosen piirrekartta kykenee löytämään annetusta vektorijoukosta silmiinpistävimmät piirteet ja kuvaamaan ne kaksiulotteiselle pinnalle säilyttäen vektorijoukon topologiset suhteet.

Tutkimuksessa I esitellään Kohosen piirrekarttoihin perustuva hierarkkinen äänenvärintunnistusmalli, jossa aikaan ja taajuuteen liittyvät tekijät prosessoidaan eri tasoilla. Tarkoituksena on tarjota neurobiologisesti uskottavampi, ajan suhteen dynaaminen vaihtoehto perinteisille staattisille äänenvärintunnistusmalleille. Tehdyissä simulaatiossa mallin havaittiin pystyvän kuvaamaan sille syötetty aineisto tavalla, joka enimmäkseen vastasi ääniesimerkkien havaittuja samankaltaisuuksia.

Tutkimuksessa I äänien esikäsittely perustui nopeaan Fouriermuunnokseen, kun taas tutkimuksissa II ja III käytettiin laskennallista korvan mallia. Viimeksimainituissa tutkimuksissa käytettiin 27 additiivisella synteesillä tuotettua äänistimulusta. Kohosen piirrekartan vasteita näihin ääniin verrattiin Pearsonin korrelaatiokertoimen avulla yhdeksän koehenkilön samalla stimulusjoukolla tekemiin samankaltaisuusarviointeihin. Tulosten havaittiin korreloivan merkitsevästi. Tämän tulkittiin tukevan hypoteesia, että äänenvärien samankaltaisuuden arviointia voidaan kuvata Kohosen piirrekartan kanssa analogisella metriikalla.

Tutkimuksen III tavoitteena oli selvittää, missä määrin tietyt äänen dynaamiset ominaisuudet, kuten syttymisvaiheen transientit, spektrin gradientit sekä taajuus- ja amplitudimodulaatiot, vaikuttavat äänenvärin havaitsemiseen. Tätä tutkittiin muuntelemalla käytettyjen ääniesimerkkien esikäsittelyä ja vertaamalla näin saatuja Kohosen piirrekartan

vasteita vastaaviin samankaltaisuusarviointeihin. Äänistimuluksista tehtiin gradienttikuvia, joiden oletettiin kvalitatiivisesti esittävän taajuus- ja amplitudimodulaatioille herkkien kuuloaivokuoren neuronien vasteita; äänistimulusten alukkeiden korostusta muunneltiin; Kohosen piirrekartan opetuksessa käytettyä metriikkaa muunneltiin. Näitä menetelmiä käyttäen saavutettiin merkittävä korrelaation kasvu Kohosen kartan vasteiden ja samankaltaisuusarviointien välillä. Merkittävin tähän vaikuttanut tekijä oli äänien alukkeiden sopiva korostaminen. Alukkeet olivat siis tärkeitä äänien samankaltaisuutta arvioitaessa. Sen sijaan gradienttikuvilla ei havaittu olevan suurta merkitystä korrelaation kasvussa. Kohosen kartan vasteista saadut korrelaatiot olivat säännöllisesti matalampia kuin kartalle syötetyistä vektoreista lasketut. Tämä puoltaisi sitä, että äänenväriä ei voida projisoida kahteen ulottuvuuteen vääristämättä ärsykkeiden välisiä metrisiä suhteita.

Tutkimuksessa IV esitetään dynaaminen hermoverkko, joka mallintaa bebop-tyyliselle jazzimprovisaatiolle olennaista maalisävelteknikkaa. Simulaatioissa mallin havaittiin kykenevän soveltamaan esimerkkiaineistosta oppimaansa materiaalia uusiin sointurakenteisiin ja tuottamaan pintatasolla melko tyylinmukaista improvisaatiota. Se ei kuitenkaan kyennyt käsittelemään laajempia musiikillisia rakenteita. Mallin tuottamia improvisaatioita analysoitiin tilastollisesti tutkimuksessa V. Vertailemalla näiden sävelprofiileja esimerkkiaineiston vastaaviin profiileihin havaittiin, että malli kykeni toistamaan esimerkkiaineiston tonaaliset piirteet sointutasolla muttei pystynyt löytämään aineiston globaalia tonaalista rakennetta.

Tutkimuksessa VI mallinnetaan melodioiden tunnistamista ja generoimista itsejärjestäytyvällä keinotekoisella hermoverkolla. Mallille syötettävä akustinen äänisignaali esikäsitellään kahdessa vaiheessa. Ensin signaalin kulloinenkin äänenkorkeus päätellään autokorrelaatioon perustuvalla menetelmällä. Näin saadusta äänenkorkeusvektorijonosta rakennetaan aikaintegraation ja eksitaatio-inhibitiomekanismin (SLIMP) avulla hierarkkisen muistijäljen sisältävä vektorijono; tämä syötetään keinotekoiseen hermoverkkoon. SLIMP-menetelmä on neuraalisesti uskottavampi tapa rakentaa hierarkkinen muisti tämänkaltaisiin systeemeihin kuin perinteiset aikaikkunointiin perustuvat menetelmät.

I

The Organisation of Timbres: A Two-Stage Neural Network Model

by Petri Toiviainen

II

Musical Timbre: Similarity Ratings Correlate With Computational Feature Space Distances

by Petri Toiviainen, Mauri Kaipainen & Jukka Louhivuori

Journal of New Music Research, 24(3), 282–298
(reproduced with permission of Swets & Zeitlinger)

https://www.researchgate.net/profile/Petri-Toiviainen/publication/238277752_Musical_Timbre_Similarity_Ratings_Correlate_With_Computational_Feature_Space_Distances/links/58a6efdba6fdcc0e078ae261/Musical-Timbre-Similarity-Ratings-Correlate-With-Computational-Feature-Space-Distances.pdf

III

Optimizing Auditory Images and Distance Metrics for Self-Organizing Timbre Maps

by Petri Toiviainen

Journal of New Music Research (in press)
(reproduced with permission of Swets & Zeitlinger)

https://doi.org/10.1080/09298219608570695

IV

Modeling the Target-Note Technique of Bebop-Style Jazz Improvisation: An Artificial Neural Network Approach

by Petri Toiviainen

Music Perception, 12(4), 399–413
(reproduced with permission of
the University of California Press)

https://doi.org/10.2307/40285674

V

Connectionist Jazz and Tonal Hierarchy: A Statistical Multilevel Analysis

by Topi Järvinen & Petri Toiviainen

CONNECTIONIST JAZZ AND TONAL HIERARCHY: A STATISTICAL MULTILEVEL ANALYSIS

Topi Järvinen & Petri Toiviainen

ABSTRACT

A set of methods of analysis by which the output and functionality of a connectionist system can be evaluated is presented. By using these methods, the input and output of an artificial neural network designed to learn and produce jazz improvisation was analyzed and evaluated.

First, the statistical distribution of the tones in the chromatic scale was measured both over single chords and whole chord progression on four metrical levels. The obtained tone-frequency profiles were examined by comparing the frequencies of individual tones on each level. The input and output materials were compared also on a more general level by measuring the similarity of the tone profile vectors by computing intervectorial direction cosines.

The network was found to reproduce successfully the local tonal characteristics. It was, however, unable to extract the information about the global tonal hierarchy from the input that it was given. On the basis of the obtained evidence, improvements were suggested for both the network model and the method of analysis.

1 INTRODUCTION

Computer simulations have gained a relatively important position in music related research. Especially since the late 1980's a paradigm known as connectionism has been widely used for modeling various musical processes (e.g., Bharucha 1988, Leman 1988, Desain & Honing 1989, Gjerdingen 1989, Todd 1989, Bharucha & Todd 1989, Leman 1989, Gjerdingen 1990, Baggi 1992, Leman 1992, Gjerdingen 1992); connectionist models are neurally inspired non-linear dynamic systems. This approach has broadened our knowledge on the cognitive basis of different musical activities, such as listening, playing, and composing. With the aid of connectionist systems, or artificial neural networks, it has been possible to test the validity of theories of cognition regarding music. For example, with a connectionist system that has been designed to learn and produce music in a given style, it is possible to find aspects that are essential in learning the style in question. The behavior of neural networks is, however, often difficult to interpret, contrary to rule-based expert systems, in which the reasoning chain can always be followed. And perhaps due to this ambiguity the technical details of the models have often become the main interest: as a consequence the evaluation of the output in reference to the input has received less attention.

In this paper we present some statistical and mathematical methods by which the output of an artificial neural network can be analyzed and evaluated. The system the input and output of which is examined is a network designed to learn and produce bebop styled jazz improvisation (see, Toiviainen 1995). In particular, by calculating the statistical distribution of the twelve tones in the chromatic scale it is examined to what ex-

tent the network is able to extract the tonal hierarchy from the input it received; this is measured over both single chords and the whole chord progression. The notion of tonal hierarchy is an appealing way to approach this type of problem, for it allows us to study more general qualities, which may not be apparent in the surface level. Also in addition to the psychological proof for its existence (see, Krumhansl 1990) there is also evidence based on a large body of materials that it is an important factor in bebop styled jazz improvisations as well (Järvinen 1995). Consequently, a computer model simulating this type of music should also display comparable qualities. The goals of this study are, then, two-fold. First, to present a set of methods of analysis by which the output and functionality of a sequential connectionist system can be evaluated. Second, to analyze and evaluate the input and output of one particular model with the presented methods, and to suggest some ways in which it can be improved on the basis of the obtained evidence.

2 TONAL HIERARCHIES

2.1 Krumhansl's studies on tonal hierarchy

Although some music theorists have claimed that listeners perceive the tones in the chromatic scale to be hierarchically ordered (e.g., Meyer 1956, 214-215), only after Carol L. Krumhansl's and Roger N. Shepard's (1979) empirical tests has this claim been substantiated. By using a method referred to as probe-tone technique they were able to determine the relative perceived stability of the twelve chromatic tones in the given tonal context.

The results of this experiment can be seen in Figure 1, which represents the key profile in the C major context. The graph indicates clearly that there are differences in the perceived stability of the tones: highest ratings are given to the tonic (C) and the other two tones of the tonic triad (G, E), which are followed first by the rest of the diatonic scale (F, A, D, B) and finally by the non-diatonic tones (F♯/G♭, G♯/A♭, D♯/E♭, A♯/B♭, C♯/D♭) (Krumhansl 1990, 25-31). — There is also some data on the statistical distribution of the 12 chromatic tones in actual European art music, and those findings are also consistent with the empirical data (see Krumhansl 1990, 69–70; also cf. Knopoff & Hutchinson 1983, 95).

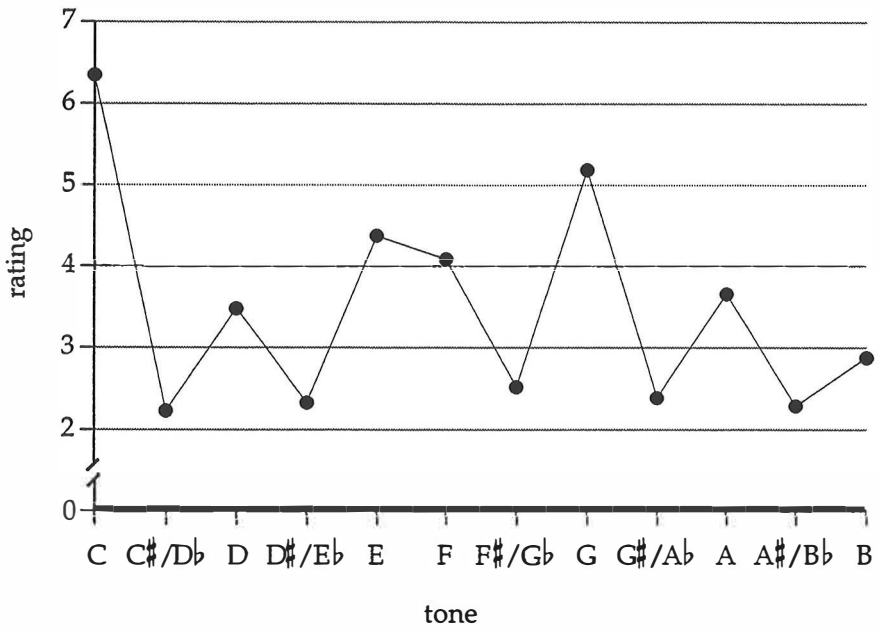


FIGURE 1. Probe tone ratings for major key context. The graph illustrates the results in reference to C major (Krumhansl 1990, 31).

Another study that further clarifies the nature of tonal hierarchy was made by Krumhansl and Edward J. Kessler (1982). They investigated how listeners' sense of key develops and changes while a given chord sequence progresses. The results suggest that the listeners develop a sense of key that is partly independent of the individual chords. At some point in the chord sequence, however, there seemed to be local effects of tonicization. Thus, sometimes the prevailing key was much stronger than the underlying chord, and sometimes the individual chords assumed a much greater role than their place in the tonal hierarchy would imply. (See Krumhansl & Kessler 1982, 356-357).

2.2 Tonal hierarchy in actual bebop styled jazz improvisations

The tonal hierarchy of bebop styled jazz was investigated in a study by Järvinen (1995) by analyzing the statistical distribution of the twelve chromatic tones in 56 improvised choruses[1] based on the so called Rhythm Changes -chord progression. In order to examine the effect meter has on the tonal hierarchy, the frequencies of tones were also contemplated on quarter (first, second, third, and fourth beat), half (first and third beat) and whole note level (first beat)[2]. Figure 2 shows the obtained global

1 In jazz the term chorus means an improvisation one time through the chord progression of the song; the length of the chorus depends on the song, but usually it is 12- or 32-bars long (blues or AABA, respectively).

2 Empirical evidence suggests that the tones on different metrical beats have different perceptual importance for the listener (Palmer & Krumhansl 1990, 734-736); it would seem plausible that the same holds true also for the improviser.

(chorus level) tone-frequency profile. These results indicate that in the analyzed improvisations there is a clear tonal hierarchy in which the tonic triad and the rest of the diatonic tones are favored over the non-diatonic tones. This is consistent with the perceived tonal hierarchy obtained in Krumhansl's psychological tests in which the same pattern of preference was found.

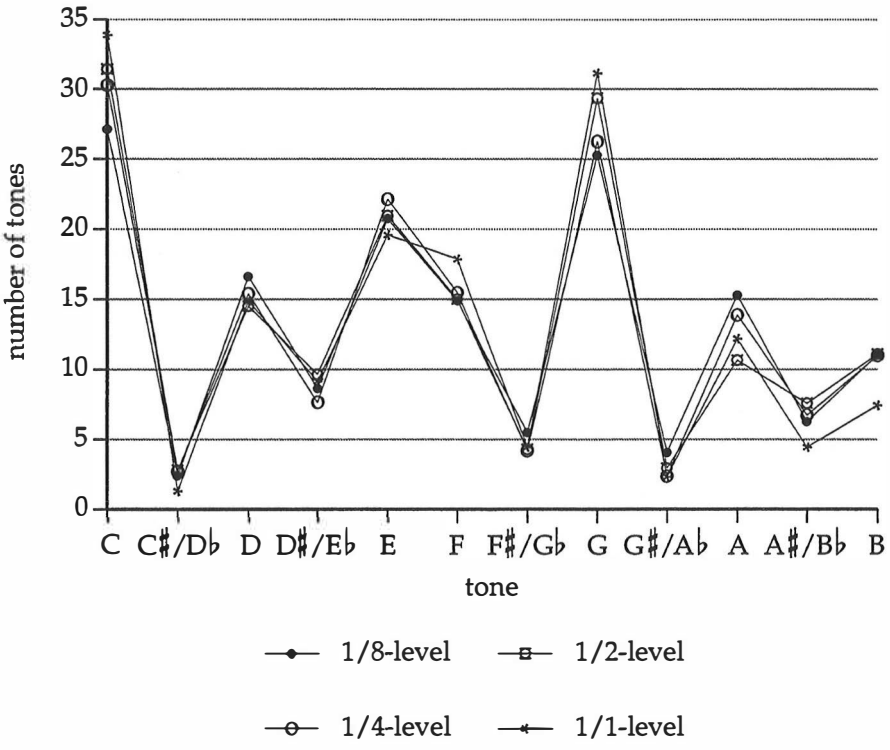


FIGURE 2. Weighted average chorus level profiles (Järvinen 1995).

Furthermore, the results indicated that there is a clear tendency to use the meter to emphasize or de-emphasize scale degrees depending on their tonal function. The important scale degrees of the C major tonality, namely the tones C and G, are used more frequently in the highest level (whole note) compared to the lower levels. The sixth scale degree (A), on the other hand, is sounded more frequently on the lower levels, which emphasizes its melodic function.

In addition to the global tonal hierarchy also the individual hierarchies of four chord functions (CM7, Dm7, FM7, and G7)[3] were investigated by analyzing thirty improvised choruses. It was found that each chord has its own tonal hierarchy in which chord tones are favored. Still,

3 The following chord type notations have been used (shown here in reference to the tone c): CM7 denotes major seventh chord, C7 dominant seventh chord, Cm7 minor seventh chord and C°7 diminished seventh chord.

the results for every chord were not as clear, for different chord functions were given unequal treatment. In the case of this particular chord sequence, namely the Rhythm Changes -progression, CM7 and FM7 were given more attention than Dm7 and G7. It appears that in the chord progression the improvisers have certain reference points that are outlined more carefully than the rest of the chords. The result of this is that in these reference points the hierarchical ordering of the tones is determined by the underlying chord (strong local hierarchy) whereas in other places the hierarchy is effected by the global tonal orientation of the chord progression (weak local hierarchy). This explanation is also in concordance with Krumhansl and Kessler's findings. — Furthermore, the role of the metrical structure was found to be important also in the chord level hierarchies.

3 CONNECTIONIST MODEL OF BEBOP STYLED JAZZ IMPROVISATION

This chapter describes cursorily the structure of the neural network model used in this study. A more thorough description can be found in Toiviainen (1995). Examples of the kind of improvisations that the network produces are presented in the same article. The model is based on target or goal note technique (see, e.g., Mehegan 1959, Berg 1990), which is a common way of explaining the production of an improvised jazz melody on the surface level. This technique can be described as follows: (1) it is based on the harmonic structure of the composition; (2) the notes of a four-note chord, and possibly its upper structure (9, 11, 13) are regarded as principal tones; (3) when approaching a chord, one of its principal tones is chosen as a target note; (4) the target note is reached through a melodic pattern. Usually the target note is preceded by a leading tone, i.e., a pitch neighbor along either the diatonic or the chromatic scale.

In order to achieve continuity in the improvised melodic lines, it is essential to be able to aim at target notes in advance. As the great jazz pianist Hal Galper (1982) puts it:

> There is an illusion going on in jazz that when you hear a cat playing a melody or a solo line the solo comes out sounding where it was played . . . , but in actuality, the player conceived it in advance of where he played it. If you start conceiving your ideas where you are, you will be late. (p. 63)

It would seem that in any approach aiming at modeling the process of jazz improvisation, this "forward motion" must be taken into account[4].

The target note technique bears a resemblance to J. J. Bharucha's (1984) concept of melodic anchoring. By this he means a principle of perceptual organization by virtue of which the listener assimilates unstable tones to the tonal schema. The basic idea of melodic anchoring is that an unstable, or dissonant, tone does not interfere with the tonal schema, if it is resolved into a tonally stable tone, or anchor, the two tones being proximal in pitch. According to Bharucha, there are two types of melodic anchoring, immediate and delayed. In immediate anchoring, a non-chord tone is immediately resolved into a chord tone that is a neighbor in either the diatonic or chromatic scale. In delayed anchoring, a nonchord tone is followed by another nonchord tone that satisfies immediate anchoring.

According to a general view among brain researchers, information processing operations within the brain can be expressed in terms of adaptive filter functions (Kohonen 1989, 14). Autoassociative recall is a particular type of adaptive filter operation. As Kohonen (1989, 15) puts it, an autoassociative memory is a system which can memorize a set $\mathbf{x}_i, i = 1,\ldots,n$, of vectors, and produce the copy of a particular vector $\mathbf{x}_k$ to the outputs, whenever the inputs are excited by a vector $\mathbf{y}$ in which a specified subset of components matches with the corresponding subset of components in $\mathbf{x}_k$. An autoassociative memory can, thus, recollect memorized information from a distorted or imperfect input pattern.

From an information theoretical point of view, the target note technique can be described as a series of temporal associations. Furthermore, the sequences of recollections are structured, i.e., they may branch into alternative sequences depending on the context information, provided by the harmonic structure. Temporal recall can be implemented by autoassociative memories using a system model depicted in Figure 3. The central block of the system is an autoassociative memory, receiving input from three sources: external input (K), context input (C), and feedback input (F). The system also has an output for recollection (R).

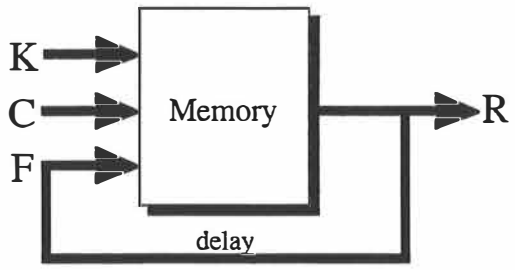


FIGURE 3. Associative memory for structured sequences (after Kohonen 1989).

4 The present model is only concerned with short-term targeting or forward motion. In actual jazz improvisation these phenomena can be found in relatively long time spans — it is only limited by the confines of the human memory.

In this approach, the autoassociative memory is implemented as an artificial neural network , which is a collection of completely interconnected simple processing units, or artificial neurons. Each neuron is associated with an activation value. The time evolution of the state of the network

$$\mathbf{a} \equiv (a_1,\ldots,a_N), \tag{1}$$

i.e., the ordered set of activation values of all neurons, is determined by the connection strengths w_{ij} between neurons. The learning of melodic patterns is carried out by strengthening the connections between active neurons, whereas the memorized patterns are recollected by a relaxation process, or flow of activation between neurons (e.g., Rumelhart & McClelland 1986, van Hemmen & Kühn 1991, Hecht-Nielsen 1990).

The learning and recollecting behavior of an autoassociative network can be described in terms of the energy function (Forrest & Wallace 1991, van Hemmen & Kühn 1991):

$$E(\mathbf{a}) \equiv -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}a_i a_j \,. \tag{2}$$

The learning process shapes the energy function so that local minima emerge at the locations of the patterns being memorized. During the relaxation process, the state of the network develops so that the energy function decreases monotonically; consequently, it finally reaches a local minimum corresponding to one of the memorized patterns. The ability of an auto-associator to recollect a memorized pattern from a distorted input depends on the size of the basin of attraction of that pattern, i.e., the part of the state space where the state of the network is attracted to that pattern.

The architecture of the network model is presented in Figure 4. The core of the model is an autoassociative network having six columns of neurons, each representing an eighth note of the melodic patterns. Each column further consists of 14 neurons, representing rest, ligature, and the 12 notes of the chromatic scale. The neurons within a same column have inhibitory interconnections in order to guarantee that, in the relaxed state, only one neuron in every group is active. An example of the representation of melodic patterns is presented in Figure 5. As can be seen, the network stores and recollects melodic patterns every half measure, with an overlap of two eighth notes between successive patterns.

The auto-associator receives input from three sources: (1) context input (C) provides information about the types of present (PC) and following (FC) chords; (2) feedback input (F) joins the melodic patterns together by providing a simple short-term memory; (3) external input (E) feeds a small random bias into each neuron in order to add variation to the sequences of output patterns. The input C deforms the energy function of the auto-associator so as to increase the sizes of the basins of attraction for those melodic patterns which are frequently used with the harmonic context in question.

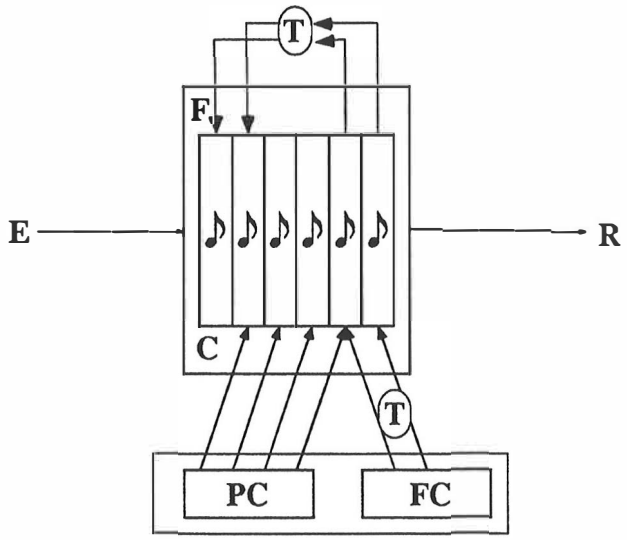


FIGURE 4. The architecture of the network model.

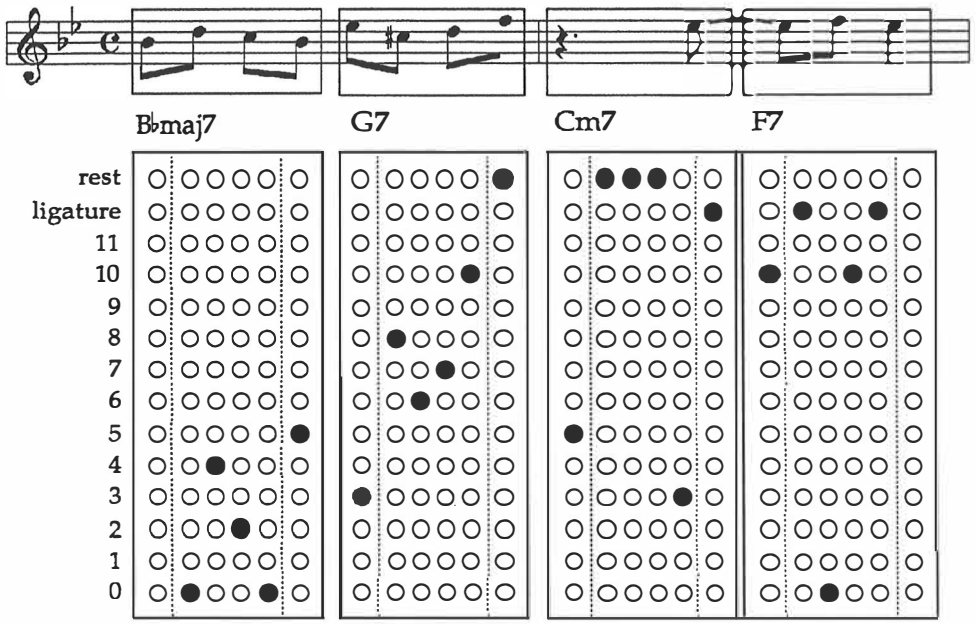


FIGURE 5. The representation of melody in the network model.

During the learning phase, one melodic pattern at a time is presented to the network: the neurons representing the notes of the melodic pattern are activated, together with the neurons representing the types of the present and the following chord. The learning occurs in two distinct groups of neurons: (1) the melodic patterns are learned through strengthening the connections between the active neurons of the auto-associator,

according to the Hebbian learning rule; (2) the relationship between the melodic patterns and the harmonic context is learned through strengthening the connections between the active neurons of the auto-associator and the active neurons of the context module.

The recollection of melodic patterns starts with activating the neurons representing the present and following chords, and the starting note. The network then relaxes, until a stable state (energy minimum) is reached; the melodic pattern represented by this state is fed into the output. After resetting the network to zero activation values, feeding back the last two notes of the melodic pattern into the starting notes of the next pattern, and updating the chords, the relaxation process is started again.

There are three parameters affecting the shaping of the energy function during the learning process and, thus, the time evolution of the system during recollection. These are (1) the strength of the inhibitory connections within each column (w_{inh}); (2) the learning rate inside the auto-associator (η_m); and (3) the learning rate between the auto-associator and the context module (η_c).

The network parameters can be thought to correspond to certain cognitive factors present in real improvisation. According to simulations, done with the model, high values of w_{inh} deform the energy surface of the auto-associator so that it has steep gradients. This causes the network to settle down quickly into a stable state, often producing new melodic patterns and target notes which never occurred in the training set. With low values of w_{inh}, again, the relaxation process takes a long time, and the recollection of learned melodic patterns is more accurate. Parameter w_{inh} could, thus, be called the "spontaneity" or "creativity" parameter. The ratio of parameters η_m and η_c contributes to the degree of emphasis on melodic and harmonic aspects of improvisation. With high values of η_m / η_c, the melodic patterns are recollected accurately, while they may not fit in the harmony. With low values of η_m / η_c, the output is faithful to the harmonic context, whereas the recollected melodic patterns often are modified versions of those in the training set.

4 MATERIALS, TRAINING PROCESS, AND METHODOLOGY

4.1 Description of the materials

As an input the artificial neural network received nine A-sections (72 measures) from an improvisation played by Hank Mobley on a chord progression known as Rhythm Changes (Mobley et al. 1956; Campbell 1989, 6-8). This particular progression, which is based on a popular composition called *I Got Rhythm* by George Gershwin, was used widely by jazz musicians in the bebop and hardbop eras. As was customary, Mobley did not use the original melody with the chord progression. Instead he substituted it with his own melody — hence the new title *Tenor Conclave.* Harmonically the Rhythm Changes progression is a very simple, major key progression, and it uses many of the same devices as most of the tonal jazz compositions. There are many alternative ways to play this particular progression; Figure 6 illustrates approximately how Mobley interpreted the basic chord changes.

Although the Rhythm Changes is a 32-bars long AABA form, the improvisations played on the B-section (Fig. 6) have been excluded. The reason is that the B-section is based on the circle of fifths -progression on dominant seventh chords, and thus no stable tonal center emerges. On the other hand, there are many chord substitutions in the actual improvisations in the B-sections: if the local hierarchies of individual chords were investigated, it would have distorted the results, because the body of ma-

terials is so small. Also, handling those substitutions would probably have been problematic with the present connectionist model.

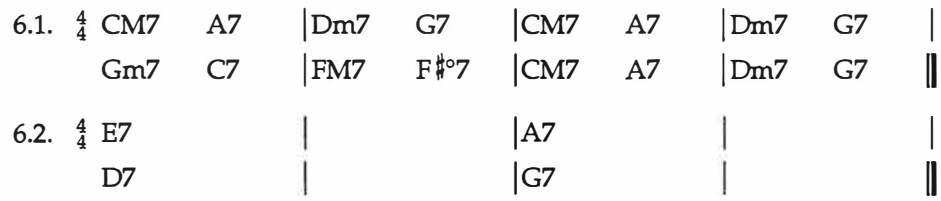


FIGURE 6. An approximation of Hank Mobley's interpretation of the Rhythm Changes -chord progression. Figure 6.1 shows the A-section and Figure 6.2 the B-section.

4.2 Network parameter values

Using the input material described in the previous section, a series of simulations was carried out. In each simulation, a differing set of values for the network parameters was used. The range of network parameter values was chosen experimentally, so that the output would, to a sufficient degree, be stylistically consistent with the input, while at the same time a sufficient amount of variation of melodic style could be achieved. Using these criteria the parameter values were limited within the ranges

$$0.10 < w_{inh} < 0.70,$$
$$0.04 < \eta_m < 0.28,$$
$$0.03 < \eta_c < 0.21.$$

Furthermore, the number of values of each parameter was set to four, those being

$$w_{inh} = 0.10, 0.30, 0.50, 0.70;$$
$$\eta_m = 0.04, 0.12, 0.20, 0.28;$$
$$\eta_c = 0.03, 0.09, 0.15, 0.21.$$

This yielded 64 values for the triplet (w_{inh}, η_m, η_c). With each value of the parameter triplet, the network produced melody on two A-sections of the Rhythm Changes -progression, i.e., 16 measures. The total output was, thus 64 x 16, or 1024 measures.

4.3 Analysis procedures

The principles behind the methods of analysis that were used in this study are explained in detail in Järvinen (1995); only the analytic procedures will be explained in the following. The basic idea is to count the frequency of each tone of the chromatic scale in the given piece of music. This is done on four metrical levels, namely the eighth, quarter, half and whole note levels. The eighth note level consists of all the sounded tones in a given piece of music. On the quarter note level the tones falling on

first, second, third and fourth beats are taken into account. Further, the half note level is obtained by counting the frequencies of the tones on first and third beats. Finally, on the whole note level only the tone on the first beat of each bar is included. The global level tone-frequency profiles are derived from the A-sections found in the body of materials, whereas the local (chord) level tone-frequency profiles are measured over five chords found in the same sections, namely CM7 (I), A7 (V/ii), Dm7 (ii), G7 (V), and FM7 (IV).

The same basic statistical procedures were performed on both the input material (Tenor Conclave) and on the output that the artificial neural network produced. The obtained tone-frequency profiles were examined by comparing the frequencies of individual tones on each level. The profiles were also compared on a more general level. A frequently used tool for comparing tone profiles is Pearson's correlation coefficient[5]. It, however, is somewhat problematic with this kind of material. The basic idea of Pearson's correlation coefficient is to estimate, on the basis of a sample of values of two random variables, to what degree it is possible to predict the values of one variable from those of the other (see, e.g., Harnett 1982). The frequencies of different tones, however, are clearly not sampled values of one and the same variable, but rather components of a vector-valued variable.

The problems caused by using the correlation coefficient for measuring the degree of similarity between profiles are not merely theoretical. From any profile, it is possible to construct another profile, such that the two profiles have a correlation value +1, but differ significantly from each other. This can be done, for instance, by shifting the values of the components away from their mean; see Appendix 1. The correlation coefficient is, thus, not an adequate similarity measure for analyzing the present data.

Such reasoning led us to treat the resulted pitch profiles as 12-dimensional vectors. This enabled us also to utilize various similarity and distance measures which better preserve the information content of the profiles. There are two commonly used approaches to measure the similarity of two vectors. (1) The distance between them can be calculated by using a given metric; usually either Euclidean or city-block metric is used. (2) A computationally simpler way, however, is to use the so-called direction cosine, which is the cosine of the mutual angle of the vectors. A more thorough description of the similarity measures can be found in Appendix 2 (see also Kohonen 1989, 59–67). The applicability of all three was tested, but no significant differences were found. It should be noted that in order to obtain reliable results with a given metric, the sizes of the materials being compared should be of similar magnitude. If the body of

5 Krumhansl, for example, has used Pearson's correlation coefficient for measuring the degree of similarity between key profiles (1990, 31-40). On p. 35 she states: "One method for assessing the degree of similarity between profiles is the statistical measure called correlation. This statistic takes a value from -1 (for patterns that are exactly opposite) to 1 (for patterns that are exactly the same)." This is not true, as has been proved in Appendix 1: the correlation of two profiles can be 1 even when the profiles differ significantly from each other.

materials is small, the distribution is highly quantized; this tends to decrease the degree of similarity between profiles. — To facilitate the comparisons between the different levels, the obtained tone-frequency profiles were normalized with respect to Euclidean metrics.

5 RESULTS

5.1 Global level tone-frequency profiles

The output material of a connectionist model of jazz improvisation was investigated by analyzing the statistical distribution of the tones of the chromatic scale within the C major context. The same procedures were also performed on the input material (an improvisation by tenor saxophonist Hank Mobley) that the neural network received. The input and output results will be compared on each of the four metrical levels, and also an average profile of both will be shown.

Figure 7 presents the tone-frequency profiles for both input (IP) and output (OP) on the eighth note level. The IP profile seems to have similar general characteristics that were found in Krumhansl's and her colleaques' tests as well as in Järvinen's study on actual improvisations: the most emphasis is given to the tonic triad, which is followed first by the rest of the diatonic scale and finally by the non-diatonic tones. While the OP profile also displays comparable properties, it differs in some crucial respects. First of all, the structurally important tones of the C major tonality, namely C and G, are used less frequently in the OP than in the IP; on the other hand, the tones D and A, which have more melodic function in C major are used quite frequently in the OP. Secondly, the neural network seems to amplify the differences between the frequencies of the tones that can be found in the input material: the non-diatonic tones D♯, F♯, G♯, and A♯, for example, are used less in OP than in IP, whereas tones C♯, E, F, and

A are used more frequently by the neural network. The most notable exceptions to this are the tones C, G, and B which receive relatively little emphasis. As a result, despite the distinctive hierarchy, the distribution of the tones in the input material is more equal (i.e. the approach is more chromatic) than in the output produced by the network.

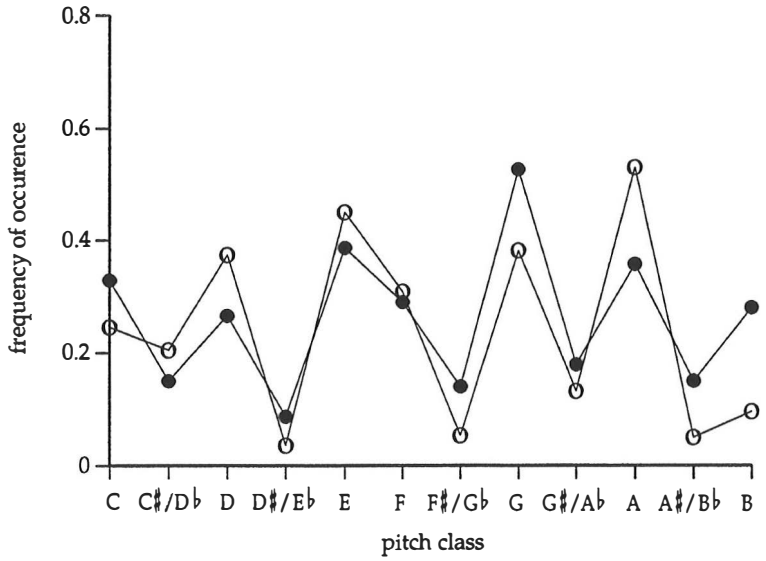


FIGURE 7. The global tone-frequency profile for the input (•) and output (o) materials on the eighth note level.

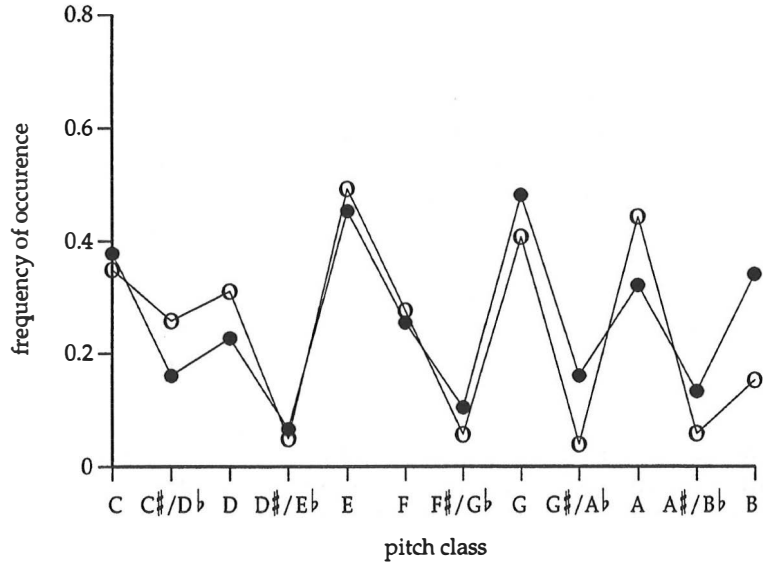


FIGURE 8. The global tone-frequency profile for the input (•) and output (o) materials on the quarter note level.

The IP and OP profiles for the quarter and half note levels (Fig. 8 and Fig. 9, respectively) show overall tendencies similar to the ones found in the

eighth note level. There are, however, some things that deserve attention. The artificial neural network seems to emphasize the tone C♯ metrically much more than the human improviser; especially on the quarter note level of OP it seems to hold a relatively stable position in the hierarchy. On the other hand, the network de-emphasizes metrically the other non-diatonic tones, most notably F♯, G♯, and A♯. The frequencies of the tones C and G are close, but unlike on the eighth and quarter note levels they are emphasized more in the OP than in the IP. Also the high frequency of the tone F in the OP should be noted. The whole note level profiles in Figure 10 are well in concordance with each other. Both, however, have some distinctive characteristics, although they show a clear and similar tonal hierarchy. The OP profile is diatonic with little emphasis on the non-diatonic tones, while the IP profile shows that on the whole note level there is relatively much chromatism.

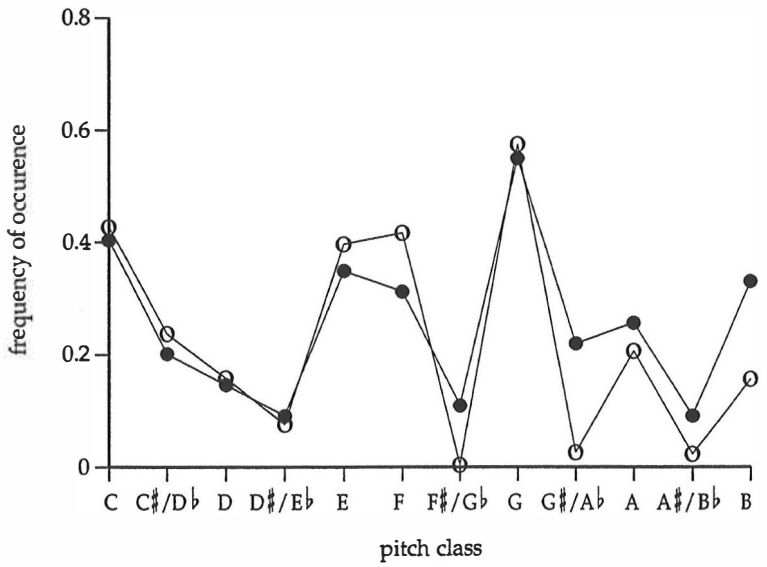


FIGURE 9. The global tone-frequency profile for the input (•) and output (o) materials on the half note level.

The weighted average profiles for both the IP and OP shown in Figure 11 summarize well the aforementioned points. It seems clear that the artificial neural network was able to produce a tonal hierarchy that is similar to the one that can be found in the input material and also in the earlier studies (see, e.g., Krumhansl 1990; Järvinen 1995).

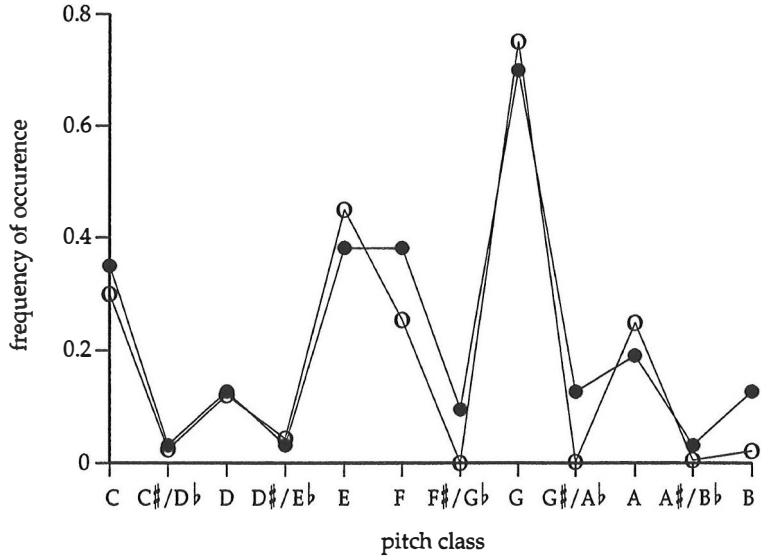


FIGURE 10. The global tone-frequency profile for the input (•) and output (o) materials on the whole note level.

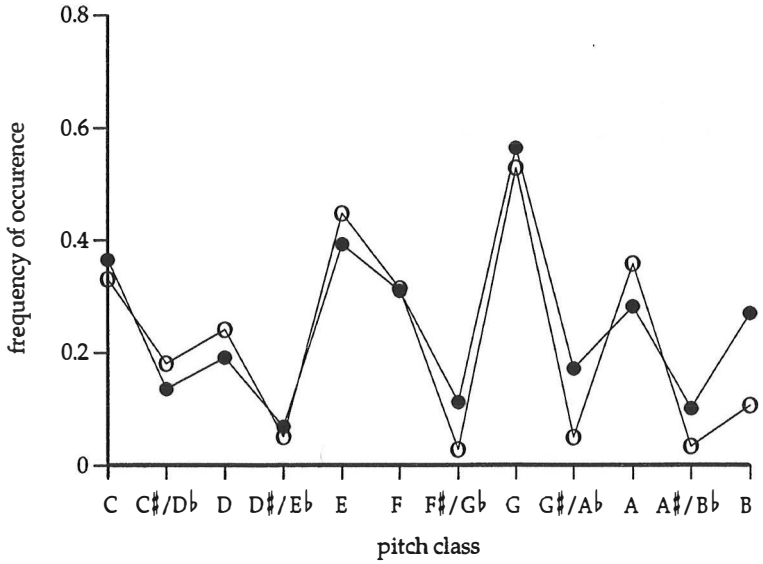


FIGURE 11. Weighted average tone-frequency profiles for the input (•) and output (o) materials.

Some of the divergencies can be probably explained by the linear nature of the learning algorithm used in this model: it may have caused some rarely used tones to occur even more infrequently in the OP (e.g., D♯, F♯, G♯, and A♯) and some frequently used tones to occur even more frequently (e.g., D, E, and A). As far as the IP is concerned the source of the relatively frequently occurring non-diatonic tones may be the blues scale (see Järvinen 1995), linear chromatic patterns, or chord substitutions (e.g.,

D7 is substituted for Dm7). From this it seems obvious that the global statistical distribution of the tones does not give clear enough indications of the possible similarities and differences the IP and OP may have. To investigate this further, we analyzed what kind of hierarchies are formed on individual chords — the results are presented in the next section.

5.2 Local level tone-frequency profiles

The individual hierarchies of five chord functions were investigated by analyzing the materials with the same statistical procedures that were used in the previous section; the only difference is that the whole note level is excluded, because in this progression each chord lasts for only two beats at a time. The chords are CM7 (I), A7 (V/ii), Dm7 (ii), FM7 (IV), and G7 (V).Figure 12 presents the weighted average values for the CM7 chord.

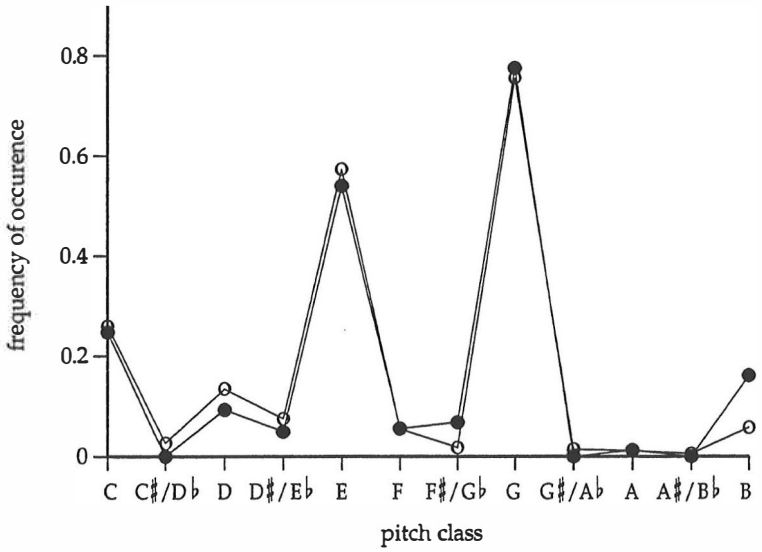


FIGURE 12. Chord level tone-frequency profiles obtained from the input (•) and output (o) materials for the CM7 chord.

The IP and OP profile follow each other very closely, and there are no significant differences. The priority of the tonic triad (C, E, and G) is clear in both profiles, for all other tones, diatonic and non-diatonic, are used considerably less often. The results for the next chord, namely A7, seem much more ambiguous (Fig. 13). The network clearly preferred tones A and C♯, i.e. the root and the third of the chord. On the other hand the IP profile shows a dissimilar pattern of preference with emphasis put on the chord tone C♯, but also on C and G, the important tones of the underlying tonality. Also there is fair amount of emphasis on the tone B, but in the improvisation it is used as an upper neighbor tone for the root of the chord.

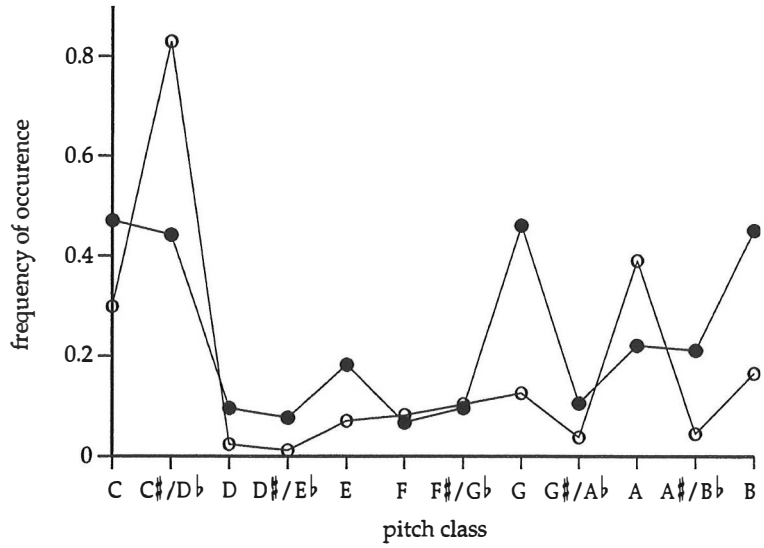


FIGURE 13. Chord level tone-frequency profiles obtained from the input (•) and output (o) materials for the A7 chord.

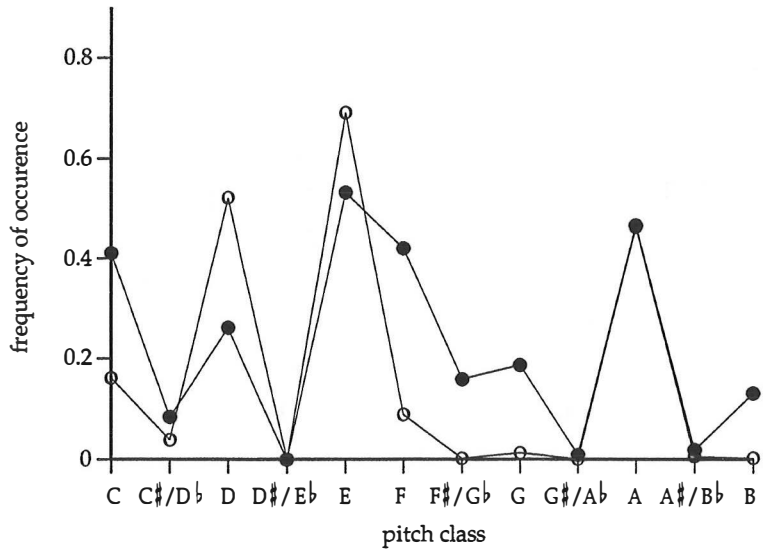


FIGURE 14. Chord level tone-frequency profiles obtained from the input (•) and output (o) materials for the Dm7 chord.

The results for the Dm7 chord are in Figure 14. The OP profile is dominated by three tones, D, E, and A, two of which are chord tones in Dm7. The third tone E can also be explained as an upper neighbor tone for the root D. In fact, the network seems to have got stuck at some point, because the motive E-C♯-D-E is used quite frequently. This may also be the reason why the third of Dm7, F, is used so little. The high frequency of the tone E in IP seems to be the result of a similar leading motion from E to D. Compared to the chord tones the central tones of the tonality are used in IP relatively frequently as was the case also in the A7 chord profile.

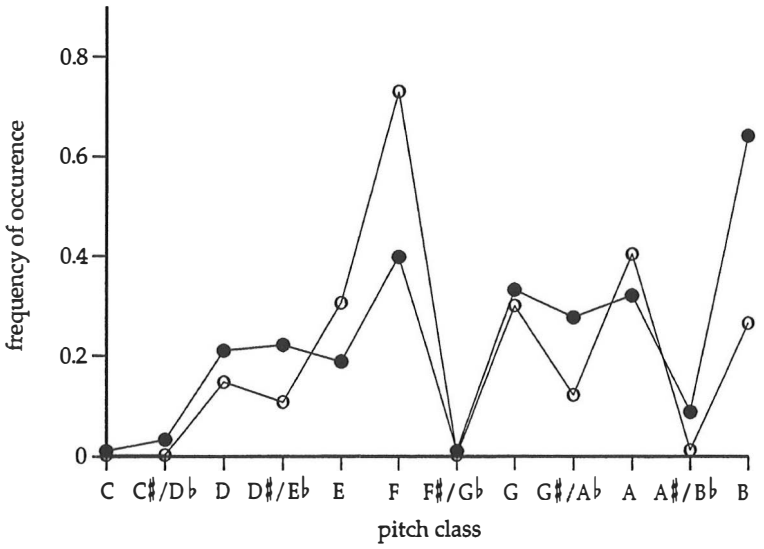


FIGURE 15. Chord level tone-frequency profiles obtained from the input (•) and output (o) materials for the G7 chord.

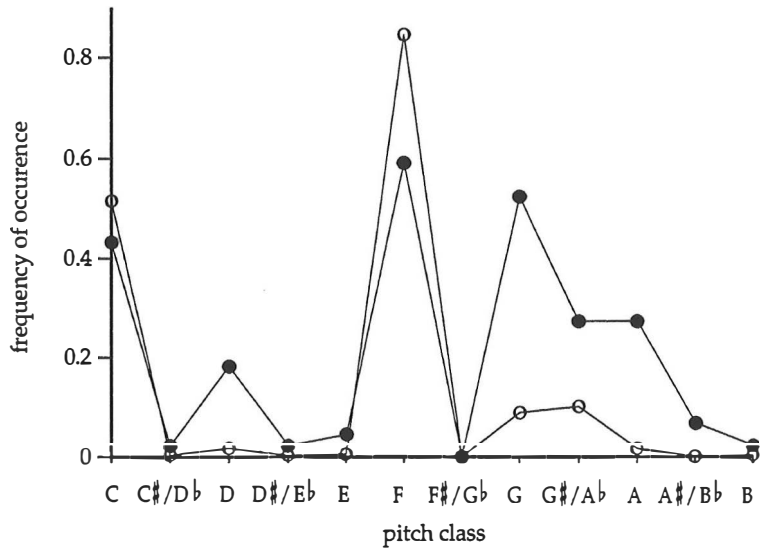


FIGURE 16. Chord level tone-frequency profiles obtained from the input (•) and output (o) materials for the FM7 chord.

Figure 15 represents the G7 profiles. The tone frequencies for the IP show a fairly chromatic approach, where non-chord tones and even non-diatonic tones are used quite frequently. Interestingly, instead of the root, the third of the chord, B, is the reference point for the improviser. The OP shows quite frequent use of most of the diatonic tones. Still, the most emphasis is put on F, which is partly attributable to a certain frequently used motive (F-G-A-G♯). The OP profile for the FM7 chord (Fig. 16) shows that the network used primarily the tones C and F. The IP profile shows a

more varied choice of tones, although the tones of the basic triad (F, A, and C) are clearly present. Again, the tone G is an upper leading tone which is resolved to F in the improvisation. These results, however, should be contemplated more cautiously than the ones on CM7, A7, Dm7, and G7, for there are three times fewer FM7 chords in the Rhythm Changes -chord progression.

To summarize these results, it seems that in the actual improvisation that was used as input there is a tendency to take both the underlying chord and the global tonality into account (cf., Järvinen 1995). In the output produced by the artificial neural network, however, the underlying chord seems to be the main factor that affects the use of tones. This is probably the cause of the discrepancies between the global IP and OP profiles: in the output produced by the network there is too much emphasis on the local hierarchies of the individual chords. As a result some tones (e.g., C♯, D, and A) occupy much higher place in the OP than in the IP tonal hierarchy. On the other hand, since the network does not take the global tonality into account, there is less emphasis on the central tones of the C major tonality, namely C and G.

5.3 Similarity of the profiles

While the profiles examined in the preceding section give us a lot of detailed information about the IP and OP, a more general and objective method seems to be needed to complement the previously made observations. In this section we will show one possible way to approach the IP and OP materials in a more comprehensive manner. The profiles are treated as 12-dimensional vectors the mutual similarity of which is computed by a mathematical measure referred to as direction cosine.

Figure 17 illustrates the obtained profile vector distances for the four metrical levels of the global tonal hierarchy. In addition to the direction cosine there are also the two distance measures that were introduced earlier for the sake of comparison. As is evident from the graph, all three give congruous results.

The direction cosine value is high for each metrical level: the IP and OP profiles are remarkably similar. Especially on the whole note level the difference is almost non-existent. The slightly lower values on the other levels are mainly due to the more frequent use of non-diatonic tones in the IP material. In fact, as the tone-frequency profiles showed in the previous section, the non-diatonic tones are used infrequently in the OP materials on every metrical level. On the other hand, it is customary in bebop styled jazz to play diatonic tones on the strong metrical positions especially on the downbeat of a measure. Consequently, on the whole note level both profiles demonstrate emphasis on the diatonic tones, and therefore the direction cosine shows high similarity on that particular level.

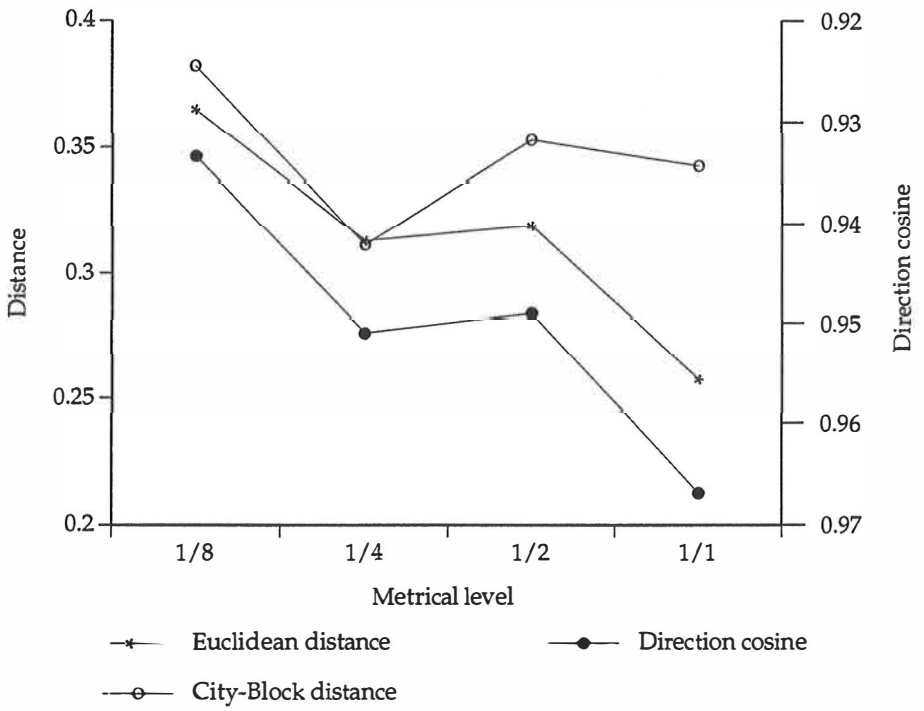


FIGURE 17. The similarity of the global input and output profiles on the four metrical levels. The results are given with three similarity measures, namely the Euclidean distance, the city-block distance, and the direction cosine. The values of the Euclidean and city-block distances are shown on the vertical axis on the left side, whereas those of the direction cosine can be read on the vertical axis on the right side.

The direction cosine values were also computed for five chords (CM7, A7, Dm7, FM7, and G7) found in the Rhythm Changes chord progression. Although all values are high as Figure 18 shows, the chord profiles do not match as neatly as the global profiles. The IP and OP profiles for the CM7 chord seem to be almost identical, for the similarity measure is almost 1. The direction cosine values are also high for the other chords, but the results are, nevertheless, markedly lower than the ones for the tonic chord. The secondary dominant for the Dm7 chord, namely A7, gets the lowest direction cosine value (less than 0.8), which may be due to the fact that it is the only chord in this five chord set that includes a non-diatonic tone as a chord tone (C♯). Consequently, since the network, unlike the human, seems to be unable to take the global tonality into account, there is discrepancy between the IP and OP profiles. The direction cosines for the Dm7, G7, and FM7 chords are slightly over 0.8. The reason for this discrepancy may be the extensive use of certain motives along with the emphasis on the local hierarchy of the underlying chord in the OP materials. It should be noted, however, that direction cosine for the FM7 chord is not totally comparable with the other direction cosines, because the body of materials is smaller than for the other chords.

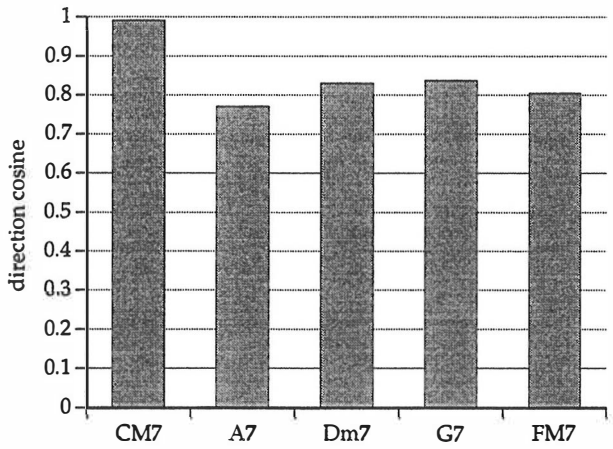


FIGURE 18. The direction cosines calculated between the input and output chord level tone-frequency profiles.

The results obtained with direction cosine measurement are similar to the ones reported in the previous section. They support the findings that the output produced by the artificial neural network appears to be similar to the input that was used in the training period. Furthermore the results show clearly that the network reproduces the global tonal hierarchy most successfully on the whole note level; on other levels there is more discrepancy. The distance cosine values for the various chord functions indicate that this divergence may be partly due to the network's inability to take the global tonal hierarchy into account. This is well demonstrated by the fact that the direction cosine value for the tonic chord (CM7) is higher than the values for the other chords. In other words, the network was most successful when it only had to be concerned with the tonal hierarchy of the underlying chord.

5.4 The effect of the network parameters

In order to analyze the effect of network parameter values on the output, tone profiles were also calculated separately for each parameter value combination. These were then compared with the corresponding input profiles by means of direction cosines. This yielded, for each metrical level, direction cosine values for 64 points of the three-dimensional network parameter space. As an example, Figures 19.1–19.3 present direction cosine values of input vs. output profiles on half-note level in three subsets of the network parameter space, keeping each time one parameter constant: (1) $w_{inh} = 0.30$; (2) $\eta_m = 0.12$; (3) $\eta_c = 0.09$. From Figure 19.1, for instance, it can be seen that, when w_{inh} had a constant value 0.3, the best match between the input and output tone profiles on half-note level was achieved with parameter values $\eta_m = 0.04$ and $\eta_c = 0.15$; moving away from this point of the parameter space resulted in a worse correspondence between the profiles. In a similar manner the optimal set of parameter values in both Figures 19.2 and 19.3 is $w_{inh} = 0.30$, $\eta_m = 0.12$, and $\eta_c = 0.09$.

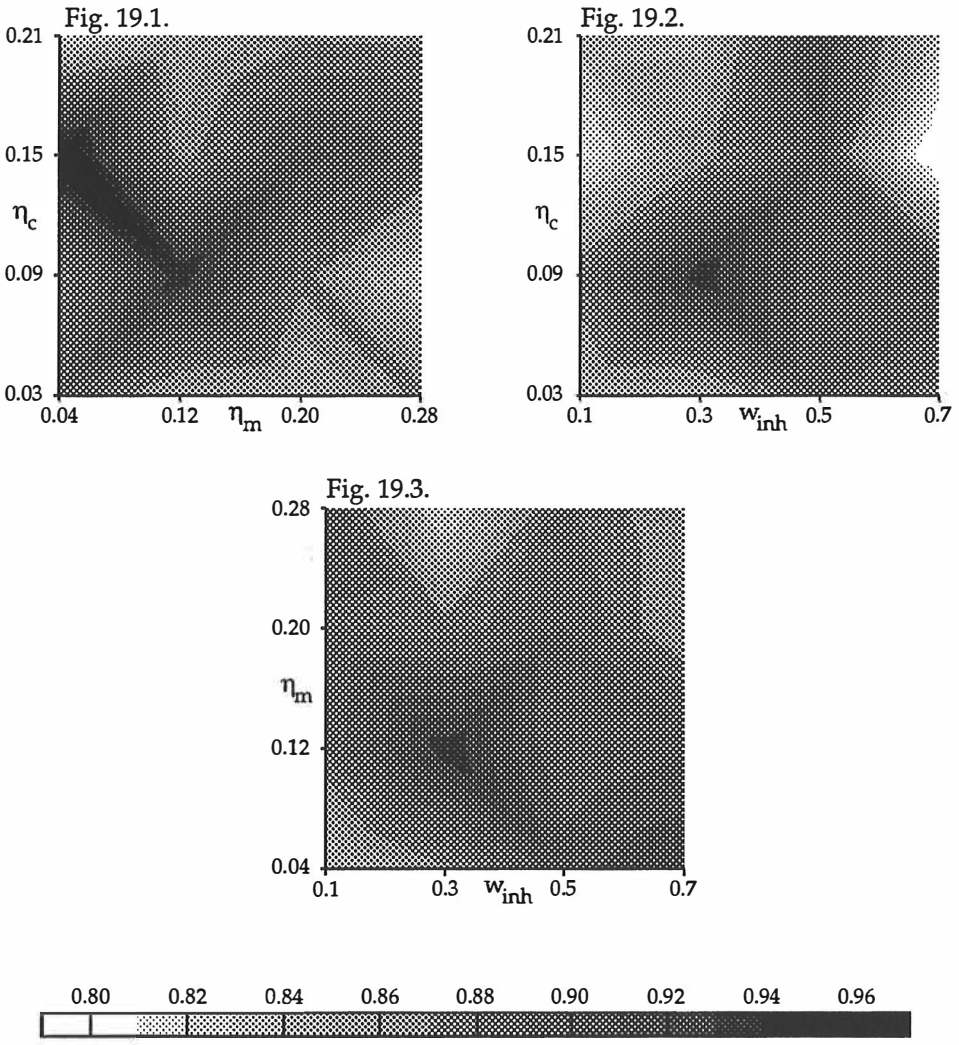


FIGURE 19. Three examples of the effect of network parameter values on the output. Each figure represents how the direction cosine calculated between the input and output profiles changes, when two network parameters are changed and one is kept constant. The constant parameters and their respective values are: w_{inh}= 0.30 (Fig. 19.1); η_m = 0.12 (Fig. 19.2); and η_c = 0.09 (Fig. 19.3). The direction cosines are calculated on the half note level. For the ease of reading, the direction cosine values have been linearly interpolated between the actual points that were used in the simulations.

In principle, this method could be used for finding the set of parameter values with highest similarity of tone profiles with the input data, and thus tuning the network to optimum performance. The present material, however, was found to be insufficient for that purpose: since the output with each parameter value combination consisted of only 16 measures, the relevant information was greatly obscured by statistical fluctuations.

6 SUMMARY AND DISCUSSION

The input and output of an artificial neural network designed to learn and produce jazz improvisation were compared by using statistical and mathematical methods. First, the statistical distribution of the tones in the chromatic scale was measured both over single chords and whole chord progression on four metrical levels. The obtained tone-frequency profiles were contemplated by comparing the frequencies of individual tones on each level. The input and output materials were compared also on a more general level by computing the intervectorial distances of the tone profiles with a similarity measure referred to as direction cosine. — As an input the network received an improvisation by tenor saxophonist Hank Mobley.

When the statistical distribution of the tones in the input and output materials were compared it was found that on the global level the general characteristics of the tone-frequency profiles were similar[6]: the tonic triad and the rest of the diatonic tones were favored over the non-diatonic tones. There were, however, some divergencies, for the input material displayed clear emphasis on the tones of the tonic chord whereas the network used most of the diatonic tones relatively frequently along with the tonic triad. These differences illustrated well how the approach used by the present computer model differs from the one that the human impro-

6 It is noteworthy that these hierarchies resemble also the kind of hierarchical ordering of the tones found in empirical (eg., Krumhansl 1990) and statistical (Järvinen, 1995; see also Knopoff & Hutchinson 1983) studies.

viser seems to utilize. Namely, the comparisons between the chord level tone-frequency profiles of five chord functions (CM7, A7, Dm7, FM7, and G7) indicated that in the output the local hierarchy of underlying chord was the main factor that affected the use of tones whereas the actual improvisation played by the musician (input) shows awareness of both the underlying chord and global tonality. Although the network reproduced successfully the local tonal characteristics, it was unable to extract the information about the global tonal tendencies from the input that it was given. Therefore it lacked the kind of overall tonal coherence that is characteristic of real jazz improvisations (cf., Järvinen 1995).

The similarity of the tone-frequency profiles was calculated with a similarity measure known as direction cosine. The results confirmed the aforementioned findings, for the direction cosines calculated for the global profiles yielded high values, although inconsistencies were apparent on the eighth, quarter, and half note levels. At least partly this can be explained by the way the artificial neural network and the musician handled the chords. The direction cosines for the chords show that the profiles for the tonic chord (CM7) are the most similar — the profiles for the other chords show considerably more discrepancy. In other words, the network was most successful when it only had to be concerned with the tonal hierarchy of the underlying chord. Another reason may be that the output displayed clear emphasis on the diatonic tones on all metrical levels. On the whole note level of the global tone-frequency profile the differences are not as evident, for there also the musician usually emphasizes diatonic tones more than on the lower levels.

The statistical and mathematical methods of analysis that were used in this study seem to be well suited for the evaluation of an artificial neural network designed to learn and produce music in a given style. These methods enable us to approach the atemporal qualities of music in a precise and relatively objective manner. The tone-frequency profiles, which were derived from the input and output materials, present us precise evidence about the tone preferences with respect to both single chords and the whole chord progression. Furthermore, the direction cosine gives easily accessible and reliable information about the overall similarity of the input and output profiles without getting into too much detail. The underlying consumptions of this study were closely related to the notion of western tonal hierarchy put forward most notably by Carol L. Krumhansl. The network that was contemplated in this study was designed to simulate bebop styled improvisation, which is a relatively tonal style of music. Therefore for our purposes Krumhansl's empirical findings along with statistical data on the tonal hierarchy in bebop styled jazz provided a good foundation. Hierarchical differentiation of musical elements, however, seems to be a basic cognitive principle regarding music in many cultures (see e.g., Castellano, Bharucha, & Krumhansl 1984, 411–412; Krumhansl 1990, 268–270), and in principle it seems reasonable to assume that a similar method could be used to evaluate connectionist systems designed to learn and produce other western as well as non-western musics.

Since one goal of this study was to find methods of analysis for the output of an artificial neural network, the extent of the input material

was kept relatively small. For this reason some of the actual results, however, should be considered cautiously. In particular, the differentiation between the four metrical levels is problematic, because even small motives which occur a few times can have an affect on the tone profiles. For example, the small three note motive in Figure 20.1 presents one such a case.

Fig. 20.1

FM7

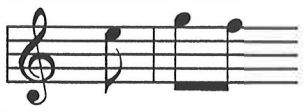


Fig. 20.2

FIGURE 20. Neighbor tone motive. Figure 20.1 shows how it is written in standard musical notation, and Figure 20.2 illustrates how a listener may perceive it.

If we consider the local tonal context, which is the FM7 chord, the first two tones, namely E and G, are clearly neighbor tones for the tone F that is played on the second eighth note of the measure. The musician both prepares the new chord by these neighbor tones and creates tension by delaying the root. Figure 20.2 illustrates the way a listener would probably perceive it: the tone F is the goal or target note of the first two tones or in Bharucha's terms E and G are be anchored to F (1984). In the statistical analysis, however, the tone G, which is on the downbeat of the measure gets eight times more emphasis than the tone F. Small idiosyncrasies of this kind are evened out in a large body of materials, but in small ones they may distort the results.

Another similar problem which might have distorted the tone profiles is that the network sometimes got stuck at some point and always produced the same melodic motive on a given chord. This behavior might be partly due to the linear learning algorithm. An autoassociative neural network which uses the Hebbian learning rule is somewhat limited in its ability to store patterns. When the number of patterns to be stored exceeds a critical storage ratio of about 0.15N (where N is the number of neurons), a perfect recall of patterns cannot be ensured. Above this limit the stored patterns start to interfere with each other, resulting in unpredictable behavior. The critical storage ratio can be improved by using non-linear error-correcting learning algorithms such as perceptron learning (Forrest & Wallace 1991, 132–133). Another essential aspect in the performance of an autoassociative network is the degree of its content-addressability, i.e., how distorted initial patterns it can tolerate. This ability depends on the sizes of basins of attraction of the stored patterns, and can be improved, for instance, by training the network with noisy patterns (Forrest & Wallace 1991, 140).

The aforementioned analysis results indicate that the present artificial neural network model, while being able to reproduce the tonal characteristics of the input material on a local level, fails in extracting the global tonality. The reason for this may be the limited temporal range of operation, as well as the lack of hierarchical structure in the model. In or-

der to be able to process events of longer temporal range, a short-term memory should be added to the model. In the connectionist paradigm this can be done, for instance, by means of tapped delay lines or leaky integrators. The contents of the short-term memory would then enable the model to better connect the output to the global tonality.

The ability of a neural network to extract global features from a given input depends to some extent on its architecture: in general, multi-layer networks seem to perform this task better than single-layer ones. Hence, a natural improvement to the present network model would be to add layers operating on greater time scales; as a first step, a layer operating with target notes within a melodic phrase of four bars could be added. A further possibility would be a hybrid model: the connectionist system could be connected with a rule-based symbolic system operating on a higher level of abstraction.

An essential aspect which affects the performance of a neural network is how the data is represented. According to a general view, making data representation more distributed seems to better exploit connectionist systems' important inherent properties, such as content-addressability, generalization capability, and noise tolerance. In the present model, there are several ways to improve the representation of data. For example, the representation of chords is localist: a given chord type is represented by activating one neuron. This could be made more distributed by using activation patterns representing the tones the chord is composed of.

The statistical methods themselves also require some additions, because the ones used in this study give only atemporal evidence about the music. In other words, they are unable to give information about how the music unfolds in time; how and to what extent, for example, motives influence the tone profiles must be determined subjectively by analyzing the music. While this is mandatory at any case, there still should be some more objective way to examine the temporal surface level events. One way to do this would be to use an n-dimensional transition matrix which would enable us to examine the statistical properties of a given piece of music on a note-to-note level. A two dimensional matrix, for example, would give us to the relative percentages by which a given tone is preceded by the other tones in the chromatic scale. Further, if a third dimension were added, we would get the transitions between any three note groups. These would give a plethora of information about the voice-leading, intervals, and small motives. On the other hand, this would require a very large body of materials in order to obtain a transition matrix with sufficient density.

REFERENCES

Baggi, D. L. 1992. NeurSwing: an intelligent workbench for the investigation of swing in jazz. In Baggi, D. L. (Ed.): *Readings in computer generated music.* Los Alamitos, CA: IEEE Computer Society Press.

Berg, S. 1990.*Jazz Improvisation: The Goal Note Method.* Evergreen: Lou Fischer Music Publishing.

Bharucha, J. J. 1984. Anchoring effects in music: the resolution of dissonance. *Cognitive Psychology,* 16, 485-518.

Bharucha, J. J. 1988. Neural net modeling of music. *Proceedings of the first workshop on AI and music.* AAAI-88. Minneapolis/St. Paul.

Bharucha, J. J. & Todd, P. M. 1989. Modeling the perception of tonal structure with neural nets. *Computer Music Journal,* 13(4), 44–53.

Campbell, G. 1989. *Hank Mobley — Transcribed Solos.* Houston: Houston Publishing Inc.

Castellano, M. A., Bharucha, J. J. & Krumhansl, C. L. 1984. Tonal hierarchies in the music of North India. *Journal of Experimental Psychology: General,* 113, 394–412.

Desain, P. & Honing, H. 1989. The quantization of musical time: a connectionist approach. *Computer Music Journal,* 13(3), 56–66.

Forrest, B. M. & Wallace, D. J. 1991. Storage capacity and learning in Ising-spin networks. In Domany, E., van Hemmen, J. L., & Schulten, K. (Eds.): *Models of neural networks.* Berlin: Springer-Verlag.

Galper, H. 1982. Forward motion. *Down Beat* 1/82.

Gjerdingen, R. O. 1989. Using connectionist models to explore complex musical patterns. *Computer music journal,* 13(3), 67–75.

Gjerdingen, R. O. 1990. Categorization of musical patterns by self-organizing neuronlike networks. *Music Perception*, 7(4), 339–370.

Gjerdingen, R. O. 1992. Learning syntactically significant temporal patterns of chords: a masking field embedded in an ART 3 architecture. *Neural Networks*, 5, 551-564.

Harnett, D. L. 1982. *Statistical methods*. Third edition. Reading, MA: Addison-Wesley.

Hecht-Nielsen, R. 1990. *Neurocomputing*. Reading, MA: Addison-Wesley.

Järvinen, T. 1995. Tonal hierarchies in jazz improvisation. *Music Perception* 12(4), 415-437.

Kohonen, T. 1989. *Self-organization and associative memory*. Third edition. Berlin: Springer-Verlag.

Knopoff, L. & Hutchinson, W. 1983. Entropy as a measure of style: The Influence of Sample Lenght. *Journal of Music Theory*, 27, 75-97.

Krumhansl, C. L. & Shepard, R. N. 1979. Quantification of the hierarchy of tonal functions within a diatonic context. *Journal of Experimental Psychology: Human Perception and Performance*, 5(4), 579-594.

Krumhansl, C. L. & Kessler, E. J. 1982. Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological Review*, 89(4), 334-368.

Krumhansl, C. L. 1990.*Cognitive foundations of musical pitch*. New York: Oxford University Press.

Leman, M. 1988. Sequential (musical) information processing with PDP-networks. *Proceedings of the first workshop on AI and music*. AAAI-88. Minneapolis/St. Paul.

Leman, M. 1989. *Artificial neural networks in music research*. Reports from the seminar of musicology — Institute for psychoacoustics and electronic music. University of Ghent.

Leman, M. 1992. Tone context by pattern integration over time. In Baggi, D. L. (Ed.): *Readings in computer generated music*. Los Alamitos, CA: IEEE Computer Society Press.

Mehegan, J. 1959. *Jazz improvisation I: Tonal and rhythmic principles*. New York: Watson-Guptill.

Meyer, L. B. 1956. *Emotion and Meaning in Music*. Chicago: The University of Chicago Press.

Mobley, H. et al. 1956. Tenor Conclave. On *Tenor Conclave*, Prestige OJCCD-127-2 (P-7074), September 7.

Palmer, C. & Krumhansl, C. L. 1990. Mental Representation for musical meter. *Journal of Experimental Psychology: Human Perception and Performance*, 16 (4), 728-741.

Rumelhart, D. E. & McClelland, J. L. 1986. PDP models and general issues in cognitive science. In D. E. Rumelhart & J. L. McClelland (Eds.): Parallel Distributed Processing: Explorations in the microstructure of cognition. Cambridge, MA: The MIT Press.

Serafine, M. L., Glassman, N. & Overbeeke, C. 1989. The Cognitive reality of hierarchic structure in music. *Music Perception*, 6 (4), 397-430.

Todd, P. M. 1989. A connectionistic approach to algorithmic composition. *Computer Music Journal*, 13(4), 27–43.

Toiviainen, P. 1995. Modeling the target-note technique of bebop-style jazz improvisations: an artificial neural network approach. *Music Perception* 12(4), 399-413.

van Hemmen, J. L. & Kühn, R. 1991. Collective phenomena in neural networks. In Domany, E., van Hemmen, J. L. & Schulten, K. (Eds.): *Models of neural networks,* Berlin: Springer-Verlag.

APPENDIX 1: Why is Pearson's correlation coefficient not an adequate similarity measure for tone profiles?

Let $\mathbf{x} = (x_1, x_2, ..., x_N)$ denote a tone profile. Let us construct tone profile $\mathbf{y} = (y_1, y_2, ..., y_N)$ by using the equation

$$y_i = x_i + \alpha(x_i - \bar{x}) \,, \tag{3}$$

where $\bar{x}$, the mean of x_i, is by definition

$$\bar{x} \equiv \frac{1}{N} \sum_i x_i \,. \tag{4}$$

It can be seen from (3) that when $\alpha = 0$, the two tone profiles are identical. When $\alpha > 0$, profile **y** is constructed from profile **x** by increasing those x_i which are greater than their mean, and decreasing those which are smaller than their mean; the opposite holds true for $\alpha < 0$. Further, it is obvious that the smaller the absolute value of α, the closer profile **y** is to profile **x**. In Figure 21, **y** is constructed from **x** by using the value $\alpha = 1$.

By combining (3) and (4) we get

$$\bar{y} = \frac{1}{N} \sum_i (x_i + \alpha(x_i - \bar{x})) = \bar{x} + \alpha(\bar{x} - \bar{x}) = \bar{x} \,, \tag{5}$$

i.e., the means of the two profiles are equal. By using (3) and (5) we further get

$$y_i - \bar{y} = x_i + \alpha(x_i - \bar{x}) - \bar{x} = (1+\alpha)(x_i - \bar{x}). \tag{6}$$

The Pearson's correlation coefficient for **x** and **y** is by definition

$$corr(\mathbf{x},\mathbf{y}) \equiv \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\left(\sum (x_i - \bar{x})^2\right)^{1/2}\left(\sum (y_i - \bar{y})^2\right)^{1/2}}. \tag{7}$$

Combining (7) and (6) yields

$$corr(\mathbf{x},\mathbf{y}) = \frac{(1+\alpha)\sum (x_i - \bar{x})^2}{|1+\alpha|\left(\sum (x_i - \bar{x})^2\right)^{1/2}\left(\sum (x_i - \bar{x})^2\right)^{1/2}}$$

$$= \frac{(1+\alpha)}{|1+\alpha|} = \begin{cases} 1 & ,if\ \alpha > -1 \\ -1 & ,if\ \alpha < -1 \end{cases}. \tag{8}$$

Pearson's correlation coefficient, thus, does not depend on the magnitude of α (except the step at $\alpha = -1$). Consequently, it does not provide an adequate similarity measure for tone profiles. See also Figure 21 and Table 1.

TABLE 1. Data for profiles presented in Figure 21.

tone	profile **x**	profile **y**	profile **z**
C	6.35	10.65125	6.9
C♯/D♭	2.23	0.35125	2.68
D	3.48	3.47625	3.83
D♯/E♭	2.33	0.60125	2.58
E	4.38	5.72625	4.53
F	4.09	5.00125	4.14
F♯/G♭	2.52	1.07625	2.47
G	5.19	7.75125	5.04
G♯/A♭	2.39	0.75125	2.14
A	3.66	3.92625	3.31
A♯/B♭	2.29	0.50125	1.84
B	2.88	1.97625	2.33

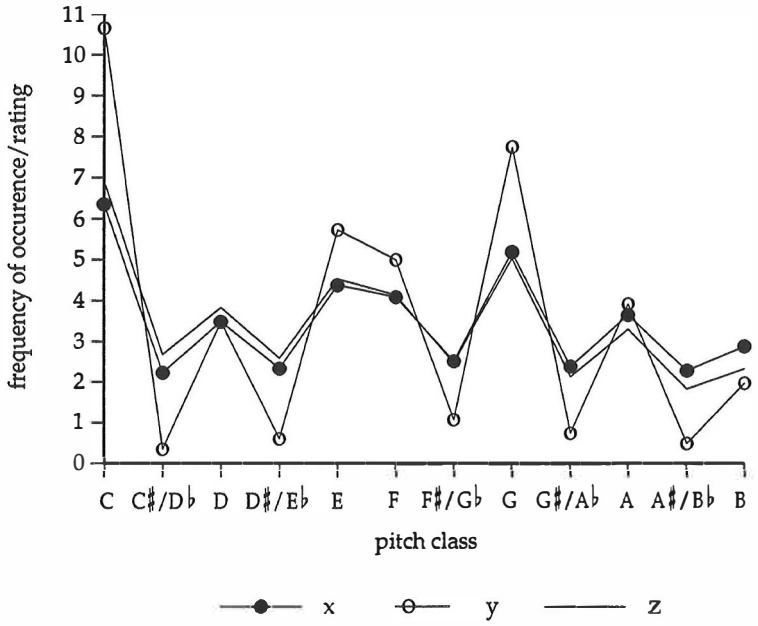


FIGURE 21. Graphical representation of the tone profiles in Table 1. Profile **x** is the C major key profile after Krumhansl (1990, 37, Table 2.3.). The values of Pearson's correlation coefficient for the pairs (**x**,**y**) and (**x**,**z**) are 1 and 0.973, respectively. These values clearly conflict with the perceived degrees of similarity between the profiles. On the other hand, the values of the direction cosine (see Appendix 2) are consistent with that: cos(**x**,**y**) = 0.925, cos(**x**,**z**) = 0.996.

APPENDIX 2: On distance and similarity measures of vectors

Let $\mathbf{p}$ and $\mathbf{q}$ denote two n-dimensional real-valued vectors:

$$\mathbf{p} \equiv (\xi_1,\ldots,\xi_n),\ \mathbf{q} \equiv (\eta_1,\ldots,\eta_n). \tag{9}$$

The magnitude, or norm, of $\mathbf{p}$ can be defined in several ways. The two most commonly used are the Euclidean norm, $\|\mathbf{p}\|_2$, and the city-block norm, $\|\mathbf{p}\|_1$, defined as

$$\|\mathbf{p}\|_2 \equiv \left[\sum_{i=1}^{n} \xi_i^2\right]^{1/2} \tag{10}$$

and

$$\|\mathbf{p}\|_1 \equiv \sum_{i=1} |\xi_i|. \tag{11}$$

Both are special cases of the Minkowski norm

$$\|\mathbf{p}\|_\mu \equiv \left[\sum_{i=1}^{n} |\xi_i|^\mu\right]^{1/\mu}. \tag{12}$$

The distance of vectors $\mathbf{p}$ and $\mathbf{q}$ is defined as the norm of their difference:

$$d(\mathbf{p},\mathbf{q}) \equiv \|\mathbf{p}-\mathbf{q}\|. \tag{13}$$

Combining definition (13) with definitions (10) and (11), we get, respectively, two frequently used distance measures for vectors: the Euclidean distance

$$d_2(\mathbf{p},\mathbf{q}) \equiv \|\mathbf{p}-\mathbf{q}\|_2 \equiv \left[\sum_{i=1}^{n}(\xi_i-\eta_i)^2\right]^{1/2} \tag{14}$$

and the city-block distance

$$d_1(\mathbf{p},\mathbf{q}) \equiv \|\mathbf{p}-\mathbf{q}\|_1 \equiv \sum_{i=1}^{n}|\xi_i-\eta_i|. \tag{15}$$

In addition, the similarity of vectors $\mathbf{p}$ and $\mathbf{q}$ can be measured in terms of the cosine of their mutual angle, or direction cosine, defined as

$$\cos\Theta \equiv \frac{(\mathbf{p},\mathbf{q})}{\|\mathbf{p}\|_2\|\mathbf{q}\|_2}, \tag{16}$$

where the scalar product $(\mathbf{p},\mathbf{q})$ has the form

$$(\mathbf{p},\mathbf{q}) \equiv \sum_{i=1}^{n}\xi_i\eta_i. \tag{17}$$

The value $\cos\Theta = 1$ represents an exact match: vector $\mathbf{p}$ is then equal to vector $\mathbf{q}$ multiplied by a scalar α, i.e., $\mathbf{p} = \alpha\mathbf{q}$.

VI

A Self-Organizing Map That Recognizes and Generates Melodies

by Mauri Kaipainen, Petri Toiviainen & Jukka Louhivuori

APPENDIX 1

C source code of the Kohonen map simulator used in studies II and III

```
/*
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
                        APPENDIX 1:
                        C source code for the
                        Kohonen Map Simulator
                        used in studies II and III
                        (contains Macintosh-specific functions)
                        Petri Toiviainen 1995
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
*/

/*••••••••••••••••••File "Kohos.h"••••••••••••••••••••••••*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdarg.h>
#include <string.h>
#include <console.h>

/*          MACRO DEFINITIONS              */

#define BEEP printf("\7");
#define getParam(x,y,z) {printf(y);printf(": ");scanf(z, &x);}
#define mError {printf("Out of memory.\n"); exit(0);}
#define foError(x) {printf("Can't open file %s.\n", x); return;}
#define noNetAlert {printf("Build a net first.\n"); return;}
#define noPattAlert {printf("Read patterns first.\n"); return;}

/*          FUNCTION DECLARATIONS          */

void        sayHello(),
            buildNet(void),
            readPatterns(void),
            goTesting(void),
            saveNet(void),
            readNet(void),
            wayOut(void),
            whatNext(int numFuncs, ...),
            goTraining(void),
            train(void),
            freeWeights(),
            allocateWeights(),
            randomizeWeights();

extern void train(void),
            test(void);

/*••••••••••••••••••File "Kohos.c"••••••••••••••••••••••••*/

#include "Kohos.h"

char  *dots = "••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••";

int   inputDim,             /* dimension of input vectors */
      xSize,                /* horizontal dimension of the map */
      ySize,                /* vertical dimension of the map */
      torus = 0,            /* =1, if toroidal architecture */
      noNet = 1,            /* =1, if no network has been read or created */
```

```
        noPatterns = 1;             /* =1, if no patterns have been read */

float ***weight,                    /* pointer to synaptic vectors */
        **tmpValue,
        **iPatt,                    /* pointer to input vectors */
        minkExp;                    /* Minkowski exponent */

int     nPatts;                     /* number of input vectors */

FILE *ifp, *ofp;                    /* input/output file pointers */
char ifName[128],                   /*      file
        ofName[128];                        names */

/*****************************************************/

void  main() {

        sayHello();
        while(1)                    /* choose next action */
                whatNext(7,
                        "Build network", buildNet,
                        "Read network from file", readNet,
                        "Read input patterns", readPatterns,
                        "Train", goTraining,
                        "Test", goTesting,
                        "Save network to file", saveNet,
                        "Exit", wayOut);
}

/*****************************************************/

void  whatNext(int numFuncs, ...) {
        #define MAXNFUNCS 10

        char        funcName[MAXNFUNCS][64],
                    *tmp;
        void        *funcPtr[MAXNFUNCS];
        void        (*theFunc)(void);
        int         i,
                    funcNum;
        va_list     ap;

/*              get the arguments               */

        va_start(ap, numFuncs);
        for (i=0; i<numFuncs; i++) {
                tmp = va_arg(ap, char*); strcpy(funcName[i], tmp);
                funcPtr[i] = va_arg(ap, void*);
        }
        va_end(ap);

/*              choose the next action          */

        printf("\n%s\n%s\n", dots, "Choose the next action:");
        for (i=0; i<numFuncs; i++)
                printf("%d) %s\n", i+1, funcName[i]);
        printf("%s\n\n", dots);

        scanf("%d", &funcNum);
```

```
        theFunc = funcPtr[funcNum-1];

/*              call the chosen function          */

        (*theFunc)();

}

/*****************************************************/

void  buildNet() {
        register int  i, j;

        freeWeights();
        getParam(inputDim, "inputDim", "%d");
        getParam(xSize, "xSize", "%d");
        getParam(ySize, "ySize", "%d");
        getParam(torus, "torus (1/0)", "%d");
        getParam(minkExp, "Minkowski exponent", "%f");

        allocateWeights();
        randomizeWeights();
        noNet= 0;
}

/*****************************************************/

void  readPatterns() {
        register int  i, j, eof;
        float         tmp;

        if(noNet) noNetAlert;

        getParam(ifName, "Pattern file name", "%s");
        if((ifp=fopen(ifName, "r")) == NULL) foError(ifName);

/*              free pattern pointers                   */

        if(iPatt != NULL) {
                for(i=0;i<nPatts;i++)  free(iPatt[i]);
                free(iPatt);
        }

/*              count the number of patterns            */

        nPatts = 0; eof = 0;
        while(1) {
                for(i=0; i<inputDim; i++)
                        if(fscanf(ifp, "%f", &tmp) == EOF) eof = 1;
                if(eof) break;
                nPatts++;
        }
        fclose(ifp);

/*              allocate memory for the patterns        */

        if((iPatt=(float**)calloc(nPatts, sizeof(float*)))==NULL)  mError;
        for(i=0;i<nPatts;i++)
```

```
            if((iPatt[i]=(float*)calloc(inputDim, sizeof(float)))==NULL)
            mError;

/*          read the patterns                                   */

      if((ifp=fopen(ifName, "r")) == NULL) foError(ifName);
      for(i=0; i<nPatts; i++) for(j=0; j<inputDim; j++)
            fscanf(ifp, "%f", &iPatt[i][j]);
      fclose(ifp);
      printf("%d patterns read from file %s.\n", nPatts, ifName);
      noPatterns = 0;

}

/*****************************************************/

void  saveNet() {
      register int  i,j,k;

      if(noNet) noNetAlert;
      getParam(ofName, "Output file name", "%s");
      if((ofp=fopen(ofName, "w")) == NULL) foError(ofName);
      printf("Writing to file %s.\n", ofName);
      fprintf(ofp, "%d\t%d\t%d\n", xSize, ySize, inputDim);
      for(i=0; i<xSize; i++)
            for(j=0; j<ySize; j++) {
                  for(k=0; k<inputDim; k++)
                        fprintf(ofp, "%.3f\t", weight[i][j][k]);
                  fprintf(ofp, "\n");
            }
      fclose(ofp);
}

/*****************************************************/

void  readNet() {
      register int  i, j, k;

      getParam(ifName, "Network file name", "%s");
      if((ifp=fopen(ifName, "r")) == NULL) foError(ifName);
      printf("Reading from file %s.\n", ifName);
      fscanf(ifp, "%d%d%d", &xSize, &ySize, &inputDim);
      printf("xSize: %d\nySize: %d\ninputDim: %d\n", xSize, ySize,
      inputDim);
      printf("Allocating ...\n"); allocateWeights();
      for(i=0; i<xSize; i++)
            for(j=0; j<ySize; j++)
                  for(k=0; k<inputDim; k++)
                        fscanf(ifp, "%f", &weight[i][j][k]);
      fclose(ifp);
      noNet = 0;
}

/*****************************************************/

void  freeWeights() {
      int           i, j;

      if(weight!=NULL) {
```

```
            for(i=0;i<xSize;i++)
                  for(j=0;j<ySize;j++)
                        free(weight[i][j]);
            for(i=0;i<xSize;i++)
                  free(weight[i]);
            free(weight);
      }
}

/*****************************************************/

void  allocateWeights() {
      int         i, j;

      if((weight=(float***)calloc(xSize,  sizeof(float**)))==NULL)
            mError;
      for(i=0;i<xSize;i++)
            if((weight[i]=(float**)calloc(ySize,  sizeof(float*)))==NULL)
                  mError;
      for(i=0;i<xSize;i++)  for(j=0;j<ySize;j++)
            if((weight[i][j]=(float*)calloc(inputDim,
      sizeof(float)))==NULL)
                        mError;

      if((tmpValue=(float**)calloc(xSize,  sizeof(float*)))==NULL)
            mError;
      for(i=0;i<xSize;i++)
            if((tmpValue[i]=(float*)calloc(ySize,  sizeof(float)))==NULL)
                  mError;


}

/*****************************************************/

void  randomizeWeights() {
      register int  i,j,k;
      srand((unsigned int) clock());

      for(i=0;i<xSize;i++)
            for(j=0;j<ySize;j++)
                  for(k=0; k<inputDim; k++)
                  weight[i][j][k] = (float) rand()/(float) RAND_MAX;
}

/*****************************************************/

void  goTraining() {
      if(noNet) noNetAlert;
      if(noPatterns) noPattAlert;
      train();
}

/*****************************************************/

void  goTesting() {
      if(noNet) noNetAlert;
      if(noPatterns) noPattAlert;
      test();
```

```
}

/*••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••*/

void  wayOut() {
      BEEP;
      exit(1);
}

/*••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••*/

void  sayHello() {
      console_options.title = "\pMinkowski-Kohonen simulator";
      console_options.pause_atexit = 0;

      printf("%s\n%s\n%s\n%s\n%s\n%s\n",
"••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••",
"            Kohonen Map Simulator",
"            Petri Toiviainen 1995",
"             internet: ptoiviai@jyu.fi",
"••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••");

}

/*••••••••••••••••••File "KohosTrain.h"••••••••••••••••••••••••*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#define MAXPHASES 5          /* maximum number of training phases */

/*                  MACRO DEFINITIONS                    */

#define getParam(x,y,z) {printf(y);printf(": ");scanf(z, &x);}
#define mError {printf("Out of memory.\n"); exit(0);}
#define foError(x) {printf("Can't open file %s.\n", x); return;}
#define min3(x,y,z) ((x)<(y) ? (x)<(z) ? (x) : (z) : (y))

/* square of topological distance between two neurons */
#define sqrOfTopDist(x1,y1,x2,y2) torus ? \
      (tmp = min3(abs(x1-x2),abs(x1+xSize-x2),abs(x1-xSize-x2))) * tmp + \
      (tmp = min3(abs(y1-y2),abs(y1+ySize-y2),abs(y1-ySize-y2))) * tmp : \
      abs(x1-x2)*abs(x1-x2)+abs(y1-y2)*abs(y1-y2)

/* neighbourhood function */
#define mexicanHat(r2,R2) exp(-(r2)/(R2))

/* adjust vector v1 towards vector v2 */
#define adjust(v1,v2,a) {for(i=0;i<inputDim;i++) v1[i]+=a*(v2[i]-v1[i]);}

/*                  EXTERNAL VARIABLES                   */
extern float      **iPatt,
                  ***weight,
                  **tmpValue,
                  minkExp;
extern int        xSize,
                  ySize,
```

```
                inputDim,
                torus,
                nPatts;
extern char     *dots;

/*              FUNCTION DECLARATIONS                   */
int         checkInterrupt(void);
void        train(void),
            getTrainParams(void),
            learnPattern(void),
            prnErrors(void);
float       minkDist(float*, float*);

/*••••••••••••••••••File "KohosTrain.c"••••••••••••••••••••••••*/

#include "KohosTrain.h"

int         phase,          /*  counter     */
            nPhases,        /*  number of training phases    */
            ePeriod = 100,  /*  number of training cycles between error reports */
            goOut = 0,      /*  flag  */
            tmp;

long        cycle,                  /*  counter     */
            nCycles[MAXPHASES],     /*  number of training cycles in each phase */
            totCycles = 0;

float       startLR[MAXPHASES],     /*  learning rates at the beginning ...     */
            endLR[MAXPHASES],       /*  ... and at the end of each phase        */
            startRadius[MAXPHASES],     /*  same for ...                */
            endRadius[MAXPHASES],       /*  ... neighbourhood radii     */
            LR,                         /*  current learning rate ...   */
            radius,                     /*  ... and neighbourhood radii */
            varLR, varRadius;           /*  flags  */

float       errorIntegrator = 0.0,      /*  leaky integrator for computing a moving
                                            average of the errors between input and
                                            synaptic vectors    */
            smoothnessIntegrator = 0.0, /*  leaky integrator for computing a moving
                                            average of the local smoothness of the
                                            map    */
            decay;                      /*  decay parameter of the integrators */

FILE        *errfp;

EventRecord     theEvent;           /*  Apple OS stuff...       */
clock_t     clock1;                 /*  ... for calculating ...     */
time_t      now;                    /*  ... the time elapsed ...    */
struct tm   *tmNow;                 /*  ... in the training         */

void  train() {

    getTrainParams();
    FlushEvents(everyEvent, 0);
    clock1 = clock(); totCycles = 0;
    for(phase=0; phase<nPhases; phase++) {
        LR = startLR[phase]; radius = startRadius[phase];
        varLR = startLR[phase] == endLR[phase] ? 0 : 1;
        varRadius = startRadius[phase] == endRadius[phase] ? 0 : 1;
```

```
            for(cycle=0; cycle<nCycles[phase]; cycle++) {
                if(varLR) LR = startLR[phase]+((float)cycle/nCycles[phase])*
                    (endLR[phase]-startLR[phase]);
                if(varRadius) radius =
                    startRadius[phase] + ((float)cycle/nCycles[phase])*
                    (endRadius[phase]-startRadius[phase]);
                learnPattern();

/*          check if it is time to report the errors          */
                if((++totCycles % ePeriod) == 0) prnErrors();
                if(checkInterrupt())
                    {goOut = 1; break;}
            }
            if(goOut) break;
        }
        fclose(errfp);
}

/*********************************************************/

void  getTrainParams() {      /* get training parameters */
        int         i;
        char  errfName[128];

        getParam(nPhases, "Number of phases", "%d");
        nPhases = nPhases<=MAXPHASES ? nPhases : MAXPHASES;
        printf("%s\n", dots);
        for(i=0; i<nPhases; i++) {
            printf("Phase %d:\n", i+1);
            getParam(startLR[i], "Learning rate at start", "%f");
            getParam(endLR[i], "Learning rate at end", "%f");
            getParam(startRadius[i], "Radius at start", "%f");
            getParam(endRadius[i], "Radius at end", "%f");
            getParam(nCycles[i], "Number of cycles", "%ld");
            printf("%s\n", dots);
        }
        getParam(ePeriod, "Error functionals output interval", "%d");
        decay = exp(-1.0/ePeriod);
        now = time(NULL); tmNow = localtime(&now);
        sprintf(errfName, "Errfuncs%2d%2d%2d%2d",
            tmNow->tm_mon,tmNow->tm_mday,tmNow->tm_hour,tmNow->tm_min);
        if((errfp=fopen(errfName,"w")) == NULL)
            foError(errfName);

}
/*********************************************************/

/*  this function return the value 1 if a key has been pressed with the Command key */
int         checkInterrupt() {
        int         ok;

        ok = GetNextEvent(everyEvent, &theEvent);
        if(ok)
            switch(theEvent.what) {
                case keyDown:
                case autoKey:
                if ((theEvent.modifiers & cmdKey) != 0)
                    return 1;
                    break;
```

```
        }
        return 0;
}

/*●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●*/

void learnPattern() {
        register int        x,y,i;
        register float      tmp,
                            minErr = 1.0e+12,
                            sOTD,               /* square of topological distance */
                            adjustAmount;       /* amount of synaptic vector to be adjusted */
        int                 pattNum = rand()%nPatts,    /* number of current input vector */
                            xFocus,     /*    coordinates of the ...    */
                            yFocus,     /*    ... focus of response     */
                            cnt;

        for(x=0; x<xSize; x++) for(y=0; y<ySize; y++)
                if((tmpValue[x][y] = minkDist(weight[x][y], &iPatt[pattNum][0])) <minErr) {
                        xFocus = x; yFocus = y; minErr = tmpValue[x][y];
                }
        for(x=0; x<xSize; x++) for(y=0; y<ySize; y++)
                if((sOTD = sqrOfTopDist(x,y,xFocus,yFocus)) < 4.0*radius*radius) {
                        adjustAmount = LR * mexicanHat(sOTD, (radius*radius));
                        adjust(weight[x][y], iPatt[pattNum], adjustAmount);
                }
/*              integrate error                 */
        errorIntegrator *= decay; errorIntegrator += (1.0-decay)*minErr;
/*              integrate smootness                 */
        smoothnessIntegrator *= decay; tmp = 0.0; cnt=0;
        for(x=xFocus-1; x<=xFocus+1; x++) for(y=yFocus-1; y<=yFocus+1; y++) {
                if(torus) {
                        tmp += tmpValue[(x+xSize)%xSize][(y+ySize)%ySize];
                        cnt++;
                }
                else {
                        if(x>=0 && x<xSize && y>=0 && y<ySize) {
                                tmp += tmpValue[x][y];
                                cnt++;
                        }
                }
        }
        smoothnessIntegrator += (1.0-decay)*tmp/cnt;
}

/*●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●*/

/* print the mean error and smoothness values to a file */

void prnErrors() {
        fprintf(errfp,"%ld\t%f\t%f\n", totCycles, errorIntegrator, smoothnessIntegrator);
        now = time(NULL);
        printf("%ld\t%f\t%f\t%.3lf\t%s", totCycles, errorIntegrator,
                smoothnessIntegrator,
                (float)(clock()-clock1)/(ePeriod*(float)CLOCKS_PER_SEC), ctime(&now));
        clock1 = clock();
}

/*●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●*/
```

```
/*    this function returns the minkowski distance between vectors v1 and v2    */

float minkDist(float *v1, float*v2) {
      register float      sum = 0.0;
      register int  i;

      if(minkExp == 1) {
            for (i=0; i<inputDim; i++)
                  sum += fabs(v1[i] - v2[i]);
            return sum;
      }
      else if(minkExp == 2) {
            for (i=0; i<inputDim; i++)
                  sum += (v1[i] - v2[i]) * (v1[i] - v2[i]);
            return sqrt(sum);
      }
      else {
            for (i=0; i<inputDim; i++)
                  sum += pow(fabs(v1[i] - v2[i]), minkExp);
            return pow(sum, 1.0/minkExp);
      }
}


/*••••••••••••••••••File "KohosTest.h"••••••••••••••••••••••••*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define getParam(x,y,z) {printf(y);printf(": ");scanf(z, &x);}

#define foError(x) {printf("Can't open file %s.\n", x); return;}

/*                EXTERNAL VARIABLES                    */

extern float      ***weight,
                  **tmpValue,
                  **iPatt;
extern int        xSize,
                  ySize,
                  inputDim,
                  torus,
                  nPatts;
extern float      minkExp;

extern float      minkDist(float*, float*);

/*••••••••••••••••••File "KohosTest.c"••••••••••••••••••••••••*/

#include "KohosTest.h"

FILE *rffp,       /* pointer to the file of response foci */
     *cafp;       /* pointer to the file of centers of activation */
char prefix[128],
     rffName[128],
```

```
        cafName[128];
int     xFocus, yFocus;     /*    coordinates of the focus of response          */
float   minDist,
        act,
        COAX, COAY,     /*    coordinates of the center of activation       */
        COA_NominX, COA_NominY, COA_Denomin;

void    test() {
        register int  x, y;
        int           pattNum,
                      xFocus, yFocus;
        float         tmp,
                      min = 1.0e+12;

        getParam(prefix, "Prefix of output file names", "%s");
        strcpy(rffName, prefix);strcat(rffName, ".respFoci");
        if((rffp=fopen(rffName, "w")) == NULL) foError(rffName);
        strcpy(cafName, prefix);strcat(cafName, ".centAct");
        if((cafp=fopen(cafName, "w")) == NULL) foError(cafName);
        for(pattNum=0; pattNum<nPatts; pattNum++) {
                minDist = 1.0e+12;
                COA_NominX = 0.0; COA_NominY = 0.0; COA_Denomin = 0.0;
                for(x=0; x<xSize; x++) for(y=0; y<ySize; y++) {
                        if((tmpValue[x][y] = minkDist(iPatt[pattNum], weight[x][y]))
                                < minDist) {
                                        minDist = tmpValue[x][y];
                                        xFocus = x; yFocus = y;
                        }
                        act = tmpValue[x][y] == 0.0 ? 1.0 :
                                1.0 - tmpValue[x][y] /
                                        (minkDist(iPatt[pattNum],iPatt[pattNum])
                                        + minkDist(weight[x][y],weight[x][y]));
                        act = act>0.0 ? act: 0.0;
                        COA_NominX += x * act;
                        COA_NominY += y * act;
                        COA_Denomin +=act;
                }
                fprintf(rffp, "%d\t%d\n", xFocus, yFocus);
                COAX = COA_NominX / COA_Denomin;
                COAY = COA_NominY / COA_Denomin;
                fprintf(cafp, "%f\t%f\n", COAX, COAY);
                printf("%d\t%d\t%f\t%f\t%f\n", xFocus, yFocus, COAX, COAY, minDist);
        }
        fclose(rffp); fclose(cafp);


/*•••••••••••••••••• THE END ••••••••••••••••••••••••*/
```

APPENDIX 2

C source code of the jazz improvisation model used in studies IV and V

```
/*
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
                        APPENDIX 2:
                        C source code for the
                        jazz improvisation model
                        used in studies IV and V
                        Petri Toiviainen 1995
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
*/

/*••••••••••••••••••• File "StructDefs.h" ••••••••••••••••••••••*/

#define NSLICES 8
#define NCHORDTYPES 9
#define NSUBSLICES 6
#define NTONES 14 /* 12 tones + sustain + rest */
#define DECAY 0

typedef struct neurontype NEURON;
struct neurontype {
        double activation;
        double new_activation;
        double bias;
};

typedef struct ctype CHORDtoMELODY;   /*     connection matrix between ...
                                              ... chord and melody neurons */
struct ctype {
        int mel[NSLICES][NCHORDTYPES][NSUBSLICES][NTONES];
};

typedef struct m2type MtoM;
struct m2type {
        int mel[NSUBSLICES][NSUBSLICES][NTONES][NTONES];
};

typedef struct mtype MELODYtoMELODY;        /* connection matrix between melody
neurons */
struct mtype {
        MtoM* me[NSLICES];
};

/*••••••••••••••••••• File "Externals.h" ••••••••••••••••••••••*/

extern      MELODYtoMELODY    *m;
extern      CHORDtoMELODY     *c;
extern      NEURON melody[NSUBSLICES][NTONES];
extern      int chord[100],note[NSUBSLICES], output_to_file, output_to_MF;
extern      double CtoMscale,MtoMscale,inhibition;
extern      FILE *log_file, *MIDI_file;
```

```
/*•••••••••••••••••• File "ActFunc.c" ••••••••••••••••••••••*/

double actfunc(double x) {
        if (x<0) return 0;
        else if (x<1) return x;
        else return 1;
}

/*•••••••••••••••••• File "Conversions.c" ••••••••••••••••••••••*/

#include "StructDefs.h"

int chordtype(int chordnumber) {
        return chordnumber/12;
}

int chordroot(int chordnumber) {
        return chordnumber-12*chordtype(chordnumber);
}

/*•••••••••••••••••• File "Main.c" ••••••••••••••••••••••*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "StructDefs.h"

MELODYtoMELODY     *m;
CHORDtoMELODY      *c;
NEURON melody[NSUBSLICES][NTONES];
int chord[100],note[NSUBSLICES];
double CtoMscale, MtoMscale, inhibition;

extern int errno;

choose_procedure(i)
        int i; {
        switch(i) {
                        case 1: learn(); break;
                        case 2: test(); break;
                        case 3: exit;
        }
        return;
}

main() {
        #define n_menuitems 3
        int i,menuitem;
        static char *menu[] = {   "Learn      = 1",
                                  "Test       = 2",
                                  "Exit       = 3" };
```

```
        if((m = calloc(1,sizeof(MELODYtoMELODY))) == NULL) {
                printf("Out of memory!\n");
                exit;
        }
        if((c = calloc(1,sizeof(CHORDtoMELODY))) == NULL) {
                printf("Out of memory!\n");
                exit;
        }
        for (i=0; i<NSLICES; i++) {
                if((m->me[i] = calloc(1,sizeof(MtoM))) == NULL) {
                        printf("Out of memory!\n");
                        exit;
                }
        }
        do {
                printf("*******************************************************\n");
                for(i = 0; i < n_menuitems; i++) printf("%s\n",menu[i]);
                if(scanf("%d",&menuitem) == 1)
                        if ((menuitem > 0) && ( menuitem <= n_menuitems))
                                choose_procedure(menuitem);
        } while (menuitem != n_menuitems);
}

/*••••••••••••••••• File "Learn.c" ••••••••••••••••••••••*/

#include <stdio.h>
#include <stdlib.h>
#include "StructDefs.h"
#include "Externals.h"
#define MAXSLICES 65

learn() {
        int     answer,
                i,j,k,
                chordnum,
                nslices,
                notenum[NSUBSLICES],
                interval[NSUBSLICES],
                chord[MAXSLICES],
                note[MAXSLICES][NSUBSLICES],
                slice;
        char *fname = "************************";
        FILE *fp;
        extern int      chordtype(),chordroot(),errno;

        do {
                printf("Read matrices? (yes=1, no=0)\n");
        }       while (scanf("%d",&answer) != 1);
        if(answer) read_matrices();
        do {
```

```
            printf("Reading example - give filename! (* to stop)\n");
            scanf("%s",fname);
            if (*fname == '*') break;
            if((fp = fopen(fname,"r")) == NULL) {
                  printf("Error in opening file, errno %d\n",errno);
                  return;
            }
            fscanf(fp,"%d",&nslices);
            note[0][0]= -1;
            note[nslices-1][NSUBSLICES-1] = -1;
            for (i=0; i<nslices; i++) {
                  fscanf(fp,"%d",&chord[i]);
                  for (j=1; j<NSUBSLICES-1; j++)
                        fscanf(fp,"%d",&note[i][j]);
            }

            for(i=0; i<nslices-1; i++) note[i][NSUBSLICES-1]=note[i+1][1];
            for(i=1; i<nslices; i++) note[i][0]=note[i-1][NSUBSLICES-2];

            for(i=0; i<nslices; i++) {
                  slice = i%NSLICES;
      /*Learn chord to melody*/
                  for (j=0; j<NSUBSLICES; j++) {
                        if (note[i][j] >= 0) {
                              interval[j] = note[i][j] - chordroot(chord[i]);
                              if(interval[j]<0) interval[j] += 12;
                        }
                        else if (note[i][j] == -2) interval[j]=NTONES-2; /* REST */
                        else interval[j]=NTONES-1; /* SUSTAIN */
                        c->mel[slice][chordtype(chord[i])][j][interval[j]] = 1;
                  }
      /* Learn melody to melody */
                  for (j=0; j<NSUBSLICES; j++)
                        for (k=0; k<NSUBSLICES; k++)
                              if (j!=k)
                                    m->me[slice]->mel[j][k][interval[j]][interval[k]] = 1;
            }
            fclose(fp);
      } while (*fname != '*');
      do {
            printf("Save matrices? (yes=1, no=0)\n");
      }     while (scanf("%d",&answer) != 1);
      if(answer)  save_matrices();
}

/*•••••••••••••••••  File "ReadSaveMatrices.c" ••••••••••••••••••••••*/

#include <stdlib.h>
#include <stdio.h>
#include "StructDefs.h"
#include "Externals.h"
```

```
read_matrices() {
        int     i,j,k,n,p,errno;
        char  *fname  =  "**************************";
        FILE          *fp;

        printf( "Give filename!\n");
        if(output_to_file) fprintf(log_file,  "Give filename!\n");
        scanf("%s",fname);
        if (output_to_file) fprintf(log_file,"%s",*fname);
        if((fp = fopen(fname,"r")) == NULL) {
                printf("Error in opening file, errno %d\n",errno);
                return;
        }
        for(i=0; i<NSLICES; i++)
                for(j=0; j<NCHORDTYPES; j++)
                        for(k=0; k<NSUBSLICES; k++)
                                for(n=0; n<NTONES; n++)
                                        fscanf(fp,"%d ",&c->mel[i][j][k][n]);
        for(i=0; i<NSLICES; i++)
                for(j=0; j<NSUBSLICES; j++)
                        for(k=0; k<NSUBSLICES; k++)
                                for(n=0; n<NTONES; n++)
                                        for(p=0; p<NTONES; p++)
                                                fscanf(fp,"%d ",&m->me[i]->mel[j][k][n][p]);
        fclose(fp);
}

save_matrices() {
        int     i,j,k,n,p,errno;
        char  *fname  =  "**************************";
        FILE          *fp;

        printf("Give filename!\n");
        scanf("%s",fname);
        if((fp = fopen(fname,"w")) == NULL) {
                printf("Error in opening file, errno %d\n",errno);
                return;
        }
        for(i=0; i<NSLICES; i++)
                for(j=0; j<NCHORDTYPES; j++)
                        for(k=0; k<NSUBSLICES; k++)
                                for(n=0; n<NTONES; n++)
                                        fprintf(fp,"%d ",c->mel[i][j][k][n]);
        for(i=0; i<NSLICES; i++)
                for(j=0; j<NSUBSLICES; j++)
                        for(k=0; k<NSUBSLICES; k++)
                                for(n=0; n<NTONES; n++)
                                        for(p=0; p<NTONES; p++)
                                                fprintf(fp,"%d ",m->me[i]->mel[j][k][n][p]);
```

```
        fclose(fp);
}

/*••••••••••••••••• File "Test.c" ••••••••••••••••••••*/

#include "StructDefs.h"
#include <stdio.h>
#include <stdlib.h>

extern      MELODYtoMELODY    *m;
extern      CHORDtoMELODY     *c;
extern      NEURON melody[NSUBSLICES][NTONES];
extern      int chord[100],note[NSUBSLICES];
extern      double CtoMscale,MtoMscale,inhibition;

FILE        *log_file,  *MIDI_file;
char        *output_file_name, *MF_name;
int         output_to_file, output_to_MF;
int         oldMIDInote=60, rest=1;
long int    delta_time=0,length;
extern int  errno;

char        *MFdata;

test() {
        int     i,nchords,chordnumber,starting_note,oldMIDInote=-1;
        extern int input_starting_note();

        if((MFdata = (char *) calloc(10000,sizeof(char))) == NULL) {
                printf("Out of memory!\n");
                exit;
        }
        printf("Do you want output to text file (1/0) ?");
        scanf("%d", &output_to_file);
        if(output_to_file) {
                printf("Give name of output text file!");
                scanf("%s", output_file_name);
                if((log_file = fopen(output_file_name,"w")) == NULL)
                        printf("Error in opening file, errno %d\n",errno);
        }
        printf("Do you want output to MIDI file (1/0) ?");
        scanf("%d", &output_to_MF);
        if(output_to_MF) {
                printf("Give name of output MIDI file!");
                scanf("%s", MF_name);
                if((MIDI_file = fopen(MF_name,"w")) == NULL)
                        printf("Error in opening file, errno %d\n",errno);
        }
        if(output_to_MF) {
                writeMFHeaderChunk(MIDI_file);
                calcMFMeterChunk(&length,MFdata);
```

```
            writeMFChunk(MIDI_file,length,MFdata);
        }
        length=1;
        nchords = read_testfile();
        do {
            scale_connections();
            do {
                if((starting_note = input_starting_note()) < 0) break;
                initialize(starting_note);
                for (chordnumber=0; chordnumber<nchords-1; chordnumber++) {
                    relax(chordnumber);
                    decode(chordnumber);
                    reinitialize(chordnumber);
                }
            } while (starting_note >= 0);
        } while (starting_note >= -1);
        MFdata[0] = 0; MFdata[1] = 0x90;
        delta_time += 48;
        if(!rest) {putVarLen(delta_time);
            MFdata[length++]=oldMIDInote; MFdata[length++]=0;}
        MFdata[length++]=0;MFdata[length++]=0xff;
            MFdata[length++]=0x2f; MFdata[length++]=0x00; /* end */
        if(output_to_MF) writeMFChunk(MIDI_file,length,MFdata);
        if(output_to_file) fclose(log_file);
        if(output_to_MF) fclose(MIDI_file);

}

/*••••••••••••••••••• File "ScaleConnections.c" ••••••••••••••••••••••*/

#include <stdio.h>
#include "StructDefs.h"
#include "Externals.h"

scale_connections() {
        int             i,j,k,n,p;
        double          sum=0,CtoMtotal,MtoMtotal;

        printf("Give CtoMtotal, MtoMtotal, inhibition!\n");
        if(output_to_file)
            fprintf(log_file, "Give CtoMtotal, MtoMtotal, inhibition!\n");
        scanf("%lf %lf %lf",&CtoMtotal,&MtoMtotal,&inhibition);
        if (output_to_file)
            fprintf(log_file,"%lf %lf %lf\n",CtoMtotal, MtoMtotal, inhibition);
        for(i=0; i<NSLICES; i++)
            for(j=0; j<NCHORDTYPES; j++)
                for(k=0; k<NSUBSLICES; k++)
                    for(n=0; n<NTONES; n++)
                        sum += (double) c->mel[i][j][k][n];
        CtoMscale = NSLICES * CtoMtotal/(double) sum;
        sum=0;
```

```
        for(i=0; i<NSLICES; i++)
                for(j=0; j<NSUBSLICES; j++)
                        for(k=0; k<NSUBSLICES; k++)
                                for(n=0; n<NTONES; n++)
                                        for(p=0; p<NTONES; p++)
                                                sum += (double) m->me[i]->mel[j][k][n][p] ;
        MtoMscale = NSLICES * MtoMtotal/(double) sum;

}

/*••••••••••••••••••• File "ReadTestFile.c" ••••••••••••••••••••••*/

#include <stdio.h>
#include <stdlib.h>
#include "StructDefs.h"
#include "Externals.h"

int read_testfile() {
        extern int chordtype(),chordroot(),errno;

        int             answer,i,j,k,nchords;
        char            *fname = "**************************";
        FILE            *fp;

        printf("Give filename!\n");
        if(output_to_file) fprintf(log_file, "Give name of test file!\n");
        scanf("%s",fname);
        if (output_to_file) fprintf(log_file,"%s\n",fname);
        if((fp = fopen(fname,"r")) == NULL) {
                printf("Error in opening file, errno %d\n",errno);
                return;
        }
        fscanf(fp,"%d",&nchords);
        for(i=0; i<nchords; i++) fscanf(fp,"%d",&chord[i]);
        fclose(fp);
        return nchords;
}

/*••••••••••••••••••• File "InputStartingNote.c" ••••••••••••••••••••••*/

#include <stdio.h>
#include <stdlib.h>
#include "StructDefs.h"
#include "Externals.h"

int input_starting_note() {
        int starting_note;
        printf("Give starting note number! ( -1 to rescale, -2 to stop testing) \n");
        if(output_to_file)
                fprintf(log_file,
                        "Give starting note number! ( -1 to rescale, -2 to stop testing) \n");
```

```
        scanf("%d",&starting_note);
        if (output_to_file) fprintf(log_file,"%d\n",starting_note);
        return starting_note;
}

/*••••••••••••••••• File "Initialize.c" ••••••••••••••••••••*/

#include <time.h>
#include <stdlib.h>
#include "StructDefs.h"
#include "Externals.h"
#define SHAKE_RANGE 0.1

int             maxcycles,errorperiod;

initialize(int starting_note) {
        extern int      chordtype(),chordroot();
        int             i,j;

        srand(clock());
        printf("Give max number of cycles!\n");
        if(output_to_file) fprintf(log_file, "Give max number of cycles!\n");
        scanf("%d",&maxcycles );
        if (output_to_file) fprintf(log_file,"%d\n",maxcycles);
        for(i=0; i<NSUBSLICES; i++)
                for(j=0; j<NTONES; j++)
                        melody[i][j].activation = SHAKE_RANGE*
                                (double)rand()/(double)RAND_MAX;
        melody[1][starting_note].activation = 1;
        compute_biases(0);
}

reinitialize(int chordnumber) {
        extern int      chordtype(),chordroot();
        int             i,j,note0,note1,interval;

        for(i=0; i<NSUBSLICES; i++)
                for(j=0; j<NTONES; j++)
                        melody[i][j].activation = SHAKE_RANGE*
                                (double)rand()/(double)RAND_MAX;
        interval = chordroot(chord[chordnumber])-chordroot(chord[chordnumber+1]);
        if(interval<0) interval += 12;
        if (note[NSUBSLICES-2] < 12)
                note0 = (note[NSUBSLICES-2]+interval)%(12);
        else
                note0 = note[NSUBSLICES-2];
        if (note[NSUBSLICES-1] < 12)
                note1 = (note[NSUBSLICES-1]+interval)%(12);
        else
                note1 = note[NSUBSLICES-1];
        melody[0][note0].activation = 1;
```

```
        melody[1][note1].activation = 1;
        compute_biases(chordnumber+1);
}

/*••••••••••••••••••• File "ComputeBiases.c" ••••••••••••••••••••••*/

#include <stdlib.h>
#include "StructDefs.h"
#include "Externals.h"

compute_biases(int chordnumber) {
        int             i,j,k,slice1,slice2,interval;

        slice1 = chordnumber%NSLICES;
        slice2 = (chordnumber+1)%NSLICES;
        for(i=1; i<NSUBSLICES-2; i++) {
                for(j=0; j<NTONES; j++)
                        melody[i][j].bias =
                                ((double) c->mel[slice1][chordtype(chord[chordnumber])][i][j])*
                                        MtoMscale;
        }
        interval = chordroot(chord[chordnumber])-chordroot(chord[chordnumber+1]);
        if(interval<0) interval += 12;
        for(i=NSUBSLICES-1; i<NSUBSLICES; i++) {
                for(j=0; j<12; j++)
                        melody[i][j].bias =
                                ((double) c->mel[slice2][chordtype(chord[chordnumber+1])]
                                [i-NSUBSLICES+2][(j+interval)%12])*CtoMscale;
                for(j=12; j<NTONES; j++)
                        melody[i][j].bias +=
                                ((double) c->mel[slice2][chordtype(chord[chordnumber+1])]
                                        [i-NSUBSLICES+2][j])*CtoMscale;
        }
}

/*••••••••••••••••••• File "Relax.c" ••••••••••••••••••••••*/

relax(int chordnumber) {
        int             i;
        extern int      maxcycles;

        for(i=0; i<maxcycles; i++) {
                compute_new_activations(chordnumber);
                update_activations();
        }
}

/*••••••••••••••••••• File "ComputeNewAct.c" ••••••••••••••••••••••*/

#include <stdlib.h>
#include "StructDefs.h"
```

```
#include "Externals.h"

compute_new_activations(int chordnumber) {

        extern int          chordroot(),chordtype();
        extern double       actfunc();
        int                 i,j,k,n,slice,counter;
        double              excinput,inhinput;

        slice = chordnumber%NSLICES;
        for(counter=0; counter<(NSUBSLICES-2)*NTONES; counter++)  {
                i = 2 + (int)((NSUBSLICES-2) * (double)(rand()-1) / (double) RAND_MAX);
                j = (int)( NTONES * (double)(rand()-1) / (double)RAND_MAX);
                excinput = melody[i][j].bias;inhinput=0;
                for(k=0; k<NSUBSLICES; k++)             /* from subslice */
                        if (i != k)
                                for (n=0; n<NTONES; n++)      /* from tone   */
                                        excinput += ((double) m->me[slice]->mel[k][i][n][j])*
                                                MtoMscale*melody[k][n].activation;
                for(n=0; n<NTONES; n++)
                if (j != n) inhinput += melody[i][n].activation*inhibition;
                melody[i][j].new_activation =
                        actfunc((1-DECAY)* melody[i][j].activation +excinput-inhinput);
                melody[i][j].activation = melody[i][j].new_activation;
        }
}

/*••••••••••••••••••  File "UpdateActivations.c" ••••••••••••••••••••••*/

#include "StructDefs.h"
#include "Externals.h"

update_activations() {
        int     i,j;

        for(i=2; i<NSUBSLICES; i++)
                for(j=0; j<NTONES; j++)
                        melody[i][j].activation = melody[i][j].new_activation;

}

/*••••••••••••••••••  File "Decode.c" ••••••••••••••••••••••*/

#include <stdio.h>
#include <math.h>
#include "StructDefs.h"
#include "Externals.h"

decode(int chordnumber) {
        int             i,j;
        double          maxact;
```

```
        extern int    oldMIDInote;
        static char  *root[] =
                {"C ","Db","D ","Eb","E ","F ","Gb","G ","Ab","A ","Bb","B "},
                        *type[] =
                {"maj7/I","maj7/IV","m7/II","m7/III","m7/VI","13#11",
                                                "13b9#11","alt","m7b5"},
                        *notechar = "****";

        for (i=1; i<NSUBSLICES; i++) {
                maxact = 0;
                for (j=0; j<NTONES; j++)
                        if (melody[i][j].activation > maxact) {
                                note[i] = j;
                                maxact = melody[i][j].activation;
                        }
        }

        printf("%s%s: ",root[chordroot(chord[chordnumber])],
                type[chordtype(chord[chordnumber])]);
        if(output_to_file)
                fprintf(log_file, "%s%s:\t",root[chordroot(chord[chordnumber])],
                        type[chordtype(chord[chordnumber])]);
        for (i=1; i<NSUBSLICES-1; i++) {
                if (output_to_MF)
                        oldMIDInote =
                                decodeToMIDI(chord[chordnumber], note[i], oldMIDInote);
                if (note[i] == NTONES-2) notechar ="* ";
                else if (note[i] == NTONES-1) notechar ="+ ";
                else
                        notechar = root[(note[i]+chordroot(chord[chordnumber]))%12];
                printf ("%s ",notechar);
                if(output_to_file) fprintf(log_file, "%s ",notechar);
        }
        printf("\n");
        if(output_to_file)  fprintf(log_file,"\n");
}

/*••••••••••••••••••• File "ComputeError.c" ••••••••••••••••••••••*/

#include <math.h>
#include "StructDefs.h"
#include "Externals.h"

double error() {
        extern double       actfunc();
        extern int          chordroot(),chordtype();
        int                 i,j,k, interval;
        double              error=0;

        for(i=2; i<NSUBSLICES; i++)
                for (j=0; j<NTONES; j++)
```

```
                error += fabs(melody[i][j].new_activation-
                        melody[i][j].activation);
    return error;
}

/*••••••••••••••••••  File "WriteToMIDI.c" ••••••••••••••••••••••*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "StructDefs.h"
#include "Externals.h"

writeMFHeaderChunk(fPointer)
    FILE        *fPointer;
{
    char data[14];
    int i;
    char *dStr =
        "4d 54 68 64 00 00 00 06 00 01 00 02 00 60 ";

    for(i=0; i<14; i++) data[i]=(char) strtol(&dStr[3*i],NULL,16);
    for(i=0; i<14; i++) fprintf(fPointer,"%c",data[i]);

}

writeMFChunk(fPointer,length,data)
    FILE        *fPointer;
    char        *data;
    long int    length;
{
    char temp[8];
    long int i, buffer;

    temp[0] = 0x4d; temp[1] = 0x54; temp[2] = 0x72; temp[3] = 0x6b;
    buffer = length;
    for(i=0; i<4; i++) {
        temp[7-i] = buffer&255;
        buffer /= 256;
    }
    for(i=0;i<8;i++)
        fprintf(fPointer,"%c",temp[i]);
    for(i=0; i<length; i++)
        fprintf(fPointer,"%c",data[i]);

}

calcMFMeterChunk(length,data)
    char        *data;
    long int    *length;
{
```

```
        int i;

        char *dStr =
                "00 ff 58 04 04 02 18 08 00 ff 51 03 07 a1 20 83 00 ff 2f 00 ";
        *length = (long int) strlen(dStr)/3;
        for(i=0;i<*length;i++)  data[i]=(char)  strtol(&dStr[3*i],NULL,16);

}

decodeToMIDI(chord, note, old_note)
        int chord, note, old_note;
{
        extern long int delta_time, length;
        extern          char *MFdata;
        extern int rest;
        int temp;

        delta_time += 48;
        if (note == NTONES-2) {                 /* REST */
                if(!rest) {
                        putVarLen(delta_time);
                        MFdata[length++]=old_note;
                        MFdata[length++]=0;
                        delta_time=0; note = old_note; rest = 1;
                }
                note=old_note;
        }
        else if (note == NTONES-1) {            /* LIGATURE */
                note=old_note;
        }
        else {
                putVarLen(delta_time);MFdata[length++]=old_note;MFdata[length++]=0;
                temp = (note+chordroot(chord))%12;
                while (abs(temp-old_note) > 6) temp += 12;
                if(temp<55) temp += 12; if(temp>85) temp -= 12;
                MFdata[length++]=0;MFdata[length++]=temp;MFdata[length++]=64;
                delta_time = 0; note = temp; rest = 0;
        }
        return note;
}

putVarLen(value) register long int value; {
        extern          char *MFdata;
        extern          long int length;
        register long buffer;

        buffer = value & 0x7f;
        while ((value>>=7) > 0) {
                buffer<<=8;buffer |= 0x80;buffer+=(value & 0x7f);
        }
```

```
        while(TRUE) {
                MFdata[length++]=buffer;
                if(buffer & 0x80) buffer >>= 8;
                else break;
        }
}

/*•••••••••••••••••• THE END ••••••••••••••••••••••*/
```