JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Richard Vogl

Submitted at
Institute of Computational Perception

Supervisor and
First Examiner
Gerhard Widmer

Second Examiner
Matthew Davies

Co-Supervisor
Peter Knees

November 2018

Deep Learning Methods for Drum Transcription and Drum Pattern Generation

Doctoral Thesis
to obtain the academic degree of
Doktor der technischen Wissenschaften
in the Doctoral Program
Technische Wissenschaften

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

STATUTORY DECLARATION

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references. This printed thesis is identical with the electronic version submitted.

Linz, November 2018

ABSTRACT

This thesis is situated in the field of *music information retrieval* and addresses the tasks of automatic drum transcription and automatic drum pattern generation. Automatic drum transcription deals with the problem of extracting a symbolic representation of the notes played by drum instruments from an audio signal. Automatic drum pattern generation aims at generating novel, musically meaningful and interesting rhythmic patterns involving several percussion instruments.

The first part of this thesis focuses on automatic drum transcription. Music transcription from audio is a hard task, which can be challenging even for trained human experts. Challenges in drum transcription are the large variety of sounds for individual instrument types as well as groups of similar sounding instruments like different types of cymbals or tom-toms of varying sizes. The contributions covered by the drum transcription part introduce end-to-end deep learning methods for this task. With these, a new state of the art is established on a variety of public drum transcription datasets, as well as in the MIREX drum transcription competition. Furthermore, two additional objectives are met: *(i)* adding meta information like bar boundaries, meter, and local tempo to the transcripts, as well as *(ii)* increasing the number of instruments under observation. While traditionally, only bass drum, snare drum, and hi-hat have been focused on, in this thesis up to 18 different instrument classes are considered.

The second part of this thesis deals with automatic drum pattern generation. The goal is to generate patterns which are musically meaningful and indistinguishable from human-created ones, and at the same time are not trivial but interesting. Evaluating generative methods is non-trivial, since quality in this context is subjective. This issue is addressed by conducting qualitative and quantitative user studies for evaluation purposes. Two different models are proposed for drum pattern generation: restricted Boltzmann machines (RBMs) and generative adversarial networks (GANs). While RBMs are comparably easy to train, GANs are more problematic in this respect, requiring more training data; on the other hand, GANs can better handle a greater variety of instruments and higher temporal resolutions.

The need for data is met through two different approaches: *(i)* by creating synthetic large scale drum pattern datasets, and *(ii)* by leveraging the drum transcription methods from the first part of the thesis to extract drum patterns from real audio. Besides these methodological contributions, different user interfaces for drum pattern generation are implemented and evaluated in user studies.

In addition, this thesis offers publicly available datasets and trained models for drum transcription as resources for the research community.

ZUSAMMENFASSUNG

Die vorliegende Dissertation ist im Bereich *Music Information Retrieval* anzusiedeln und befasst sich mit automatischer Schlagzeugtranskription und automatischer Generierung von Drum-Patterns. Unter Schlagzeugtranskription versteht man den Prozess eine symbolische Darstellung der von Schlaginstrumenten gespielten Noten aus einem Audiosignal zu extrahieren. Bei der automatischen Generierung von Drum-Patterns gilt es Methoden zur Erzeugung von musikalisch sinnvollen, neuartigen und interessanten Rhythmen für Schlaginstrumente zu finden.

Der erste Teil dieser Arbeit befasst sich mit automatischer Schlagzeugtranskription. Transkription von Musik ist eine schwierige Aufgabe, die selbst für Fachkundige anspruchsvoll sein kann. Herausforderungen bei der Schlagzeugtranskription sind einerseits die klangliche Vielfalt einzelner Instrumenttypen, andererseits die Differenzierung innerhalb Gruppen ähnlich klingender Instrumente wie z.B. verschiedene Arten von Becken oder Trommeln unterschiedlicher Größe. In dieser Arbeit werden end-to-end Deep-Learning-Methoden für Schlagzeugtranskription verwendet. Mithilfe dieser werden neue Bestresultate auf öffentlichen Datensätzen sowie beim MIREX Schlagzeugtranskriptions-Task erreicht. Darüber hinaus werden zwei weitere Ziele erreicht: *(i)* Extrahieren zusätzlicher Metainformationen wie Taktgrenzen, Taktart und lokales Tempo, sowie *(ii)* Erhöhung der Anzahl der Instrumente bei der Transkription. Während in anderen Arbeiten aus diesem Themenbereich nur Bassdrum, Snare und Hi-Hat berücksichtigt werden, kommen hier bis zu 18 verschiedene Instrumentklassen zum Einsatz.

Der zweite Teil dieser Arbeit beschäftigt sich mit der automatischen Generierung von Drum-Patterns. Dabei sollen interessante musikalische Drum-Patterns erzeugt werden, die wie von Menschen kreierte klingen. Die Evaluierung solch generativer Methoden ist im allgemeinen diffizil, da Qualität in diesem Kontext subjektiv ist. Dieses Problem wird mittels qualitativer Interviews und quantitativer Umfragen gelöst. Zur Generierung von Drum-Patterns werden zwei verschiedene Modelle verwendet: *Restricted Boltzmann Machines* (RBMs) und *Generativ Adversarial Networks* (GANs). Während RBMs vergleichsweise einfach zu trainieren sind, gestaltet sich dies bei GANs problematischer. GANs benötigen außerdem mehr Trainingsdaten, können jedoch dafür besser mit einer größeren Vielfalt an Instrumenten und höheren zeitlichen Auflösungen umgehen.

Der Bedarf großer Mengen an Trainigsdaten wird auf zwei Arten gedeckt: *(i)* durch das Erstellen eines großen synthetischen Drum-

Pattern-Datensatzes und *(ii)* mittels der im ersten Teil vorgestellten Transkriptionsmethoden, mit denen Drum-Patterns aus Musik extrahieren werden. Weiters werden verschiedene Softwareprototypen für die Erzeugung von Drumpatterns implementiert und evaluiert.

Als zusätzliches Ergebnis werden erstellte Datensätze und vortrainierte Transkriptionsmodelle der Forschungsgemeinschaft frei zur Verfügung gestellt.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Gerhard Widmer, for his support and for giving me the opportunity to work towards a PhD at the Institute of Computational Perception.

I am very grateful to my co-supervisor, Peter Knees, who acted as a mentor throughout my PhD studies and who allowed me to work on interesting research projects.

I would also like to thank Matthew Davies, for taking the time and energy to review this thesis.

Many thanks go to all the co-authors of the publications for this thesis (ordered as they appear in this work): Matthias Dorfer, Peter Knees, Gerhard Widmer, Matthias Leimeister, Cárthach Ó Nuanáin, Sergi Jordà, Michael Hlatky, Hamid Eghbal-zadeh, Chih-Wei Wu, Christian Dittmar, Carl Southall, Jason Hockman, Meinhard Müller, Alexander Lerch, Angel Faraldo, Perfecto Herrera, Sebastian Böck, Florian Hörschläger, and Mickael Le Goff.

Conducting my research would have been impossible without the aid and support of my colleagues at the Institute of Computational Perception at JKU Linz, at the Information and Software Engineering Group at TU Wien, and at the Austrian Research Institute for Artificial Intelligence.

Special thanks go to my family who supports me in all I do, as well as to my friends for their encouragement in difficult times.

Finally, I want to thank Katja, my girlfriend, who had to put up with me having only little time besides work for the last four years, and still provided her continued support and encouragement.

The research in this thesis was funded by the European Union Seventh Framework Programme FP7 / 2007-2013 through the *GiantSteps* projects (grant agreements no. 610591), and by the Austrian FFG under the BRIDGE 1 project *SmarterJam* (858514).

CONTENTS

Part I

INTRODUCTION AND BACKGROUND

1 INTRODUCTION

The field of music information retrieval (MIR), also *music information research*, comprises research in the intersection of many different disciplines. Important contributing areas are signal processing, musicology, psychoacoustics, data science, human–computer interaction (HCI), and lately increasingly machine learning and artificial intelligence. MIR deals, amongst other things, with the extraction of meaningful information about music. In this context, music can be represented as audio signals (monophonic, multi track, etc.), images (sheet music, album artwork, tablature, lyrics, etc.), or even other symbolic representations (e.g.: MIDI files, text, websites, etc.). To extract meaningful information from these sources, different data manipulation and analysis methods from the fields listed above are applied. Research in this field has enabled new technologies for commercial products as well as musicology research. The range of applications reaches from entertainment and experimental apps (Shazam[1], Smule[2], etc.), via productivity tools for music production (onset detection, beat tracking, transcription, etc.), to large scale analytic tools for musicology and archive applications (optical music recognition (OMR), transcription, classification, etc.).

One of the central topics is automatic music transcription (AMT), which is an active area of research in the field of MIR. The goal in AMT is to extract a complete symbolic representation of a piece of music from a digital audio recording. The extracted symbolic representation may be presented in different formats, e.g.: as sheet music or musical instrument digital interface (MIDI) files. The process of transcription can be seen as the inversion of a musician interpreting and performing from sheet music. Trained humans are generally capable of solving this task satisfactory well, depending on complexity of the music and quality of the audio source. However, even for experienced professionals this process takes a lot of time. A reliable AMT solution would facilitate finding solutions for other MIR tasks. This is due to the fact that working with symbolic data is often easier since it circumvents the error prone, and often ill-defined step of extracting information from the audio signal. AMT solutions would also be useful as end-user products for musicians, music producers, and librarians.

The first part of this work deals with a subtask of AMT, namely automatic drum transcription (ADT). ADT focuses on transcribing drum instruments from an audio recording of a music piece. Although being a narrower and better defined subtask, ADT is still far from being

1 https://www.shazam.com/

2 https://www.smule.com/

solved. Nevertheless, in recent years significant progress has been made within the scientific community. The existence of a variety of publicly available ADT datasets simplifies evaluation and comparison of different approaches. This said, currently existing datasets often have limitations, and especially for data-hungry machine learning methods, these datasets are often not sufficiently large and diverse.

Another area of research in the field of MIR is automatic music generation. Although strictly speaking not *information retrieval*, music generation has a close relation to MIR via a common basis consisting of musicology, psychoacoustics, machine learning, and artificial intelligence. In the case of automatic music generation, the aforementioned interpretation of MIR as *music information research* is more fitting. In the past, many attempts and approaches to automatically generate music have been presented. While the music generated by those approaches is often interesting and promising, until now, a system that automatically creates convincing and pleasing music has not been found. Since the problem in its entirety is quite complex, a reasonable approach is to break it down into easier-to-solve subtasks. Similar as with AMT, focusing on drums represents a better defined and arguably easier subtask, since the complex topics of melody and harmony can mostly be ignored and the main focus can be shifted to rhythmic patterns, structure, and arrangement. However, especially the question of global structure and arrangement are similarly challenging for drum tracks when compared to harmonic instruments.

The second part of this work deals with automatic generation of drum patterns. In drum pattern generation the goal is to find a method which is able to create drum patterns or even a whole drum track given a certain context. This context might be provided implicitly by a piece of music or explicitly by certain parameters like tempo, musical style, complexity, et cetera. A special case of this is automatic drum pattern variation. Here, the necessary context is provided through a short drum pattern. Given this seed drum pattern, the goal is to generate similar patterns with different characteristics regarding certain aspects. For example a variation of a pattern might be more *complex* or less *intense* than the original. In this work both drum pattern variation and the more general drum pattern generation will be discussed. As part of the music generation task, the topic of drum pattern generation is quite artistic in nature, i.e. there is no clear definition of correct results. Thus, one of the main challenges is to evaluate and objectively compare different approaches.

The two parts of this work (ADT and automatic drum pattern generation) are linked by the need for huge amounts of diverse training data. This requirement for pattern generation can be met by using ADT methods to generate large amounts of drum patterns using datasets which contain the required additional annotations needed for pattern generation (e.g. genre annotations). Furthermore, findings from ADT

research (e.g. which models work best and why), can lead to a better understanding of properties of drum patterns, which in turn can be used to improve generative models.

1.1 OUTLINE

The remainder of this work is structured as follows: First, this introduction will continue with mandatory sections covering the scientific contributions and publications which make up this thesis. After that *(i)* an introduction to the ADT task, *(ii)* an introduction to the drum pattern generation task, and *(iii)* a basic introduction of machine learning methods used throughout this thesis are provided in Chapter 2. Two distinct parts (Parts ii and iii) dealing with ADT and drum pattern generation follow. These parts consist of original publications, each prefaced with a short overview section that provides the context and a short summary. Parts ii and iii each start with a synopsis that put the individual publications into context and provide a narrative for the part. In the last part, published datasets are summarized, a discussion of the publications of this work from a critical standpoint is provided, and an overview of future research directions for this field of research is provided. The work concludes with a discussion of the findings of this thesis in a broader context.

1.2 CONTRIBUTIONS

The main scientific contributions of this thesis, which are presented in the individual chapters and publications, are as follows:

1. **ADT state of the art:** The first major contribution of this thesis is the proposed end-to-end ADT architecture utilizing trained neural networks as the main transcription engine. The approach is first presented in Chapter 4 and used in the subsequent publications and chapters. Using this architecture, different methods for ADT were implemented which set a new state of the art for drum transcription performance and were able to outperform other methods in competitive evaluations (MIREX'17 [95], MIREX'18 [99], Chapter 8).

2. **Deep learning for ADT:** The work demonstrates successful application and provides a detailed evaluation of different neural network architectures in the context of ADT. This includes recurrent (Chapter 4), convolutional, and convolutional recurrent neural networks (Chapter 6) architectures, as well as special training techniques like data augmentation (Chapter 5), multi-task learning (Chapter 6), and pre-training (Chapter 7) [93, 94, 101].

3. **Expanding the capabilities of ADT systems:** Efforts to extend the capabilities of ADT systems are presented. These consist of adding meta information to transcripts using multi-task learning [94] (Chapter 6) as well as extending the range of drum instruments considered for transcription [101] (Chapter 7).

4. **Drum pattern generation methods:** In this work, machine learning techniques to generate drum patterns are introduced [20, 98] (Chapter 10) and thoroughly evaluated [100] (Chapter 11).

5. **Drum pattern generation user interfaces:** Chapters 10, 12, and 13 cover the implementation and evaluation of novel user interface (UI) approaches for drum pattern generation tools [20, 98, 100].

6. **Datasets and models:** This work also provides trained models for ADT as well as several datasets which are available to the research community: RBMA drums, a synthetic large-scale drum transcription dataset, and the GiantSteps dataset (key and tempo part). Chapter 14 discusses these datasets, while trained models are provided for the MIREX submissions covered in Chapter 8. The files are available under the following URL: `http://ifs.tuwien.ac.at/~vogl/` .

1.3 MAIN PUBLICATIONS

This list provides an overview of the publications which represent the main scientific content of this thesis. These publications can be split into two groups: the first four publications deal with ADT while the rest cover drum pattern creation. This split into two topics is also reflected by the two parts of the thesis.

1. Richard Vogl, Matthias Dorfer, and Peter Knees. "Recurrent neural networks for drum transcription". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*. New York, NY, USA, 2016 [92].

2. Richard Vogl, Matthias Dorfer, and Peter Knees. "Drum Transcription from Polyphonic Music with Recurrent Neural Networks". In: *Proceedings of the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. New Orleans, LA, USA, 2017 [93].

3. Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees. "Drum Transcription via Joint Beat and Drum Modeling using Convolutional Recurrent Neural Networks". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017 [94].

4. Richard Vogl, Gerhard Widmer, and Peter Knees. "Towards Multi-Instrument Drum Transcription". In: *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx)*. Aveiro, Portugal, 2018 [101].

5. Richard Vogl and Peter Knees. "An Intelligent Musical Rhythm Variation Interface". In: *Companion Publication 21st International Conference on Intelligent User Interfaces*. Sonoma, CA, USA, 2016 [97].

6. Richard Vogl, Matthias Leimeister, Cárthach Ó Nuanáin, Sergi Jordà, Michael Hlatky, and Peter Knees. "An Intelligent Interface for Drum Pattern Variation and Comparative Evaluation of Algorithms". In: *Journal of the Audio Engineering Society* 64.7/8 (2016) [100].

7. Richard Vogl and Peter Knees. "An Intelligent Drum Machine for Electronic Dance Music Production and Performance". In: *Proceedings of the 17th International Conference on New Interfaces for Musical Expression (NIME)*. Copenhagen, Denmark, 2017 [98].

8. Hamid Eghbal-Zadeh*, Richard Vogl*, Gerhard Widmer, and Peter Knees. "A GAN based Drum Pattern Generation UI Prototype". In: *Late Breaking/Demos, 19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018 [20]. *Equal contribution.

1.4 ADDITIONAL PUBLICATIONS

The following list contains publications which are relevant in the context of this thesis. The list comprises works that were published as contributing author [47, 105], competition submissions [95, 99], and a submission covering an installation demonstrating generative models for an art exhibition [96].

- Chih-Wei Wu, Christian Dittmar, Carl Southall, Richard Vogl, Gerhard Widmer, Jason Hockman, Meinhard Müller, and Alexander Lerch. "A Review of Automatic Drum Transcription". In: *IEEE Transactions on Audio, Speech and Language Processing* 26.9 (2018) [105].

- Peter Knees, Angel Faraldo, Perfecto Herrera, Richard Vogl, Sebastian Böck, Florian Hörschläger, and Mickael Le Goff. "Two Data Sets for Tempo Estimation and Key Detection in Electronic Dance Music Annotated from User Corrections". In: *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*. Malaga, Spain, 2015 [47].

- Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees. "MIREX Submission For Drum Transcription 2017". In: *MIREX extended abstracts, 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017 [95].

- Richard Vogl and Peter Knees. "MIREX Submission For Drum Transcription 2018". In: *MIREX extended abstracts, 19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018 [99].

- Richard Vogl*, Hamid Eghbal-Zadeh*, Gerhard Widmer, and Peter Knees. "GANs and Poses: An Interactive Generative Music Installation Controlled by Dance Moves". In: *Interactive Machine-Learning for Music @Exhibition at ISMIR*. Paris, France, 2018 [96]. *Equal contribution.

2 BACKGROUND

This chapter provides all the necessary information and terminology that is required as a basis to understand the technical descriptions in the individual publications of this thesis. The chapter is structured into three sections, which cover *(i)* introduction, related work, and challenges of ADT (Section 2.1), *(ii)* the same for automatic drum pattern generation (Section 2.2.1), and *(iii)* an introduction to required deep learning basics (Section 2.3). Sections 2.1 and 2.2.1 also provide an overview of former state-of-the-art methods and their working principles.

2.1 AUTOMATIC DRUM TRANSCRIPTION

As already mentioned, ADT is a subtask of AMT, which in turn is about generating a symbolic representation of a music piece, given a digital audio recording. Transcription is quite a hard task, but the many potential applications for even only partial solutions provide a great incentive to work on this task. When transcribing a piece of music, the goal is to extract the following information for each played note:

- Which instrument plays the note
- When does the played note start
- When does the played note end
- Which pitch does the played note have
- How loud is the note
- Which special playing technique is used

Some of this information is not always applicable for notes of different instruments, e.g. a harpsichord has only a very limited (in an arrangement usually indistinguishable) range of loudness, cymbals do not have a precise pitch, and for drums there is usually no clear end of a note. However, every note is assignable to an instrument, and has an onset time (note start), even if it might not be unambiguously detectable using only the audio recording.

A common model that is used to synthesize musical notes is the so called attack-decay-sustain-release (ADSR) model. While this model is mainly used to generate sound, it is also helpful to understand the different phases the signal of a played note goes through. Figure 2.1

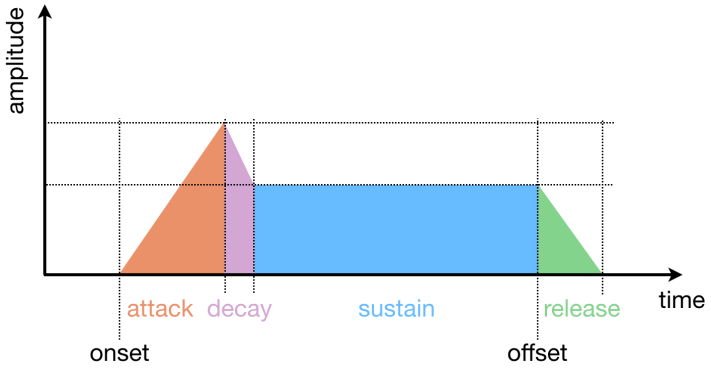


Figure 2.1: Model of phases of a note played by a musical instrument. The hight of the individual blocks indicates the typical relative amplitude and its development over time.

visualizes the phases of the ADSR model. In the following the single phases of the model are explained for the case of an acoustic piano playing a note: At the onset a piano key is pressed, this results in setting the piano action in motion, lifting the damper of the corresponding set of strings and pushing the hammer towards the strings. During the attack phase, the sound quickly builds up and a percussive sound generated by the hammer hitting the strings is produced. After this first percussive burst of energy has faded away (decay phase), the strings start to oscillate in their respective tuned fundamental and harmonic frequencies (sustain phase). As soon as the key of the piano is released, the damper is lowered again, which results in the strings quickly stopping to oscillate (release phase).

In the context of AMT all of these phases contain valuable information. Of special interest are the onset time, attack phase, sustain phase, and offset time of the note. When performing transcription, the properties of interest identified earlier can be deduced by focusing on these parts of each note.

The onset of a played note is arguably the most important part: humans can, for certain instruments, identify pitch and even type of instrument when provided with only attack and decay phases of a played note. For percussive instruments (i.e. instruments that are struck or plucked in some form - e.g.: piano, xylophone, cymbals, drums), the attack phase contains broadband noise, which makes it easier to identify the exact point in time of the onset. In contrast to that, bowed or blown instruments often have the capability to slowly build up the volume of a note, making it hard to exactly determine the exact onset time.

During the sustain phase, pitched instruments produce tones with a fundamental frequency (f_0) and harmonics (overtones at frequencies $f_i = f_0 \cdot i, i \in \mathbb{N}_{>0}$), which result in a perceptible note pitch. Pitched instruments are usually capable of producing tones with similar timbre but different pitches (e.g. piano, violin, trumpet, xylophone, organ).

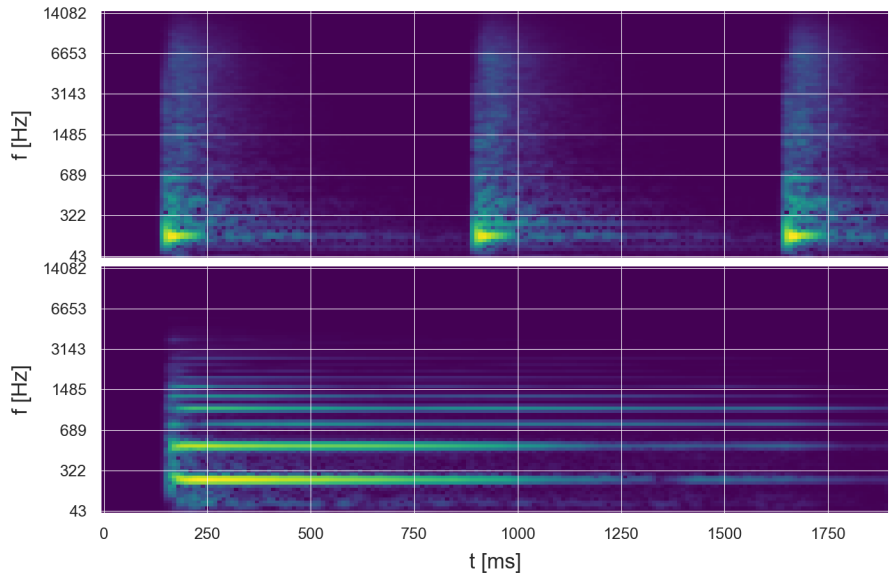


Figure 2.2: Spectrogram of snare drum hits (top) and a piano playing a middle C $f_0 = 261.63$Hz (bottom). The harmonic frequencies are clearly visible for the piano note (bright parallel lines), whereas the drum exhibits a more noisy signal with stray frequencies.

The frequency spectrum of unpitched instruments either consists of inharmonic frequencies (overtones at frequencies $f_i = f_0 \cdot i, i \in \mathbb{R}_{>0}$, e.g. membranophones) or irregular broadband noise. Some instruments, like timpani, while technically being unpitched instruments (a kind of drum i.e. a membranophone), appear to have pitch. This is often due to special acoustic designs of the instrument; e.g. in case of the timpani, the drum is designed in a way that amplifies harmonic frequencies while inharmonic frequencies are dampened, leaving a pseudo-harmonic pitch to be perceived. Figure 2.2 shows the spectrum of snare drum hits and the spectrum of a piano note, visualizing the difference of the distribution of spectral energy between pitched and unpitched instruments.

Detecting the exact point in time of the offset has proven to be difficult, but is required to identify note length of the played note. Usually this is due to the fact that the sound produced by most instruments tends to fade out slowly after the note is stopped (release phase) compared to the rapid increase in loudness during the attack phase. It remains to be discussed and tested how well humans are able to exactly identify and locate offsets. Arguably, experience and knowledge about musical genre, played instruments and their playing techniques, as well as musical notation will have a significant impact when transcribing music manually and deciding about note lengths. This leads to the conclusion that for efficiently identifying offsets, all this should also be taken into consideration.

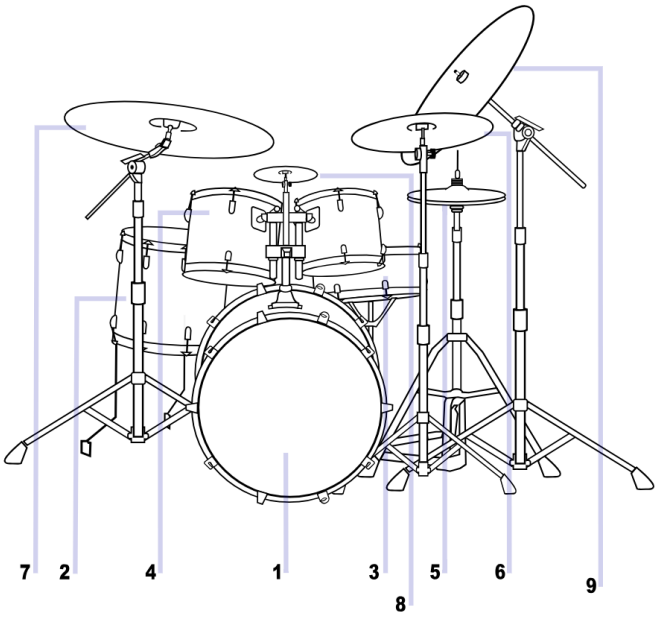


Figure 2.3: Visualization of an example drum kit consisting of different drum instruments. (1) Bass (or kick) drum (2) floor tom (3) snare drum (4) hanging/rack toms (5) hi-hat (6) crash cymbal (7) ride cymbal (8) splash cymbal (9) china cymbal. Source: Wikipedia [103].

Special playing techniques are unique for every instrument and may change different attributes of the produced notes and sound. For some instruments it is possible to manipulate the timbre of the played note (e.g. muted trumpet), the duration (e.g. plucked violin, choked cymbals), the pitch (e.g. vibratos, pitch bends, glissando), or to add percussive elements or change from pitched to unpitched sound (e.g. col legno/battuto on violin). Identifying these special techniques is challenging, primarily because of their diversity, and instrument dependency.

Even after retrieving all of this note-related information, a full transcript still requires additional meta-information which is required to reconstruct complete sheet music. This additional meta-information covers overall tempo of the piece, bar division and meter, key annotations, as well as local interpretation indicators like tempo and dynamics changes, et cetera. While some of these could be deduced from the note information (dynamic changes and tempo), some can be extracted using other MIR techniques. For example, beat and meter using beat-tracking [94], or key using key-estimation [48]. When ignoring this additional meta-information, the task should be correctly denoted *note detection*, but is often sloppily referred to as transcription in the literature, since it is an easy and more common way to clarify the goal of the task.

Arguably, transcribing unpitched percussive instruments is a comparably easy sub-task of AMT. This is due to the fact that with these

types of instruments the offset time is virtually at the same time as the onset, followed by a longer release phase, which is usually not of much relevance concerning the transcript. Attack and decay contain most of the required information for note detection and classification in these cases. Compare Figure 2.2, which displays spectrograms of snare drum hits and a piano note. Most of the works on ADT additionally focused only on drum instruments used in western music. These cover real drums like bass drum and tom toms; cymbals like crash, ride, and hi-hat; and other instruments like cowbells, shakers, etc.; and are often collectively referred to as a *drum kit*. Figure 2.3 visualizes a typical drum kit consisting of a subset of possible instruments.

While detecting and classifying unpitched percussive instrument onsets might be simpler than doing so for harmonic instruments, the ADT task involves other challenges. Compared to harmonic instruments, where timbral properties of *good* sounding instruments are usually pretty narrowly defined, certain drum instrument types exhibit large variations of timbre and pseudo-pitch across different drum kits. For example, the bass drum of a certain drum kit might resonate at a higher pseudo-pitch than the low tom of another kit. Similarly, different cymbal types might be hard to distinguish between certain drum kits; i.e. for a drum kit cymbals are usually chosen in a way that they sound distinguishable, but this is not necessarily true across different drum kits. A special challenge of ADT is thus, that instruments are often classified by function/task within the arrangement rather than actual instrument sound: e.g. a drum playing the backbeat in a song might be notated as snare drum by a human transcriber, even though it does not sound like a snare drum. Similarly, the lowest pitched drum accentuating downbeats will usually be notated as the bass drum, regardless of the actual absolute pseudo-pitch. Examples for this are quite common, and can be found in public drum transcription datasets, e.g. when listening to the bass drums of drummer 1 and drummer 2 in the ENST-Drums[1] dataset.

Many different publications proposing methods for ADT have been published in recent years. However, at the time of writing this work, the task is not satisfactory solved, even for publicly available datasets.

2.1.1 *Overview of ADT Approaches*

First efforts to develop methods for ADT can be found as early as in 1985 [68]. With the rise of MIR, many different approaches were presented in the early 2000s, primarily focusing on drum note detection from drum solo tracks. An attempt to summarize early methods was made by FitzGerald and Paulus in 2006 [23], categorizing methods into either pattern-recognition-based or separation-based approaches.

1 http://www.tsi.telecom-paristech.fr/aao/en/2010/02/19/enst-drums-an-extensive-audio-visual-database-for-drum-signals-processing/

Pattern-recognition-based approaches often use machine learning to classify snippets of spectrograms. Nevertheless, other methods exist which e.g. use bandpass filters to perform the classification [44, 91]. Gillet [25] uses an onset detection detection method based on sub-band decomposition to segment the spectrogram into drum notes, and uses support vector machine (SVM)s and hidden Markov model (HMM)s to classify the drum notes into eight categories (bass drum, snare drum, hi-hat, clap, cymbal, rim shot, tom tom, and other percussions). Herrera et al. [32] compare and evaluate different classification approaches on different sound sample datasets. One of the datasets consists of drum sound samples categorized into 21 classes.

Separation-based approaches use source separation algorithms to identify the single drum instruments in mixed signals. In 2003 Fitzgerald et al. [21, 22] introduce prior subspace analysis (PSA), an independent component analysis (ICA) variant leveraging prior knowledge about the audio signal to be separated, and demonstrate the application for drum transcription on solo drum tracks as well as on polyphonic music. Smaragdis [76] introduces an extension to non-negative matrix factorization (NMF) [53, 62], the so-called non-negative matrix factor deconvolution (NMFD) method (sometimes sloppily referred to as convolutional NMF or CNMF c.f. Section 2.1.2.1).

In 2008, Gillet and Richard [26] added a third category to the ADT method classification scheme introduced in [23], and also propose new naming conventions: *(i)* segment and classify (pattern recognition), *(ii)* separate and detect (separation-based), and *(iii)* match and adapt. This list was further extended by Paulus in 2009 [64], adding a category for HMM-based system, noting that these do not fit in any of the existing categories.

The *segment and classify* category is equivalent to the previously introduced pattern recognition category. An representative of this class from this era is a method introduced by Gillet and Richard [26]: First, the drum track of a polyphonic audio mixture is enhanced, using an ICA-based approach. Then, onset detection is performed and a SVM classifier is used to classify the instrument onsets into three drum classes (bass drum, snare drum, and hi-hat). Miron et al. [57] use frequency filters for preprocessing, and subsequently perform onset detection and feature extraction to classify the drum onsets using a k-nearest neighbor (KNN) classifier to detect drum notes in solo drum audio signals, in real-time.

The *separate and detect* category represents methods formerly grouped as separation-based methods. In 2010, Spich et al. [81] build on the approach introduced in [22] and extend it by adding a statistical music language model. Dittmar and Gärtner [14] introduce a method based on an NMF extension, building a real-time ADT system for solo drum tracks, using prior knowledge of the used drum sounds.

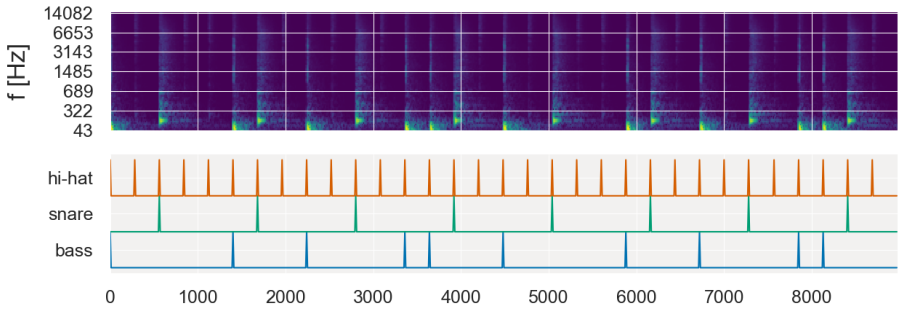


Figure 2.4: Example of an activation function for three drum instruments alongside the spectrogram of the audio.

The NMF-based method by Dittmar and Gärtner [14] could also be categorized into the *match and adapt* category, since the initially extracted templates are continuously adapted. Another representative of this category is the method introduced by Yoshii et al. [109] in 2004 and further extended in 2007 [110]. It represents an ADT system based on template matching and adaptation, similar to sparse coding approaches, which can also be interpreted as a variant of NMFD.

Formerly, HMM-based approaches have been categorized into the *segment and classify* category, nevertheless, their lack of the crucial onset detection step justifies a separate category. An example of such an approach was published in 2009 by Paulus and Klapuri [64]. The method utilizes HMMs to model the sequence of mel-frequency cepstral coefficient (MFCC)s of drum tracks. Then the state transitions of the HMMs are used to identify the individual drum note onsets.

The past continuous adaptations of categorization schemes for ADT approaches made clear that a more flexible and extendible categorization system would be needed to avoid further adaptations. Such a system was introduced by Wu et al. in 2018 [105], proposing a modular kit of components and nomenclature to classify ADT methods. The proposed components are: *(i)* feature representation (FR), *(ii)* feature transformation (FT), *(iii)* event segmentation (ES), *(iv)* event classification (EC), *(v)* activation function (AF), and *(vi)* language model (LM). These components represent processing steps in pipelines of method classes. Different methods can then be assigned into classes, constructed from these components. In [105] four classes are proposed: *(i)* segmentation-based (FR, ES, EC), *(ii)* classification-based (FR, ES, FT, EC), *(iii)* language model-based (FR, FT, LM), and *(iv)* activation-based (FR, AS, ES). Both the component list as well as the method classes are designed to be expandable, thus allowing classification of yet unknown future approaches.

2.1.2 *State of the Art Methods*

Current state-of-the-art ADT systems share certain commonalities: *(i)* they aim at detecting activation functions for each instrument under observation, *(ii)* using as little processing steps as possible (end-to-end), and *(iii)* they use gradient-descent-based numeric optimization to train the models on a set of diverse training data. Figure 2.4 shows an example of target activation functions for three drum instruments alongside the spectrogram, highlighting the onsets for each instrument. End-to-end systems tend to perform better than methods consisting of several processing steps, each adding a chance of error and dismissing information of the signal for the next stage. In other words: In a multi-staged systems every processing step can be seen as a link in a chain, and the chain is only as strong as its weakest link. With many steps in the chain, also the probability of a single step failing naturally increases.

Activation functions can be interpreted as pseudo probabilities for each time step indicating if an onset for the corresponding instrument occurred. This opens diverse possibilities of post processing, which can be a simple peak-picking algorithm, or more complex systems like *musical language models*, considering the global musical context of drum patterns within the song. However, in the spirit of real end-to-end learning, especially with (artificial) neural network (NN)-based systems the goal is that such an *musical language model* is implicitly learned during training.

In [105] an attempt at comparing state-of-the-art methods was made, showing that, depending on the application, both NMF and NN-based approaches have their strengths. Figure 2.5 compares the activation functions of an NMF and a NN-based ADT approach, applied on a simple drum only audio file. While the NMF approach used is rather basic, this example represents a best case scenario for this approach: The NMF basis vectors are perfectly fitted using the mean spectrogram of isolated drum hits of the single instruments used, and are fixed during optimization. For the NN-based approach, a simple CRNN trained on a different dataset was used. The CRNN represents a quite powerful NN model, but performs transcription on unseen data, while the used NMF model is quite basic (plain NMF with fixed bases), but overfitted on the data. The diagram can be seen as an example of how well correctly trained NN approaches are able to generalize, while producing almost perfect activation functions for simple data.

2.1.2.1 *Approaches Using Non-Negative Matrix Factorization*

In this section on overview of the working principles of NMF-based methods will be provided. The goal of NMF algorithms is to factorize a matrix into two smaller matrices using iterative numerical approximation. The factor matrices are constrained to have no negative

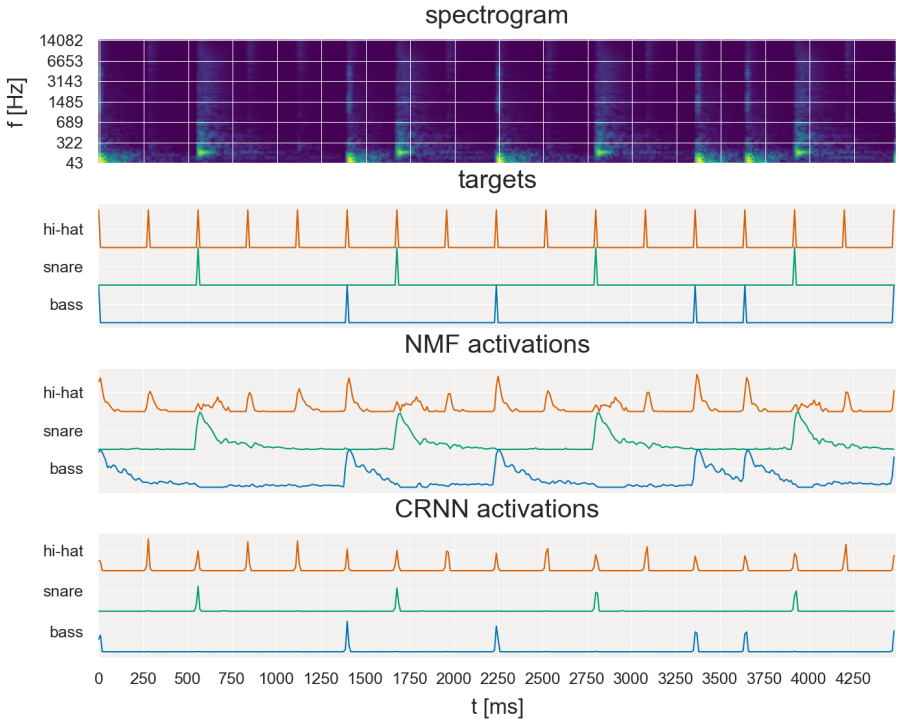


Figure 2.5: Example of activation functions extracted using an NMF and a CRNN-based ADT approach. The first image (top) represents the input spectrogram used, the second one displays the target activation function (i.e. ground truth), while the third and fourth plot show extracted activation functions using the NMF and CRNN transcription system, respectively. The used NMF system is a simple *fixed-bases* method as described in Section 2.1.2.1, while the CRNN system used is the on introduced in Chapter 6 [94].

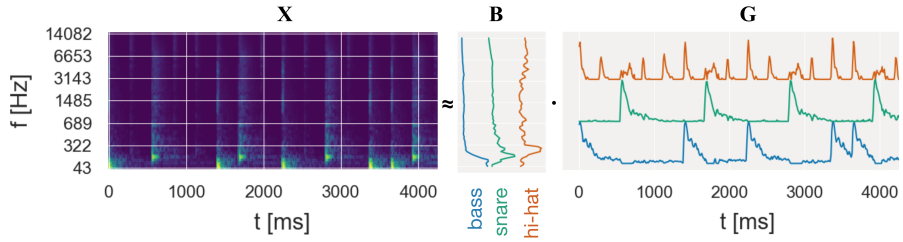


Figure 2.6: X, B, and G matrices after performing NMF on a simple drum loop. While X contains the input spectrogram, B represents the average spectral energy distribution for each instrument, and G displays the activation function for each instrument. Note the relative long tail of onset peaks in the activation function, caused by the sustain of the instrument.

elements (non-negativity), which makes sense for the application in signal and audio processing, when dealing with magnitude (or similar) spectrograms. In the context of ADT, NMF is usually performed on a spectrogram representation of the audio, and the two factorization products can be interpreted as spectral templates for each of the instruments under observation, and their activation functions. In this context, the spectrogram, spectral templates, and activation functions are represented by the matrices X, B, and G, respectively:

$$X \approx \hat{X} = B \cdot G\,, \tag{2.1}$$

where $\hat{X}$ is the approximation of X when using the current values for B and G. See Figure 2.6 in which the relation of the matrices and the arrangement of the data within these matrices is visualized. To adapt B and G to better fit X, usually an iterative algorithm is used. To this end a loss function $\mathcal{L}$ is defined. Often the generalized Kullback-Leibler (KL) Divergence [54] is used:

$$\mathcal{L} = D_{KL}(X|\hat{X}) = \sum \left(X \odot \log\left(\frac{X}{\hat{X}}\right) - X + \hat{X} \right)\,, \tag{2.2}$$

where $\odot$ denotes element-wise multiplication, and also the fraction is executed as an element-wise division of the two matrices. The sum is performed over the single elements of the resulting matrix. To adapt B and G, often stochastic gradient descent (SGD) is used. After calculating the partial derivatives for the loss function $\mathcal{L}$ with respect to B and G, the following update rules can be derived:

$$B \Leftarrow B \odot \frac{\frac{X}{\hat{X}} \cdot G^{\intercal}}{J \cdot G^{\intercal}} \tag{2.3}$$

$$G \Leftarrow G \odot \frac{B^{\intercal} \cdot \frac{X}{\hat{X}}}{B^{\intercal} \cdot J}\,, \tag{2.4}$$

where J is a matrix of ones of the same size as X. For a detailed derivation of these multiplicative update rules via the partial derivatives

of the loss, please consult the work of Juan José Burred [11], who provides an in depth discussion.

In practice, choosing the correct size of B (i.e. its rank), and its initialization are crucial. Plain NMF works best for drum solo recordings, thus choosing the rank of B to be the number of distinct drum instruments in the recording (e.g. 3 if transcribing bass drum, snare drum, and hi-hat). For initialization, the mean spectra for each drum instrument can be used, if available. If single hit training data is available, it can be used to extract the mean spectra for each drum instrument. For certain applications it can even make sense not to update the basis matrix B, which is called *fixed-bases* NMF. While this provides good results for scenarios where each drum hit sounds exactly the same (transcription of sampled or synthesized drum track), in most cases the pre-defined templates will not match the audio well and thus results will deteriorate. To counteract this issue, methods to add columns which can be adapted to B, and/or to make B semi-adaptive have been proposed (*partially-fixed* NMF [106] and *semi-adaptive* NMF [14]).

When using NMF the assumption is that a drum hit sound can be modeled using one spectrum with varying intensity over time—i.e. the relative distribution of energy in the spectrum for one drum onset does not change. This is, of course, a simplification. As already discussed earlier, in the attack phase of percussive signals, more broad band noise is present, which later changes to inharmonic oscillations. This simplification is a shortcoming which can be overcome in several ways, e.g. Battenberg et al. use different basis vectors for different phases of the notes [3]. Another way is to use *Non-Negative Matrix Factor Deconvolution*, where instead of a single column as basis vector, a whole matrix for each basis is used [52, 55, 67, 75]. Doing so, B becomes a 3-tensor P consisting of one matrix for each basis, and the matrix multiplication in $\hat{X} = B \cdot G$ becomes the sum of convolving each matrix in P over the corresponding row of G:

$$\hat{X} = \sum_m B_m * G_m \,. \tag{2.5}$$

In practice, the convolution is usually avoided by shifting the activations in G and performing the normal matrix multiplication step as with vanilla NMF for each frame of the basis vectors, and then summing over the result—c.f. approach in [75]. Using matrices as spectrogram templates for NMF allows a more detailed modeling of the time series for each onset, but this improvement comes at a higher computational cost for optimization.

2.1.2.2 *Approaches based on Neural Networks*

A detailed discussion of the working principles of NNs will follow in Section 2.3, and a detailed explanation of state-of-the-art NN ADT systems is the focus of Part ii of this thesis. Therefore, this section

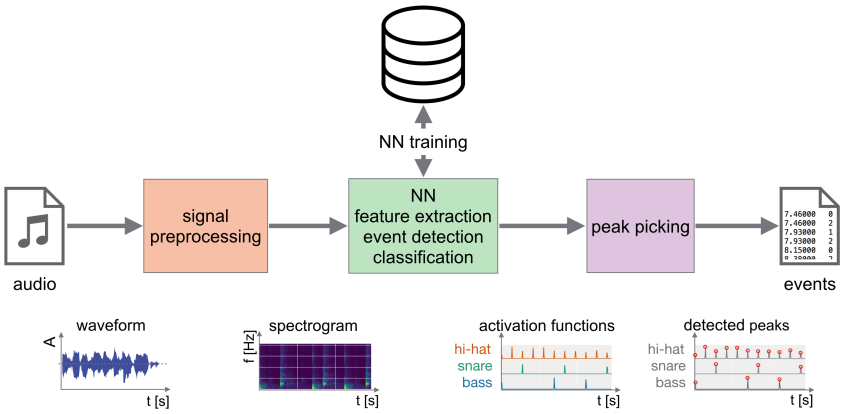


Figure 2.7: An outline of NN ADT systems. First, spectrogram frames are calculated from the audio source. The spectrogram frames are then used as input features for previously trained NNs. Finally, a peak-picking algorithm is used to determine the instrument onset locations using the instrument activation functions provided by the NNs.

will focus on giving a brief overview of the history of publications for NN-based ADT systems, and then explain how in principle neural networks as black box machine learning tools can be used for ADT.

After the success of artificial neural networks and the subsequent hype reached the MIR community, NNs were applied to many different MIR problems, e.g.: onset detection [70], piano transcription [9, 45], beat and downbeat tracking [7, 18, 19, 49], key estimation [48], singing voice detection [71], et cetera.

The first works focusing on ADT were published in 2016 [78, 92] and use recurrent neural network (RNN)s to extract activation functions for three different drum instruments (bass drum, snare drum, and hi-hat) from spectrogram representations of drum tracks. Subsequent works in 2017 use different training techniques [93], architectures [79, 94], and training paradigms [94] to further increase performance and generalization capabilities. Further work focused on enriching the transcript with more drum instruments [12, 101], and additional meta information [94].

Neural networks are still a young technology in ADT, nevertheless these systems seem to have the capability to outperform traditional systems [95, 105] (also see MIREX drum transcription task results[2]).

If a NN should be used in an activation-function-based ADT system, the neurons of the output layer are responsible for generating activation functions for assigned drum instruments. Figure 2.7 provides a schematic over of such a system. To this date, several NN-based ADT systems have been proposed [78, 79, 92–94, 101]. All of these systems use a similar processing pipeline, with the main difference being if a

2 http://www.music-ir.org/mirex/wiki/2017:Drum_Transcription_Results

single [12, 92–94, 101] or multiple networks [78, 79] are used to extract the activation function for the instruments under observation.

The extraction of the activation function using NNs is usually designed as a frame-wise logistic regression for each instrument. This is realized using a spectrogram representation of the audio as input for the network and binary cross-entropy as loss function. As target functions for training, ground-truth activation functions are created using the annotations of the training data: For each of the drum instruments under observation, frames are either labeled as not containing an onset (0) or containing an onset (1). The trained NN should then produce similar activation functions when provided with the spectrogram input data of unseen audio files. As with all activation-function-based methods, a peak picking algorithm is required to determine the onset locations of the drum instruments. Experience shows that these activation functions can be trained to be very spiky, which leads to peak picking being a relatively easy task, especially when compared to NMF-based methods—also see Figure 2.5 for a comparison of typical NMF and NN activation functions.

2.1.3 *Evaluation*

In the simplest form, drum note detections consist of only the (absolute) onset time and instrument class. A more detailed transcript would additionally contain onset intensity or loudness, as well as relative position to the rhythmic grid or bar. However, most ADT systems only focus on absolute time and instrument class. In this section, metrics usually used for evaluation of drum transcription systems are discussed.

In the case of systems focusing on onset time and instrument class only, evaluating ADT systems is almost the same as evaluating onset detection systems, with the only difference that for one music track, multiple onset categories (one for each drum instrument under observation), must be evaluated. For onset detection, first the number of true positive (TP) onsets tp, false positives (FP) fp, and false negatives (FN) fn is determined by comparing detected and annotated onsets using an onset time tolerance window of a certain size. Using these values, precision P, recall R (also called sensitivity), and F-measure F (harmonic mean of precision and recall) can be calculated:

$$P = \frac{tp}{tp + fp} \tag{2.6}$$

$$R = \frac{tp}{tp + fn} \tag{2.7}$$

$$F = 2 \cdot \frac{P \cdot R}{P + R}\,. \tag{2.8}$$

Figure 2.8 shows the relationship between these measures and visualizes their calculation.

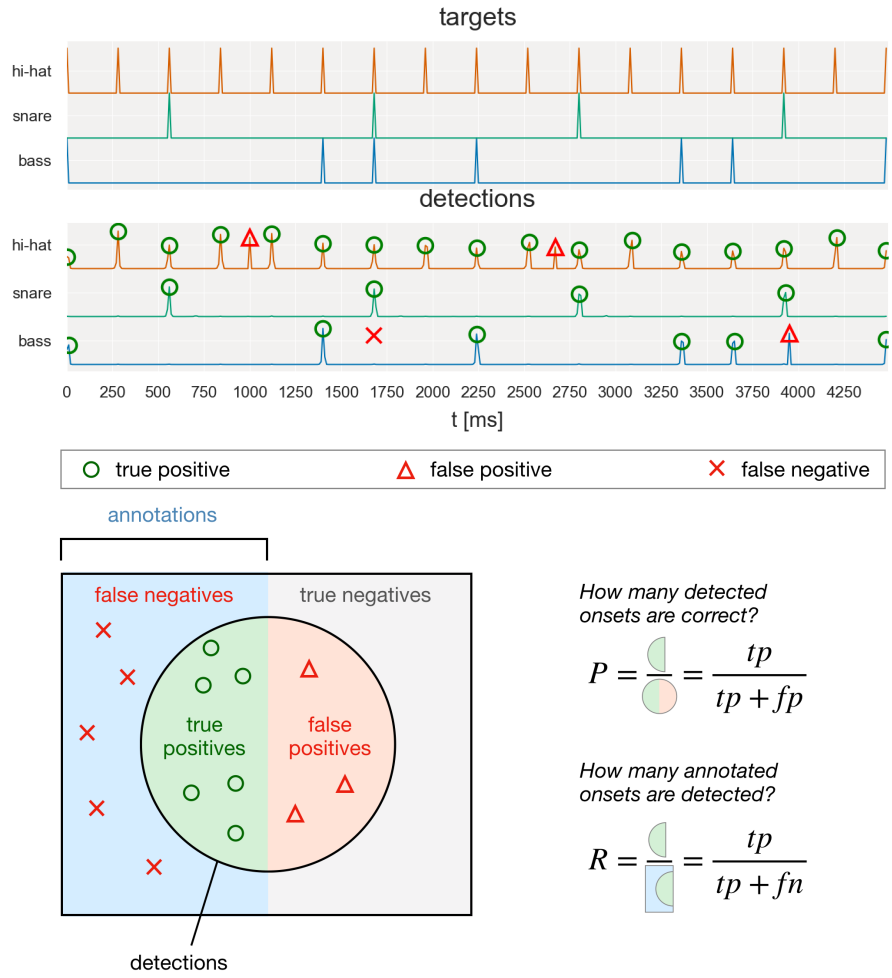


Figure 2.8: Calculating evaluation metrics for a drum loop. The top plot shows the annotated activation function. The second plot shows the predictions of the transcription algorithm alongside detected peaks and classification into true positives, false positives, and false negatives. Note that true negatives are represented by all other frames of the activation functions. The diagram in the bottom area visualizes the relationship between the measures used for evaluation. C.f. precision and recall on Wikipedia [104].

To obtain an overall result for one track over the individual instruments, two strategies can be used: *(i)* calculate a mean value for precision, recall, and F-measure values, or *(ii)* count TP, FP, and FN values jointly over all instruments. These two approaches will be referred to as *mean* and *sum* evaluation for the rest of this work. If the evaluation set contains more than just one audio track, an overall value for all audio tracks in the evaluation set must be calculated. Again, either the *sum* or *mean* approach can be used for this. Note that *sum* over all tracks using *mean* over all instruments is not possible. While it would be possible to use *mean* for instruments and *sum* for tracks, it is not common. Usually either *sum* or *mean* is used for both instruments and tracks. While in most cases the resulting values will be very similar, both approaches behave differently for certain extreme conditions. While *mean* is more sensitive to a low performance on individual sub-results (tracks and instruments) even if they contain only few onsets (relative underrepresented classes), *sum* is more robust in terms of very sparse sub-results (i.e. tracks or instruments with no, or only very few onsets). E.g. if an instrument is not present in most tracks, the F-measure for the instrument in the cases where no onsets are annotated will be 1.0 if no detections are predicted. For the whole dataset the *mean* F-measure result for this instrument might now be close to 1.0, even if the only onsets in the whole dataset were missed (FN). In contrast the *sum* evaluation for this instrument will yield a value close to, or exactly 0.0, if this was the case. On the other hand, if a instrument in the dataset only contributes a small fraction of onsets, its F-measure value will not influence the overall result much when using *sum* evaluation, whereas *mean* evaluation will yield a more defensive overall value, if certain instruments or tracks perform comparably worse, even if the number of onsets is relatively small. Each evaluation strategy highlights slightly different aspects, thus it often makes sense to include them both and discuss whenever there are large discrepancies.

To count TP, FP, and FN values, usually an onset time tolerance window of a certain width is used. The size of the tolerance window has an impact on the results: the larger the tolerance window the easier it is to achieve high F-measure values. Desirable is thus a tolerance window with a size as small as possible. This usually depends on two factors: *(i)* temporal resolution of the drum transcription method, and *(ii)* temporal accuracy of the annotations. If the method only provides a low temporal resolution for transcribed drum onsets, the tolerance window needs to be at least the same size as a discrete time step of the transcription methods time resolution. In practice a larger value is reasonable (e.g. two times the size). The second aspect is the temporal accuracy of annotations. If annotations are only provided with an accuracy of $\pm T_a$, the tolerance window must be at least of the same size, otherwise even given a perfect transcription

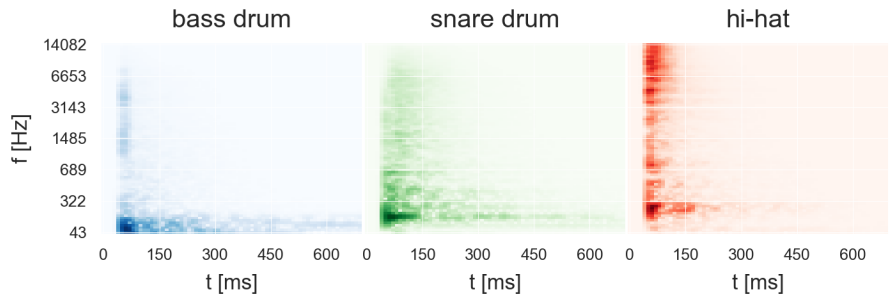


Figure 2.9: Example spectrograms for bass drum (left), snare drum (middle), and hi-hat (right).

system correctly detected onsets might not match the annotations provided during evaluation. Again, in practice this value must be larger for machine learning approaches, since annotations with a large tolerance introduce a lot of noise during training, and thus the resulting transcripts may display even larger deviations. In the literature different sizes of onset time tolerance windows are common, ranging from 50ms down to 20ms. Arguably, since drum instruments are used to accentuate rhythms, and their broadband energy at onset time makes it easy for the human ear to identify the exact temporal location, a relatively low tolerance window is preferable. Depending on the annotation quality, tolerance windows as low as 20ms can make sense. This is due to the fact, that the lower bound of the time interval for which a human ear is capable of discerning distinct onsets is in the order of magnitude of 10ms [5].

2.1.4 *Challenges and Open Research Questions*

While in the early years, works on ADT were focusing on different subsets and groupings of drum instruments, it quickly became clear that due to a lack of high quality datasets, as wells as certain signal properties, a focus on three most common drum kit instruments was reasonable. These instruments are bass drum, snare drum, and hi-hat, which usually shape the basic rhythmic patterns and are statistically most represented in the available datasets of popular western music. Furthermore, because of the timbral properties of these instruments, they are well separable considering their spectral energy distribution: The bass drum generates a broadband-noise shortly after the onset, and low-frequency resonances can be observed after that. The spectrogram of the snare drum displays a similar broadband impulse after the onset, and mid-frequency resonances as well as mid-frequency noise, later. The hi-hat generally produces higher frequency noise and resonance contents. Compare Figure 2.9, which shows separate spectra for these instruments. This distribution explains the relative success of simple methods using only band-pass filtering for signal separa-

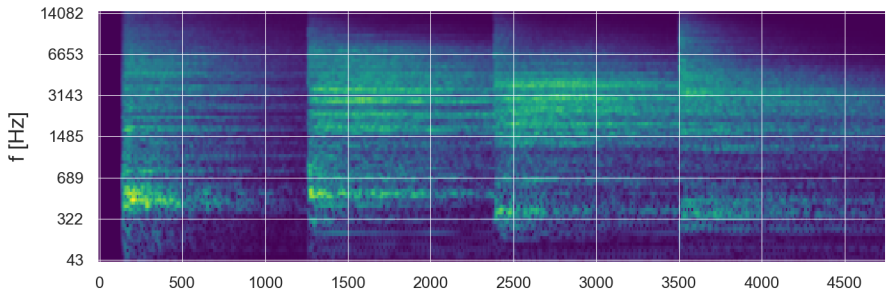


Figure 2.10: Spectrograms of an open hi-hat, two different crash cymbals, and a crashed ride cymbal. While different resonant frequencies are identifiable e.g. for the two crash cymbals (middle two onsets) the overall shape is very similar, since the timbre and behavior of these instruments is also quite similar.

tion [44, 91]. These properties are also related to the issues usually encountered when working with more than only these three drum instruments: Other drum instruments often sound very similar (cymbals and hi-hat), and/or their resonance frequencies are not uniquely assignable to a certain instrument (e.g. a mid tom in one track might be higher than the high tom in another track). Figure 2.10 shows a spectrogram of open hi-hat, two crash cymbals, and a crashed ride cymbal, visualizing the similarity between those, in contrast to the different spectra in Figure 2.9. Additionally, the relatively low number of occurrence for certain instruments in common datasets leads to many problems during training and evaluation [12, 101].

Another open problem for ADT is extracting relative loudness of drum events. Hawthorne et al. [31] present an interesting approach for piano transcription, using multi-task learning for framewise note detection, onset and offset detection, as well as velocity extraction. A similar approach could be applicable for drums, using extra outputs for velocity curves for each drum instrument. However, obtaining sufficient training data for this approach remains a challenge. While it would be possible to use artificial, generated data, in the case of real world recordings it is almost impossible to manually create these annotations in sufficient quantities.

An additional aspect of transcribing drum instrument events is the variety of playing techniques that can be employed on the individual instruments. These playing techniques comprise different aspects: First, drums in western music are usually struck using drum sticks, but other mallet/beater types for striking are also quite common: e.g. rutes/rods, brushes, soft mallets, or even bare hands. Furthermore, diverse striking techniques are commonly used on the different instruments, e.g.: single and double strokes, rolls, drags, and flams can be used on almost all instruments; rim-shot, cross or side stick on drums; edge, bell, and bow strikes, crashing and choking on cymbals; different hi-hat opening positions and techniques; et cetera. All these can have a significant

impact on the produced sound, and are relevant in terms of notations for a final transcript. With this wide variety, and the sparseness with which these techniques are usually used, all the problems that exist for the standard stroke drum transcription task are exponentiated. Several attempts to deal with well defined tasks using only a small subset of these techniques have been made in the past. Tindale et al. [88] investigated classification of snare drum techniques like rimshots considering also stroke locations on the drumhead. In 2011, Hochenbaum and Kapur [34] investigate methods to identify if a stroke was played using the left or right hand ultimately using additional accelerometer data. Prockup et al. [65] introduce a dataset consisting of drum hits on three instruments (snare, high and low tom) using different intensities and playing techniques, and evaluate different features using a SVM classifier. Souza et al. [80] introduce a dataset containing sound samples of different cymbal playing techniques and evaluate different features using SVM classifiers, in a similar fashion. In their 2016 publication, Wu et al. [107] present an approach to tackle drum instrument playing technique identification during transcription from polyphonic music. Classifying playing techniques is a difficult task with many aspects, and the question remains how to model the many parameters that this task brings, effectively.

In general, the need for large amounts of high quality, manually annotated training data is a limiting factor for data-driven machine learning methods, which seem to be the most successful in this context. The annotation's accuracy is also a limiting factor for high temporal resolution and accuracy which is desirable, especially in the context of drums and rhythm. Manual annotation is very time consuming, not only but also because high temporal accuracy is required. Generating a complete transcript for a five minute long track, including all used drum instruments and considering playing technique and dynamics information, requires many hours up to days of work and can only be performed by a trained and experienced person. Using tools like drum-triggers or accelerometers [34] can help with creating annotated datasets. However, not all required nuances can be captured using these primitive sensors. Generating synthetic data using drum samplers and synthesizer is often the only feasible option if only limited resources are available. However, these datasets always bear the risk of resulting in models which do not generalize well on real, recorded drum tracks.

2.2 DRUM PATTERN GENERATION

With a shift of studio technology towards fully digital systems running on computers, many production and recording technologies changed. Due to greatly lowering the costs of high quality equipment, this digitalization also enabled non-professionals to build home recording

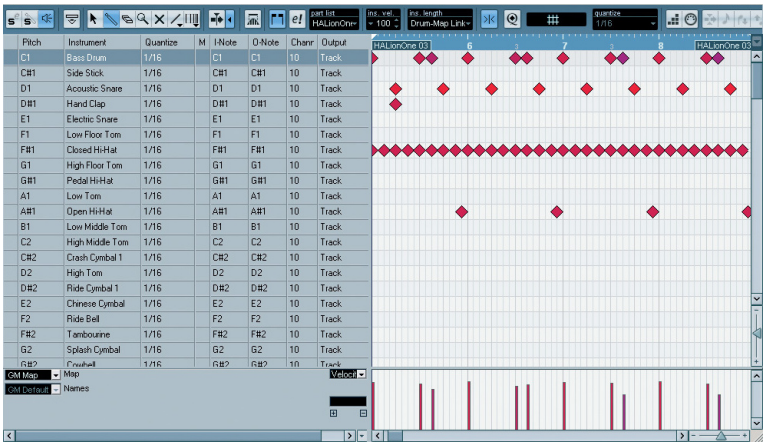


Figure 2.11: A screenshot showing a drum track manually set in a drum roll editor within the Steinberg Cubase† DAW .

† https://www.steinberg.net/en/products/cubase/

and production studios. Especially for electronic dance music (EDM), fully digital studios, using only synthesizers, samplers, and snippets of audio recordings are, quite common. However, even for pop/rock genres, replacing much of the traditional equipment with digital versions is quite common, e.g. guitar amplifiers, pianos, effects and sound processing like equalizers. Even using sampled drums for non-EDM genres, especially in semi-professional settings, has become quite common. This is due to the fact that recording drums requires expensive equipment (studio drum kit, multiple microphones) which is notoriously difficult to set up, and the recording process can be very time consuming, also depending on the capabilities of the musician.

An approach often used to create drum tracks is to enter drum patterns using a piano-roll-like editor within a DAW. Figure 2.11 shows a typical drum roll editor for editing and creating a drum track. Other typical input modalities for drums include step-sequencer interfaces, pads, or e-drums. While a step sequencer is very similar to drum roll editors, it is not as flexible but has certain advantages especially for live setups. Playing drums on pads using a technique called *finger drumming* requires practice and is thus not as accessible as other input methods. This is even more the case with using e-drums, which additionally require more space and are more costly. Figure 2.12 shows examples of a hardware step sequencer, drum programming pads, and an e-drum kit. Because of its flexibility and ease of use, programming drums in a drum roll editor is one of the most widespread techniques. However, using this technique to create synthetic drum tracks can be quite time consuming, while much of the work involves repetitive tasks. Due to this, producers of DAWs have started adding libraries of drum patterns to their software products. These come in two main variants, providing different output formats, which can be either audio or discrete notes (e.g. MIDI tracks). Both have their advantages: While

Figure 2.12: Examples of a hardware step sequencer (a) (Roland TR808), hardware MIDI controller featuring pads for finger drumming (b) (Native Instruments Maschine†), and an e-drum kit (c) (Roland V-Drums‡).

†https://www.native-instruments.com/en/products/maschine/production-systems/maschine/

‡https://www.roland.com/global/categories/drums_percussion/v-drums_kits/

audio drum loops are an easy-to-use shelf product, they lack flexibility. With symbolic drum loops and patterns, it is easier to change tempo, make small modifications, and choose the desired sound by feeding the patterns into drum samples or synthesizers. While these approaches speed up production, using predefined drum loops and patterns has two downsides. Foremost, using a commercial pattern library bears the risk of sounding unoriginal since many other artists might use the same samples. This can partly be overcome by increasing the library size. Doing so leads to another problem: finding and selecting patterns or loops from large sample libraries can be equally time consuming and frustrating. Figure 2.13 shows a screenshot of the UI for Native Instruments' Maschine[3], featuring an arranger, a drum roll editor, and a browser window for instruments, sounds, and drum patterns. The browser features a category selection element, and a simple list where fitting results are presented.

To improve this situation and circumvent problems of browsing-based approaches, more sophisticated supportive tools have been developed in recent years. For unorganized libraries, automatic categorization and recommender systems can improve accessibility for both audio loops and symbolic drum patterns. A completely different approach is to use generative methods creating patterns or audio loops according to certain parameters. While generative methods offer solutions to many problems of pattern-library-based tools, they bring certain problems on their own. A main challenge of generative methods is to generate patterns that match the provided parameters, sound original, but also conform to certain expectations.

3 https://www.native-instruments.com/en/products/maschine/production-systems/maschine/

Figure 2.13: Software UI of Native Instrument Maschine. On the left side in the browser panel the user can select instruments, sounds, and patterns. In the top portion the arrangement is visualized, while in the bottom portion a drum roll editor displays the currently selected element.

2.2.1 *Overview of Drum Pattern Generation Methods*

In the context of pattern generation and variation, the question if a generated pattern is suitable is of central interest. Automatic evaluation of such systems is often difficult, because this question is hard to answer and often depends on many latent variables. However, for symbolic data, attempts to calculate similarity and quality of rhythmic patterns have been made. In his 2004 publication, Toussaint [89] discusses similarity measures for rhythmic patterns. The measures investigated are: Hamming distance, edit distance, Euclidean distance of inter-onset-interval vectors, and the interval-ratio distance. To compare these measures, *phylogenetic trees* based on the computed distance matrices—a visualization tool borrowed from biology—are built and compared. In 2010, Toussaint [90] investigates and proposes properties of "good" rhythms, and additionally introduces algorithmic approaches to create rhythm patterns. Calculating rhythmic similarity for audio is more difficult, since either a transcription step is necessary, or relevant features have to be extracted from the audio first. For the case of using audio as source material, Holzapfel and Stylianou [36] propose a tempo invariant rhythmic similarity measure utilizing the scale transform. Similarly, Jensen et al. [41] as well as Gruhne and Dittmar [29] approach the goal of extracting tempo invariant rhythmic

features by using logarithmic autocorrelation functions calculated on different onset density functions.

One of the earliest approaches to drum pattern generation was published in 1994 by Horowitz [37], using a genetic algorithm (GA). Since GAs represent an important group of methods to generate drum patterns besides the techniques used in this thesis, Section 2.2.1.1 will provide a brief introduction to GAs. Kaliakatsos-Papakostas et al. [42] also use a GA to build an automatic drum rhythm generation tool. They do so by defining an L-system, a rule-based approach to generate fractal patterns, for drum patterns. Using a GA to evolve the grammar, a set of five rhythmic indicators is used to measure fitness. The system can generate rhythm patterns controlled by the rhythmic indicators *density*, *pauses*, *self-similarity*, *symmetry*, *syncopation*. In their follow-up work [43] they switch to a matrix representation of the drum patterns which are directly evolved using a GA and similar fitness functions based on 40 feature values. The system is designed to generate variations while allowing interactive control of the divergence between a seed pattern and the generated ones. In 2015, Ó Nuanáin et al. [60] propose a similar rhythm pattern variation system based on GAs. The focus of this work is the comparison of two fitness functions based on different distance functions for rhythm patterns. A GA approach that uses a dataset to check the fitness of current generations was presented by Bernardes et al. [6].

An advantage of genetic algorithms is that they only rely on a fitness function. Another option is to use supervised machine learning (ML) approaches to train a pattern generator. However, these usually require large amounts of training data. In 2007, Paiement et al. [63] propose to use a probabilistic model fitted on rhythm pattern subsequence distances. With the help of this probabilistic model, continuations of rhythm patterns are generated, utilizing a HMM. In a recent work, Huang et al. [38] demonstrate the power of RNN-based systems using self-attention, to generate symbolic music. A similar approach could be applied to drum tracks.

Another class of probabilistic models which can be used in a generative way are Restricted Boltzmann Machines (RBM). Section 2.3.6 will provide an introduction to RBMs Boulanger-Lewandowski et al. [10] use a recurrent extension of RBMs to model and generate polyphonic music. Battenberg and Wessel [4] use a conditional RBM, to extract meter information from rhythm patterns. In this work, the authors mention the capability of the model to generate drum patterns, when provided with seed patterns. The works which make up the rhythm pattern variation part of this thesis take up this idea [97, 98, 100] to build drum rhythm variation systems based on RBMs. Evaluation of the variation systems is done by embedding them in a step sequencer interface [97] using a touch interface [98], and performing a qualitative evaluation as well as a quantitative study using user surveys [100].

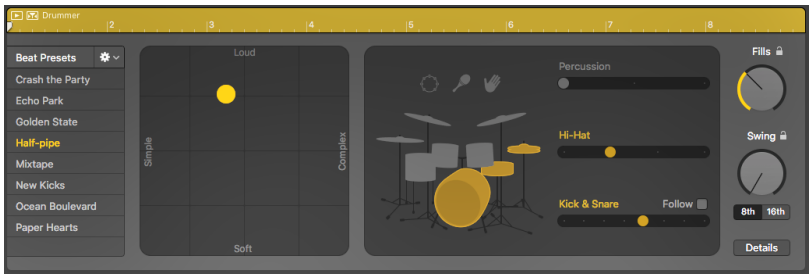


Figure 2.14: Screenshot of the *Drummer* plugin in commercial DAW Apple Logic Pro X. All rights by Apple Inc.

A relatively new technology primarily aimed at generative tasks are generative adversarial networks (GAN)—see Section 2.3.7. One of the first approaches of using GANs to create symbolic music has been introduced by Yang et al. [108]. They use a convolutional architecture trained on MIDI data to develop a model that generates symbolic notes. A similar work by Dong et al. [16] is trained to synchronously generate five tracks of symbolic notes for bass, drums, guitar, strings, and piano. In [20], we use a controllable GAN based on a convolutional-recurrent architecture to directly generate symbolic drum patterns of different lengths.

Publications dealing with generating audio drum patterns directly, have emerged only very recently. An early related approach was proposed by Ravelli et al. [66], who present a system performing adaptation of audio loops, based on rhythmic information extracted from other audio loops. A similar system could be used to transform symbolic patterns into audio, using a source audio loop, providing the sound. More recent works mostly use GANs to generate audio, e.g. Donahue [15] introduce a method called *WaveGAN* which is used to generate speech, as well as drum and piano sequences. Krishnamurthy [50] created a demo application of *WaveGAN* aiming to generate audio drum loops, called *BeatGAN*. Directly creating audio is an interesting approach, and results are often of surprising quality. Nevertheless, generating symbolic music is far from becoming obsolete since common tools and workflows used by producers and musicians often build on symbolic music.

In current commercial products which provide automatic drum loop generation, the predominant approach is to generate symbolic tracks. For example, the Logic Pro X DAW[4] by Apple provides a *Drummer* plugin, which generates a drum track for a given style. Figure 2.14 shows the user interface to control the *Drummer* plugin. The complexity and loudness can be controlled dynamically in realtime or using automation. It is not documented if for this plugin the patterns are generated or are selected from a library, but it appears that an internal pattern database is used.

4 http://www.apple.com/logic-pro/

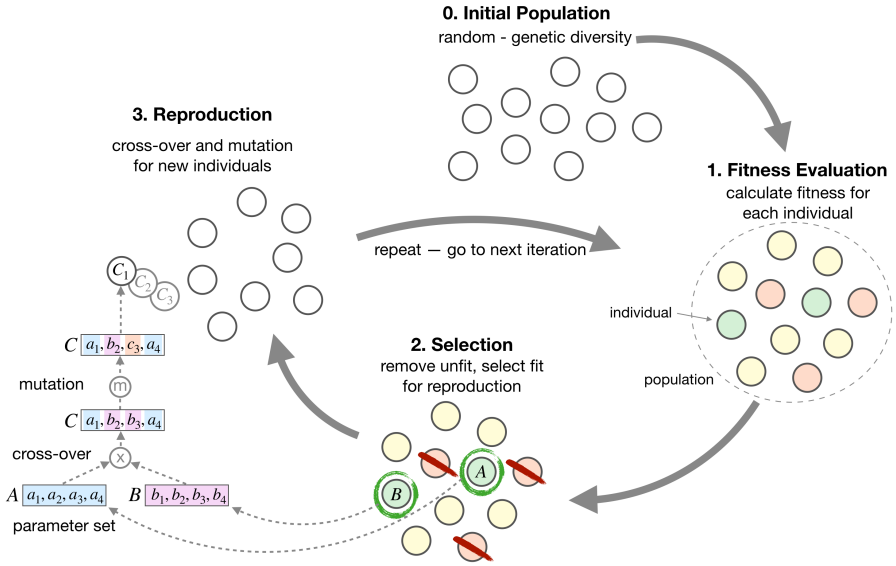


Figure 2.15: Overview of the iterative optimization cycle for a genetic algorithm.

The working principles of RBMs and GANs, which represent the technology behind most current state-of-the-art drum pattern generation methods, will be explained in Section 2.3.6 and Section 2.3.7, respectively. Details on the application of these methods for drum pattern generation are the focus of the second main part of this thesis.

2.2.1.1 *Genetic Algorithms*

The basic idea of a GA, is to mimic the real world natural selection and evolution of living organisms based on deoxyribonucleic acid (DNA), which encodes the building plan of living organisms. To do so, a *population* of *individuals*, representing solutions for the optimization problem to be solved, is evolved to find better solutions. An individual is represented by a parameter set defining a candidate solution. During one epoch every individual is evaluated using a *fitness function*. Individuals with a low fitness are removed, while individuals with high fitness are kept in the population and used for reproduction. In the reproduction step, the parameter sets from two individuals are combined by using *cross-over* and *mutation*. Cross-over consists of creating a new parameter set by combining a randomly selected subset of parameters from individual A, and the disjunct set of parameters from individual B. Mutation allows for random alteration of the parameter set of a new individual, given a certain mutation rate or probability. The goal of cross-over is to create new individual that combine strengths of the *parent* individuals, while mutation alongside the diversity of the population is responsible for sufficient exploration of the parameter space. These steps of a training epoch or generation are iterated until a satisfying solution is found, or no further improvement

can be achieved. The whole process is mimicking how natural selection (survival of the fittest) is driving biological evolution in nature. Figure 2.15 provides an overview of a typical iterative optimization using a GA. To be able to use a GA for optimization, two things must be modeled: *(i)* the representation of a solution by the parameter set of an individual, and *(ii)* a fitness function that provides a heuristic for the quality or performance of an individual.

2.2.2 *Challenges and Open Research Questions*

The question, whether a rhythm pattern is good or not is per se ambiguous. Evaluation is generally a major challenge in the context of generative methods. A simple approach is to use similarity measures to calculate the distance to provided examples [42, 43, 60, 89]. Efforts have been made to study properties of good rhythm patterns [90], however, a way to generalize these approaches to a larger variety of music genres is desirable. When working with probabilistic models, tests comparing feature distributions of training data and generated samples can be useful. However, conclusions are only meaningful if a reasonable feature space is used [85]. The challenge in this case is the same as when using distance metrics. Often, performing user studies is the most direct and effective way to evaluate generative systems [98, 100]. However, performing user studies is usually laborious while careful selection of the participants is crucial.

Many hidden parameters for successfully creating meaningful and appropriate drum patterns depend on the musical context. Experienced musicians are able to implicitly consider those, e.g.: genre of the music, general mood and tempo of the piece, available instruments and their sound, et cetera. In addition, conscious decisions expressing artistic intent have to be made: create or release tension, thematically connect or separate different parts of a piece, pay homage to other pieces et cetera. As of now, very few of these parameters are explicitly captured by generative methods and used to control generation of patterns. A main challenge is to find a set of parameters which provide sufficient flexibility and allow room for sufficient artistic expression while not overly complicating a potential interface.

It is to be discussed how far certain parameters should be detected automatically by such a system, and in which context. E.g. in a production environment, musicians and producer may not want to use a system that makes decisions which have a strong impact on the final song on its own, since that might take away the creative responsibility from the artist. On the other hand, in an experimental music setup, or as a proof of concept, a fully automatic drummer, using many different input modalities to decide what to play, might be desirable. The choice of directly generating audio or symbolic notes also depends on the context and field of application. As mentioned earlier, in a production

environment, generating symbolic notes, which can easily be further manipulated, is more common than generative methods producing audio. On the other hand, if style transfer systems can be used to successfully transfer sound and feel of preexisting drum loops, such a system might make sense in a more sampling focused production.

2.3 DEEP LEARNING BASICS

Most of the publications which form the basis for this manuscript are based on deep learning. To provide a basic understanding of the concepts and techniques referred to and used throughout the thesis, a compact introduction to deep learning will be provided in the next section. Note that the focus will be to give a basic understanding and an overview. Further reading with more details will be provided through links and references, since exhaustive coverage and in-depth mathematical discussion for all covered topics is not possible in this format.

The expression *deep learning* is a term used to refer to a family of machine learning methods. They are usually focused on un-, semi-, or supervised data driven optimization of hierarchical models which are structured into layers. These models are usually called (deep) artificial neural networks.

2.3.1 *Neural Networks*

A NN can be understood as a network of weighted, directed connections between nodes which represent artificial neurons. Along these connections, values are synchronously propagated through the network in discrete time steps. At a neuron, the weighted values of multiple incoming connections are summed and an activation function ϕ is applied to calculate the output of such a neuron.

$$h = \phi(b + \sum_{i=0}^{N_i} x_i \cdot w_i)\,, \tag{2.9}$$

where h represents the output value (also: activation or hidden state) of the neuron, b denotes a bias value (wighted connection from constant value 1), N_i number of inputs, and w_i the weighting associated with incoming value x_i.

The activation function used in neurons can also be called *nonlinearity*. This is due to the fact that only nonlinear activation functions enable the network to learn solutions for nontrivial problems. The activation function represents an important hyperparameter for NN

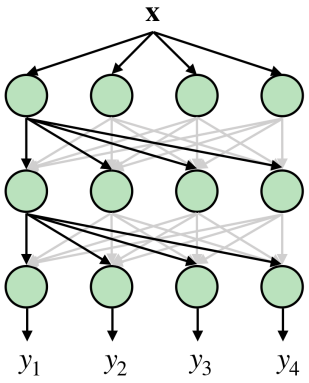


Figure 2.16: Three layers of a fully connected (dense) neural network.

design which influences modeling capabilities of the network. A basic and widely used activation function is the hyperbolic tangent:

$$\tanh(v) = \frac{1 - e^{-2v}}{1 + e^{-2v}}\,. \tag{2.10}$$

Since it is such an important parameter, a variety of activation functions can be found in the literature[5], e.g.: arc-tangent, rectified-linear (and variants), exponential-linear, and sigmoid (σ):

$$\sigma(v) = \frac{1}{1 + e^{v}}\,. \tag{2.11}$$

Special activation functions can also consider the hidden state of the whole layer, e.g. softmax:

$$\text{softmax}(v) = \frac{e^{v}}{\sum e^{v}}\,, \tag{2.12}$$

which is used in the output layer of one-hot classification problems.

A NN usually has a set of special neurons without incoming connections, which are used to feed data into the network. Similarly, the hidden states of another set of neurons are used to output data from the network. These sets need not to be disjoint in the general case—see RBMs, Section 2.3.6. They are disjoint, however, for the type of networks which are used for ADT in this work. The weights and biases of neurons represent the tunable parameters Θ which are to be adapted to make a NN calculate the desired output when provided with input data. Adaptation of these parameters is usually done using iterative numerical approximation methods and is called *training* or *optimization*. Different methods which specify algorithms to perform this task are usually referred to as *training methods* or *optimizers*. A trick which makes the numerical parameters optimization of complex NNs manageable is to organize them in layers. These layers of neurons

5 This Wikipedia article provides a good overview: `https://en.wikipedia.org/wiki/Activation_function`

must only have connections to the neurons in layers above (inputs) and below (outputs) it, c.f. Figure 2.16. If the network consists of at least three layers (input, hidden, and output layer), it is called a multilayer perceptron (MLP), or deep neural network. The inputs, outputs, and biases of the individual layers can be represented by a vector, the weights as matrices:

$$\mathbf{h}_l = \phi(\mathrm{W}_l \cdot \mathbf{x}_l + \mathbf{b}_l)\,, \tag{2.13}$$

where $\mathbf{h}_l$ and $\mathbf{b}_l$ are vectors of size N_l, the number of neurons within the current layer; $\mathbf{x}_l$ is a vector of size N_{l-1} the number of neurons of the previous layer, or input. W_l represents the input weights for the layer and is a matrix of size $N_l \times N_{l-1}$.

This structure enables the application of a parameter adaptation concept called *backward propagation of errors* or simply *backpropagation*. Utilizing the layered structure, backpropagation employs automatic differentiation to calculate a gradient of an error measure. This gradient is then used to update the model parameters to reduce the error. The error is usually measured using a loss function $\mathcal{L}$ which calculates the difference between targets $\mathbf{y}$ and actual output values $\hat{\mathbf{y}}$ of the network. The output of the network $\hat{\mathbf{y}}$, is calculated using the input data $\mathbf{x}$ and the network's forward path transfer function $\hat{\mathbf{y}} = f_{nt}(\Theta, \mathbf{x})$. The forward path transfer function of the network is simply a nesting of the transfer functions for the individual layers provided in Equation 2.13. There exist numerous candidates to be used as loss function, depending on the task and model choices. For regression tasks a linear output layer ($\phi(a) = a$) in combination with a mean squared error loss function is typically used:

$$\mathcal{L}(\Theta, \mathbf{x}, \mathbf{y}) = \frac{1}{N_{op}} \sum_{n=1}^{N_{op}} (y_n - \hat{y}_n)^2\,, \tag{2.14}$$

where N_{op} is the number of output neurons and y_n is the n-th element of the output vector $\mathbf{y}$ of the network.

In case of multi-class one-hot classification tasks, a softmax output layer (see Equation 2.12) in combination with a (categorical) cross-entropy loss is usually used:

$$\mathcal{L}(\Theta, \mathbf{x}, \mathbf{y}) = -\sum_{n=1}^{N_{op}} y_n \log(\hat{y}_n)\,. \tag{2.15}$$

For binary classification tasks (logistic regression), a sigmoid output layer in combination with a (mean binary) cross-entropy loss can be used:

$$\mathcal{L}(\Theta, \mathbf{x}, \mathbf{y}) = -\frac{1}{N_{op}} \sum_{n=1}^{N_{op}} \left[y_n \log(\hat{y}_n) + (1 - y_n) \log(1 - \hat{y}_n) \right]\,. \tag{2.16}$$

Given the loss function and the set of model parameters, a gradient $\mathcal{G}$ in respect to the model parameters Θ can be calculated:

$$\mathcal{G} = \nabla_{\Theta}\mathcal{L}(\Theta, \mathbf{x}, \mathbf{y})\,. \tag{2.17}$$

Calculating the gradient $\mathcal{G}$ of the loss function $\mathcal{L}$ for each individual parameter in the network (elements in Θ), is where the idea of backpropagation comes into play. In principle it is a clever way of computing the partial derivatives by recursively applying the chain rule in backward direction of the network. This is enabled by the network's structure of being organized in layers—thus there are no circular connections and the network transfer function consists of nesting and summing the individual neuron transfer functions. If for every activation function used in the network a derivative is known, the calculation of the gradient is tedious, but trivial and can thus be done automatically (automatic differentiation).

To update the model parameters using the gradient, different strategies can be employed. For plain gradient descent, the gradients are weighted with a learning rate α and subtracted from the model parameters to get the updated parameters:

$$\Theta \Leftarrow \Theta - \alpha \cdot \mathcal{G}\,. \tag{2.18}$$

The required training data consists of multiple pairs of corresponding input and output data. During training this data can be used in different ways. A parameter update can be calculated using the complete set of data (batch gradient descent), a single input/output pair (stochastic gradient descent, SGD), or a small sample of data points (mini batch gradient descent) for a single update. One iteration of updating the network parameters is usually called *update*, and an iteration over the whole training data set is called an *training epoch*. Thus, for batch gradient descent one training epoch consists of exactly one update.

Vanilla gradient descent does have some limitations when used on complex problems. An issue that can occur is that the method gets stuck in a local minimum of the loss function $\mathcal{L}$. Another problem which is encountered is slow progress in ravines and flat areas of the loss function. To accelerate the convergence of gradient descent and to avoid local minima, modifications[6] have been proposed: *(i)* momentum approaches use past update values to speed up convergence in ravines and flat areas of $\mathcal{L}$, e.g., SGD with momentum [83] and Nesterov accelerated gradient [59]; *(ii)* methods with learning rate adaptation use a history of past gradients to accelerate convergence more intelligently, e.g., Adagrad[17], Adadelta [112], RMSprop [87], and Adam [46]. As an example, RMSprop incorporates a recursively

6 This blog post provides a detailed overview of common methods: `http://sebastianruder.com/optimizing-gradient-descent`

defined decaying average of the past squared gradients $\mathbb{E}[\mathcal{G}^2]$ to adapt the learning rate:

$$\mathbb{E}[\mathcal{G}^2] \Leftarrow \lambda \mathbb{E}[\mathcal{G}^2] + (1-\lambda)\mathcal{G}^2 \tag{2.19}$$

$$\Theta \Leftarrow \Theta - \frac{\alpha}{\sqrt{\mathbb{E}[\mathcal{G}^2] + \epsilon}} \mathcal{G} \,. \tag{2.20}$$

Although not widely used, alternative training methods for NNs have been introduced in the past e.g. genetic algorithms [58], augmented Lagrangian methods [84], et cetera.

2.3.2 *Convolutional Neural Networks*

A simple MLP as described in Section 2.3.1 may also be called a *fully connected* or *dense* network, since every neuron in each layer has a connection to every neuron in the following layer. It is noteworthy that by stacking multiple layers of neurons this way, the number of parameters, and thus memory requirements as well as training time, grow quickly.

A property of fully connected networks is that they are oblivious of any ordering of the input data. We can scramble the input (and output) data of the individual training examples, as long as it is scrambled consistently, using the same ordering for each instance of the dataset; the network will still be able to adapt to the data the same way. In fact, if we scramble the network parameter matrices columnwise the same way as the input data, we can even scramble the data for already trained networks.

While this is desirable for certain types of data, for structured data it can be beneficial if the network has the capability to use local structures during training. An example of such data which is structured are images: the ordering of the pixels within an image is very important.

To allow the network to learn spatial relations in the input data and thus simplify training for structured data (like images), convolutional layers can be used. The idea is that instead of feeding every pixel as input to every neuron, a neuron in a convolutional layer only accepts a subset of input pixels (a smaller matrix). The inputs for this neuron are then shifted over the full input matrix, like a kernel during convolution. Because of this, a neuron in the case of convolutional neural network (CNN)s is usually called a *filter* or *kernel*. Usually multiple filters are used per layer, leading to multiple output channels or feature maps. When working on images, it is also common to use single channels for red, green, and blue colors (RGB channels) as input matrices for the whole network. To express this mathematically the single input values for a neuron x_i are replaced with a matrix

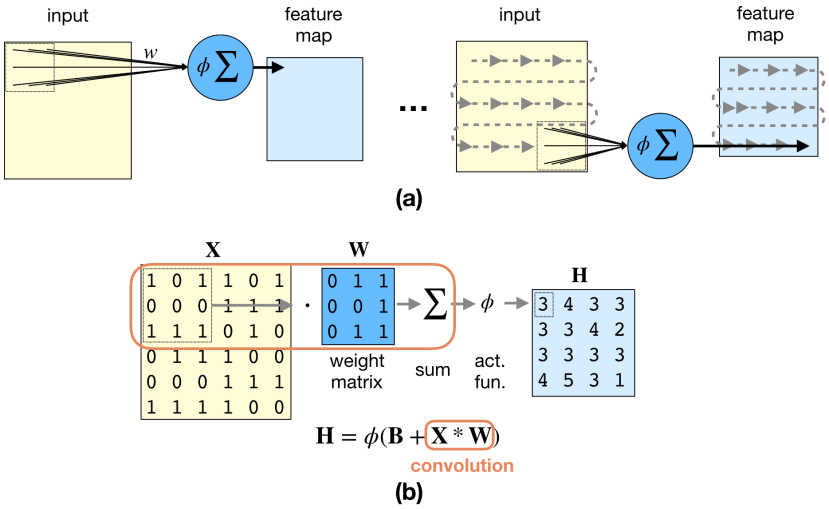


Figure 2.17: Mode of operation of a convolutional layer. This example shows a simplified scenario visualizing only one input and output channel. Usually multiple input channels are fed into multiple filters, leading to more than one output matrix (feature maps). Realizing the convolution using neurons similar to the ones used in MLPs, the convolution operation can be thought of as an element-wise multiplication with subsequent summing. Instead of moving the filter over the input matrix, the convolution can be thought of as multiple neurons using only the corresponding sub-matrix of X as input, while sharing input weights W and biases B.

X_i, the weights for the input w_i with a weight matrix W_{ij}, where the subscripts i and j indicate the input and output channel (filter), respectively. Finally, the multiplication is replaced by a convolution $*$. Doing so the output of a convolutional unit can be calculated using:

$$H_j = \phi(B_j + \sum_{i=0}^{l} X_i * W_{ij}) \,, \tag{2.21}$$

where l indicates the number of input channels. Note that the output for each filter H_j is again represented by a matrix. Figure 2.17 visualizes how a convolutional layer operates on input data. When performing convolution, the output matrix will be smaller than the input matrix:

$$c_H = \frac{c_X - c_W + 1}{s_c} \tag{2.22}$$

$$r_H = \frac{r_X - r_W + 1}{s_r} \,, \tag{2.23}$$

where c_H and r_H are the number of columns and rows of output matrix H, c_X and r_X columns and rows for input matrix X, c_W, r_W columns and rows for weight matrix W, and s_c and s_r the stride for convolutions (stepping size) in column and row directions. This

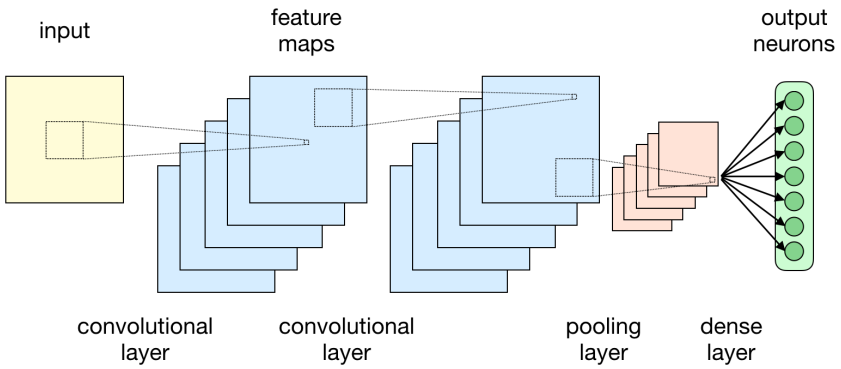


Figure 2.18: A convolutional network stack consisting of two convolutional layers and a pooling layer. While the input data only consists of one channel, every convolutional layer consists of five filters, generating five feature maps.

shrinking in matrix size is in some cases desirable. However, it can be avoided by using padding of the matrices before convolution is applied. There are several different methods for padding: *(i)* zero padding, *(ii)* repetition of last value, *(iii)* mirroring data, etc.; and the correct choice depends on the application and data.

A layer type which is commonly found in combination with convolutional layers are pooling layers. In pooling layers the neurons operate similar to neurons of convolutional layers on the input matrices, except they do not perform element-wise multiplication and summation. They calculate the output in other ways: e.g. max-pooling calculates the maximum while average-pooling calculates the average value of the input sub-matrix.

A typical convolutional network can be created by stacking multiple convolutional and pooling layers, with optional additional fully connected (dense) layers as output layers. If the dense output layers are omitted the network is usually called *fully convolutional*. Fully convolutional network stacks usually use the shrinking of the input matrix created by convolving without padding (valid convolutions) and pooling layers to reduce the size of the input array down to the desired output dimensionality. These kind of networks tend to be especially small, in terms of number of parameters. Figure 2.18 shows a simple convolutional network stack to exemplify typical CNN architectures.

Besides allowing the network to identify local structures, using convolutions also greatly reduces the number of parameters. This is often a positive side-effect which reduces memory consumption during training, and generally reduces calculation time per update due to algebraic optimizations for convolution.

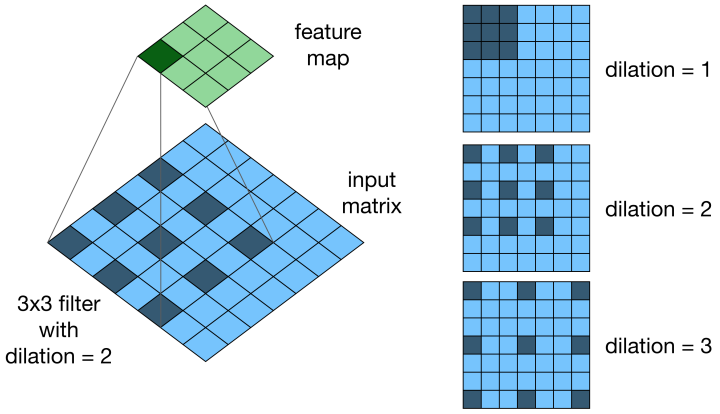


Figure 2.19: By using dilated convolutions, a larger receptive field can be achieved without increasing filter size.

2.3.2.1 *Dilated Convolution*

A special case of convolutional layers are so-called dilated convolutions [111]. The idea is to allow the used filters to cover a larger area of the input matrix, without actually increasing filter size. This can be useful for detecting large scale structures spread across a larger area of the input matrix. To this end, a dilation parameter is added which specifies how far the single elements of the filter matrix are separated when being applied on the input matrix. Figure 2.19 visualizes the working principles of dilated convolutions.

2.3.2.2 *Transposed Convolution*

Regularly misleadingly referred to as *deconvolution*, transposed convolutions [113] are often used as an inversion of normal convolutional layers in symmetric network architectures like autoencoders and GANs (c.f. Section 2.3.7). The basic idea is to transform the convolution into a matrix multiplication by creating a convolution matrix C and rearranging the input and output as vectors. Doing so, the transposed convolution matrix C^T can be used to create a pseudo-inverse transformation. An interpretation of this operation that is equivalent but easier to understand is the following: First, every value of the input matrix is padded with zeros (number depends on filter size and strides). Then normal convolutions are performed. This also results in an upsampling, i.e. an increase in size of the resulting feature map compared to the input matrix. Since the filter values are learned and initialized randomly anyway, it is not necessary to transform the filter matrix. Figure 2.20 visualizes the working principles of transposed convolutions.

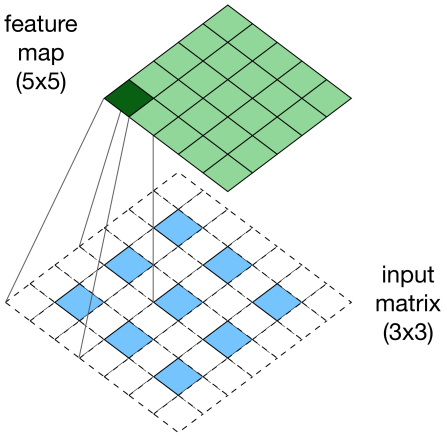


Figure 2.20: A transposed convolutional layer pads each value of the input matrix (blue) as shown in this figure. Doing so it is possible to increase the size of the resulting feature map (green) compared to the input matrix.

2.3.3 *Recurrent Neural Networks*

While CNNs are able to identify local structures and are thus well suited for images, they do not capture temporal context. I.e. when presented with two images at the input, the output will always be the same for each image, no matter in which order the network is presented with the images. For certain applications this behavior is not desirable, e.g. when dealing with time series data.

One way to approach this issue is to use special connections which provide the outputs of a layer for the previous time step as additional inputs. Because of their function, these connections are usually called *recurrent* connections. An RNN is a neural network containing neurons with recurrent connections within all or certain layers. The left diagram of Figure 2.21 visualizes a standard RNN neuron (sometimes also referred to as RNN *cell*, or *node* in the context of the network). For an RNN consisting of multiple recurrent hidden layers, the equation for each layer l at time step t is:

$$\mathbf{h}_l^t = \phi(\mathrm{W}_l \left[\mathbf{x}_l^t, \mathbf{h}_l^{t-1}\right] + \mathbf{b}_l)\,. \tag{2.24}$$

This equation is inferred from Equation 2.13, adding indexing of time for the inputs $\mathbf{x}_l^t$ and outputs $\mathbf{h}_l^t$. Furthermore, the output vector from the previous time step $\mathbf{h}_l^{t-1}$ is concatenated to the inputs $[\mathbf{x}_l^t, \mathbf{h}_l^{t-1}]$. In a network with multiple hidden layers, the input for a hidden layer is set to the output of the previous layer $\mathbf{x}_l^t = \mathbf{h}_{l-1}^t$. The feedback of the layer output acts as a simple form of memory and make RNNs the appropriate tool to process time series data.

To be able to train RNNs using backpropagation, the network is unfolded in time for the length of the time series data sequence. To unfold an RNN in time, the whole network is copied for each input

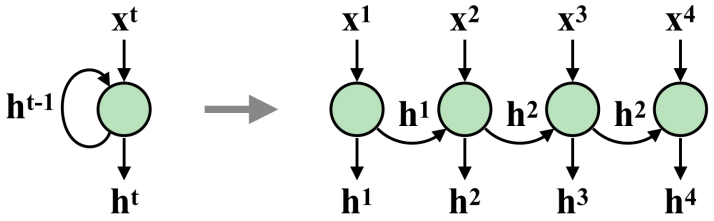


Figure 2.21: A simple RNN node (left) and the same node unfolded in time for four time steps (right).

value and recurrent connections now represent horizontal connections between the copied recurrent layers. Using this technique in combination with backpropagation to calculate the gradients is called *backpropagation through time (BPTT)* [102]. Figure 2.21 demonstrates the unfolding for a simple one layer, one neuron RNN. Note that the horizontal connections do not violate the constraint of no inter-layer connections, since the copied nodes for later time steps can be thought of as being located in an extra layer below the nodes of the preceding time step. This means that a new layer is added for each time step in the training sequence. The parameters (weights, biases) for nodes which were unfolded for different time steps are shared, i.e. the matrices and vectors only exist once and all node instances share the same values. Unfolded RNNs can become very deep networks, depending on the sequence length used for training. Since very deep networks have the tendency of being harder to train, often the time series data is split into subsequences to reduce the depth of the network during training.

2.3.3.1 *Bidirectional RNNs*

Simple recurrent networks as discussed so far have the ability to consider past input values when calculating the output for the current time step. However, for certain problems considering the full sequence can be advantageous, i.e. information from past as well as future inputs when producing the output for a certain time step. This can be achieved by implementing bidirectional connections, creating a so-called bidirectional recurrent neural network (BDRNN) [73]. A bidirectional layer consist of two recurrent sub-layers, one with connections in forward direction ($t-1 \rightarrow t$) and the other with connections in backward direction ($t+1 \rightarrow t$). Such an bidirectional architecture is visualized in Figure 2.22. The bidirectional connections allow the network to take past as well as future information into consideration for the output at time step t, which has been shown to be beneficial for many different tasks. The division of the bidirectional layer into two sub-layers is necessary to avoid circular connections and maintain the constraint of no inter-layer connections. The equations to calculate the output of a bidirectional layer are as follows:

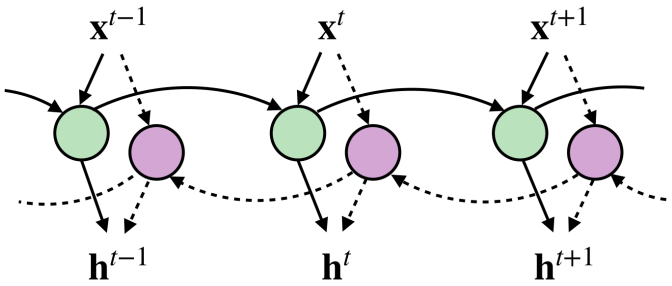


Figure 2.22: An unfolded bidirectional RNN layer revealing forward and backward connections. The solid (forward) connections are also found in a standard RNN (c.f. Figure 2.21) while the BDRNN contains additional backward connections (dashed arrows). $\mathbf{x}^t$ and $\mathbf{h}^t$ are the inputs and outputs at time step t, with the circles representing the network's neurons (green for forward layer, violet for backward layer).

$$\mathbf{h}_{lf}^t = \phi(\mathrm{W}_{lf}\left[\mathbf{x}_l^t, \mathbf{h}_{lf}^{t-1}\right] + \mathbf{b}_{lf}) \tag{2.25}$$

$$\mathbf{h}_{lb}^t = \phi(\mathrm{W}_{lb}\left[\mathbf{x}_l^t, \mathbf{h}_{lb}^{t+1}\right] + \mathbf{b}_{lb})\,, \tag{2.26}$$

using the same structure and nomenclature as in Equation 2.24. For the input of the following layer both hidden activations $\mathbf{h}_{lf}^t$ and $\mathbf{h}_{lb}^t$ are concatenated: $\mathbf{x}_{l+1}^t = [\mathbf{h}_{lf}^t, \mathbf{h}_{lb}^t]$.

2.3.3.2 *Long Short-Term Memory*

A problem when training plain RNNs in practice is that learning of long-term dependencies from the input data is often hard to achieve. This is mainly due to a system inherent problem called *vanishing and exploding gradients* [35]. Problems with gradients are caused by high numbers of multiplications with values smaller than (vanishing) or higher than (exploding) one. The chances of these scenarios to occur are increased by the many layers generated while unfolding the recurrent network for training.

To address this issue for recurrent architectures Hochreiter and Schmidhuber [35] introduce LSTM cells. LSTMs feature an internal memory $\mathbf{c}$ controlled by multiplicative gates, which counteract vanishing and exploding gradients and thus allow the network to better learn long-term dependencies. The internal memory is accessed and updated using three gates (input gate $\mathbf{i}$, forget gate $\mathbf{f}$, and output gate $\mathbf{o}$) controlled by the input $\mathbf{x}^t$, the hidden state $\mathbf{h}^{t-1}$ and, in case of

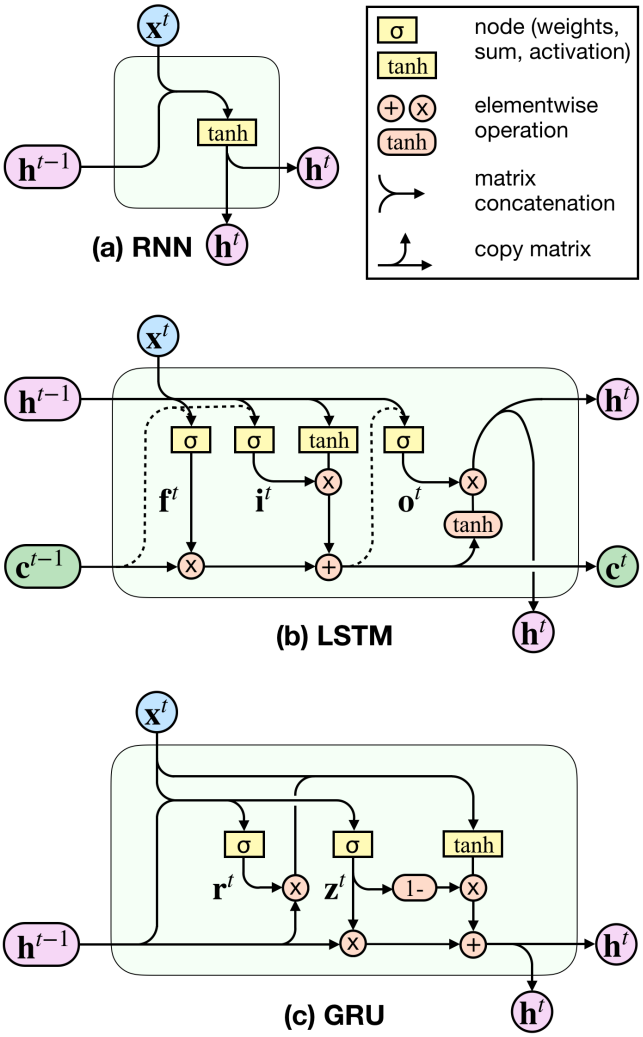


Figure 2.23: Overview of LSTM (b) and GRU (c) cell architectures, alongside a simple RNN cell (a). Symbol $\mathbf{x}^t$ represents the input at time step t, $\mathbf{h}$ is the hidden state, and $\mathbf{c}$ is the cell memory. Note that, while plain recurrent cells and GRUs only forward the hidden state $\mathbf{h}$ to the next time step, LSTM cells forward the hidden state as well as the cell memory $\mathbf{c}$. The yellow blocks represent NN nodes, applying weights and bias, taking the sum and applying the indicated activation function, where *tanh* is the hyperbolic tangent function and σ is the sigmoid function, e.g.: $\tanh(W \cdot \mathbf{x} + \mathbf{b})$—compare Equation 2.13. Orange round blocks stand for elementwise multiplication (x), elementwise addition (+), and elementwise application of the tanh function. Merging lines imply concatenation of matrices while splitting lines imply copying of the matrix represented by the connection. Dashed lines within the LSTM cell represent peephole connections for LSTMPs, i.e. they are not present for simple LSTM cells. Note that these diagrams do not represent single nodes but a whole layer, thus the input and output for $\mathbf{h}$ are both vectors. In case of single cells, c and outputs for h would be scalar values. Compare figures and details provided in [61].

LSTMs with peephole connections (LSTMP) [24], the cell memory $\mathbf{c}^{t-1}$. The equations for an RNN with LSTM cell architecture are:

$$\mathbf{i}_l^t = \sigma(\mathrm{W}_{il}\left[\mathbf{x}^t, \mathbf{h}_l^{t-1}\right] + \mathbf{b}_{il}) \tag{2.27}$$
$$\mathbf{f}_l^t = \sigma(\mathrm{W}_{fl}\left[\mathbf{x}^t, \mathbf{h}_l^{t-1}\right] + \mathbf{b}_{fl}) \tag{2.28}$$
$$\tilde{\mathbf{c}}_l^t = \tanh(\mathrm{W}_{cl}\left[\mathbf{x}^t, \mathbf{h}_l^{t-1}\right] + \mathbf{b}_{cl}) \tag{2.29}$$
$$\mathbf{c}_l^t = \mathbf{f}^t \odot \mathbf{c}^{t-1} + \mathbf{i}^t \odot \tilde{\mathbf{c}}_l^t \tag{2.30}$$
$$\mathbf{o}_l^t = \sigma(\mathrm{W}_{ol}\left[\mathbf{x}^t, \mathbf{h}_l^{t-1}\right] + \mathbf{b}_{ol}) \tag{2.31}$$
$$\mathbf{h}_l^t = \mathbf{o}_l^t \odot \tanh(\mathbf{c}_l^t)\,. \tag{2.32}$$

For an RNN with LSTMP cell architecture, equations for $\mathbf{i}$, $\mathbf{f}$, and $\mathbf{o}$ change:

$$\mathbf{i}_l^t = \sigma(\mathrm{W}_{il}\left[\mathbf{x}^t, \mathbf{h}_l^{t-1}, \mathbf{c}^{t-1}\right] + \mathbf{b}_{il}) \tag{2.33}$$
$$\mathbf{f}_l^t = \sigma(\mathrm{W}_{fl}\left[\mathbf{x}^t, \mathbf{h}_l^{t-1}, \mathbf{c}^{t-1}\right] + \mathbf{b}_{fl}) \tag{2.34}$$
$$\mathbf{o}_l^t = \sigma(\mathrm{W}_{ol}\left[\mathbf{x}^t, \mathbf{h}_l^{t-1}, \mathbf{c}^{t}\right] + \mathbf{b}_{ol})\,. \tag{2.35}$$

An overview of the LSTM cell architecture is given in the middle diagram of Figure 2.23. The contents of this figure and explanation of LSTMs and related technologies follow the ones provided in Olah's blog post from 2015 [61].

2.3.3.3 *Gated Recurrent Units*

Similar to LSTMPs, GRUs [13] can be seen as a modification of standard LSTMs, however, the changes are more grave. GRUs have a significantly lower number of parameters compared to LSTMs. This is achieved by reducing the number of gates, using only an update $\mathbf{z}$ and reset $\mathbf{r}$ gate, as well as by merging the memory into the hidden state $\mathbf{h}^{t-1}$. The equations which define an RNN with GRUs are:

$$\mathbf{z}_l^t = \sigma(\mathrm{W}_{zl}\left[\mathbf{x}_l^t, \mathbf{h}_l^{t-1}\right] + \mathbf{b}_{zl}) \tag{2.36}$$
$$\mathbf{r}_l^t = \sigma(\mathrm{W}_{rl}\left[\mathbf{x}_l^t, \mathbf{h}_l^{t-1}\right] + \mathbf{b}_{rl}) \tag{2.37}$$
$$\tilde{\mathbf{h}}_l^t = \tanh(\mathrm{W}_{hl}\left[\mathbf{x}^t, \mathbf{r}_l^t \odot \mathbf{h}_l^{t-1}\right] + \mathbf{b}_{hl}) \tag{2.38}$$
$$\mathbf{h}_l^t = \mathbf{z}_l^t \odot \mathbf{h}_l^{t-1} + (1 - \mathbf{z}_l^t) \odot \tilde{\mathbf{h}}_l^t\,. \tag{2.39}$$

An overview of the GRU architecture is shown in the bottom diagram of Figure 2.23.

GRUs can be seen to be more or less equivalent to LSTMs in terms of modeling capacities, with the difference that usually more GRU cells per layer are necessary to achieve the same model capacity as with LSTM cells. In practice, however, it shows that GRUs are more forgiving in terms of hyperparameter tuning.

2.3.4 *Convolutional Recurrent Neural Networks*

After discussing convolutional and recurrent layers and networks, an obvious question is if these two concepts can be combined. The short answer is yes: by combining convolutional and recurrent layers into one neural network a CRNN is created (sometimes but less commonly also called recurrent convolutional neural network). Since CNNs are the tool of choice for structured data like images, and RNNs are suited to process sequences like time series signals, CRNNs combine both of these abilities. Thus, as an example application for which CRNNs are well suited a video stream (time series of images) is an obvious candidate.

The structure of CRNNs can take different shapes, the most common is to use convolutional input layers and to use recurrent layers as output layers (instead of normal feed forward layers). Since both layer types were already discussed this is just a matter of correctly stacking those layer types. Figure 2.24 shows a schematic representation of such a stack similar to Figure 2.18's CNN version.

Similar as with RNNs the network has to be unfolded in time for training using backpropagation. In practice, data preparation and generating the input matrices for training, as well as transforming data between convolutional and recurrent layers correctly is not trivial. Apart from convolutional context (input matrix size) and training sequence length for the recurrent part, additional dimensions for batch size and convolution filter channels (or feature maps) are necessary and have to be handled correctly. Figure 2.25 visualizes how the CRNN is unfolded in time and how data is fed into the convolutional part during training. Note that Figure 2.25 only represents the convolutional input and recurrent training sequence, but ignores batch and feature maps, which represent additional data dimensions for training.

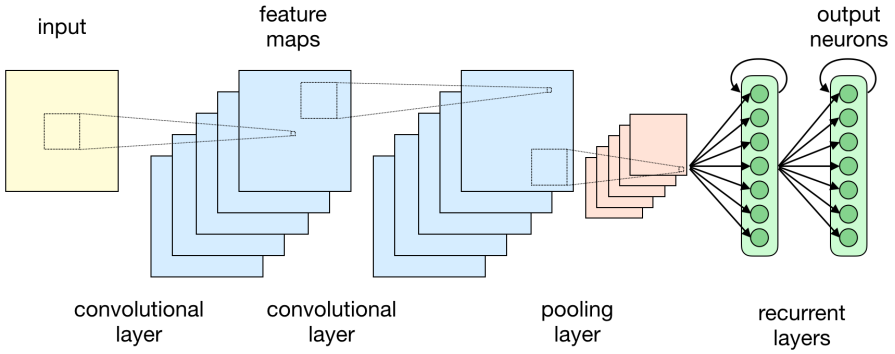


Figure 2.24: A schematic representation of a convolutional recurrent network stack. The convolutional and pooling layers are equal to the CNN stack in Figure 2.18, but the dense layer is replaced by two recurrent layers.

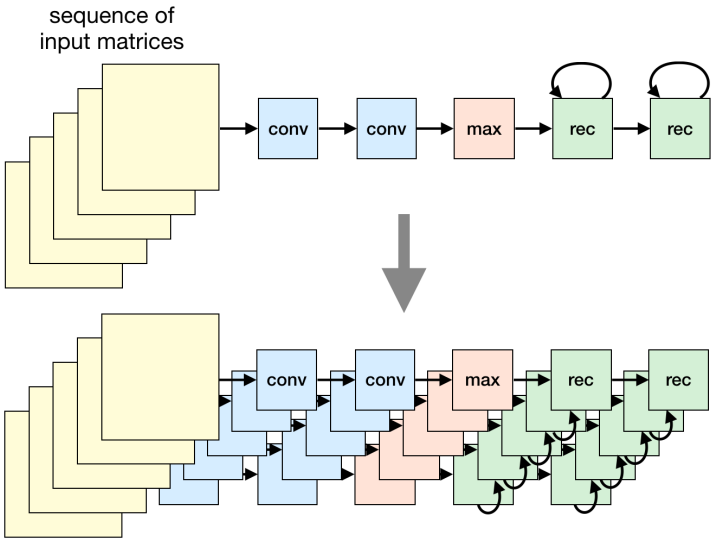


Figure 2.25: Diagram representing the data shape and mode of operation on the data for a CRNN network during training. As with normal RNNs, the CRNN has to be unfolded in time during training to enable the use of backpropagation.

2.3.5 *Regularization*

One of the major challenges with machine learning in general and with deep learning particularly is to avoid overfitting. The data used for training is usually only a small sample of the real data on which the model should be used. More often than not, the sample contains annotation errors, inaccuracies, or is not perfectly representative of the statistical population. Adapting the model's parameters to fit the training data too closely usually yields a suboptimal model, which performs worse on other unseen data—this is called overfitting. Figure 2.26 visualizes a simple two-dimensional feature space for a two-class classification problem. The three panels show *(a)* a too simple model, *(b)* a too complex, overfitted model, and *(c)* a model which learned the true underlying decision boundary. While model *(a)* might do surprisingly well on unseen data, model *(b)* will perform poorly compared to the performance on the training set.

To counter the tendency of NNs to overfit the training data, especially when complex network architectures are used, many different approaches have been presented in the past. Methods dealing with these problems can be collectively referred to as *regularizers*. One of the simplest and most effective forms of regularization is to keep the model complexity as low as possible, i.e. work with a network as small as possible. This usually involves reducing the network's number of layers and number of nodes per layer after a successful hyperparameter setup was found, by finding a good trade-off between performance and network size. Often this is not possible, or too resource intensive, e.g. if training takes a very long time. Because training often is very

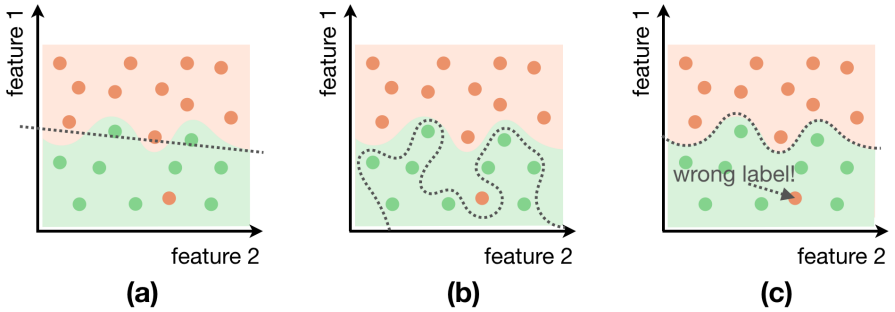


Figure 2.26: Three diagrams that show a simple data distribution for a two-class classification problem. Colored areas represent the two classes, and colored circles represent samples of training data. The decision boundary learned by the model is represented by a dotted line. Panels show different decision boundaries that are *(a)* too simple (underfitted), *(b)* too complex (overfitted), and *(c)* optimal or real boundary revealing wrong class labels in the training data.

time consuming, a common approach is to use models which have more capacity than necessary, and counteract overfitting by using other regularization techniques.

2.3.5.1 *Early Stopping*

Another quite simple approach to regularization is to use early stopping. To counteract overfitting, a small sample is excluded from the training data and used after each training epoch to validate the performance on data which was not used for training. Doing this, the degree of overfitting can be estimated by watching the ratio between validation and training loss. When the validation loss starts deteriorating it is a strong indication that the model begins to severely overfit the training data. As soon as this is observed, the training of the model is stopped, even before the training loss converges (= early stopping). Figure 2.27 visualizes this scenario.

2.3.5.2 *L1 and L2 norm Regularization*

L1 and L2 norm regularization consists of adding a term to the loss function which penalizes large absolute values for weights in the network:

$$\mathcal{L} = \mathcal{L} + \lambda \cdot \sum |w| \tag{2.40}$$

$$\mathcal{L} = \mathcal{L} + \lambda \cdot \sum w^2, \tag{2.41}$$

where Equation 2.40 represents the term for L1 and Equation 2.41 the term for L2 regularization, and λ represents another tunable hyperparameter. Adding this penalty to the loss function results in the weights to be pushed towards zero during training. Figuratively

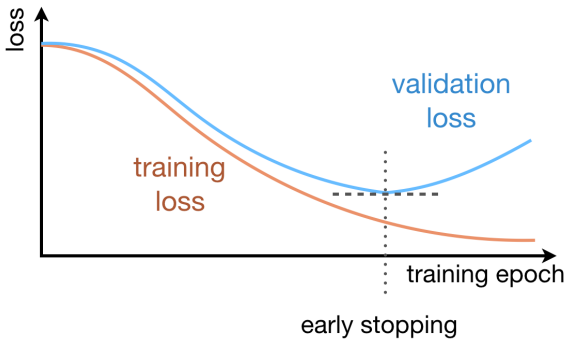


Figure 2.27: Training and validation loss curves for a network trained without early stopping. When using early stopping, the training is stopped as soon as the validation loss does not decrease anymore (indicated by dotted line). This usually marks the point where the network starts to severely overfit the training data.

speaking, this forces the network to distribute the calculation it needs to perform for its task evenly across all nodes, which ultimately limits the networks modeling capabilities and thus counteracts overfitting.

2.3.5.3 *Dropout*

When using dropout [82], randomly selected connections between two layers are disabled for one training step. Usually a value which determines which percentage of connections is disabled for each layer is used as an additional hyperparameter. Dropout effectively improves generalization and counteracts overfitting. During inference, dropout is not applied. Using dropout can also be interpreted as implicit random partitioning of the network during training and using an ensemble during inference.

2.3.5.4 *Batch Normalization*

Batch normalization [39] is a method which accelerates training of deep networks, but also has regularizing properties. The idea behind batch normalization is to normalize the activations between each layer and adding learnable translation and scaling parameters. First, the mean and variance of activations, for each layer which uses batch normalization, is calculated over the current training batch:

$$\mathcal{B} = \{\mathbf{x}_1 \dots \mathbf{x}_m\} \tag{2.42}$$

$$\boldsymbol{\mu}_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i \tag{2.43}$$

$$\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i - \boldsymbol{\mu}_{\mathcal{B}})^2 \,, \tag{2.44}$$

where $\mathcal{B}$ is the set of input values for a node for the current batch, m is the batch size, and $\boldsymbol{\mu}_{\mathcal{B}}$ is the mean and $\sigma^2_{\mathcal{B}}$ is the variance of the values.

Then, the input values are normalized and subsequently scaled and shifted using the learnable parameters $\lambda_{\mathcal{B}}$ and $\beta_{\mathcal{B}}$:

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{2.45}$$

$$\mathbf{y}_i = \lambda_{\mathcal{B}} \hat{\mathbf{x}}_i + \beta_{\mathcal{B}}\,, \tag{2.46}$$

where ϵ is a small value, added for computational stability and $\mathbf{y}_i$ is the batch normalized activation. During inference, the mean and variance of the current input batch are replaced by mean and variance for the whole training set, providing constant values, which leads to a deterministic behavior of the model during inference, which is desirable.

2.3.5.5 *Data Augmentation*

Especially if the training data is limited, it is hard to limit overfitting and achieve good generalization for the resulting trained models. If special domain knowledge is available and certain properties of the training data are known, it can be helpful to apply appropriate transformations to the data to increase variety and amount of training data. In case of images of objects to train an image recognition CNN, possible transformations can be: scaling, translation, rotation, and adding noise. Additionally color information adjustments like changing lightness, saturation, hue, and contrast can also be applied. By applying these data augmentation methods, the trained system will be more robust, tend less to overfit the training data, and exhibit better generalization capabilities. In the context of audio signals and music, the principle stays the same, while other transformations have to be used. Depending on the context, for audio signals these transformations might be applicable: pitch shifting, time stretching, volume changes, as well as adding noise and overlaying other signals.

Which of the possible transforms is applicable and valid depends on the target application and properties of the trained model. It is also necessary to consider if and in which form the annotations have to be adapted; e.g. in the context of piano transcription, pitch shifting might be applicable, but pitch annotations of played notes have to be adapted.

2.3.6 *Restricted Boltzmann Machines*

RBMs are a class of special neural networks consisting of two layers (visible and hidden layer) of neurons with undirected connections. There are no connections within a layer, in contrast to the general case of (unrestricted) Boltzmann machines. RBMs are thus a special case of Boltzmann machines, which are in turn a special case of Markov networks. Figure 2.28 shows a simple RBM structure. RBMs can be used

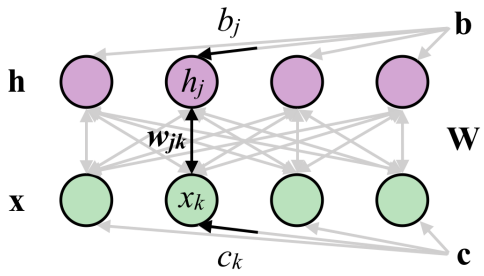


Figure 2.28: Structure and elements of an RBM. The nodes in the top layer represent the hidden or latent values **h** of the RBM, while the nodes in the lower layer represent the visible values **x**. The undirected connections between the two layers are weighted by the weight matrix W. Bias values for the hidden and visible units are represented by the vectors **b** and **c**, respectively.

for unsupervised feature learning, classification, and dimensionality reduction. RBMs were introduced by Smolensky in 1986 [77], and gained popularity after Hinton et al. [33] introduced fast training algorithms for them in 2006.

An RBM consists of neurons in a visible $x_k \in \mathbf{x}$ and hidden $h_j \in \mathbf{h}$ layer. Values for these layers are binary, i.e. can only assume values $x_k, h_j \in \{0, 1\}$. The two layers are (fully) connected with undirected weighted connections. The weight matrix W assigns each connection a weight. Additionally every neuron has a bias value, b_j for hidden and c_k for visible units.

Given these variables an energy function $E(x, h)$ can be defined:

$$E(x,h) = -\mathbf{h}^\intercal \mathrm{W}\mathbf{x} - \mathbf{c}^\intercal \mathbf{x} - \mathbf{b}^\intercal \mathbf{h} \tag{2.47}$$

$$= -\sum_j \sum_k \mathrm{W}_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \,. \tag{2.48}$$

Using this energy function, a probability that a certain state $(\mathbf{x}, \mathbf{h})$ can be observed is defined. Note that the probability distributions for elements in **x** and **h** are Bernoulli distributions since they are binary variables:

$$p(\mathbf{x}, \mathbf{h}) = \frac{e^{-E(\mathbf{x},\mathbf{h})}}{Z} \,, \tag{2.49}$$

where Z is the so-called partition function:

$$Z = \sum_{\mathbf{x}} \sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})} \,, \tag{2.50}$$

which is in practice intractable due to the exponential number of possible combinations of **x** and **h**. The sum over **x** and **h** in this and all subsequent equations means that it is taken over all possible $\mathbf{x} \in \{0, 1\}^X$ and $\mathbf{h} \in \{0, 1\}^H$, where X and H are the lengths of **x** and **h**, respectively.

Given those definitions, we can calculate the conditional probability for a certain state of $\mathbf{h}$, given $\mathbf{x}$: $p(\mathbf{h}|\mathbf{x})$ and vice-versa: $p(\mathbf{x}|\mathbf{h})$. Since there are no connections within a layer, the individual variables of a layer are independent: $p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h})$. This can also be seen as a special case of the local Markov property since the RBM is a Markov network. Thus, when substituting the definition of $p(\mathbf{x}, \mathbf{h})$ into

$$p(\mathbf{h}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{h})}{\sum_{\mathbf{h}'} p(\mathbf{x}, \mathbf{h}')}, \tag{2.51}$$

the following definition can be derived (note that Z can be factored out):

$$p(h_j = 1|\mathbf{x}) = \sigma(\mathbf{b}_j + \mathrm{W}_j \cdot \mathbf{x}), \tag{2.52}$$

where σ is the sigmoid function (see Equation 2.11). Similarly the probability for an element in $\mathbf{x}$ to be 1, given the hidden state $\mathbf{h}$ is defined by:

$$p(x_k = 1|\mathbf{h}) = \sigma(\mathbf{c}_k + \mathrm{W}_k \cdot \mathbf{h}). \tag{2.53}$$

Note that these equations are very similar to the definition of the hidden activation in Equation 2.9, with the only difference being that the output is the probability for the value being 1, instead of the activation for the neuron.

An RBM essentially models the distribution of $\mathbf{x}$ using the network parameters. The probability distribution for a input vector $\mathbf{x}$, $p(\mathbf{x})$, can be expressed using the networks parameters as:

$$\begin{aligned}
p(\mathbf{x}) &= \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h}) && (2.54)\\
&= \sum_{\mathbf{h}} \frac{e^{-E(\mathbf{x},\mathbf{y})}}{Z} && (2.55)\\
&= \exp(\mathbf{c}^\intercal \mathbf{x} + \sum_j \log(1 + e^{\mathbf{b}_j + \mathrm{W}_j \mathbf{x}}))/Z && (2.56)\\
&= \exp(\mathbf{c}^\intercal \mathbf{x} + \sum_j \mathrm{softplus}(\mathbf{b}_j + \mathrm{W}_j \mathbf{x}))/Z = e^{-F(\mathbf{x})}/Z. && (2.57)
\end{aligned}$$

A new activation function is introduced at this point: $\mathrm{softplus}(x) = \log(1 + e^x)$. This equation also introduces F which is commonly known as the *free energy* of an RBM. Equation 2.57 show how the parameters of the RBM ($\mathbf{c}$, $\mathbf{b}$ and W) influence the probability distribution $p(\mathbf{x})$. To train the RBM, these parameters have to be adapted in a way to maximize $p(\mathbf{x})$ for samples of $\mathbf{x}^{(t)} \in \mathrm{X}$.

To train an RBM, a loss function is defined and gradient descent is used to update the parameters. As loss, the average *negative log probability* is used:

$$\mathcal{L}(\mathbf{x}) = \frac{1}{T} \sum_t -\log\left(p(\mathbf{x}^{(t)})\right), \tag{2.58}$$

where T is the total number of samples in X and t is the index of a sample $\mathbf{x}^{(t)} \in \mathrm{X}$. Using this loss function the gradient with respect to the model parameters (Θ) can be computed:

$$\mathcal{G}(\mathbf{x}) = \frac{\partial - \log\left(p(\mathbf{x}^{(t)})\right)}{\partial \Theta} \tag{2.59}$$

$$= \mathbb{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x}^{(t)}, \mathbf{h})}{\partial \Theta}\middle| \mathbf{x}^{(t)}\right] - \mathbb{E}_{\mathbf{x},\mathbf{h}}\left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \Theta}\right]. \tag{2.60}$$

Since the real state of $\mathbf{h}$ is unknown, expectations $\mathbb{E}$ for the partial differentials of the energy $E(\mathbf{x}, \mathbf{h})$ are used instead. The index of $\mathbb{E}$ denotes variables over which values the expectation is taken, while the vertical line expresses a conditional expectation. Unfortunately, calculating the gradient for this loss involves calculating a partial derivative of the energy of the network over all $\mathbf{x}$, and $\mathbf{h}$, $E(\mathbf{x}, \mathbf{h})$, which is equally intractable as calculating the partition function Z. To solve this issue, a method to approximate $\mathbb{E}_{\mathbf{x},\mathbf{h}}$, called contrastive divergence (CD) [33], is usually used: Starting from a sample $\mathbf{x}^{(t)}$ from the training data, Gibbs sampling is applied to obtain a point $\tilde{\mathbf{x}}$, which is then used to calculate a point estimate for the expectation of the partial derivative of the energy for $\tilde{\mathbf{x}}$ and $\mathbf{h}$, $\mathbb{E}_{\tilde{\mathbf{x}},\mathbf{h}}$. This can be seen as Monte Carlo sampling of the expectation $\mathbb{E}_{\mathbf{x},\mathbf{h}}$ using a single data point $\tilde{\mathbf{x}}$ sampled from the model distribution. As starting point for sampling the expectation $\mathbb{E}_{\mathbf{x},\mathbf{h}}$, an actual training sample $\mathbf{x}^{(t)}$, instead of sampling from a (e.g. uniform) random distribution for the hidden variables $\mathbf{h}$, is used. Gibbs sampling of an RBM consists of two calculation steps:

$$\tilde{h}_j \sim p(h_j|\tilde{\mathbf{x}}) \tag{2.61}$$

and

$$\tilde{x}_k \sim p(x_k|\tilde{\mathbf{h}})\,. \tag{2.62}$$

After calculating the hidden state given the values in the visible layer (Equation 2.61), the updated values for the visible are calculated using the hidden state (Equation 2.62). For classic Gibbs sampling, this is usually done until the samples $\mathbf{x}$ represent the true distribution. In practice for sampling the expectation $\mathbb{E}_{\mathbf{x},\mathbf{h}}$, only a limited number of Gibbs sampling steps are performed (even only one step has been shown to work). This in combination with the fact that the variables in a layer are independent, which makes calculating the conditional probabilities very efficient, makes this approximation method feasible in practice.

Substituting the approximations introduced by CD and calculating the gradient for the model parameters W, $\mathbf{b}$, and $\mathbf{c}$, the following update rules can be obtained:

$$\mathrm{W} \Leftarrow \mathrm{W} + \alpha\left(\mathbf{h}(\mathbf{x}^{(t)})\mathbf{x}^{(t)\intercal} - \mathbf{h}(\tilde{\mathbf{x}})\tilde{\mathbf{x}}^{\intercal}\right) \tag{2.63}$$

$$\mathbf{b} \Leftarrow \mathbf{b} + \alpha\left(\mathbf{h}(\mathbf{x}^{(t)}) - \mathbf{h}(\tilde{\mathbf{x}})\right) \tag{2.64}$$

$$\mathbf{c} \Leftarrow \mathbf{c} + \alpha\left(\mathbf{x}^{(t)} - \tilde{\mathbf{x}}\right), \tag{2.65}$$

where, α is a small factor called *learning rate*.

A modification to that algorithm called persistent contrastive divergence (PCD) was introduced in 2009 by Tieleman et al. [86]. Instead of starting Gibbs sampling for $\tilde{\mathbf{x}}$ using the current sample $\mathbf{x}^{(t)}$, $\tilde{\mathbf{x}}$ from the previous iteration is used. Using this simple trick, Gibbs sampling produces samples with a higher variance, thus helping the network to generalize better.

An extension of RBM training used in works of this thesis is a method to encourage latent variable selectivity and sparsity [27]. Both are desired for multiple reasons, and will in the end aid the network to learn better suited hidden representations.

In this work RBMs are used to model the distribution of drum patterns. By starting with a given drum pattern sample, Gibbs sampling can be used to generate patterns with similar latent variable characteristics, which, in theory, should produce patterns with similar rhythmic properties.

2.3.7 *Generative Adversarial Networks*

In 2014, Goodfellow et al. [28] introduced GANs. While being a relatively new technology, GANs are applied on a wide variety of tasks and the field has a very active research community, making it nearly impossible to stay up-to-date with current developments. In the context of this work, only a very focused introduction of the working principles, and some training methods is given.

The basic idea behind GANs is to use two competing (deep) neural networks trained in a minimax fashion. One network's task is to generate realistic examples (generator G), while the other network's task is it to discriminate between generated examples and real examples from a sample library (discriminator D). The generator is defined by its model parameters θ_g and is a function generating an example $\hat{x}$ when provided with a latent variable z as input:

$$\hat{x} = G(z; \theta_g) \tag{2.66}$$

The dimensionality of the latent input depends on the application but is in general of a lower dimensionality than the generated data. The discriminator, likewise, has its own trainable model parameters θ_d and

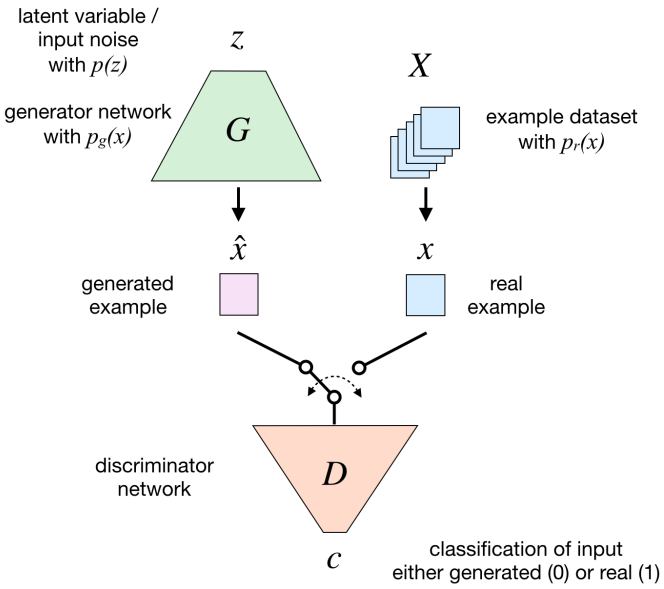


Figure 2.29: Overview of a GAN during training.

is a binary classifiers (output c, generated=0, real=1) discriminating input examples x:

$$c = D(x; \theta_d) \tag{2.67}$$

Both networks are designed in a way that they can be trained using backpropagation (c.f. Section 2.3.1). Training is then performed iteratively, feeding the discriminator with alternating samples from the generator and real examples from the training set.

A GAN learns an implicit estimation of the distribution of x, $p_r(x)$ represented by the training set, $x \in X$. The distribution learned by the generator will be denoted as $p_g(x)$. The discriminator tries to distinguish real $x \sim p_r(x)$ and generated $\hat{x} \sim p_g(x)$ samples, i.e. identifying generated samples. The training, thus, can be interpreted as a two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \,. \tag{2.68}$$

D tries to maximize this value, and can achieve this by improving detecting real examples ($\lim_{D(x)\to 1} \log D(x) = 0$), or reveal generated examples ($\hat{x} = G(z), \lim_{D(\hat{x})\to 0} \log(1 - D(\hat{x})) = 0$). On the other hand G tries to minimize this value by trying to trick the discriminator D into classifying generated examples as real ones: $\lim_{D(\hat{x})\to 1} \log 1 - D(\hat{x}) = -\inf$. The optimal discriminator D for a generator G can be derived from that as:

$$D^*_G(x) = \frac{p_r(x)}{p_r(x) + p_g(x)} \,. \tag{2.69}$$

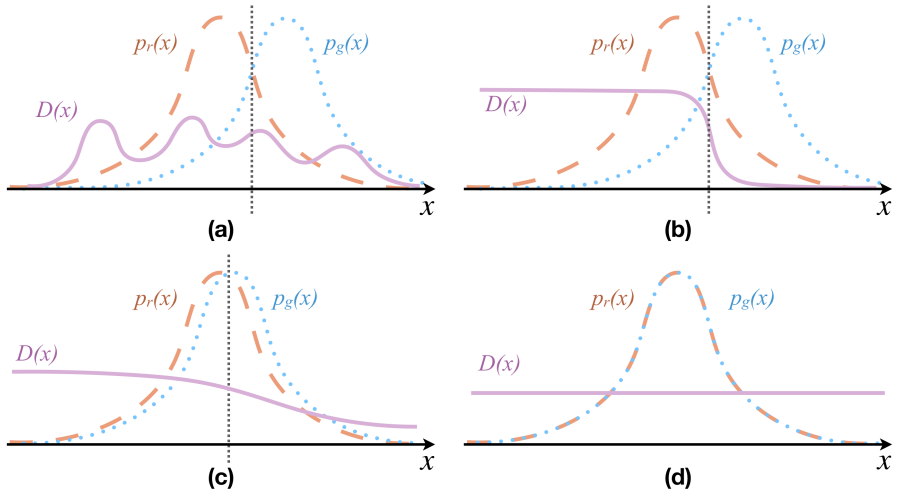


Figure 2.30: Probability distributions for generator $p_g(x)$ versus the distribution of real examples $p_r(x)$ during training. In the beginning, the two distributions are different. Panel (a) shows the discriminator before training. After some iterations of discriminator training an optimal discriminator was found (b). After several iterations of generator training, $p_g(x)$ becomes more similar to $p_r(x)$ (b). Finally, when $p_g(x) = p_r(x)$ the generator produces examples indistinguishable from real ones, and the discriminator can only guess. The optimal discriminator function in that case is $D^*_G = \frac{1}{2}$, see Equation 2.69.

Figure 2.30 shows the distributions of the generator $p_g(x)$ compared to the distribution of real samples $p_r(x)$ for a one dimensional case for X. The visualized discriminator $D(x)$ for panels (b), (c), and (d) of Figure 2.30 always close to the optimal discriminator provided by Equation 2.69.

Training of a GAN is performed updating the discriminator's parameters θ_d until convergence, and then the generator's parameters (θ_g) once. This is repeated until convergence of the generator. For this, two loss functions are needed which can be derived from Equation 2.68. To update the parameters we descend the following gradients:

$$\mathcal{G}_d = \nabla_{\theta_d} \mathcal{L}_d = \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} - \left[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \right] \tag{2.70}$$

$$\mathcal{G}_g = \nabla_{\theta_g} \mathcal{L}_g = \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)}))) \,. \tag{2.71}$$

Using these definitions for gradients, the GAN can be trained applying backpropagation and any gradient descent method, as with other deep neural networks. Note that in practice, convergence of the discriminator is not desired, rather an equilibrium between discriminator and generator is sought after. To achieve this, the discriminator is usually only trained for several iterations; typically for more iterations

in the beginning and fewer after several generator updates. The idea is that once the discriminator was sufficiently trained, and the generator only slowly changes, the discriminator only needs to adapt slightly to the updated generator for each generator update.

In general, vanilla GANs (the original version introduced in [28]) are hard to train, partly due to the fact that with high dimensionality of X both $p_g(x)$ and $p_r(x)$ residing on a lower dimensional manifold (represented by the latent variable space z), the probability of overlapping distributions for $p_g(x)$ and $p_r(x)$ is virtually zero. This leads to the problem that it is easy to find a perfect discriminator in such a high dimensional space and if the discriminator is perfect, the gradient for generator training vanishes (becomes zero). In these cases the GAN training is very unstable and successful training is tricky. This is because either the discriminator has not learned anything reasonable and there is no reasonable gradient for the generator to improve, or the discriminator is perfect and the gradient for the generator training becomes zero. In some cases that can lead to a behavior termed *mode collapse*, where the generator produces the same output which may or may not trick the discriminator. In any case, the generator does not learn a reasonable distribution for x.

2.3.7.1 *Wasserstein Distance and Loss*

To overcome this problem, many different tricks have been proposed. A major improvement brings the use of the Wasserstein distance instead of Jensen-Shannon (or Kullback-Leibler) divergence for distributions $p_g(x)$ and $p_r(x)$ for generator and discriminator training [1, 2]. The advantage of using the Wasserstein distance is that it provides a continuous measure of distances also for probability distributions that do not overlap, thus improving convergence in situations which have been identified as problematic. In other words, the Wasserstein distance provides a more interpretable relation between real and generated examples, indicating how far off the generator is; in contrast to the 0/1 behavior of the discriminator of the vanilla GAN. The Wasserstein distance between the two distributions p_r and p_g is given by:

$$W(p_r, p_g) = \inf_{\gamma \sim \prod(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]\,. \tag{2.72}$$

In general, it is intractable to use the Wasserstein distance as a loss function, since it involves calculating the *infimum* (greatest lower bound) of all joint distributions $\inf_{\gamma\sim\prod(p_r,p_g)}$. Due to this a transformation of the Wasserstein loss function utilizing the Kantorovich-Rubinstein duality is applied, which leads to:

$$\mathcal{L}_d = W(p_r, p_g) = \max_{w\in W} \mathbb{E}_{x\sim p_r}[f_w(x)] - \mathbb{E}_{z\sim p_r(z)}[f_w(G(z))]\,. \tag{2.73}$$

The function f_w is a K-Lipschitz continuous function, $\{f_w\}_{w\in\mathcal{W}}$, where $\mathcal{W}$ is a family of K-Lipschitz functions. Thus, in this GAN setup, the

discriminator D is the parameterized f_w and the loss function $\mathcal{L}$ represents the Wasserstein distance between p_r and p_g. That means that in a Wasserstein GAN, the discriminator does not classify examples into real and generated, but rather learns a function f_w helping to compute the Wasserstein distance between them. A requirement that comes with these design choices is that the function f_w represented by the discriminator has to be K-Lipschitz continuous. In practice this is enforced by clipping the weights of D to a small interval, e.g. ± 0.01.

While the idea of using Wasserstein distance is great and circumvents many problems of vanilla GANs, the approximation and use of weight clipping introduces new problems. Wasserstein GAN (WGAN) training can show slow convergence when the clipping window is chosen too large, or can still run into the problem of vanishing gradients for too small clipping windows. Gulrajani et al. [30] introduce gradient penalty as a way to circumvent weight clipping, which offers some improvement to these problems.

Part II

AUTOMATIC DRUM TRANSCRIPTION

3

SYNOPSIS

The following part covers the contributions focusing on automatic drum transcription. The individual chapters introduce the main publications of this thesis dealing with ADT. Each publication and thus chapter focuses a different problem in this area. As discussed in the introduction, methods and solutions introduced for this task in this work are all based on deep learning. All publications have a focus on end-to-end activation-function-based processing pipelines while different network architectures are used and evaluated. The methods introduced in the following chapters set new state-of-the-art performance results at the time of publication, and some still do.

In Chapter 4 deep learning methods for drum transcription are introduced for the first time. For the implemented neural networks a recurrent architecture is chosen, since it is well suited for time-series data like audio signals. The basic processing pipeline, modeling principles, and evaluation strategy for drum transcription experiments using artificial neural networks are established . The focus is on drum-only audio signals to investigate the applicability of deep learning using the most basic problem of ADT. In the evaluation, state-of-the-art performance on public drum solo datasets is achieved. Southall et al. [78] independently presented a related system for drum transcription at the same time. While the works are similar in term of used method for drum transcription, they focus on different aspects in their evaluation sections.

After that, in Chapter 5, the focus shifts to transcribing drums from music containing other instruments alongside the drum tracks. This represents a much harder problem than transcribing drums from drum-only tracks. This is due to several factors: Primarily because onsets of other harmonic and percussive instruments will often be close to drum instrument onsets, making it difficult to identify the exact onset time as well as the correct drum instruments. Furthermore, the spectral energy distributions for different instruments in full music tracks usually heavily overlap, making it difficult to classify the individual instruments correctly. To tackle this issue, state-of-the-art deep learning techniques like data augmentation [71] and dropout [82] along with changes to the network architecture are applied to improve generalization capabilities and robustness of the trained models. Using these methods, new state-of-the-art performance results on public datasets covering polyphonic music are achieved.

As discussed in the introduction, most drum transcription systems presented in recent years focus on drum instrument onset detection

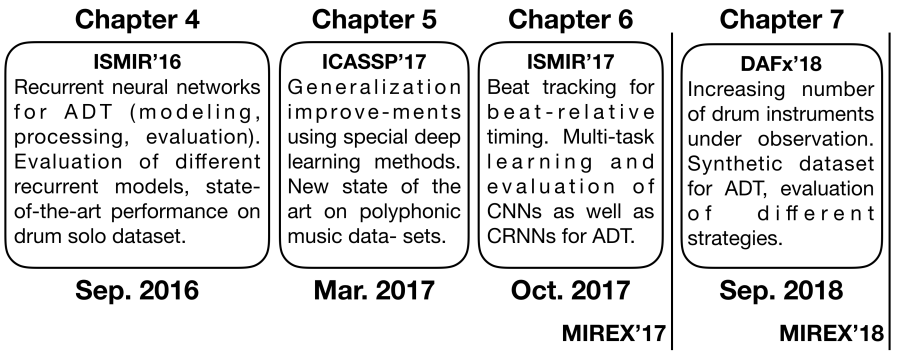


Figure 3.1: Timeline of publications and brief overview of topics covered in them, with corresponding chapters.

only. This is also the case for the neural-network-based approaches introduced in this thesis, so far. To put the drum instrument onset times into context and create real transcripts, additional metadata is required and must be extracted from the audio signal. The additional metadata which has to be extracted covers beat and downbeat positions, which is needed to transform onset times into bar and beat relative timing (note positions and durations). Since this data strongly correlates with drum instrument onsets, it is reasonable to try to leverage multi-task-learning effects for this data. The publication presented in Chapter 6 deals with these topics, while also introducing and evaluating CNN and CRNN network architectures for ADT.

Another shortcoming of most ADT systems introduced in the past was the limitation of considering only three drum instruments for transcription. As discussed in the introduction, making this simplification is reasonable since it circumvents many problems caused by shortcomings of the available datasets and other challenges of the task. Nevertheless, for satisfying transcripts all the other drum instruments cannot be ignored since they are equally important, even if being used more sparsely than bass drum, snare drum, and hi-hat. Chapter 7 thus focuses on increasing the number of instruments considered for transcription. In the corresponding publication, a large scale synthetic dataset is introduced to overcome shortcomings of available datasets. A thorough evaluation of different training strategies provides insights and indicates further directions to solve this problem.

Last, Chapter 8 covers attempts of evaluating different state-of-the-art ADT methods. An overview of an extensive review article on drum transcription which also provides an evaluation of NMF-based and NN-based ADT methods, is provided. Furthermore, the drum transcription challenge organized by the Music Information Retrieval Evaluation eXchange (MIREX) team is introduced and results from the last two years are presented. The works covered in this chapter are all part of the supplemental publications for this thesis.

Figure 3.1 puts the individual publications of this part into perspective and provides an overview of the timeline while indicating the chapters in which those publications can be found.

4 NEURAL NETWORKS FOR DRUM TRANSCRIPTION

4.1 OVERVIEW

The methods and experiments presented in this chapter were published in the following work:

Richard Vogl, Matthias Dorfer, and Peter Knees. "Recurrent neural networks for drum transcription". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*. New York, NY, USA, 2016.

In this chapter, neural networks, more specifically RNNs, are introduced as a novel method to solve the drum transcription task. As discussed in the introduction, RNNs are well suited for processing sequential data. Furthermore, Böck et al. [7, 8] demonstrate that RNNs can successfully be employed for beat tracking, which is a related task. For these reasons, RNNs present themselves as a viable choice, in this context.

In this work, different RNN architectures are evaluated within an end-to-end, activation-function-based transcription system. The neural network in this pipeline is tasked with extracting activation functions for the individual drum instruments under observation when provided with a spectrogram representation of the audio signal. The label *end-to-end* is used although a transformation of the audio signal into a spectrogram representation is required, as well as a peak picking method to extract onset positions from the resulting activation functions. This is done to distinguish the proposed new models from traditional hand-crafted methods that often use several different and more complex processing steps. Furthermore, the spectrogram can be seen as just another representation of the audio signal, while peak picking on the very spiky activation functions produced by the neural network is almost trivial. The evaluation covers different recurrent architectures, namely *(i)* forward, *(ii)* backward, and *(iii)* bidirectional RNNs. Additionally, a forward RNN with label time shift is evaluated. For evaluation, two publicly available drum transcription datasets comprising drum-solo tracks are used. This is done in order to establish a baseline for neural networks on a basic task.

The results show that RNNs can successfully be employed for activation function extraction for ADT. While state-of-the-art results on a simple drum-only public dataset are achieved, further work is required to make the model suitable for polyphonic music. Additionally,

the evaluation reveals that bidirectional RNNs perform better than unidirectional networks, but similar results can be obtained using a short label time shift. This is an interesting finding, since it reveals that the first few milliseconds are sufficient to correctly distinguish the onsets of the three drum instruments under observation. It further indicates that the bidirectional architecture used is not able to leverage global structures of the drum patterns.

4.2 CONTRIBUTIONS OF AUTHORS

As first author of this work, I contributed most of its content. My contributions comprise: idea of using RNNs for drum transcription, collection and preparation of used datasets, writing most of the source code in python, designing/running experiments and evaluation, as well as writing most of the paper.

Matthias Dorfer's contributions focused on helping me getting started with neural network training and ideas for evaluating the proposed system. He helped with choice of network architectures, provided a helper library for training (batch iterator, training setup, ...), and created Figure 1 visualizing the use of an RNN in the context of ADT. Finally, he supported me with writing the paper (proofreading, suggestions, etc.).

Peter Knees acted as supervisor for this paper. In this role, he supported me throughout the process of creating and conducting the experiments, and especially during writing the paper, by proofreading and providing valuable feedback.

RECURRENT NEURAL NETWORKS FOR DRUM TRANSCRIPTION

Richard Vogl, Matthias Dorfer, Peter Knees
Department of Computational Perception
Johannes Kepler University Linz, Austria
richard.vogl@jku.at

ABSTRACT

Music transcription is a core task in the field of music information retrieval. Transcribing the drum tracks of music pieces is a well-defined sub-task. The symbolic representation of a drum track contains much useful information about the piece, like meter, tempo, as well as various style and genre cues. This work introduces a novel approach for drum transcription using recurrent neural networks. We claim that recurrent neural networks can be trained to identify the onsets of percussive instruments based on general properties of their sound. Different architectures of recurrent neural networks are compared and evaluated using a well-known dataset. The outcomes are compared to results of a state-of-the-art approach on the same dataset. Furthermore, the ability of the networks to generalize is demonstrated using a second, independent dataset. The experiments yield promising results: while F-measures higher than state-of-the-art results are achieved, the networks are capable of generalizing reasonably well.

1. INTRODUCTION AND RELATED WORK

Automatic music transcription (AMT) methods aim at extracting a symbolic, note-like representation from the audio signal of music tracks. It comprises important tasks in the field of music information retrieval (MIR), as — with the knowledge of a symbolic representation — many MIR tasks can be address more efficiently. Additionally, a variety for direct applications of AMT systems exists, for example: sheet music extraction for music students, MIDI generation/re-synthesis, score following for performances, as well as visualizations of different forms.

Drum transcription is a sub-task of AMT which addresses creating a symbolic representation of all notes played by percussive instruments (drums, cymbals, bells, etc.). The source material is usually, as in AMT, a monaural audio source—either from polyphonic audio containing multiple instruments, or a solo drum track. The symbolic representation of notes played by the percussive instruments can be used to derive rhythmical meta-information like tempo, meter, and downbeat. The repetitive rhythmical structure of the drum track, as well as changes therein, can be used as features for high-level MIR tasks. They provide information about the overall structure of the song which can be utilized for song segmentation [18]. The drum rhythm patterns can also be utilized for genre classification [7]. Other applications for rhythmic patterns include query-by-tapping and query-by-beat-boxing [11,19].

A common approach to the task of drum transcription is to apply methods used for source separation like non-negative matrix factorization (NMF), independent component analysis (ICA), or sparse coding. In recent work, Dittmar and Gärtner [5] use an NMF approach to transcribe solo drum tracks into three drum sound classes representing bass drum, snare drum, and hi-hat. They achieve F-measure values of up to 95%. Their approach focuses on real-time transcription of solo drum tracks for which training instances of each individual instrument are present. This is a very specific use case and in many cases separate training instances for each instrument are not available. A more general and robust approach which is able to transcribe different sounding instruments is desirable. Smaragdis [28] introduces a convolutional NMF method. It uses two-dimensional matrices (instead of one-dimensional vectors used in NMF) as *temporal-spectral bases* which allow to consider temporal structures of the components. Smaragdis shows that this method can be applied to transcribe solo drum tracks. Lindsay-Smith and McDonald [21] extend this method and use convolutive NMF to build a system for solo drum track transcription. They report good results on a non-public, synthetic dataset.

Fitzgerald et al. [9] introduce prior subspace analysis, an ICA method using knowledge of the signals to be separated, and demonstrate the application for drum transcription. Spich et al. [29] extend this approach by incorporating a statistical music language model. These works focus on transcription of three and two instruments, respectively.

Scholler and Purwins [26] use a sparse coding approach to calculate a similarity measure for drum sound classification. They use eight basis vectors to represent the sounds for bass drum, snare drum, and hi-hat in the time domain. Yoshii et al. [33] present an automatic drum transcription system based on template matching and adaptation, similar to sparse coding approaches. They focus on transcription of snare and bass drum only, from polyphonic audio signals.

Algorithms based on source separation usually use the input signal to produce prototypes (or components) rep-

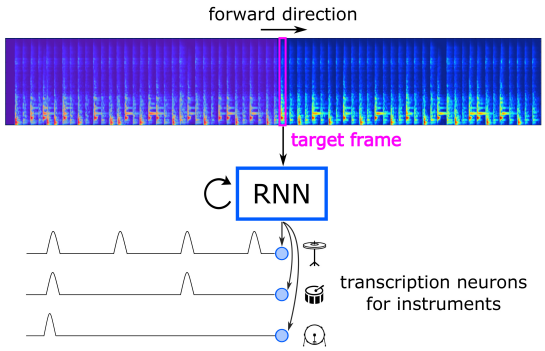


Figure 1. Overview of the proposed method. The extracted spectrogram is fed into the trained RNN which outputs activation functions for each instrument. A peak picking algorithms selects appropriate peaks as instrument onset candidates.

resenting individual instruments and so called activation curves which indicate the activity of them. A peak picking algorithm is needed to identify the instrument onsets in the activation curves. Additionally, the identified component prototypes have to be assigned to instruments. This is usually done using machine learning algorithms in combination with standard audio features [6, 16].

Another approach found in the literature is to first segment the audio stream using onset detection and classify the resulting fragments. Gillet and Richard [13] use a combination of a source separation technique and a support vector machine (SVM) classifier to transcribe drum sounds from polyphonic music. Miron et al. use a combination of frequency filters, onset detection and feature extraction in combination with a k-nearest-neighbor [23] and a k-means [22] classifier to detect drum sounds in a solo drum audio signal in real-time. Hidden Markov Models (HMMs) can be used to perform segmentation and classification in one step. Paulus and Klapuri [24] use HMMs to model the development of MFCCs over time. Decoding the most likely sequence yields activation curves for bass drum, snare drum, and hi-hat and can be applied for both solo drum tracks as well as polyphonic music.

Artificial neural networks consist of nodes (neurons) forming a directed graph, in which every connection has a certain weight. Since the discovery of gradient descent training methods which make training of complex architectures computationally feasible [17], artificial neural networks regained popularity in the machine learning community. They are being successfully applied in many different fields. Recurrent neural networks (RNNs) feature additional connections (recurrent connections) in each layer, providing the outputs of the same layer from the last time step as additional inputs. These connections can serve as memory for neural networks which is beneficial for tasks with sequential input data. RNNs have been shown to perform well, e.g., for speech recognition [25] and handwriting recognition [15]. Böck and Schedl use RNNs to improve beat tracking results [3] as well as for polyphonic

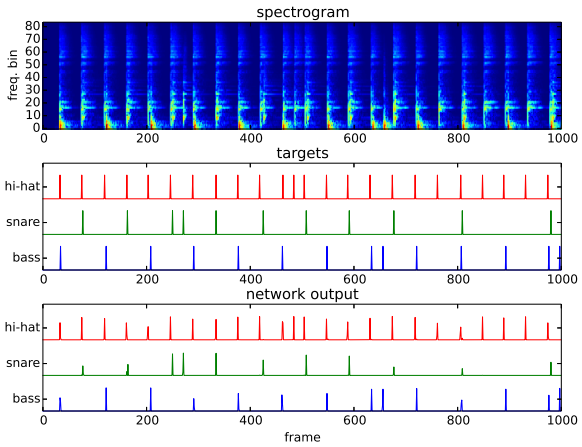


Figure 2. Spectrogram of a drum track and target functions for bass drum, snare drum, and hi-hat. The target function has a value of 1.0 at the frames at which annotations for the instruments exist and 0.0 otherwise. The frame rate of the target function is 100Hz, the same as for the spectrogram. The third graph shows the output of the trained RNN for the spectrogram in the first image.

piano transcription [4]. Sigtia et al. [27] use RNNs in the context of automatic music transcription as music language models to improve the results of a frame-level acoustic classifier. Although RNNs have been used in the past for transcription systems [4], we are not aware of any work using RNNs for transcription of drum tracks.

2. TASK AND MOTIVATION

In this work, we introduce a new method for automatic transcription of solo drum tracks using RNNs. While it is a first step towards drum transcription from polyphonic music, there also exist multiple applications for the transcription of solo drum tracks. In electronic music production, it can be used to transcribe drum loops if a re-synthesis using different sounds is desired. The transcription of recorded solo drum tracks can be used in the context of recording and production of rock songs. Nowadays it is not unusual to use sampled drums, e.g., in low-budget productions or in modern heavy metal genres. One the one hand, this is due to the complexity and costs of recording drums. On the other hand, with sampled drums it is easier to achieve the high precision and even robotic sounding style desired in some genres. Instead of manually programming the drum track, automatic transcription of a simple low-quality drum recording can be used as basis for the production of a song. As in other works, we focus on the transcription of bass drum, snare drum, and hi-hat. These instruments usually define the main rhythmic patterns [24], depending on genre and play style. They also cover most (>80% in the case of the ENST-Drums dataset, see Section 4.1) of the played notes in full drum kit recordings. Since simple RNN architectures already provide good transcription results (cf. Section 5), it is worthwhile exploring their application in this task further.

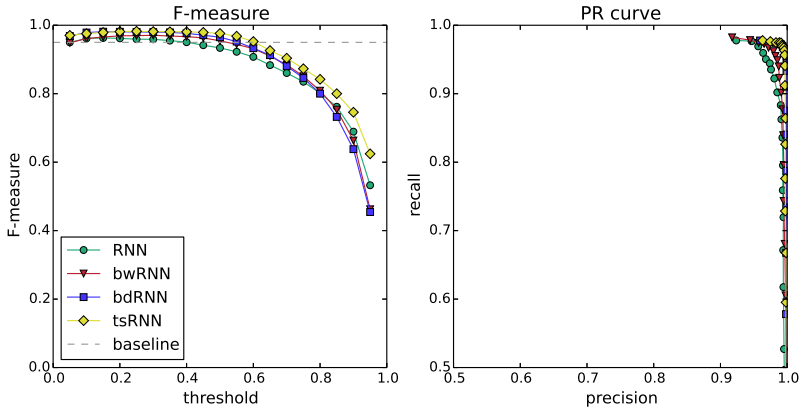


Figure 3. Result visualization for the evaluation on the IDMT-SMT-Drums dataset. The left plot shows the F-measure curve, the right plot the precision-recall curve for different threshold levels for peak picking.

3. METHOD

To extract the played notes from the audio signal, first a spectrogram of the audio signal is calculated. This is frame-wise fed into an RNN with three output neurons. The outputs of the RNN provide activation signals for the three drum instruments. A peak picking algorithm then identifies the onsets for each instrument's activation function, which yields the finished transcript (cf. Figure 1).

In this work, we compare four RNN architectures designed for transcribing solo drum tracks. These are: *i.* a simple RNN, *ii.* a backward RNN (bwRNN), *iii.* a bidirectional RNN (bdRNN), and *iv.* an RNN with time shift (tsRNN). The next section will cover the preprocessing of the audio signal, which is used for all four RNNs. After that, the individual architectures are presented in detail.

3.1 Signal Preprocessing

All four RNN architectures use the same features extracted from the audio signal. As input, mono audio files with 16 bit resolution at 44.1 kHz sampling rate are used. The audio is normalized and padded with 0.25 seconds of silence, to avoid onsets occurring immediately at the beginning of the audio file. First a logarithmic power spectrogram is calculated using a 2048 samples window size and a resulting frame rate of 100Hz. The frequency axis is then transformed to a logarithmic scale using twelve triangular filters per octave for a frequency range from 20 to 20,000 Hz. This results in a total number of 84 frequency bins.

3.2 Network Architectures

In this work, four different architectures of RNNs are compared. The four architectures comprise a plain RNN and three variations which are described in detail in the following.

3.2.1 Recurrent Neural Network

The plain RNN features a 84-node input layer which is needed to handle the input data vectors of the same size. The recurrent layer consists of 200 recurrently connected rectified linear units (ReLUs [14]). Although RNNs with ReLU activations can be difficult to train [20], good results without special initialization or treatment were achieved in this work. The connections between the input and the recurrent layer, the recurrent connections, and the connections between the recurrent layer and the output layer are all realized densely (every node is connected to all other nodes). The output layer consists of three nodes with sigmoid transfer functions, which provide the activation functions for the three instrument classes defined earlier. The sigmoid transfer function was chosen because binary cross-entropy was used as loss function for training, which turned out to be easier to train in the experiments.

3.2.2 Backward RNN

This RNN is very similar to the basic RNN with the only difference being that the recurrent connections are backward instead of forward in time. This was done in order to evaluate if the short sustain phase of percussive instruments provides additional information for the classification. The plain RNN has to identify the instruments at exactly the time frame of the onset annotation, thus the sustain phase of the notes can not be considered by it. This architecture is not real-time-capable since the audio to be transcribed is analyzed in reverse. Moreover, it might be more hard for this architecture to find the exact position of the onsets since the steep slope of the onset is only seen in forward direction.

3.2.3 Bidirectional RNN

The architecture of the bidirectional RNN used in this work consists of 100 nodes in a forward layer and 100 nodes in a backward layer. Both the forward and backward layers are directly connected to the input layer. Bidirectional RNNs often produce better results than unidirectional RNNs because they can also use the context of future frames for classification. In this work, they are meant to combine both the strengths of the forward and backward RNN. Unfortunately, this system has the same limitations as the backward RNN, making it not usable for real-time applications.

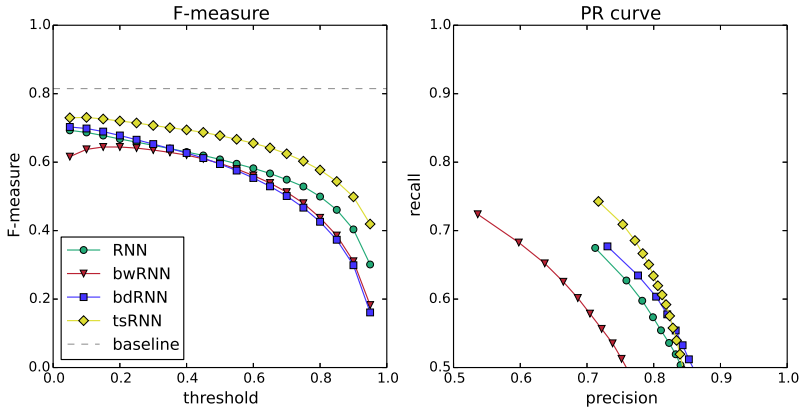


Figure 4. Result visualization for the evaluation on the ENST-Drums dataset. The left plot shows the F-measure curve, the right plot the precision-recall curve for different threshold levels for peak picking.

3.2.4 RNN with Time Shift

This approach is architecturally the same as the simple forward RNN, with the addition that this network can see more frames of the spectrogram to identify the instruments active at an onset. For training, the annotations are shifted into the future by 25ms and after transcription the detected onsets are shifted back by the same time. Doing this, the RNN can take a small portion of the sustain phase of the onset's spectrogram also into account. This is meant to improve the performance of the classification in the same way the backward connections do, without losing the real-time capabilities. The system is to a limited degree still real-time capable—depending on the length of the time shift. The used delay of 25ms in this work might still be sufficiently small for certain applications like score following and other visualizations and it can be tuned to meet the demands of certain applications.

3.3 Peak Picking

The output neurons of the RNNs provide activation functions for every instrument. To identify the instrument onsets, a simple and robust peak picking method designed for onset detection is used [2]. Peaks are selected at a frame n of the activation function $F(n)$ if the following three conditions are met:

1. $F(n) = max(F(n - pre\_max : n + post\_max))$,
2. $F(n) \geq mean(F(n - pre\_avg : n + post\_avg)) + \delta$,
3. $n - n_{lastpeak} > combination\_width$,

where δ is a threshold varied for evaluation. Simply put, a peak has to be the maximum of a certain window, and higher than the mean plus some threshold of another window. Additionally there has to be a distance of at least $combination\_width$ to the last peak. Parameters for the windows were chosen to achieve good results on a development data set while considering that 10 ms is the threshold of hearing two distinct events (values are converted from frames to ms): $pre\_max = 20ms$, $post\_max = 0ms$, $pre\_avg = 20ms$, $post\_avg = 0ms$, and $combination\_width = 20ms$. Setting $post\_max$ and $post\_avg$ to zero allows the application in online scenarios.

3.4 RNN Training

The task which has to be solved by the RNNs in this work is a three-way binary classification problem. When provided with the input spectrogram, the RNN has to identify the onsets of the three instrument classes by predicting the activation functions at the output neurons. The training algorithm has to adapt the weights and biases of the network in a way to achieve this functionality. In this work, the *rmsprop* method proposed by Hinton and Tieleman [31] is used as training algorithm. Additionally, dropout [30] between the recurrent and the output layer of the RNNs is used for training. When using dropout, randomly chosen connections are disabled for a single training iteration. The amount of disabled connections is determined by the dropout rate.

The goal of the training algorithm is to minimize a loss function. The loss function measures how much error the networks makes while reproducing the target functions. As loss function for training, the mean of the binary cross-entropy of the values of the three output neurons and the target functions is used (see Figure 2). The training with *rmsprop* is based on mini batches. In this work, mini batches with a size of eight instances were used. The training instances consist of 100-frame-segments of the extracted spectrogram. These are generated by extracting the spectrogram as described in Section 3.1 from the training files and cutting it into 100-frame-segments with 90 frames overlap (i.e. 10 frames hop-size). The order of the segments for training is randomized.

During one epoch the training data is used to adapt the weights and biases of the network. At the end of an epoch, the validation data is used to estimate the quality of the trained network. The training of the RNNs is aborted as soon as the resulting loss for the validation set has not decreased for 10 epochs. As learning rate decay strategy, the following method is applied: after every seven epochs the

Results for IDMT-SMT-Drums

algorithm	best F-measure[%]	at threshold
RNN	96.3	0.15
bwRNN	97.1	0.30
bdRNN	98.1	0.15
tsRNN	**98.2**	0.25
NMF [5]	**95.0**	-

Table 1. Evaluation results on the IDMT-SMT-Drums dataset. The NMF approach serves as state-of-the-art baseline.

Results for ENST-Drums

algorithm	best F-measure[%]	at threshold
RNN	69.3	0.05
bwRNN	64.4	0.15
bdRNN	70.3	0.05
tsRNN	**73.1**	0.10
HMM [24]	**81.5**	-

Table 2. Evaluation results on the ENST-Drums dataset. The HMM approach serves as state-of-the-art baseline.

learning rate is halved. For the simple RNN, backward RNN, and time shifted RNN the following parameter settings are used: initial learning rate $r_l = 0.001$ and dropout rate $r_d = 0.2$. In case of the bidirectional RNN the following parameter settings are used: initial learning rate $r_l = 0.0005$ and the dropout rate $r_d = 0.3$. The network is initialized with weights randomly sampled from a uniform distribution in the range ± 0.01, and zero-value biases.

All hyperparameters like network architecture, dropout rate, and learning rate were chosen according to empirical experimentation on a development data set, experience, and best practice examples.

3.5 Implementation

The implementation was done in Python using Lasagne [8] and Theano for RNN training and evaluation. The madmom [1] framework was used for signal processing and feature extraction, as well as for peak picking and evaluation metric calculation (precision, recall, and F-measure).

4. EVALUATION

To evaluate the presented system, the audio files of the test subset are preprocessed as explained in Section 3.1. Subsequently the spectrogram of the audio file is fed into the input layer of the RNN. The three neurons of the output layer provide the activation functions for the three instruments for which the peak picking algorithm then identifies the relevant peaks. These peaks are interpreted as instrument onsets. The true positive and false positive onsets are then identified by using a 20 ms tolerance window. It should be noted that in the state-of-the-art methods for the ENST-Drums dataset [24] as well as for the IDMT-SMT-Drums dataset [5], less strict tolerance windows of 30 ms and 50 ms, respectively, are used. Using these values, precision, recall, and F-measure for the onsets are calculated.

4.1 Datasets

For training and evaluation the IDMT-SMT-Drums [5] dataset was used. Some missing annotations have been added and additionally annotations for the *#train* tracks have been created. The *#train* tracks are tracks containing separated strokes of the individual instruments. These are only used as additional training examples and not used in the test set, to maintain a fair comparison with the results in [5]. The dataset was split into train, validation, and test subsets using 70%, 15%, and 15% of the files, respectively.

Additionally, the audio portion of the ENST-Drums [12] dataset was used as a second independent dataset to evaluate the generalization capabilities of the RNNs. From this dataset, the wet mixes of the drum-only tracks of all three drummers were used. Since all models were trained on the IDMT-SMT-Drums dataset, no splitting of this dataset was necessary.

For both datasets the three instruments' target functions are created by calculating the correct active frames (for a target frame rate of 100 Hz) using the annotations for each instrument. The target functions are one at the frames in which an annotation is present and zero otherwise. See Figure 2 for a visualization of the target functions in the context of the input spectrogram.

4.2 Experiments

For all four architectures, two different experiments were performed. First, the model was trained using the training and validation subsets of the IDMT-SMT-Drums dataset. Then, using the trained model, the tracks of the test split of the dataset were transcribed and the resulting precision, recall, and F-measure were calculated. Second, the trained model was evaluated by transcribing the ENST-Drums dataset and calculating the validation metrics. This was done to evaluate how well the trained models are able to generalize and if the models are over-fitted to the training dataset. Since the ENST-Drums dataset contains more than just the three instruments with which the model was trained, only the snare, bass, and hi-hat annotations were used. This makes it on the one hand easier to identify all annotated notes, on the other hand, there are some percussive onsets in the audio, which should not be transcribed and which are counted as false positives if the network falsely interprets them as snare, bass, or hi-hat hits. The percentage of snare, bass, and hi-hat annotations is 81.2% (i.e., 18.8% are other instruments which are ignored and potential false positives). The ENST-Drums dataset contains more expressive and faster drumming styles than the IDMT-SMT-Drums dataset, making it a more difficult dataset to transcribe. This fact is reflected in the transcription performances of both the state-of-the-art algorithms as well as the proposed methods. This behavior can also be observed in the work of Wu and Lerch [32] who apply their method to both datasets.

5. RESULTS AND DISCUSSION

Table 1 summarizes the results of all methods on the IDMT-SMT-Drums dataset. It can be seen that the F-measure values for all RNNs are higher than the state-of-the-art. It should be noted at this point, that the approach presented in [5] was introduced for real-time transcription. Nevertheless, the used NMF approach uses audio samples of the exact same instruments which should be transcribed as prototypes, which is the best-case scenario for NMF transcription. In contrast, our approach is trained on a split of the full dataset which contains many different instrument sounds, and thus is a more general model than the one used in the state-of-the-art approach used as baseline. It can be observed that the backward RNN performs slightly better than the plain RNN, which indicates that indeed the short sustain phases of the drum instruments contain information which is useful for classification. The bidirectional RNN again performs slightly better than the backward RNN, which comes as no surprise since it combines the properties of the plain forward and backward RNNs. The results of the forward RNN with time shift are not significantly different from the results of the bidirectional RNN. This indicates that the short additional time frame provided by the time shift provides sufficient additional information to achieve similar classification results as with a bidirectional RNN. Figure 3 shows a F-measure curve as well as a precision-recall curve for different threshold levels for peak picking.

The results of the evaluation of the models trained on the IDMT-SMT-Drums dataset used to transcribe the ENST-Drums dataset are shown in Table 2. The achieved F-measure values are not as high as the state-of-the-art in this case but this was expected. In contrast to [24], the model used in this work is not trained on splits of the ENST-Drums dataset and thus not optimized for it. Nevertheless, reasonable high F-measure values are achieved with respect to the fact that the model was trained on completely different and more simple data. This can be interpreted as an indication that the model in fact learns, to some degree, general properties of the three different drum instruments. Figure 4 shows an F-measure curve as well as a precision-recall curve for different threshold levels for peak picking.

In Figures 3 and 4, it can be seen that the highest F-measure values are found for low values for the threshold of the peak picking algorithm. This suggests that the RNNs are quite selective and the predicted activation functions do not contain much noise—which can in fact be observed (see Figure 2). This further implies that choices for peak picking window sizes are not critical, which was also observed in empiric experiments.

6. FUTURE WORK

Next steps for using RNNs for drum transcription will involve adapting the method to work on polyphonic audio tracks. It can be imagined to combine the presented method with a harmonic/percussive separation stage, using, e.g., the method introduced by Fitzgerald et al. [10], which would yield a drum track transcript from a full polyphonic audio track. As we show in this work, the transcription methods using RNNs are quite selective and therefore expected to be robust regarding artifacts resulting from source separation. Training directly on full audio tracks may also be a viable option to work on full audio tracks.

Another option is to use more instrument classes than the three instruments used in this and many other works. Theoretically, RNNs are not as vulnerable as source separation approaches when it comes to the number of instruments to transcribe. It has been shown that RNNs can perform well when using a much greater number of output neurons, for example 88 neurons in the case of piano transcription [4]. Although, for this, a dataset which has a balanced amount of notes played by different instruments has to be created first.

7. CONCLUSION

In this work, four architectures for drum transcription methods of solo drum tracks using RNNs were introduced. Their transcription performances are better than the results of the state-of-the-art approach which uses an NMF method—even with the NMF approach having the advantage of being trained on exactly the same instruments used in the drum tracks. The RNN approaches seem to be able to generalize quite well, since reasonable high transcription results are yielded on another, independent, and more difficult dataset. The precision-recall curves show that the best results are obtained when using a low threshold for peak picking. This implies that the used transcription methods are quite selective, which is an indication that they are robust and not bound to be influenced by noise or artifacts when using additional preprocessing steps.

8. ACKNOWLEDGMENTS

This work is supported by the European Union's Seventh Framework Programme FP7 / 2007-2013 for research, technological development and demonstration under grant agreement no. 610591 (GiantSteps). This work is supported by the Austrian ministries BMVIT and BMWFW, and the province of Upper Austria via the COMET Center SCCH. We gratefully acknowledge the support of NVIDIA Corporation with the donation of one Titan Black and one Tesla K40 GPU used for this research.

9. REFERENCES

[1] Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, and Gerhard Widmer. madmom: a new python audio and music signal processing library. `https://arxiv.org/abs/1605.07008`, 2016.

[2] Sebastian Böck, Florian Krebs, and Markus Schedl. Evaluating the online capabilities of onset detection methods. In *Proc 13th Intl Soc for Music Information Retrieval Conf*, pages 49–54, Porto, Portugal, October 2012.

[3] Sebastian Böck and Markus Schedl. Enhanced beat tracking with context-aware neural networks. In *Proc 14th Conf on Digital Audio Effects*, Paris, France, September 2011.

[4] Sebastian Böck and Markus Schedl. Polyphonic piano note transcription with recurrent neural networks. In *Proc IEEE Intl Conf on Acoustics, Speech and Signal Processing*, pages 121–124, Kyoto, Japan, March 2012.

[5] Christian Dittmar and Daniel Gärtner. Real-time transcription and separation of drum recordings based on nmf decomposition. In *Proc 17th Intl Conf on Digital Audio Effects*, Erlangen, Germany, September 2014.

[6] Christian Dittmar and Christian Uhle. Further steps towards drum transcription of polyphonic music. In *Proc 116th Audio Engineering Soc Conv*, Berlin, Germany, May 2004.

[7] Simon Dixon, Fabien Gouyon, Gerhard Widmer, et al. Towards characterisation of music via rhythmic patterns. In *Proc 5th Intl Conf on Music Information Retrieval*, Barcelona, Spain, October 2004.

[8] Sander Dieleman et al. Lasagne: First release. `http://dx.doi.org/10.5281/zenodo.27878`, 2015.

[9] Derry FitzGerald, Robert Lawlor, and Eugene Coyle. Prior subspace analysis for drum transcription. In *Proc 114th Audio Engineering Soc Conf*, Amsterdam, Netherlands, March 2003.

[10] Derry FitzGerald, Antoine Liukus, Zafar Rafii, Bryan Pardo, and Laurent Daudet. Harmonic/percussive separation using kernel additive modelling. In *Irish Signals & Systems Conf and China-Ireland Intl Conf on Information and Communications Technologies*, pages 35–40, Limerick, Ireland, June 2014.

[11] Olivier Gillet and Gaël Richard. Drum loops retrieval from spoken queries. *Journal of Intelligent Information Systems*, 24(2-3):159–177, 2005.

[12] Olivier Gillet and Gaël Richard. Enst-drums: an extensive audio-visual database for drum signals processing. In *Proc 7th Intl Conf on Music Information Retrieval*, pages 156–159, Victoria, BC, Canada, October 2006.

[13] Olivier Gillet and Gaël Richard. Transcription and separation of drum signals from polyphonic music. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(3):529–540, 2008.

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proc 14th Intl Conf on Artificial Intelligence and Statistics*, pages 315–323, Ft. Lauderdale, FL, USA, April 2011.

[15] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5), 2009.

[16] Marko Helen and Tuomas Virtanen. Separation of drums from polyphonic music using non-negative matrix factorization and support vector machine. In *Proc 13th European Signal Processing Conf*, Antalya, Turkey, September 2005.

[17] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006.

[18] Kristoffer Jensen. Multiple scale music segmentation using rhythm, timbre, and harmony. *EURASIP Journal on Applied Signal Processing*, 2007(1):159–159, 2007.

[19] Ajay Kapur, Manj Benning, and George Tzanetakis. Query-by-beat-boxing: Music retrieval for the dj. In *Proc 5th Intl Conf on Music Information Retrieval*, pages 170–177, Barcelona, Spain, October 2004.

[20] David Krueger and Roland Memisevic. Regularizing rnns by stabilizing activations. In *Proc 4th Intl Conf on Learning Representations*, San Juan, Puerto Rico, May 2015.

[21] Henry Lindsay-Smith, Skot McDonald, and Mark Sandler. Drumkit transcription via convolutive nmf. In *Intl Conf on Digital Audio Effects*, York, UK, September 2012.

[22] Marius Miron, Matthew EP Davies, and Fabien Gouyon. Improving the real-time performance of a causal audio drum transcription system. In *Proc Sound and Music Computing Conf*, pages 402–407, Stockholm, Sweden, July 2013.

[23] Marius Miron, Matthew EP Davies, and Fabien Gouyon. An open-source drum transcription system for pure data and max msp. In *Proc IEEE Intl Conf on Acoustics, Speech and Signal Processing*, pages 221–225, Vancouver, BC, Canada, May 2013.

[24] Jouni Paulus and Anssi Klapuri. Drum sound detection in polyphonic music with hidden markov models. *EURASIP Journal on Audio, Speech, and Music Processing*, 2009.

[25] Haşim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proc 15th Annual Conf of the Intl Speech Communication Association*, pages 338–342, Singapore, September 2014.

[26] Simon Scholler and Hendrik Purwins. Sparse approximations for drum sound classification. *IEEE Journal of Selected Topics in Signal Processing*, 5(5):933–940, 2011.

[27] Siddharth Sigtia, Emmanouil Benetos, Nicolas Boulanger-Lewandowski, Tillman Weyde, Artur S d'Avila Garcez, and Simon Dixon. A hybrid recurrent neural network for music transcription. In *Proc IEEE Intl Conf on Acoustics, Speech and Signal Processing*, pages 2061–2065, Brisbane, Australia, April 2015.

[28] Paris Smaragdis. Non-negative matrix factor deconvolution; extraction of multiple sound sources from monophonic inputs. In *Independent Component Analysis and Blind Signal Separation*, volume 3195 of *Lecture Notes in Computer Science*, pages 494–499. Springer, Granada, Spain, September 2004.

[29] Andrio Spich, Massimiliano Zanoni, Augusto Sarti, and Stefano Tubaro. Drum music transcription using prior subspace analysis and pattern recognition. In *Proc 13th Intl Conf of Digital Audio Effects*, Graz, Austria, September 2010.

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, June 2014.

[31] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5rmsprop: Divide the gradient by a running average of its recent magnitude. In *COURSERA: Neural Networks for Machine Learning*, October 2012.

[32] Chih-Wei Wu and Alexander Lerch. Drum transcription using partially fixed non-negative matrix factorization with template adaptation. In *Proc 16th Intl Soc for Music Information Retrieval Conf*, pages 257–263, Málaga, Spain, October 2015.

[33] Kazuyoshi Yoshii, Masataka Goto, and Hiroshi G. Okuno. Automatic drum sound description for real-world music using template adaptation and matching methods. In *Proc 5th Intl Conf on Music Information Retrieval*, pages 184–191, Barcelona, Spain, October 2004.

5 IMPROVING GENERALIZATION CAPABILITIES

5.1 OVERVIEW

The contents of this chapter were published in the following paper:

Richard Vogl, Matthias Dorfer, and Peter Knees. "Drum Transcription from Polyphonic Music with Recurrent Neural Networks". In: *Proceedings of the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. New Orleans, LA, USA, 2017.

The focus of this work is on tackling drum transcription from more diverse audio material which also features other instruments alongside the drums. To this end, the previously established RNN-based methods are adapted to be able to deal with the more difficult data. To surpass performance of previous state-of-the-art methods on this data, a recurrent architecture featuring GRUs is used. Additionally, advanced deep learning methods are employed to improve the generalization capabilities of the trained models. In detail, data augmentation and dropout as well as an improved loss function which assigns different weights to the loss functions of individual instruments, are used. Again, in this work a label time shift for onset labels is used. Using a small shift around 30ms backwards in time for all onset labels allows the network to access required spectral information to classify instrument onsets. This is due to the fact that the sound, and thus spectral energy patterns, which characterize the individual instruments occur only after an onset. In bidirectional networks this information is naturally provided by the backward connections, but this comes at the cost of losing realtime capability.

The evaluation shows that unidirectional forward RNNs with additional label time shift can surpass state-of-the-art systems (NMF-based as well as BDRNN-based) when using data augmentation and dropout. While this was shown on simple, drum-only tracks in the previous work, in this work similar performance improvements compared to other systems could be achieved on more complex and realistic data. Analysis of the datasets reveals that certain drum instruments exhibit strong timbral variation over the data-splits for cross-validation, which proves to be a generalization challenge when evaluating in a three-fold cross-validation setup. Data augmentation has been shown to be an

effective way to circumvent this issue and improve generalization capabilities of the trained model.

5.2 CONTRIBUTIONS OF AUTHORS

As for the last work, I contributed most of this work's content: ideas for using data augmentation and more complex RNN architecture for drum transcription, preparation of datasets, updating the source code in python, designing and running experiments and evaluation, as well as writing most of the paper.

Matthias Dorfer's main contribution for this work was helping with designing the networks' architectures. He also provided support with writing the paper (proofreading, suggestions).

Peter Knees acted as supervisor for this submission: he provided feedback throughout the process of creating and conducting the experiments, and helped writing the paper, by proofreading and suggesting improvements.

DRUM TRANSCRIPTION FROM POLYPHONIC MUSIC WITH RECURRENT NEURAL NETWORKS

Richard Vogl,[1,2] *Matthias Dorfer,*[1] *Peter Knees* [2]

[1] Dept. of Computational Perception, Johannes Kepler University Linz, Austria
[2] Institute of Software Technology & Interactive Systems, Vienna University of Technology, Austria
richard.vogl@jku.at, matthias.dorfer@jku.at, knees@ifs.tuwien.ac.at

ABSTRACT

Automatic drum transcription methods aim at extracting a symbolic representation of notes played by a drum kit in audio recordings. For automatic music analysis, this task is of particular interest as such a transcript can be used to extract high level information about the piece, e.g., tempo, downbeat positions, meter, and genre cues. In this work, an approach to transcribe drums from polyphonic audio signals based on a recurrent neural network is presented. Deep learning techniques like dropout and data augmentation are applied to improve the generalization capabilities of the system. The method is evaluated using established reference datasets consisting of solo drum tracks as well as drums mixed with accompaniment. The results are compared to state-of-the-art approaches on the same datasets. The evaluation reveals that F-measure values higher than state of the art can be achieved using the proposed method.

Index Terms— Drum transcription, neural networks, deep learning, automatic transcription, data augmentation

1. INTRODUCTION

The goal of automatic drum transcription (ADT) systems is to create a symbolic representation of the drum instrument onsets contained in monaural audio signals. A reliable ADT system has many applications in different fields like music production, music education, and music information retrieval. Good transcription results can be achieved on simple solo drum tracks [1], but for complex mixtures in polyphonic audio, the problem is still not solved satisfactorily. In this work, a robust method to transcribe solo drum tracks using RNNs [2] is further extended to be applicable on polyphonic audio.

As in other work (e.g. [3, 1, 4, 5, 6]), the transcribed instrument classes are limited to the three main instruments used in most drum kits: bass drum, snare drum, and hi-hat. This is a reasonable simplification as these three classes usually suffice to capture the main rhythm patterns of the drum track [3] and cover most of the played notes in full drum kit recordings.[1]

[1] >80% in the case of the ENST-Drums [7] dataset, see sec. 4.1.

2. RELATED WORK

The majority of ADT methods can be categorized into three classes: *i. segment and classify*, *ii. match and adapt*, and *iii. separate and detect* methods (cf. [9]).

Segment and classify methods first segment the audio signal using, e.g., onset detection and then classify the resulting fragments regarding contained drum instruments [8, 9]. Miron et al. use a combination of frequency filters, onset detection and feature extraction in combination with a k-nearest-neighbor [10] and a k-means [8] classifier to detect drum sounds in a solo drum audio signal in real-time.

Match and adapt methods use temporal or spectral templates of the individual drum sounds to detect the events. These templates are iteratively adapted during classification to better match the events in the input signal. Yoshii et al. [11] present an ADT system based on template matching and adaptation incorporating harmonic suppression.

The *separate and detect* methods utilize source separation techniques to separate the drum sounds from the mix. Subsequently the onsets for the individual drums are detected. The most successful technique in this context is non-negative matrix factorization (NMF). Dittmar and Gärtner [1] use an NMF approach for a real-time ADT system of solo drum tracks. Their approach utilizes training instances for the individual drum instrument of each track.

Additionally there are methods which combine techniques from these categories. Hidden Markov Models (HMMs) can be used to perform segmentation and classification in one step. Paulus and Klapuri [3] use HMMs to model the development of MFCCs over time and incorporate an unsupervised acoustic model adaptation. Decoding the most likely sequence yields activation curves for bass drum, snare drum, and hi-hat and can be applied for both solo drum tracks as well as polyphonic music. Wu and Lerch [5] use an extension of NMF, the so-called partially fixed NMF (PFNMF), for which they also evaluate two different template adaptation methods.

Artificial neural networks represent a powerful machine learning technique which is being successfully applied in many different fields. Recurrent neural networks (RNNs) are neural networks with additional connections (recurrent connections) in each layer, providing the outputs of the same

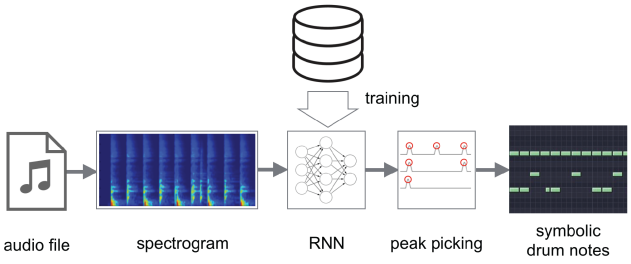


Fig. 1. Overview of the proposed method. The extracted spectrogram is fed into the trained RNN which outputs activation functions for each instrument. A peak picking algorithms selects appropriate peaks as instrument onset candidates.

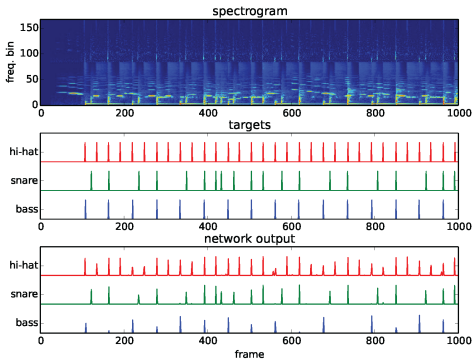


Fig. 2. Spectrogram of a drum track with accompaniment (top) and target functions for bass drum, snare drum, and hi-hat (middle). The target functions have a value of 1.0 for the frames which correspond to the annotations of the individual instruments and 0.0 otherwise. The frame rate of the target function is 100Hz, the same as for the spectrogram. The third plot (bottom) shows the output of a trained RNN for the spectrogram in the top plot.

layer from the last time step as additional inputs. These recurrent connections serve as a kind of memory which is beneficial for tasks with sequential input data. For example, RNNs have been shown to perform well for speech recognition [12] and handwriting recognition [13].

RNNs have several advantages in the context of automatic music transcription. As shown in the context of automatic piano transcription by Böck and Schedl [14], RNNs are capable of handling many different classes better than NMF. This becomes particularly relevant when classifying pitches (typically up to 88) [14] or many instruments. Southall et al. [15] apply bidirectional RNNs (BDRNNs) for ADT and demonstrate the capability of RNNs to detect snare drums in polyphonic audio better than state of the art. In [2], we show that time-shift RNNs (tsRNNs) perform as well as BDRNNs when used for ADT on solo drum tracks, while maintaining online capability and also demonstrate the generalization capabilities of RNNs in the context of ADT. In the present work, this method is further developed into an online-capable ADT system for polyphonic audio, which further improves the state of the art.

3. METHOD

Fig. 1 shows an overview of the proposed method. First, the input features derived from the power spectrogram of the audio signal are calculated. The result is frame-wise fed into an RNN with three output neurons. The outputs of the RNN provide activation signals for the three drum instruments considered (bass drum, snare drum, and hi-hat). A peak picking algorithm then identifies the onsets for each instrument's activation function, which yields the finished transcript. The next sections will cover the individual steps of the method in detail.

3.1. Feature Extraction

As input, mono audio files with 16 bit resolution at 44.1kHz sampling rate are used. The audio is normalized and padded with 0.25 seconds of silence at the start to avoid undesired artifacts resulting from onsets which occur immediately at the beginning. A logarithmic power spectrogram is calculated using a 2048-samples window size and a resulting frame rate of 100Hz. The frequency axis is transformed to a logarithmic scale using twelve triangular filters per octave over a frequency range from 20 to 20,000 Hz. This results in a total number of 84 frequency bins. Additionally the positive first-order-differential over time of this spectrogram is calculated. The resulting differential-spectrogram-frames are stacked on top of the normal spectrogram frames, resulting in feature vectors with a length of 168 (2x84) values.

3.2. Recurrent Neural Network

In this work, a two-layer RNN architecture with label time shift is used (tsRNN). It has been shown that these networks perform as well as BDRNNs on solo drum tracks, while having the advantage of being online capable [2]. The RNN features a 168 node input layer which is needed to handle the input data vectors of the same size. Two recurrent layers, consisting of 50 gated recurrent units (GRUs [16]) each, follow. The connections between the input and the recurrent layers, the recurrent connections, as well as the connections between the recurrent layer and the next layer are all realized densely (every node is connected to all other nodes). A so-called dropout layer [17] is situated between the recurrent and the output layer. In this layer, connections are randomly disabled for every iteration during training. This helps preventing overfitting to the training data. The amount of disabled connections is controlled by the dropout rate, which was set to $r_d = 0.3$. The output layer consists of three nodes with sigmoid transfer functions, which output the activation functions for the three instrument classes defined earlier.

Label time shift refers to the process of shifting the orig-

inal annotations. After transcription, the detected onsets are shifted back by the same time. In doing so, the RNN can also take a small portion of the sustain phase of the onset's spectrogram into account. The used delay of $30ms$ (corresponds to three spectrogram frames) in this work is still sufficiently small for certain applications like score following and other visualizations, while it can be tuned to meet the demands of other applications.

3.3. Peak Picking

The neurons of the output layer generate activation functions for the individual instruments (see fig. 2). The instrument onsets are identified using the same peak picking method as in [2]: A point n in the function $F(n)$ is considered a peak if these terms are fulfilled:

1. $F(n) = max(F(n-m), \cdots, F(n))$,
2. $F(n) \geq mean(F(n-a), \cdots, F(n)) + \delta$,
3. $n - n_{lp} > w$,

where δ is a variable threshold. A peak must be the maximum value within a window of size $m+1$, and exceeding the mean value plus a threshold within a window of size $a+1$. Additionally, a peak must have at least a distance of $w+1$ to the last detected peak (n_{lp}). Values for the parameters were tuned on a development dataset to be: $m = a = w = 2$.

3.4. RNN Training

When fed with the features at the input nodes, the RNN should reproduce the activation functions of the individual instruments at the output neurons. During training, the update function adapts parameters of the network (weights and biases of neurons) using the calculated error (loss) and the gradient through the network. As update function, the *rmsprop* method is used [18].

As loss function, the mean of the binary cross-entropy between outputs of the network and target functions is used (see fig. 2). Snare drum and hi-hat onsets are considered more difficult to transcribe than bass drum [3, 5, 15]. Due to this fact, the loss functions of the output neurons for bass drum (1.0), snare drum (4.0), and hi-hat (1.5) are weighted differently. This way, errors for snare drum and hi-hat are penalized more, which forces the training to focus on them.

RNN training using *rmsprop* involves so-called mini-batches. In this work, a mini-batch consists of eight training instances. The training instances are obtained by cutting the extracted spectrograms into 100-frame-segments with 90 frames overlap. The order of the segments for training is randomized. To further increase generalization, data-augmentation [19] is used. The training instances are randomly augmented using pitch shift (-5 to $+10$ frequency bins) and time stretching (scale factors: 0.70, 0.85, 1.00, 1.30, 1.60).

Training is structured into epochs, during which the training data is used to optimize the parameters of the network. At the end of an epoch a validation set (25% excluded from the training set) is used to estimate the performance of the trained network on data not used for training. The training of the RNN is aborted as soon as the resulting loss on the validation set has not decreased for 10 epochs. The initial learning rate was set to $r_l = 0.007$, the learning rate is reduced to a fifth every 7 epochs.

All hyperparameters like network architecture, dropout rate, augmentation parameters, and learning rate were chosen accordingly to experiments on a development dataset, experience, and best practice examples.

4. EVALUATION

The well-known metrics precision, recall, and F-measure are used to evaluate the performance of the presented system. True positive, false positive, and false negative onsets are identified by using a $20ms$ tolerance window. It should be noted that state-of-the-art methods for the ENST-Drums dataset [3] as well as for the IDMT-SMT-Drums dataset [1], use less strict tolerance windows of $30ms$ and $50ms$, respectively, for evaluation. However, listening experiments showed that distinct events with a delay of $50ms$ are already perceivable. Therefore, in this work, $20ms$ windows are used.

4.1. Datasets

For evaluation, two well-known datasets are used. The IDMT-SMT-Drums [1] contains recorded (*RealDrum*), synthesized (*TechnoDrum*), and sampled (*WaveDrum*) drum tracks. It comprises 560 files of which 95 are simple drum tracks (of approx. 15sec). The rest are single-instrument training tracks.

As second dataset the ENST-Drums set [7] is used. The dataset consists of real drum recordings of three drummers performing on three different drum kits. The recordings are available as solo instrument tracks and as two mixtures (dry and wet). For a subset, accompaniment tracks are included (minus-one tracks). The total length of the recorded material is roughly 75 minutes per drummer. In this work, the wet mixes of the minus-one tracks plus accompaniment of all three drummers were used. Since the ENST-Drums dataset contains more than the three main instruments, only the snare, bass, and hi-hat annotations were used. 81.2% of onsets are annotated as snare drum, bass drum, and hi-hat while the remaining 18.8% cover other cymbals and tom-tom drums.

4.2. Experiments

The proposed method was evaluated in four different experiments. These were performed using *i.* the drum tracks of the IDMT-SMT-drums dataset (SMT solo), *ii.* the minus-one tracks of the ENST-drums dataset without accompaniment (ENST solo), and *iii.* the minus-one tracks mixed with accompaniment of aforementioned (ENST acc.). In the experiments, on *SMT solo* a three-fold cross validation on the

F-measure [%] for individual methods on datasets

Method	SMT solo		ENST solo	ENST acc.
NMF [1]	—	(95.0)	—	—
PFNMF[5]	81.6	(—)	77.9	72.2
HMM [3]	—	(—)	81.5	74.7
BDRNN [15]	83.3	(96.1)	73.2	66.9
tsRNN	**92.5**	**(96.6)**	**83.3**	**75.0**

Table 1. Top four rows show results of state-of-the-art algorithms. Highest values were achieved at peak picking thresholds of 0.10 and 0.15 (*ENST solo*, *SMT solo opt.* cf. fig. 3). Values in brackets represent results for optimized models (*SMT solo opt.* see sec. 4.2).

three splits (*RealDrum*, *TechnoDrum*, and *WaveDrum*) of the dataset was performed (comparable to the *automatic* experiment in [15] and [5]). Additionally a six-fold cross validation on six randomized splits was performed (*SMT solo opt.*). This task is comparable to the *semi-automatic* experiments in [15], and [1]—it is arguably a even harder task, since in a model is trained on more than the training data of just one track. In both cases the corresponding splits of the training tracks are additionally used only for training.

In the case of *ENST solo* and *ENST acc.* the dataset was split into three parts consisting of the tracks of one drummer and a three-fold cross validation was performed. Training for each fold was performed on all tracks of two drummers while testing was done on the minus-one tracks (without and with accompaniment resp.) of the third drummer and thus on unseen data. This is consistent with the experiments performed in [3, 5, 15].

5. RESULTS

Tab. 1 summarizes the results of the presented method and state-of-the-art methods on the used datasets. It can be seen that the F-measure values for the tsRNN approach are higher than the state of the art for *SMT solo* and *ENST solo*, and on the same level for *ENST acc.*Since for training, both tracks with and without accompaniment were used, the same models are applied to *ENST solo* and *ENST acc.* splits, which further demonstrates the capability of the presented method to generalize well. Fig. 3 shows F-measure and precision-recall curves for the cross-validation results on the individual datasets. For these curves the threshold level for peak picking was varied in the range 0.05 to 0.95 using steps of 0.05. It can be seen that the highest F-measure values are found for threshold values of 0.10 and 0.015, which is lower than the expected value of around 0.5 (target functions range is 0–1). This is due to the fact that the output of the RNN does not contain much noise (see fig. 2), which implies that the trained RNN is capable of effectively filtering accompaniment.

Since the target functions contain little noise while strong peaks are present for instrument onsets, only little time was

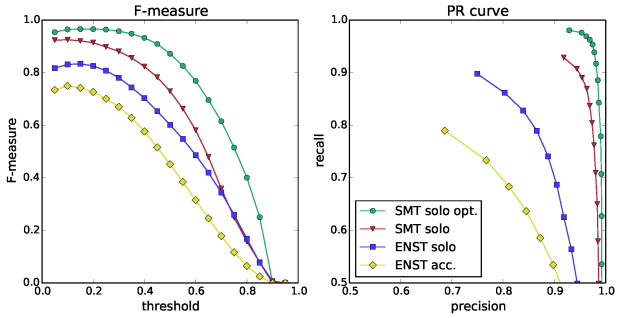


Fig. 3. Results of the evaluation on the individual datasets. Left plot shows F-measure curve, right plot precision-recall curves for different threshold levels (δ) for peak picking. Best results were achieved at thresholds of 0.10 and 0.15.

invested optimizing peak picking. Noticeable improvements to [2] were achieved by using data augmentation and GRUs instead of RNN units for the network.

6. CONCLUSION

In this work, an approach for drum transcription from solo drum tracks and polyphonic music was introduced. The proposed method uses an RNN with two recurrent layers consisting of GRUs in combination with label time shift and introduces loss function weighting for the individual instruments to increase transcription performance. Additionally dropout and data augmentation are successfully applied to overcome overfitting to the individual drum sounds in the different dataset splits. The presented system is online capable with a latency of $30ms$ introduced by the label time shift.

In contrast to hand-crafted systems and features, where the architecture is often difficult to adapt when shortcomings are detected, RNNs have shown to be more flexible. A major advantage of such a technique is that the system can be focused on training instances on which the model previously failed. In table 1 it can be seen that RNNs are capable of learning to filter accompaniment and perform well also on polyphonic music. It has been shown that the transcription F-measure performance of the proposed method is higher than the results of state-of-the-art approaches, even when using a more stringent tolerance window for evaluation.

7. ACKNOWLEDGMENTS

This work has been partly funded by the EU's seventh Framework Programme FP7/2007-13 for research, technological development and demonstration under grant agreement no. 610591 (*GiantSteps*) and by the Austrian FFG under the BRIDGE 1 project *SmarterJam* (858514), We gratefully acknowledge the support of the NVIDIA Corporation by donating one Titan Black GPU for research purposes.

8. REFERENCES

[1] Christian Dittmar and Daniel Gärtner, "Real-time transcription and separation of drum recordings based on NMF decomposition," in *Proc 17th International Conference on Digital Audio Effects*, Erlangen, Germany, Sept. 2014.

[2] Richard Vogl, Matthias Dorfer, and Peter Knees, "Recurrent neural networks for drum transcription," in *Proc 17th International Society for Music Information Retrieval Conference*, New York, NY, USA, Aug. 2016.

[3] Jouni Paulus and Anssi Klapuri, "Drum sound detection in polyphonic music with hidden markov models," *EURASIP Journal on Audio, Speech, and Music Processing*, 2009.

[4] Andrio Spich, Massimiliano Zanoni, Augusto Sarti, and Stefano Tubaro, "Drum music transcription using prior subspace analysis and pattern recognition," in *Proc 13th International Conference of Digital Audio Effects*, Graz, Austria, Sept. 2010.

[5] Chih-Wei Wu and Alexander Lerch, "Drum transcription using partially fixed non-negative matrix factorization with template adaptation," in *Proc 16th International Society for Music Information Retrieval Conference*, Málaga, Spain, Oct. 2015.

[6] Derry FitzGerald, Robert Lawlor, and Eugene Coyle, "Prior subspace analysis for drum transcription," in *Proc 114th Audio Engineering Society Conference*, Amsterdam, Netherlands, Mar. 2003.

[7] Olivier Gillet and Gaël Richard, "Enst-drums: an extensive audio-visual database for drum signals processing," in *Proc 7th International Conference on Music Information Retrieval*, Victoria, BC, Canada, Oct. 2006.

[8] Marius Miron, Matthew EP Davies, and Fabien Gouyon, "Improving the real-time performance of a causal audio drum transcription system," in *Proc Sound and Music Computing Conference*, Stockholm, Sweden, July 2013.

[9] Olivier Gillet and Gaël Richard, "Transcription and separation of drum signals from polyphonic music," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 3, 2008.

[10] Marius Miron, Matthew EP Davies, and Fabien Gouyon, "An open-source drum transcription system for pure data and max msp," in *Proc IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, Canada, May 2013.

[11] Kazuyoshi Yoshii, Masataka Goto, and Hiroshi G Okuno, "Drum sound recognition for polyphonic audio signals by adaptation and matching of spectrogram templates with harmonic structure suppression," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 1, 2007.

[12] Haşim Sak, Andrew W. Senior, and Françoise Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proc 15th Annual Conference of the International Speech Communication Association*, Singapore, Sept. 2014.

[13] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber, "A novel connectionist system for improved unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, 2009.

[14] Sebastian Böck and Markus Schedl, "Polyphonic piano note transcription with recurrent neural networks," in *Proc IEEE International Conference on Acoustics, Speech and Signal Processing*, Kyoto, Japan, Mar. 2012.

[15] Carl Southall, Ryan Stables, and Jason Hockman, "Automatic drum transcription using bidirectional recurrent neural networks," in *Proc 17th International Society for Music Information Retrieval Conference*, New York, NY, USA, Aug. 2016.

[16] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," `http://arxiv.org/abs/1409.1259`, 2014.

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, June 2014.

[18] Tijmen Tieleman and Geoffrey Hinton, "Lecture 6.5rmsprop: Divide the gradient by a running average of its recent magnitude," in *COURSERA: Neural Networks for Machine Learning*, Oct. 2012.

[19] Jan Schlüter and Thomas Grill, "Exploring data augmentation for improved singing voice detection with neural networks," in *Proc 16th International Society for Music Information Retrieval Conference*, Málaga, Spain, Oct. 2015.

6 META INFORMATION FOR DRUM TRANSCRIPTS

6.1 OVERVIEW

The contents of this chapter were published in the following work:

Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees. "Drum Transcription via Joint Beat and Drum Modeling using Convolutional Recurrent Neural Networks". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017.

In this chapter, a shortcoming of previous works is addressed: systems introduced so far only focus on drum instrument onset times. However, depending on the application, additional data which has to be extracted from the audio signal may be required to create the desired transcripts. This metadata can, for example, comprise: bar lines, global and local tempo, time signature, dynamics, and playing technique for the application of sheet music extraction. Important metadata that is required for many different applications are the bar lines (bar boundaries), time signature, and local tempo. This data can be obtained by using a downbeat and beat tracking system. To this end, in the publication for this chapter, a beat tracking system is trained alongside the ADT system in a multi-task fashion. This is done since beats and drum instrument onsets are usually highly correlated, and it can be assumed that training on correlated data in a multi-task fashion will further improve the individual performances on the subtasks.

Additionally, in this work, CNNs and CRNNs for ADT are introduced. The motivation for using convolutions and combining convolutional with recurrent layers for a drum transcription system is as follows: convolutional layers, which process small subsections of the input spectrograms, are well suited to identify the percussive onsets, which are represented by temporally localized energy bursts. By adding recurrent layers after the convolutional stack, the network has the means to model long term temporal structures, like rhythmic patterns and beats.

For evaluation, an RNN system similar to previously introduced ones, and the newly introduced CNN-based and CRNN-based ADT systems are compared. In previous experiments it had been shown that bidirectional architectures do not necessarily provide an edge in performance if a label time shift is used. However, a desirable behavior of RNNs, especially in the context of beat detection, would be to consider

global rhythmical structures. Since the only main advantage of using label time shift over bidirectional networks is gaining realtime capability, a switch to bidirectional architectures was made in this work. Additionally to the beat tracking and drum transcription performance evaluation, an evaluation of performance differences when using *(i)* multi-task training for beats and drums, *(ii)* annotated beat information as additional input features, and *(iii)* no beat information, was performed.

The evaluation reveals that beats and downbeat information is beneficial for drum transcription and that training on beats and drums in a multi-task fashion can improve the drum transcription performance for recurrent architectures. Additionally, the results show that CRNNs perform better than simple RNN networks, while CNNs exhibit a comparable performance for ADT on the used datasets.

6.2 CONTRIBUTIONS OF AUTHORS

As first author, I contributed the largest part of this work's content. Evaluating CNNs was, on one hand, long due, because of their success in many other tasks. On the other hand, we knew that detecting bass and snare drum with CNNs works reasonably well from unpublished experiments conducted by Matthias Leimeister. Using CRNNs, the idea of using beat tracking to add meta information, as well as using multi-task learning to improve performance was the logical consequence of previous work and the other ideas for this work. Besides that, I again prepared the datasets, updated the experiment source code, designed and ran experiments, as well as wrote most of the paper.

For this work, Matthias Dorfer contributed a special universal batch-iterator which can prepare data in the format required for RNN, CNN, as well as CRNN training. Additionally, he helped with designing the CNN and CRNN network architecture. He supported me with writing the paper by helping with the title, abstract, and proofreading.

Gerhard Widmer and Peter Knees acted as supervisors for this work. They both provided valuable feedback, and helped to improve the paper.

As mentioned in the acknowledgments, Mark Übel and Jo Thalmayer, both Red Bull Music Academy alumni, provided the basis for drum annotations for the used multi-task dataset. Sebastian Böck helped with the beat and downbeat annotations, and also provided some valuable input for beat tracking in general. All annotations of the data used in this work were double-checked and corrected by myself.

DRUM TRANSCRIPTION VIA JOINT BEAT AND DRUM MODELING USING CONVOLUTIONAL RECURRENT NEURAL NETWORKS

Richard Vogl[1,2] **Matthias Dorfer**[2] **Gerhard Widmer**[2] **Peter Knees**[1]
[1] Institute of Software Technology & Interactive Systems, Vienna University of Technology, Austria
[2] Dept. of Computational Perception, Johannes Kepler University Linz, Austria
{richard.vogl, peter.knees}@tuwien.ac.at

ABSTRACT

Existing systems for automatic transcription of drum tracks from polyphonic music focus on detecting drum instrument onsets but lack consideration of additional meta information like bar boundaries, tempo, and meter. We address this limitation by proposing a system which has the capability to detect drum instrument onsets along with the corresponding beats and downbeats. In this design, the system has the means to utilize information on the rhythmical structure of a song which is closely related to the desired drum transcript. To this end, we introduce and compare different architectures for this task, i.e., recurrent, convolutional, and recurrent-convolutional neural networks. We evaluate our systems on two well-known data sets and an additional new data set containing both drum and beat annotations. We show that convolutional and recurrent-convolutional neural networks perform better than state-of-the-art methods and that learning beats jointly with drums can be beneficial for the task of drum detection.

1. INTRODUCTION

The automatic creation of symbolic transcripts from music in audio files is an important high-level task in music information retrieval. Automatic music transcription systems (AMT) aim at solving this task and have been proposed in the past (cf. [1]), but there is yet no general solution to this problem. The transcription of the drum instruments from an audio file of a song is a sub-task of automatic music transcription, called automatic drum transcription (ADT). Usually, such ADT systems focus solely on the detection of drum instrument note onsets. While this is the necessary first step, for a full transcript of the drum track more information is required. Sheet music for drums—equally to sheet music for other instruments—contains additional information required by a musician to perform a piece. This information comprises (but is not limited to): meter, overall tempo, indicators for bar boundaries, indications for local changes in tempo, dynamics, and playing style of the piece. To obtain some of this information, beat and downbeat detection methods can be utilized. While beats provide tempo information, downbeats add bar boundaries, and the combination of both provides indication for the meter within the bars.

In this work, neural networks for joint beat and drum detection are trained in a multi-task learning fashion. While it is possible to extract drums and beats separately using existing work and combine the results afterwards, we show that it is beneficial to train for both tasks together, allowing a joint model to leverage commonalities of the two problems. Additionally, recurrent (RNN), convolutional (CNN) and convolutional-recurrent neural network (CRNN) models for drum transcription and joint beat and drum detection are evaluated on two well-known, as well as a new data set.

The remainder of this work is structured as follows. In the next section, we discuss related work. In sec. 3, we describe the implemented drum transcription pipeline used to evaluate the network architectures, followed by a section discussing the different network architectures (sec. 4). In sec. 5, we explain the experimental setup to evaluate the joint learning approach. After that, a discussion of the results follows in sec. 6 before we draw conclusions in sec. 7.

2. RELATED WORK

While in the past many different approaches for ADT have been proposed [11,13,15,16,22,24,25,34,38], recent work focuses on end-to-end approaches calculating activation functions for each drum instrument. These methods utilize non-negative matrix factorization (NMF, e.g. adaptive-NMF in Dittmar et al. [7] and partially fixed NMF in Wu et al. [37]) as well as RNNs (RNNs with label time-shift in Vogl et al. [35, 36] and bidirectional RNNs in Southall et al. [31]) to extract the activation functions from spectrograms of the audio signal. Such activation-function-based end-to-end ADT systems circumvent certain issues associated with other architectures. Methods which first segment the song (e.g. using onset detection) and subsequently classify these segments [22, 23, 38] suffer from a loss of information after the segmentation step—i.e. whenever the system fails to detect a segment, this information is lost. Such systems heavily depend on the accuracies of the single components, and can never perform better than the weakest component in the pipeline. Additionally, information of the input signal which is discarded after a processing step might still be of value for later steps.

Since RNNs, especially long short-term memory (LSTM) [17] and gated recurrent unit (GRU) [5] networks, are designed to model long term relationships, one might suspect that systems based on RNNs [31, 35, 36] can leverage the repetitive structure of the drum tracks and make use of this information. Contrary to this intuition this is not the case for RNN-based systems proposed so far. Both the works of Vogl et al. [35, 36] and Southall et al. [31] use snippets with length of only about one second to train the RNNs. This prohibits learning long-term structures of drum rhythms which are typically in the magnitude of two or more seconds. In [35], it has been shown that RNNs with time-shift perform equally well as bidirectional RNNs, and that backward directional RNNs perform better than forward directional RNNs. Combining these findings indicates that the learned models actually mostly consider local features. Therefore, RNNs trained in such a manner seem to learn only an acoustic, but not a structural model for drum transcription.

Many works on joint beat and downbeat tracking have been published in recent years [2, 9, 10, 19–21, 26]. A discussion of all the different techniques would go beyond the scope of this work. One of the most successful methods by Böck et al. [2] is a joint beat and downbeat tracking system using bidirectional LSTM networks. This approach achieves top results in the 2016 MIREX task for beat detection and can be considered the current state of the art. [1]

In this work, a multi-task learning strategy is used to address the discussed issues of current drum transcription systems, cf. [4]. The use of a model jointly trained on drum and beat annotations, combined with longer training snippets, allows the model to learn long-term relations of the drum patterns in combination with beats and downbeats. Furthermore, learning multiple related tasks simultaneously at once can improve results for the single tasks. To this end, different architectures of RNNs, CNNs, and a combination of both, convolutional-recurrent neural networks (CRNNs) [8, 27, 39], are evaluated.

The rationale behind selecting these three methods for comparison is as follows. RNNs have proven to be well-suited for both drum and beat detection, as well as learning long-term dependencies for music language models [30]. CNNs are among the best performing methods for many image processing and other machine learning tasks, and have been used on spectrograms of music signals in the past. For instance, Schlüter and Böck [28] use CNNs to improve onset detection results, while Gajhede et al. [12] use CNNs to successfully classify samples of three drum sound classes on a non-public data set. CRNNs should result in a model, in which the convolutional layers focus on acoustic modeling of the events, while the recurrent layers learn temporal structures of the features.

3. DRUM TRANSCRIPTION PIPELINE

The implemented method is an ADT system using a similar pipeline as presented in [31] and [36]. Fig. 1 visualizes the overall structure of the system. The next subsections discuss the single blocks of the system in more detail.

[1] http://www.music-ir.org/mirex/wiki/2016:MIREX2016_Results

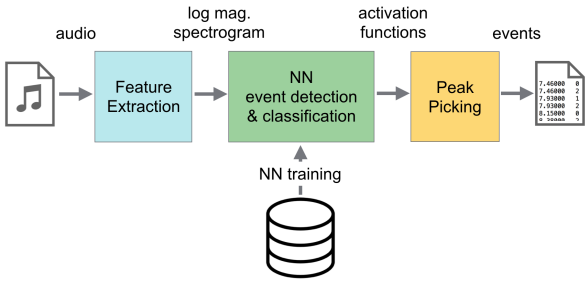


Figure 1. System overview of the implemented drum transcription pipeline used to evaluate the different neural network architectures.

3.1 Feature Extraction

First, a logarithmic magnitude spectrogram is calculated using a 2048-samples window size and a resulting frame rate of 100Hz from a 44.1kHz 16bit mono audio signal input. Then, the frequency bins are transformed to a logarithmic scale using triangular filters (twelve per octave) in a frequency range from 20 to 20,000 Hz. Finally, the positive first-order-differential over time of this spectrogram is calculated and concatenated. This results in feature vectors with a length of 168 values (2x84 frequency bins).

3.2 Activation Function Calculation

The central block in fig. 1 represents the activation function calculation step. This task is performed using a neural network (NN) trained on appropriate training data (see sec. 4). As in most of the related work, we only consider three drum instruments: bass- or kick drum, snare drum, and hi-hat.

While the architectures of the single NNs are different, they share certain commonalities: *i.* all NNs are trained using the same input features; *ii.* the RNN architectures are implemented as bidirectional RNNs (BRNN) [29]; *iii.* the output layers consist of three or five sigmoid units, representing three drum instruments under observation (drum only) or three drum instruments plus beat and downbeat (drum and beats), respectively; and *iv.* the NNs are all trained using the RMSprop optimization algorithm proposed by Tieleman et al. [33], using mini-batches of size eight. For training, we follow a three-fold cross validation strategy on all data sets. Two splits are used for training, 15% of the training data is separated and used for validation after each epoch, while testing/evaluation is done on the third split. The NNs are trained using a fixed learning rate with additional refinement if no improvement on the validation set is achieved for 10 epochs. During refinement the learning rate is reduced and training continues using the parameters of the best performing model so far.

More details on the individual NN architectures are provided in sec. 4.

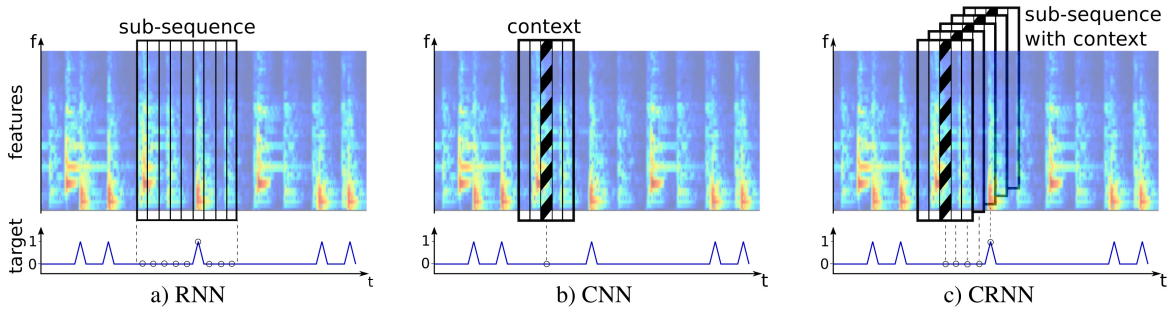


Figure 2. Comparison of mode of operation of RNNs, CNNs, and CRNNs on spectrograms of audio signals. RNNs process the input in a sequential manner. Usually, during training, only sub-sequences of the input signal are used to reduce the memory footprint of the networks. CNNs process the signal frame by frame without being aware of sequences. Because of this, a certain spectral context is added for each input frame. CRNNs, like RNNs, process the input sequentially, but additionally, a spectral context is added to every frame on which convolution is performed by the convolutional layers.

3.3 Preparation of Target Functions

For training the NNs, target functions of the desired output are required besides the input features. These target functions are generated by setting frames of a signal with the same frame rate as the input features to 1 whenever an annotation is present and to 0 otherwise. A separate target function is created for each drum instrument as well as for beats and downbeats.

3.4 Peak Picking

In the last step of our pipeline (rightmost block of fig. 1), the drum instrument onsets (and beats if applicable) are identified using a simple peak picking method introduced for onset detection in [3]: A point n in the activation function $f_a(n)$ is considered a peak if these terms are fulfilled:

1. $f_a(n) = max(f_a(n-m), \cdots, f_a(n))$,
2. $f_a(n) \geq mean(f_a(n-a), \cdots, f_a(n)) + \delta$,
3. $n - n_{lp} > w$,

where δ is a variable threshold. A peak must be the maximum value within a window of size $m+1$, and exceeding the mean value plus a threshold within a window of size $a+1$. Additionally, a peak must have at least a distance of $w+1$ to the last detected peak (n_{lp}). Values for the parameters were tuned on a development data set to be: $m = a = w = 2$.

The threshold for peak picking is determined on the validation set. Since the activation functions produced by the NN contain little noise and are quite spiky, rather low thresholds $(0.1 - 0.2)$ give best results.

4. NEURAL NETWORK MODELS

In this section, we explore the properties of the neural network models considered more closely. Of the NN categories mentioned before, we investigate three different types: bidirectional recurrent networks (BRNN), convolutional networks (CNN), and convolutional bidirectional recurrent networks (CBRNN). For every class of networks, two different architectures are implemented: *i.* a smaller network, with less capacity, trained on shorter subsequences (with focus only on acoustic modeling), and *ii.* a larger network, trained on longer subsequences (with additional focus on pattern modeling).

Even though we previously showed that RNNs with label time-shift achieve similar performance as BRNNs [35, 36], in this work, we will not use time-shift for target labels. This is due to three reasons: *i.* the focus of this work is not real-time transcription but a comparison of NN architectures and training paradigms, therefore using a bidirectional architecture has no downsides; *ii.* it is unclear how label time-shift would affect CNNs; *iii.* in [2], the effectiveness of BRNNs (BLSTMs) for beat and downbeat tracking is shown. Thus, in the context of this work, using BRNNs facilitates combining state-of-the-art drum and beat detection methods while allowing us to compare CNNs and RNNs in a fair manner.

4.1 Bidirectional Recurrent Neural Network

Gated recurrent units (GRUs [5]) are similar to LSTMs in the sense that both are gated RNN-cell types that facilitate learning of long-term relations in the data. While LSTMs feature forget, input, and output gates, GRUs only exhibit two gates: update and output. This makes the GRU less complex in terms of number of parameters. It has been shown that both are equally powerful [6], with the difference that more GRUs are needed in an NN layer to achieve the same model capacity as with LSTMs, resulting in more or less equal number of total parameters. An advantage of using GRUs is that hyperparameter optimization for training is usually easier compared to LSTMs.

In this work, two bidirectional GRU (BGRU) architectures are used. The small model (BGRU-a) features two layers of 50 nodes each, and is trained on sequences of 100 frames; the larger model (BGRU-b) consists of three layers of 30 nodes each, and is trained on sequences of 400 frames. For training an initial learning rate of 0.007 is used.

	Frames	Context	Conv. Layers	Rec. Layers	Dense Layers
BGRU-a	100	—	—	2 x 50 GRU	—
BGRU-b	400	—	—	3 x 30 GRU	—
CNN-a	—	9	1xA + 1xB	—	2 x 256
CNN-b	—	25	1xA + 1xB	—	2 x 256
CBGRU-a	100	9	1xA + 1xB	2 x 50 GRU	—
CBGRU-b	400	13	1xA + 1xB	3 x 60 GRU	—

Table 1. Overview of used neural network model architectures and parameters. Every network additionally contains a dense sigmoid output layer. Conv. block A consists of 2 layers with 32 3x3 filters and 3x3 max-pooling; conv. block B consists of 2 layers with 64 3x3 filters and 3x3 max-pooling; both use batch normalization.

4.2 Convolutional Neural Network

Convolutional neural networks have been successfully applied not only in image processing, but also many other machine learning tasks. The convolutional layers are constructed using two different building blocks: block *A* consists of two layers with 32 3x3 filters and block *B* consists of two layers with 64 3x3 filters; both in combination with batch normalization [18], and each followed by a 3x3 max pooling layer and a drop-out layer ($\lambda = 0.3$) [32].

For both CNN models, block *A* is used as input, followed by block *B*, and two fully connected layers of size 256. The only difference between the small (CNN-a) and the large (CNN-b) model is the context used to classify a frame: 9 and 25 frames are used for CNN-a and CNN-b respectively. While plain CNNs do not feature any memory, the spectral context allows the CNN to access surrounding information during training and classification. However, a context of 25 frames (250ms) is not enough to find repetitive structures in the rhythm patterns. Therefore, the CNN can only rely on acoustic, i.e., spectral features of the signal. Nevertheless, with advanced training methods like batch normalization, as well as the advantage that CNNs can easily learn pitch invariant kernels, CNNs are well-equipped to learn a task adequate acoustic model. For training an initial learning rate of 0.001 is used.

4.3 Convolutional Bidirectional RNN

Convolutional recurrent neural networks (CRNN) represent a combination of CNNs and RNNs. They feature convolutional layers as well as recurrent layers. Different implementations are possible. In this work, the convolutional layers directly process the input features, i.e. spectrogram representations, meant to learn an acoustic model (cf. 2D image processing tasks). The recurrent layers are placed after the convolutional layers and are supposed to serve as a means for the network to learn structural patterns.

For this class of NN, the two versions differ in the following aspects: CBGRU-a features 2 recurrent layers with 30 GRUs each, uses a spectral context of 9 frames for convolution, and is trained on sequences of length 100; while CBGRU-b features 3 recurrent layers with 60 GRUs each, uses a spectral context of 13 frames, and is trained on sequences of length 400. For training an initial learning rate of 0.0005 is used.

Table 1 recaps the information of the previous sections in a more compact form. Figure 2 visualizes the modes of operation of the different NN architectures on the input spectrograms.

5. EVALUATION

For evaluation of the introduced NN architectures, the different models are individually trained on single data sets in a three-fold cross-validation manner. For data sets which comprise beat annotations, three different experiments are performed (explained in more detail in section 5.2); using data sets only providing drum annotations, just the drum detection task is performed.

5.1 Data Sets

In this work, the different methods are evaluated using three different data sets, consisting of two well-known and a newly introduced set.

5.1.1 IDMT-SMT-Drums v.1 (SMT)

Published along with [7], the IDMT-SMT-Drums [2] data set comprises tracks containing three different drum-set types. These are: *i.* real-world, acoustic drum sets (titled *RealDrum*), *ii.* drum synthesizers (*TechnoDrum*), and *iii.* drum sample libraries (*WaveDrum*). It consists of 95 simple drum tracks containing bass drum, snare drum and hi-hat only. The tracks have an average length of 15s and a total length of 24m. Also included are additional 285 shorter, single-instrument training tracks as well as 180 single instrument tracks for 60 of the 95 mixture tracks (from the *WaveDrum02* subset)—intended to be used for source separation experiments. These additional single instrument tracks are used as additional training samples (together with their corresponding split) but not for evaluation.

5.1.2 ENST Drums (ENST)

The ENST-Drums set [14] contains real drum recordings of three different drummers performing on different drum kits. [3] Audio files for separate solo instrument tracks

[2] https://www.idmt.fraunhofer.de/en/business_units/m2d/smt/drums.html

[3] http://perso.telecom-paristech.fr/~grichard/ENST-drums/

	Input Features		Target Functions	
	Spectrogram	Beats	Drums	Beats
DT	✓		✓	
BF	✓	✓	✓	
MT	✓		✓	✓

Table 2. Overview of experimental setup. Rows represent individual tasks and show their input feature and target function combinations.

as well as for two mixtures are included. Additionally, accompaniment tracks are available for a subset of the recordings—the so called minus-one tracks. In this work, the wet mixes (contains standard post-processing like compression and equalizing) of the minus-one tracks were used. They make up 64 tracks of 61s average length and a total length of 1h.

Evaluation was performed on the drum-only tracks (ENST solo) as well as the mixes with their accompaniment tracks (ENST acc.). Since the ENST-Drums data set contains more than the three instruments under observation, only the snare, bass, and hi-hat annotations were used.

5.1.3 RBMA Various Assets 2013 (RBMA13)

This new data set consists of the 30 tracks of the freely available 2013 Red Bull Music Academy Various Assets sampler. [4] The sampler covers a variety of electronically produced music, which encompasses electronic dance music (EDM) but also singer-songwriter tracks and even fusion-jazz styled music. Three tracks on the sampler do not contain any drums and are therefore ignored. Annotations for drums, beats, and downbeats were manually created. Tracks in this set have an average length of 3m 50s. The total length of the data set is 1h 43m.

This data set is different from the other two data sets in three aspects: *i.* it contains quite diverse drum sounds, *ii.* the drum patterns are arranged in the usual song-structure within a full length track, and *iii.* most of the tracks contain singing voice, which showed to be a challenge for systems solely trained on music without singing voice. The annotations for drums and beats have been manually created and are publicly available for download. [5]

5.2 Experimental Setup

To compare the different NN architectures, and evaluate them in the context of ADT using joint learning of beat and drum activations, the following experiments were performed.

5.2.1 Drum Detection (DT)

In this set of experiments, the features as explained in sec. 3.1 and target functions generated from the drum annotations described in sec. 3.3 are used for NN training.

[4] https://rbma.bandcamp.com/album/various-assets-not-for-sale-red-bull-music-academy-new-york-2013

[5] http://ifs.tuwien.ac.at/~vogl/datasets/

	SMT	ENST		RBMA13		
		solo	acc.	DT	BF	MT
GRUts [36]	*92.5*	*83.3*	*75.0*	-	-	-
BGRU-a	93.0	80.9	70.1	59.8	63.6	64.6
BGRU-b	93.3	82.9	72.3	61.8	64.5	64.3
CNN-a	87.6	78.6	70.8	66.2	66.7	63.3
CNN-b	93.4	**85.0**	78.3	66.8	65.2	64.8
CBGRU-a	**95.2**	84.6	76.4	65.2	66.1	66.9
CBGRU-b	93.8	83.9	**78.4**	**67.3**	**68.4**	**67.2**

Table 3. F-measure results for the evaluated models on different data sets. The columns DT, BF, and MT show results for models trained only for drum detection, trained using oracle beats as additional input features, and simultaneously trained on drums and beats, respectively. Bold values represent the best performance for an experiment across models. The baseline can be found in the first row.

These experiments are comparable to the ones in the related work, since we use a similar setup. As baseline, the results in [36] are used. The results of this set of experiments allow to compare the performance of different NN architectures for drum detection.

5.2.2 Drum Detection with Oracle Beat Features (BF)

For this set of experiments, in addition to the input features explained in sec. 3.1, the annotated beats, represented as the target functions for beats and downbeats, are included as input features. As targets for NN training only the drum target functions are utilized. Since beat annotations are required for this experiment, only data sets comprising beat annotations can be used. Using the results of these experiments, it can be investigated if the prior knowledge of beat and downbeat positions is beneficial for drum detection.

5.2.3 Joint Drum and Beat Detection (MT)

This set of experiments represents the multi-task learning investigation. As input for training, again, only the spectrogram features are used. Targets for training of the NNs comprise, in this case, drum and beat activation functions. As discussed in the introduction, in some cases it can be beneficial to train related properties simultaneously. Beats and drums are closely related, because usually drum pattern are repetitive on a bar-level (separated by downbeats) and drum onsets often correlate with beats.

The insight which can be drawn from these experiments is whether simultaneous training of drums, beats, and downbeats is beneficial. It is of interest if the resulting performance is higher than the one achieved for DT; and also if it is below, comparable, or even surpasses the results in the BF experiment series.

Table 2 gives an overview of the properties of the experiments and the used feature/target combination.

5.3 Evaluation Method

To evaluate the performance of the different architectures and training methods, the well-known metrics precision,

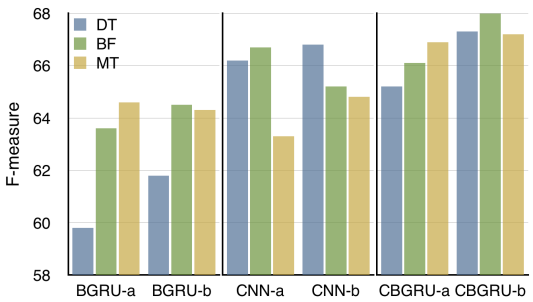


Figure 3. Results for RBMA13 data set, highlighting the influence of oracle beat features (BF) and multi-task learning (MT). While recurrent models (left and right) benefit, convolutional models (center) do not.

recall, and F-measure are used. These are calculated for drum instrument onsets as well as beat positions. True positive, false positive, and false negative onset and beat positions are identified by using a $20ms$ tolerance window. This is in line with the evaluation in [36] which is used as baseline for the experiments of this work. Note that other work, e.g. [7, 25, 31], uses less strict tolerance windows of $30ms$ or $50ms$ for evaluation.

6. RESULTS AND DISCUSSION

Table 3 shows the F-measure results for the individual NN architectures on the data sets used for evaluation. The results for BGRU-a and BGRU-b on the ENST data set are lower than for the baseline, although the models should be comparable. This is due to the fact that in [36] data augmentation is applied. This is especially helpful in the case of the ENST data set, since e.g. the pitches of the base drums vary greatly over the different drum kits. The results for CNN-a are lower than the state of the art, which implies that the context of 9 frames is too small to detect drum events using a CNN. All other results on the ENST and SMT data sets represent an improvement over the state of the art. This shows that CNN with a large enough spectral context (25 frames in this work) can detect drum events better than RNNs. A part of the large increase for the ENST data set can be attributed to the fact that CNNs can model pitch invariance easier than RNNs.

The results for the MT experiments show the following tendencies: For the BGRU-a and BGRU-b models, an improvement can be observed when applying multi-task learning. Compared to using oracle beats (BF) for training, the improvement is higher for BGRU-a and similar in the case of BGRU-b. This result is interesting for two reasons: *i.* although BGRU-a is trained on short sequences, an improvement can be observed, and *ii.* the improvement is comparable to that when using oracle beats (BF) although the beat tracking results are low. This could imply that multi-task learning is also beneficial for the acoustic model of the system. As expected, the CNNs (CNN-a, CNN-b) can not improve when using multi-task learning, but rather the results deteriorate. In case of the convolutional-recurrent models, the result for CBGRU-a is similar to BGRU-a. In case of CBGRU-b no improvement of drum detection performance using multi-task learning can be observed, although it is the case using oracle beats (BF). We attribute this to the fact that CBGRU-b has enough capacity for good acoustic modeling, while the low beat detection results limit the effects of multi-task learning on this level.

BLSTM [2]	*85.6*
BGRU-a	46.4
BGRU-b	46.2
CNN-a	44.9
CNN-b	46.9
CBGRU-a	47.6
CBGRU-b	48.8

Table 4. F-measure results for beat detection for the multi-task learning experiments compared to a state-of-the-art approach (first row) on the RBMA13 set.

Table 4 shows the F-measure results for beat and downbeat tracking. The results are all below the state-of-the-art beat tracker used as baseline [2]. This is due to several factors. In [2], *i.* much larger training sets for beat and downbeat tracking are used, *ii.* the LSTMs are trained on full sequences of the input data, giving the model more context, and *iii.* an additional music language model in the form of a dynamic Bayesian network (DBN) is used.

The results for CNNs and CRNNs show that convolutional feature processing is beneficial for drum detection. The finding considering drum detection results for multi-task learning are also promising. The low results of beat and downbeat tracking are certainly a limiting factor and probably the reason for the lack of improvement for MT over DT in the case of BGRU-b. As a next step, to better leverage multi-task learning effects, beat detection results must be improved using similar techniques as in [2].

7. CONCLUSIONS

In this work, convolutional and convolutional-recurrent NN models for drum transcription were introduced and compared to the state of the art of recurrent models. The evaluation shows that the new models are able to outperform this state of the art. Furthermore, an investigation whether *i.* beat and downbeat input features are beneficial for drum detection, and *ii.* this benefit is also achievable using multi-task learning of drums, beats, and downbeats, was conducted. The results show that this is the case, although the low beat and downbeat detection results achieved with the implemented architectures is a limiting factor. While the goal of this work was not to improve the capabilities of beat and downbeat tracking per se, future work will focus on improving these aspects, as we believe this will have an overall positive impact on the performance of the joint model. The newly created data set consisting of freely available music and annotations for drums, beats and downbeats will be an asset for this line of research to the community.

8. ACKNOWLEDGEMENTS

This work has been partly funded by the Austrian FFG under the BRIDGE 1 project *SmarterJam* (858514), by the EU's seventh Framework Programme FP7/2007-13 for research, technological development and demonstration under grant agreement no. 610591 (*GiantSteps*), as well by the Austrian ministries BMVIT and BMWFW, and the province of Upper Austria via the COMET Center SCCH. We would like to thank Wulf Gaebele and the annotators Marc Übel and Jo Thalmayer from the Red Bull Music Academy, as well as Sebastian Böck for advice and help with beat and downbeat annotations and detection.

9. REFERENCES

[1] Emmanouil Benetos, Simon Dixon, Dimitrios Giannoulis, Holger Kirchhoff, and Anssi Klapuri. Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41(3):407–434, 2013.

[2] Sebastian Böck, Florian Krebs, and Gerhard Widmer. Joint beat and downbeat tracking with recurrent neural networks. In *Proc. 17th Intl Society for Music Information Retrieval Conf (ISMIR)*, 2016.

[3] Sebastian Böck and Gerhard Widmer. Maximum filter vibrato suppression for onset detection. In *Proc. 16th Intl Conf on Digital Audio Effects (DAFx)*, 2013.

[4] Rich Caruana. Multitask learning. In Thrun and Pratt (eds.) *Learning to learn*, pages 95–133. Springer, 1998.

[5] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. Learning phrase representations using rnn encoderdecoder for statistical machine translation. In *Proc. Conf on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. `http://arxiv.org/abs/1412.3555`, 2014.

[7] Christian Dittmar and Daniel Gärtner. Real-time transcription and separation of drum recordings based on nmf decomposition. In *Proc. 17th Intl Conf on Digital Audio Effects (DAFx)*, 2014.

[8] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proc. IEEE conference on computer vision and pattern recognition (CVPR)*, 2015.

[9] Simon Durand, Juan P Bello, Bertrand David, and Gaël Richard. Downbeat tracking with multiple features and deep neural networks. In *Proc. 40th IEEE Intl Conf on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[10] Simon Durand, Juan P Bello, Bertrand David, and Gaël Richard. Feature adapted convolutional neural networks for downbeat tracking. In *Proc. 41st IEEE Intl Conf on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[11] Derry FitzGerald, Bob Lawlor, and Eugene Coyle. Drum transcription in the presence of pitched instruments using prior subspace analysis. In *Proc. Irish Signals & Systems Conf*, 2003.

[12] Nicolai Gajhede, Oliver Beck, and Hendrik Purwins. Convolutional neural networks with batch normalization for classifying hi-hat, snare, and bass percussion sound samples. In *Proc. Audio Mostly Conf*, 2016.

[13] Olivier Gillet and Gaël Richard. Automatic transcription of drum loops. In *Proc. 29th IEEE Intl Conf on Acoustics, Speech, and Signal Processing (ICASSP)*, 2004.

[14] Olivier Gillet and Gaël Richard. Enst-drums: an extensive audio-visual database for drum signals processing. In *Proc. 7th Intl Conf on Music Information Retrieval (ISMIR)*, 2006.

[15] Olivier Gillet and Gaël Richard. Supervised and unsupervised sequence modelling for drum transcription. In *Proc. 8th Intl Conf on Music Information Retrieval (ISMIR)*, 2007.

[16] Olivier Gillet and Gaël Richard. Transcription and separation of drum signals from polyphonic music. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(3):529–540, 2008.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. `http://arxiv.org/abs/1502.03167`, 2015.

[19] Florian Krebs, Sebastian Böck, Matthias Dorfer, and Gerhard Widmer. Downbeat tracking using beat-synchronous features and recurrent neural networks. In *Proc. 17th Intl Society for Music Information Retrieval Conf (ISMIR)*, 2016.

[20] Florian Krebs, Sebastian Böck, and Gerhard Widmer. Rhythmic pattern modeling for beat and downbeat tracking in musical audio. In *Proc. 15th Intl Society for Music Information Retrieval Conf (ISMIR)*, 2013.

[21] Florian Krebs, Filip Korzeniowski, Maarten Grachten, and Gerhard Widmer. Unsupervised learning and refinement of rhythmic patterns for beat and downbeat tracking. In *Proc. 22nd European Signal Processing Conf (EUSIPCO)*, 2014.

[22] Marius Miron, Matthew EP Davies, and Fabien Gouyon. Improving the real-time performance of a causal audio drum transcription system. In *Proc. 10th Sound and Music Computing Conf (SMC)*, 2013.

[23] Marius Miron, Matthew EP Davies, and Fabien Gouyon. An open-source drum transcription system for pure data and max msp. In *Proc. 38th IEEE Intl Conf on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[24] Arnaud Moreau and Arthur Flexer. Drum transcription in polyphonic music using non-negative matrix factorisation. In *Proc. 8th Intl Conf on Music Information Retrieval (ISMIR)*, 2007.

[25] Jouni Paulus and Anssi Klapuri. Drum sound detection in polyphonic music with hidden markov models. *EURASIP Journal on Audio, Speech, and Music Processing*, 2009.

[26] Geoffroy Peeters and Helene Papadopoulos. Simultaneous beat and downbeat-tracking using a probabilistic framework: Theory and large-scale evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(6):1754–1769, 2011.

[27] Pedro HO Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *Proc. 31st Intl Conf on Machine Learning (ICML)*, Beijing, China, 2014.

[28] Jan Schlüter and Sebastian Böck. Improved musical onset detection with convolutional neural networks. In *Proc. 39th IEEE Intl Conf on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

[29] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[30] Siddharth Sigtia, Emmanouil Benetos, Srikanth Cherla, Tillman Weyde, Artur S d'Avila Garcez, and Simon Dixon. An RNN-based music language model for improving automatic music transcription. In *Proc. 15th Intl Society for Music Information Retrieval Conf (ISMIR)*, 2014.

[31] Carl Southall, Ryan Stables, and Jason Hockman. Automatic drum transcription using bidirectional recurrent neural networks. In *Proc. 17th Intl Society for Music Information Retrieval Conf (ISMIR)*, 2016.

[32] Srivastava, Nitish and Hinton, Geoffrey and Krizhevsky, Alex and Sutskever, Ilya and Salakhutdinov, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[33] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5rmsprop: Divide the gradient by a running average of its recent magnitude. In *COURSERA: Neural Networks for Machine Learning*, October 2012.

[34] Christian Uhle, Christian Dittmar, and Thomas Sporer. Extraction of drum tracks from polyphonic music using independent subspace analysis. In *Proc. 4th Intl Symposium on Independent Component Analysis and Blind Signal Separation*, 2003.

[35] Richard Vogl, Matthias Dorfer, and Peter Knees. Recurrent neural networks for drum transcription. In *Proc. 17th Intl Society for Music Information Retrieval Conf (ISMIR)*, 2016.

[36] Richard Vogl, Matthias Dorfer, and Peter Knees. Drum transcription from polyphonic music with recurrent neural networks. In *Proc. 42nd IEEE Intl Conf on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[37] Chih-Wei Wu and Alexander Lerch. Drum transcription using partially fixed non-negative matrix factorization with template adaptation. In *Proc. 16th Intl Society for Music Information Retrieval Conf (ISMIR)*, 2015.

[38] Kazuyoshi Yoshii, Masataka Goto, and Hiroshi G Okuno. Drum sound recognition for polyphonic audio signals by adaptation and matching of spectrogram templates with harmonic structure suppression. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(1):333–345, 2007.

[39] Zhen Zuo, Bing Shuai, Gang Wang, Xiao Liu, Xingxing Wang, Bing Wang, and Yushi Chen. Convolutional recurrent neural networks: Learning spatial dependencies for image representation. In *Proc. IEEE Conf on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015.

7 TRANSCRIBING MORE DRUM INSTRUMENTS

7.1 OVERVIEW

This contents of this chapter were published in the following work:

Richard Vogl, Gerhard Widmer, and Peter Knees. "Towards Multi-Instrument Drum Transcription". In: *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx)*. Aveiro, Portugal, 2018.

Previous works in this thesis, as well as most of the approaches proposed in recent years, focus on three drum instruments, namely bass drum, snare drum, and hi-hat. The aim of this work is to make first steps towards an ADT system that can deal with more than just these three instruments. Two label systems featuring eight and 18 different instrument classes are introduced and used besides the traditional three-class system. A major challenge of using more instrument labels is that most of the instrument classes besides the traditionally used ones are very sparsely represented in publicly available datasets. To overcome this issue, a large-scale dataset synthesized from MIDI tracks is introduced. Additionally, a second version of the synthetic data is created by balancing the dataset's instrument class distributions by exchanging individual instruments within tracks. This is done in order to check if avoiding the underrepresentation of scarcely played instruments helps during network training. Finally, different training strategies to leverage the synthetic dataset are evaluated using CNN-based and CRNN-based methods, similar to the ones used in the previous publication. A related work that uses similar approaches was independently published by Mark Cartwright and Juan P. Bello [12] at the same time.

The evaluation reveals that while cross-validation on the balanced synthetic dataset shows performance improvements on underrepresented drum instrument classes, this does not generalize to recorded datasets. Another interesting finding is that a network trained on the unbalanced artificial dataset generalizes surprisingly well to recorded datasets. The results indicate that the destruction of natural drum patterns when balancing the dataset conditions recurrent models on wrong prior distributions, which is responsible for the poor performance in that case. Finally, by using pre-training on the synthetic data and fine tuning the models on real world data, a slight improvement for the multi class problem can be observed. An in-depth evalua-

tion using confusion matrices for individual drum instruments under observation provides additional insights.

7.2 CONTRIBUTIONS OF AUTHORS

As first author of this work I contributed almost all of this work's content. Trying to work with more drum instruments was one of the reasons I started working with neural networks for ADT, since first experiments with NMF into this direction made clear that they are not well suited for this task. To evaluate multi-instrument drum transcription systematically, it was necessary to have a dataset with enough data for all instruments under observation. This was critical to be able to check if the imbalance of training data is the main reason for bad performance, or if other factors also play a role. Looking into the types of errors ADT systems make, and why, was something I have been discussing with colleagues working on ADT in general, but especially with Carl Southall. The confusion matrices used in this work are partly a result of those discussions. I collected, created, and prepared the new dataset, wrote necessary code to run the experiments, designed and performed the evaluation, as well as wrote the paper.

Gerhard Widmer and Peter Knees, again, acted as supervisors for this work. They both helped to improve the paper by proofreading and providing valuable feedback.

TOWARDS MULTI-INSTRUMENT DRUM TRANSCRIPTION

Richard Vogl[1,2]
richard.vogl@tuwien.ac.at

Gerhard Widmer[2]
gerhard.widmer@jku.at

Peter Knees[1]
peter.knees@tuwien.ac.at

[1] Faculty of Informatics
TU Wien
Vienna, Austria

[2] Institute of Computational Perception
Johannes Kepler University
Linz, Austria

ABSTRACT

Automatic drum transcription, a subtask of the more general automatic music transcription, deals with extracting drum instrument note onsets from an audio source. Recently, progress in transcription performance has been made using non-negative matrix factorization as well as deep learning methods. However, these works primarily focus on transcribing three drum instruments only: snare drum, bass drum, and hi-hat. Yet, for many applications, the ability to transcribe more drum instruments which make up standard drum kits used in western popular music would be desirable. In this work, convolutional and convolutional recurrent neural networks are trained to transcribe a wider range of drum instruments. First, the shortcomings of publicly available datasets in this context are discussed. To overcome these limitations, a larger synthetic dataset is introduced. Then, methods to train models using the new dataset focusing on generalization to real world data are investigated. Finally, the trained models are evaluated on publicly available datasets and results are discussed. The contributions of this work comprise: *(i.)* a large-scale synthetic dataset for drum transcription, *(ii.)* first steps towards an automatic drum transcription system that supports a larger range of instruments by evaluating and discussing training setups and the impact of datasets in this context, and *(iii.)* a publicly available set of trained models for drum transcription. Additional materials are available at http://ifs.tuwien.ac.at/~vogl/dafx2018.

1. INTRODUCTION

Automatic drum transcription (ADT) focuses on extracting a symbolic notation for the onsets of drum instruments from an audio source. As a subtask of automatic music transcription, ADT has a wide variety of applications, both in an academic as well as in a commercial context. While state-of-the-art approaches achieve reasonable performance on publicly available datasets, there are still several open problems for this task. In prior work [1] we identify additional information—such as bar boundaries, local tempo, or dynamics—required for a complete transcript and propose a system trained to detect beats alongside drums. While this adds some of the missing information, further work in this direction is still required.

Another major shortcoming of current approaches is the limitation to only three drum instruments. The focus on snare drum (SD), bass drum (BD), and hi-hat (HH) is motivated by the facts that these are the instruments *(i.)* most commonly used and thus with the highest number of onsets in the publicly available datasets; and *(ii.)* which often define the main rhythmical theme. Nevertheless, for many applications it is desirable to be able to transcribe a wider variety of the drum instruments which are part of a standard drum kit in western popular music, e.g., for extracting full transcripts for further processing in music production or educational scenarios. One of the main issues with building and evaluating such a system is the relative underrepresentation of these classes in available datasets (see section 2).

In this work we focus on increasing the number of instruments to be transcribed. More precisely, instead of three instrument classes, we aim at transcribing drums at a finer level of granularity as well as additional types of drums, leading to classification schemas consisting of eight and 18 different instruments (see table 1). In order to make training for a large number of instruments feasible, we opt for a single model to simultaneously transcribe all instruments of interest, based on convolutional and convolutional recurrent neural networks. Especially in the case of deep learning, a considerable amount of processing power is needed to train the models. Although other approaches train separate models for each instrument in the three-instrument-scenario [2, 3], for 18 instruments it is more feasible to train a single model in a multi-task fashion (cf. [4]). To account for the need of large volumes of data in order to train the chosen network architectures, a large synthetic dataset is introduced, consisting of 4197 tracks and an overall duration of about 259h.

The remainder of this paper is organized as follows. In section 2 we discuss related work, followed by a description of our proposed method in section 3. Section 4 provides a review of existing datasets used for evaluation, as well as a description of the new, large synthetic dataset. Sections 5 and 6 describe the conducted experiments and discuss the results, respectively. Finally, we draw conclusions in section 7.

2. RELATED WORK

There has been a considerable amount of work published on ADT in recent years, e.g., [5, 6, 7, 8, 9]. In the past, different combinations of signal processing and information retrieval techniques haven been applied to ADT. For example: onset detection in combination with *(i.)* bandpass filtering [10, 11], and *(ii.)* instrument classification [5, 6, 7]; as well as probabilistic models [8, 12]. Another group of methods focus on extracting an onset-pseudo-probability function (activation function) for each instrument under observation. These methods utilize source separation techniques like Independent Subspace Analysis (ISA) [13], Prior Subspace Analysis (PSA) [14], and Non-Negative Independent Component Analysis (NNICA) [15]. More recently, these approaches have been further developed using Non-Negative Matrix Factorization (NMF) variants as well as deep learning [1, 3, 16, 17].

The work of Wu et al. [18] provides a comprehensive overview of the publications for this task, and additionally performs in-depth evaluation of current state-of-the-art methods. Due to the large

Table 1: Classes used in the different drum instrument classification systems. Labels map to General MIDI drum instruments: e.g. bass drum: 35, 36; side stick: 37; etc. The mapping is available on the accompanying website.

number of classes 3	8	18	instrument name
BD	BD	BD	bass drum
SD	SD	SD	snare drum
		SS	side stick
		CLP	hand clap
		HT	high tom
	TT	MT	mid tom
		LT	low tom
		CHH	closed hi-hat
HH	HH	PHH	pedal hi-hat
		OHH	open hi-hat
		TB	tambourine
	RD	RD	ride cymbal
	BE	RB	ride bell
		CB	cowbell
	CY	CRC	crash cymbal
		SPC	splash cymbal
		CHC	Chinese cymbal
	CL	CL	clave/sticks

number of works and given the space limitations, in the remainder of this section, we will focus on work that is directly relevant with respect to the current state of the art and methods focusing on more than three drum instrument classes.

As mentioned, the state of the art for this task is currently defined by end-to-end activation function based methods. In this context, end-to-end implies using only one processing step to extract the activation function for each instrument under observation from a digital representation of the audio signal (usually spectrogram representations). Activation functions can be interpreted as probability estimates for a certain instrument onset at each point in time. To obtain the positions of the most probable instrument onsets, simple peak picking [19, 20, 1, 3, 2, 16, 15] or a language-model-style decision process like dynamic Bayesian networks [21] can be used. These methods can be further divided into NMF based and deep neural network (DNN) based approaches.

Wu et al. [16] introduce partially fixed NMF (PFNMF) and further modifications to extract the drum instrument onset times from an audio signal. Dittmar et al. [17] use another modification of NMF, namely semi adaptive NMF (SANMF) to transcribe drum solo tracks in real time, while requiring samples of the individual drum instruments for training. More recently, recurrent neural networks (RNNs) have successfully been used to extract the activation functions for drum instruments [19, 20, 2]. It has also been shown that convolutional (CNNs) [1, 3] and convolutional recurrent neural networks (CRNNs) [1] have the potential to even surpass the performance of RNNs.

The majority of works on ADT, especially the more recent ones, focus solely on transcribing three drum instrument (SD, BD, HH) [9, 19, 20, 1, 2, 3, 16, 8, 17, 7, 8]. In some works multiple drum instruments are grouped into categories for transcription [5] and efforts have been made to classify special drum playing techniques within instrument groups [22]. However, only little work exists which approach the problem of transcribing more than three individual drum instruments [15], furthermore, such a system has—to our knowledge—never been evaluated on currently available public drum transcription datasets.

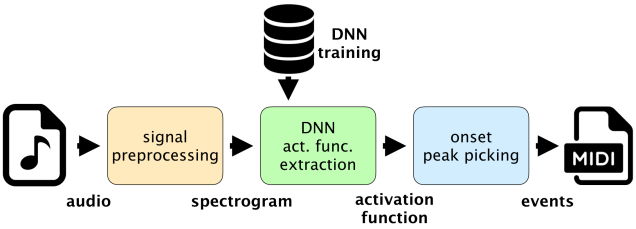


Figure 1: Overview of implemented ADT system using DNNs.

In [6], a set of MIDI drum loops rendered with different drum samples are used to create synthetic data in the context of ADT. Using synthetic data was a necessity in the early years of music information retrieval (MIR), but due to the continuous efforts of creating datasets, this has declined in recent years. However, machine learning methods like deep learning, often requirer large amounts of data, and manual annotation in large volumes is unfeasible for many MIR tasks. In other fields like speech recognition or image processing, creating annotations is easier, and large amounts of data are commonly available. Using data augmentation can, to a certain degree, be used to overcome lack of data, as has been demonstrated in the context of ADT [20]. In [23] an approach to resynthesizes solo tracks using automatically annotated f0 trajectories, to create perfect annotations, is introduced. This approach could be applicable for ADT, once a satisfactory model for the full range of drum instruments is available. At the moment such annotations would be limited to the three drum instrument classes used in state-of-the-art methods.

3. METHOD

In this work, we use an approach similar to the ones introduced in [2] and [19], for drum transcription. As mentioned in the introduction, a single model trained in a multi-task fashion will be used. Creating individual models for each instrument is an option [2, 3], however, in the context of this work it has two downsides: First, training time will scale linearly with the amount of models, which is problematic when increasing the number of instruments under observation. Second, training multi-task models in the context of ADT can improve the performance [1]. Other state-of-the-art methods based on NMF [16, 17] are less suitable for a multi-task approach, since the performance of NMF methods is prone to degrade for basis matrices with higher rank.

Thus, the method proposed in [1] seems most promising for the goal of this work. We will only use CNNs and CRNNs, since simple RNNs do not have any advantage in this context. The implemented ADT system consists of three stages: a signal preprocessing stage, a DNN activation function extraction stage, and a peak picking post processing stage, identifying the note onset. The system overview is visualized in figure 1, and the single stages will be discussed in detail in the following subsections.

3.1. Preprocessing

During signal preprocessing, a logarithmic magnitude spectrogram is calculated using a window size of 2048 samples (@44.1kHz input audio frame rate) and choosing 441 samples as hop size for a

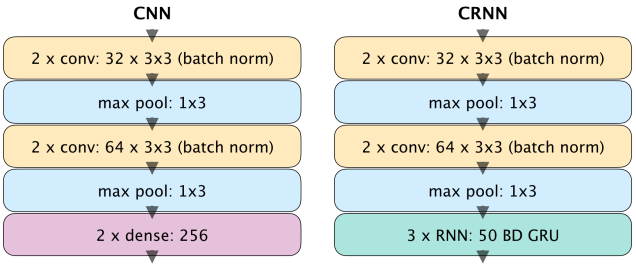


Figure 2: Architecture comparison between the CNN and CRNN used for activation function extraction.

100Hz target frame rate of the spectrogram. The frequency bins are transformed to a logarithmic scale using triangular filters in a range from 20 to 20,000 Hz, using 12 frequency bins per octave. Finally, the positive first-order-differential over time of this spectrogram is calculated and stacked on top of the original spectrogram. The resulting feature vectors have a length of 168 values (2x84 frequency bins).

3.2. Activation Function Extraction

The activation function extraction stage is realized using one of two different DNNs architectures. Figure 2 visualizes and compares the two implemented architectures. The convolutional parts are equivalent for both architectures, however, the dense output layers are different: while for the CNN two normal dense layers are used (ReLUs), in case of the CRNN two bidirectional RNN layers consisting of gated recurrent units (GRUs) [24] are used. As already noted in [1], GRUs exhibit similar capabilities as LSTMs [25], while being more easy to train.

The combination of convolutional layers which focus on local spectral features, and recurrent layers which model mid- and long-term relationships, has been found to be one of the best performing models for ADT [1].

3.3. Peak Picking

To identify the drum instrument onsets, a standard peak picking method introduced for onset detection in [26] is used. A peak at point n in the activation function $f_a(n)$ must be the maximum value within a window of size $m+1$ (i.e.: $f_a(n) = max(f_a(n-m), \cdots, f_a(n))$), and exceeding the mean value plus a threshold δ within a window of size $a+1$ (i.e.: $f_a(n) \geq mean(f_a(n-a), \cdots, f_a(n)) + \delta$). Additionally, a peak must have at least a distance of $w+1$ to the last detected peak n_{lp} (i.e.: $n - n_{lp} > w$,). The parameters for peak picking are the same as used in [1]: $m = a = w = 2$. The best threshold for peak picking is determined on the validation set. As observed in [3, 20, 1], appropriately trained DNNs produce spiky activation functions, therefore, low thresholds $(0.1 - 0.2)$ give best results.

3.4. Training and Evaluation

Training of the models is performed using *Adam* optimization [27] with mini-batches of size 100 and 8 for the CNNs and CRNNs respectively. The training instances for the CNN have a spectral context of 25 samples. In case of the CRNN, the training sequences consist of 400 instances with a spectral context of 13 samples. The DNNs are trained using a fixed learning rate ($l_r = 0.001$) with

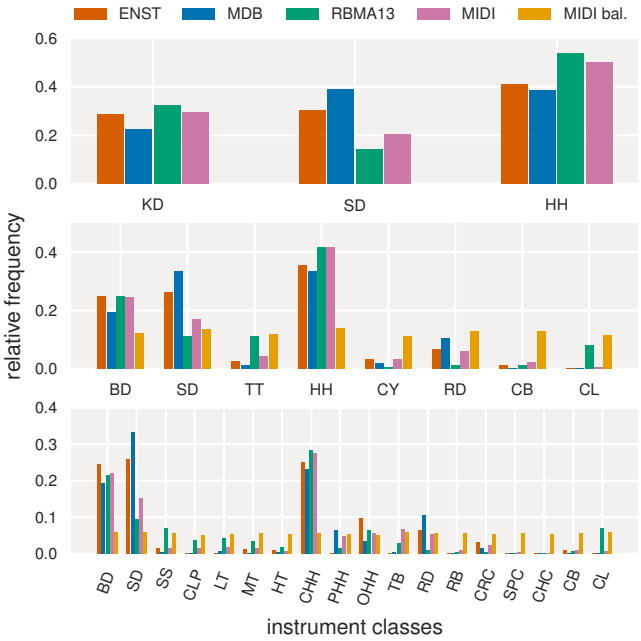


Figure 3: Label distributions of the different datasets used in this work.

additional refinement if no improvement on the validation set is achieved for 10 epochs. During refinement the learning rate is reduced ($l_r = l_r \cdot 0.2$) and training continues using the parameters of the best performing model so far.

A three-fold cross-validation strategy is employed, using two splits during training, while 15% of the training data is separated and used for validation after each epoch (0.5% in case of the large datasets, to reduce validation time). Testing is done on the third, during training unseen, split. Whenever available, drum solo versions of the tracks are used as additional training material, but not for testing/evaluation. The solo versions are always put into the same splits as their mixed counterparts, to counter overfitting. This setup is consistently used through all experiments, whenever datasets are mixed or cross-validated, corresponding splits are used.

For audio preprocessing, peak picking, and calculation of evaluation metrics, the madmom[1] python framework was used. DNN training was performed using Theano[2] and Lasagne[3]. For a more details on C(R)NN training and a comparison of their working principles in the context of ADT, we kindly refer the reader to our previous work [1] due to space limitations and a different focus of this work.

4. DATASETS

There are a number of publicly available datasets for ADT with varying size, degree of detail, and number of classes regarding the drum instrument annotations. As noted in the introduction, current state-of-the-art approaches limit the instruments under observation to the three most common ones (SD, BD, HH). This is done by ignoring other instruments like tom-toms and cymbals, as well as

[1] https://github.com/CPJKU/madmom
[2] https://github.com/Theano/Theano
[3] https://github.com/Lasagne/Lasagne

Table 2: F-measure (*mean*/*sum*) results of implemented ADT methods on public datasets for different class systems. The first line indicates state-of-the-art F-measure results in previous work using CNN and CRNN ADT systems in a three-class scenario.

CL	model	*ENST*	*MDB*	*RBMA13*
3	*SotA* [1]	— / *0.78*	— / —	— / *0.67*
3	CNN	0.75 / 0.77	0.65 / 0.72	0.53 / 0.63
	CRNN	0.74 / 0.76	0.64 / 0.70	0.55 / 0.64
8	CNN	0.59 / 0.63	0.68 / 0.65	0.55 / 0.44
	CRNN	0.65 / 0.70	0.68 / 0.63	0.55 / 0.50
18	CNN	0.69 / 0.49	0.76 / 0.47	0.62 / 0.31
	CRNN	0.75 / 0.67	0.77 / 0.55	0.64 / 0.39

grouping different play styles like closed, opened, and pedal hi-hat strokes. In order to investigate ways of generating a model which is capable to transcribe more than these three instruments, two classification systems, i.e., a medium and a large one, for drum instruments of a standard drum kit are defined. Table 1 shows the two sets of classes, which contain eight and 18 labels respectively, alongside with the classic three-class set used in state-of-the-art works and the mapping used between these classes.

In the following we discuss publicly available ADT datasets and their limitations, leading to the description of the large volume synthetic dataset introduced for training of our models.

4.1. ENST Drums (*ENST*)

The ENST Drums[4] dataset published by Gillet and Richard [28] in 2005, is commonly used in ADT evaluations. The freely available part of the dataset consists of single track audio recordings and mixes, performed by three drummers on different drum kits. It contains recordings of single strokes for each instrument, short sequences of drum patterns, as well as drum tracks with additional accompaniment (*minus-one* tracks). The annotations contain labels for 20 different instrument classes.

For evaluation, the *wet mixes* (contain standard post-processing like compression and equalizing) of the *minus-one tracks* were used. They make up 64 tracks of 61s average duration and a total duration of 1h. The rest of the dataset (single strokes, patterns) was used as additional training data.

4.2. MDB-Drums (*MDB*)

The MDB-Drums dataset[5] was published in [29] and provides drum annotations for 23 tracks of the Medley DB dataset[6] [30]. The tracks are available as drum solo tracks with additional accompaniment. Again, only the full mixes are used for evaluation, while the drum solo tracks are used as additional training data. There are two levels of drum instrument annotations, the second providing multiple drum instruments and additional drum playing technique details in 21 classes. Tracks have an average duration of 54 seconds and the total duration is 20m 42s.

[4] http://perso.telecom-paristech.fr/~grichard/ENST-drums/

[5] https://github.com/CarlSouthall/MDBDrums

[6] http://medleydb.weebly.com/

Table 3: F-measure results (*mean*/*sum*) of the implemented networks on synthetic datasets.

CL	model	*MIDI*	*MIDI 1%*	*MIDI bal.*
3	CNN	0.74 / **0.84**	0.70 / 0.79	— / —
	CRNN	0.74 / **0.84**	0.68 / 0.77	— / —
8	CNN	0.64 / 0.63	0.63 / 0.69	0.54 / 0.58
	CRNN	0.74 / **0.82**	0.69 / 0.73	0.58 / 0.70
18	CNN	0.66 / 0.39	0.65 / 0.39	0.59 / 0.18
	CRNN	0.73 / **0.70**	0.69 / 0.62	0.63 / 0.52

4.3. RBMA13 (*RBMA13*)

The RBMA13 datasets[7] was published alongside [1]. It consists of 30 tracks of the freely available 2013 Red Bull Music Academy Various Assets sampler.[8] The tracks' genres and drum sounds of this set are more diverse compared to the previous sets, making it a particularly difficult set. It provides annotations for 23 drum instruments as well as beat and downbeats. Tracks in this set have an average duration of 3m 50s and a total of 1h 43m.

4.4. Limitations of current datasets

A major problem of publicly available ADT datasets in the context of deep learning is the volume of data. To be able to train DNNs efficiently, usually large amounts of diverse data are used (e.g. in speech and image processing). One way to counter the lack of data is to use data augmentation (as done in [20] for ADT). However, data augmentation is only helpful to a certain degree, depending on the applicable augmentation methods and the diversity of the original data.

Given the nature of drum rhythms found in western popular music, another issue of ADT datasets is the uneven distribution of onsets between instrument classes. In case of the available datasets, this imbalance can be observed in figure 3. While it is advantageous for the model to adapt to this bias, in terms of overall performance, this often results in the trained models to never predict onsets for sparse classes. This is due to the number of potential false negatives being negligible, compared to the amount of false positives produced in the early stages of training. To counter a related effect on slightly imbalanced classes (BD, SD, HH in the three-class scenario), a weighting of the loss functions for the different classes can be helpful [20]. Nevertheless, a loss function weighting cannot compensate for the problem in the case of very sparse classes.

Since manual annotation for ADT is a very resource intensive task, a feasible approach to tackle these problems is to create a synthetic dataset using the combination of symbolic tracks, e.g. MIDI tracks, drum synthesizers and/or sampler software.

4.5. Synthetic dataset (*MIDI*)

For generating the synthetic dataset, a similar approach as in [6] was employed. Since the focus of this work is the transcription of multiple drum instruments from polyphonic music, full MIDI tracks of western popular music were used instead of MIDI drum loops. First, every MIDI track from a freely available online collection[9] was split into a drum and accompaniment track. Using

[7] http://ifs.tuwien.ac.at/~vogl/datasets/

[8] https://rbma.bandcamp.com/album/

[9] http://www.midiworld.com

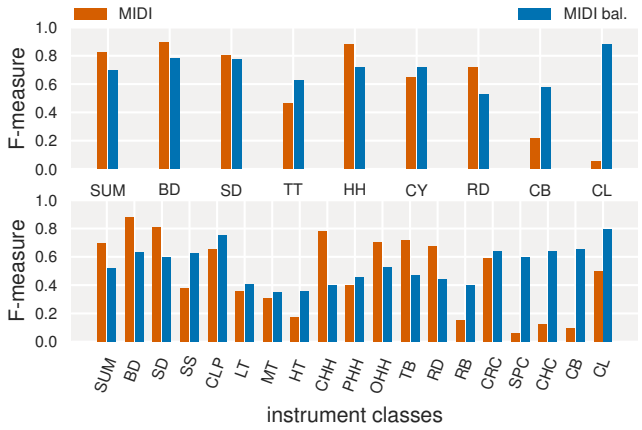


Figure 4: Instrument class details for evaluation results on *MIDI* and *MIDI bal.* for 8 and 18 instrument classes using the CRNN. First value (SUM) represents the overall sum F-measure results.

timidity++[10], the drum tracks were rendered utilizing 57 different drum SoundFonts[11]. The used SoundFonts were collected from different online sources, and great care was taken to manually check and correct the instrument mappings and overall suitability. They cover a wide range of drum sounds from electronic drum machines (e.g. TR808), acoustic kits, and commonly used combinations. The SoundFonts were divided into three groups for the three evaluation splits, to counter overfitting to drum kits. The accompaniment tracks were rendered using a full General MIDI SoundFont. Using the MIDI tracks, drum annotations as well as beat and downbeat annotations were generated. After removing broken MIDI files, very short (< 30s) as well as very long (> 15m) tracks, the set contains 4197 tracks with an average duration of 3m 41s and a total duration of about 259h. As with the other datasets, we only use the mixes for evaluation, while the drum solo tracks are used as additional train-only data.

Figure 3 shows that the general trend of the drum instrument class distribution is similar to the smaller datasets. This is not surprising since the music is of the same broad origin (western popular music). Since one of the goals of creating this dataset was to achieve a more balanced distribution, some additional processing is necessary. Due to the fact that we can easily manipulate the source MIDI drum files, we can change a certain amount of instruments for several tracks to artificially balance the classes. We did this for the 18 classes as well as for the 8 classes and generated two more synthetic datasets consisting of the same tracks, but with drum instruments changes so that the classes are balanced within their respective drum instrument class system. This was done in a way to switch instruments which have a similar expected usage frequency within a track, while keeping musicality in mind. Ideal candidates for this are CHH and RD: exchanging them makes sense from a musical standpoint, as well in terms of usage frequency. On the other hand, BD and CRC are close in expected usage frequency but switching them can be questionable from a musical standpoint, depending on the music genre. A full list of performed switches for the balanced versions can be found on the accompanying webpage.

[10] `http://timidity.sourceforge.net/`
[11] `https://en.wikipedia.org/wiki/SoundFont`

Table 4: F-measure results (*mean*/*sum*) for the CRNN model on public datasets when trained on different dataset combinations. The top part shows results for the 8 class scenario, while the bottom part shows results for the 18 class scenario. Whenever the *MIDI* set is mixed with real world datasets, only the 1% subset is used, to keep a balance between different data types.

8 instrument classes			
train set	*ENST*	*MDB*	*RBMA13*
all	0.61 / 0.64	0.68 / 0.64	0.57 / 0.52
MIDI	0.65 / 0.68	0.70 / 0.61	0.57 / 0.51
MIDI bal.	0.61 / 0.57	0.66 / 0.52	0.56 / 0.47
all+MIDI	0.58 / 0.62	0.67 / 0.57	0.57 / 0.52
all+MIDI bal.	0.61 / 0.64	0.68 / 0.56	0.56 / 0.51
pt MIDI	0.64 / **0.69**	0.72 / **0.68**	0.58 / **0.56**
pt MIDI bal.	0.61 / 0.63	0.72 / 0.67	0.58 / **0.56**
18 instrument classes			
train set	*ENST*	*MDB*	*RBMA13*
all	0.71 / 0.58	0.77 / 0.55	0.63 / 0.41
MIDI	0.73 / 0.61	0.77 / 0.53	0.64 / 0.39
MIDI bal.	0.70 / 0.52	0.76 / 0.45	0.63 / 0.35
all+MIDI	0.73 / 0.62	0.77 / 0.54	0.64 / 0.41
all+MIDI bal.	0.72 / 0.57	0.76 / 0.47	0.64 / 0.37
pt MIDI	0.74 / **0.67**	0.78 / **0.60**	0.64 / **0.47**
pt MIDI bal.	0.74 / 0.65	0.78 / 0.58	0.64 / 0.45

A downside of this approach is that the instrument switches may create artificial drum patterns which are atypical for western popular music. This can be problematic if the recurrent parts of the used CRNN architecture start to learn structures of typical drum patterns. Since these effects are difficult to measure and in order to be able to build a large, balanced dataset, this consequence was considered acceptable.

5. EXPERIMENTS

The first set of experiments evaluates the implemented ADT methods on the available public datasets, using the classic three drum instrument class labels, as well as the two new drum classification schemas with 8 and 18 classes, as a baseline. As evaluation measure primarily the F-measure of the individual drum instrument onsets is used. To calculate the overall F-measure over all instruments and all tracks of a dataset, two methods are used: First, the mean over all instruments' F-measure (=F-measure of track), as well as the mean over all tracks' F-measure is calculated (*mean*). Second, all false positives, false negatives, and true positives for all instruments and tracks are used to calculate a global F-measure (*sum*). These two values give insight into different aspects. While the *mean* value is more conservative for only slightly imbalanced classes, it is problematic when applied to sets containing only sparsely populated classes. In this case, some tracks may have zero occurrences of an instrument, thus resulting in a F-measure of 1.0 when no instrument is detected by the ADT system. In that case, the overall *mean* F-measure value for this instrument is close to 1.0 if it only occurs in a small fraction of tracks and the system never predicts it. On the other hand, the *sum* value will give a F-measure close to zero if the system never predicts an instrument, even for sparse classes—which is more desirable in this context.

The second set of experiments evaluates the performance of the ADT methods on the synthetic datasets, as well as a 1% subset

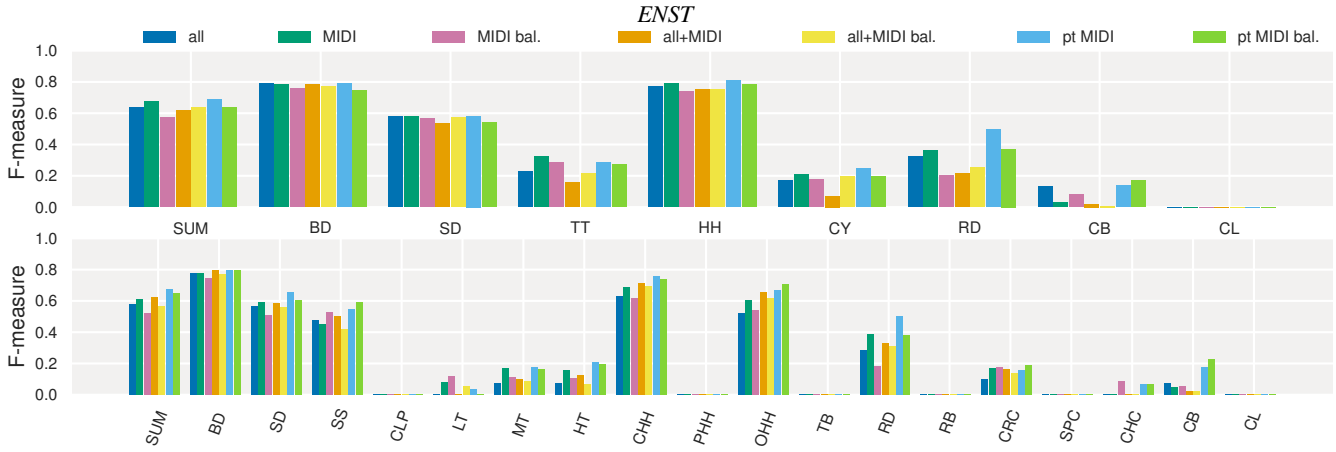


Figure 5: This figure shows F-measure results for each instrument, for both the 8 class (top) as well as the 18 class (bottom) scenarios, exemplary for the *ENST* dataset. Figures for other sets are found on the accompanying webpage (see sec. 7). The color of bars indicates the dataset or combinations trained on: *all*—three public datasets; *MIDI*—synthetic dataset; *MIDI bal.*—synthetic set with balanced classes; *all+MIDI*—three public datasets plus 1% split of synthetic dataset; *all+MIDI bal.*—three public datasets plus the 1% split of the balanced synthetic dataset; *pt MIDI* and *pt MIDI bal.*—pre-trained on the *MIDI* and *MIDI bal.* datasets respectively and fine tuned on *all*. The first set of bars on the left (SUM) shows the overall *sum* F-measure value.

for each of the instrument classification schemas. This will give insight in how the systems perform on the synthetic dataset and how relevant the data volume is for each of the schemas.

In the final set of experiments, models trained with different combinations of synthetic and real data will be evaluated. The evaluation will show how well models trained on synthetic data can generalize on real world data. Mixing the real world datasets with the symbolic data is a first, simple approach of leveraging a balanced dataset to improve detection performance of underrepresented drum instrument classes in currently available datasets. To be able to compare the results, models are trained on all of the public datasets (*all*), the full synthetic dataset (*MIDI*), the balanced versions of the synthetic dataset (*MIDI bal.*), a mix of the public datasets and the 1% subset of the synthetic dataset (*all+MIDI*), and a mix of the public datasets and a 1% subset of the balanced synthetic datasets (*all+MIDI bal.*). Additionally, models pre-trained on the *MIDI* and *MIDI bal.* datasets with additional refinement on the *all* dataset were included. We only compare a mix of the smaller public datasets to the other sets, since models trained on only one small dataset have the tendency to overfit, and thus generalize not well—which makes comparison problematic.

6. RESULTS AND DISCUSSION

The results of the first set of experiments is visualized in Table 2, which shows the 3-fold cross-validation results for models trained on public datasets with 3, 8, and 18 labels. The resulting F-measure values are not surprising: for the 3-class scenario the values are close to the reported values in the related work. Differences are due to slightly different models and hyper-parameter settings for training. As expected, especially the *sum* values drop for the cases of 8 and 18 classes. It can be observed, that the CRNN performs best for all sets in 18 class scenario and for two out of three sets for the eight class scenario.

Table 3 shows the results for models trained on synthetic datasets with 3, 8, and 18 labels. As expected, there is a tendency for the models trained on the 1% subset to perform worse, especially for the CRNN. However, this effect is not as severe as suspected. This might be due to the fact that, while different drum kits were used, the synthetic set is still quite uniform, given its size. The overall results for the balanced sets are worse than for the normal set. This is expected, since the difficulty of the balanced sets is much greater than for the imbalanced one (sparse classes can be ignored by the models without much penalty). Figure 4 shows a comparison of F-measure values for individual instruments classes when training on *MIDI* and *MIDI bal.* sets. The plot shows, that performance for underrepresented classes improves for the balanced set, which was the goal of balancing the set. A downside is that the performance for classes which have a higher frequency of occurrence in the *MIDI* dataset decreases in most cases, which contributes to the overall decrease. However, this effect is less severe in the 8 class case.

A general trend which can be observed, especially in the scenarios with more instrument class labels, is that CRNNs consistently outperform CNNs. Since this is true for all other experiments as well, and for reasons of clarity, we will limit the results for the next plots and tables to those of the CRNN model.

Table 4 shows the F-measure results for the CRNN model trained on different dataset combinations and evaluated on public datasets. In figure 5, a detailed look in the context of cross-datasets evaluation on instrument class basis for the *ENST* dataset is provided. As mentioned in section 5, results for models trained on only one public dataset are not included in this chart. While the performance for those is higher, they are slightly overfitted to the individual datasets and do not generalize well to other datasets, therefore a comparison would not be meaningful. Although an overall big performance improvement for previously underrepresented classes can not be observed, several interesting things are visible: *(i.)* both the models trained solely on the *MIDI* and the *MIDI bal.* datasets generalize surprisingly well to the real world dataset; *(ii.)* in some cases, performance improvements for underrepresented classes can be observed (e.g. for 18 classes: LT, MT, RD, CRC, CHC), when using the synthetic data; *(iii.)* bal-

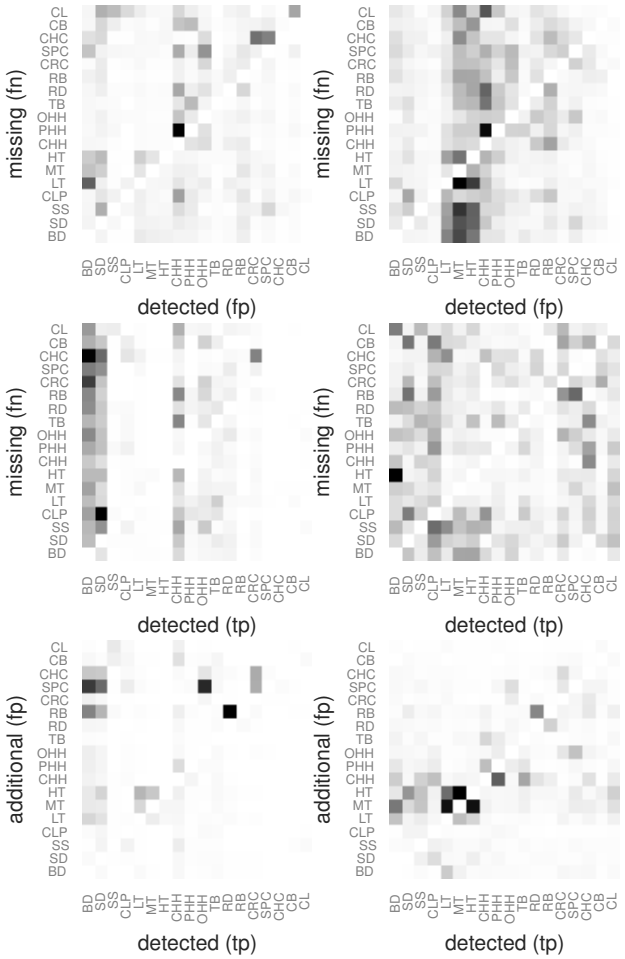


Figure 6: Left column shows matrices for *MIDI* set, right column shows matrices for *MIDI bal.* set, both for the 18 classes scenario. From top to bottom, the matrices display: classic confusions (fn/fp), masking by true positives (fn/tp), and positive masking (excitement—fp/tp).

ancing the instruments, while effective within the evaluation for the synthetic dataset, seems not to have a positive effect in the cross-dataset scenario and when mixing dataset; and *(iv.)* using pre-training on the *MIDI* set with refinement on the *all* set, seems to produce models which are better suited to detect underrepresented classes while still performing well on other classes.

To gain more insight into which errors the systems make when classifying within the 8 and 18 class systems, three sets of pseudo confusion matrices were created. We term them *pseudo* confusion matrices because one onset instance can have multiple classes, which is usually not the case for classification problems. These three pseudo confusion matrices indicate how often *(i.)* a false positive for another instrument was found for false negatives (classic confusions); *(ii.)* a true positive for another instrument was found for false negatives (onset masked or hidden); and *(iii)* a true positive for another instrument was found for a false positive (positive masking or excitement). Figure 6 shows examples of these matrices for the *MIDI* and *MIDI bal.* sets in the 18 class scenario. The images lead to intuitive conclusions: similar sounding instruments may get confused (BD/LT, CHH/PHH), instruments with energy over a wide frequency range mask more delicate instruments as well as similar sounds (HT/BD, CLP/SD), and similar sounding instruments lead to false positives (LT/MT/HT, RB/RD). Many of these errors may very well be made by human transcribers as well. This also strengthens the assumption that instrument mappings are not well defined: boundaries of the frequency range between bass drum, low, mid and high toms are not well defined, the distinction between certain cymbals is sometimes difficult even for humans, and different hi-hat sounds are sometimes only distinguishable given more context, like genre or long term relations within the piece.

To further improve performance, an ensemble of models trained on different datasets (synthetic and real, including balanced variants) can be used. However, experience shows that while these systems often perform best in real world scenarios and in competitions (e.g. MIREX), they give not so much insight in an evaluation scenario.

7. CONCLUSION

In this work we discussed a shortcoming of current state-of-the art automatic drum transcription systems: the limitation to three drum instruments. While this choice makes sense in the context of currently available datasets, some real world applications require transcription of more instrument classes. To approach this shortcoming, we introduced a new and publicly available large scale synthetic dataset with balanced instrument distribution and showed that models trained on this dataset generalize well to real world data. We further showed that balancing can improve performance for usually underrepresented classes in certain cases, while overall performance may decline. An analysis of mistakes made by such systems was provided and further steps into this directions were discussed. The dataset, trained models and further material are available on the accompanying webpage.[12]

8. ACKNOWLEDGEMENTS

This work has been partly funded by the Austrian FFG under the BRIDGE 1 project *SmarterJam* (858514), as well as by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC Grant Agreement No. 670035, project *CON ESPRESSIONE*).

9. REFERENCES

[1] Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees, "Drum transcription via joint beat and drum modeling using convolutional recurrent neural networks," in *Proc. 18th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Suzhou, CN, Oct. 2017.

[2] Carl Southall, Ryan Stables, and Jason Hockman, "Automatic drum transcription using bidirectional recurrent neural networks," in *Proc. 17th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, New York, NY, USA, Aug. 2016.

[3] Carl Southall, Ryan Stables, and Jason Hockman, "Automatic drum transcription for polyphonic recordings using soft attention mechanisms and convolutional neural net-

[12] `http://ifs.tuwien.ac.at/~vogl/dafx2018`

works," in *Proc. 18th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Suzhou, China, Oct. 2017.

[4] Rich Caruana, "Multitask learning," in *Learning to learn*. Springer, 1998.

[5] Olivier Gillet and Gaël Richard, "Automatic transcription of drum loops," in *Proc. 29th IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Montreal, QC, Canada, May 2004, vol. 4.

[6] Marius Miron, Matthew EP Davies, and Fabien Gouyon, "An open-source drum transcription system for pure data and max msp," in *Proc. 38th IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, Vancouver, BC, Canada, May 2013.

[7] Kazuyoshi Yoshii, Masataka Goto, and Hiroshi G Okuno, "Drum sound recognition for polyphonic audio signals by adaptation and matching of spectrogram templates with harmonic structure suppression," *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 15, no. 1, 2007.

[8] Jouni Paulus and Anssi Klapuri, "Drum sound detection in polyphonic music with hidden markov models," *EURASIP Journal on Audio, Speech, and Music Processing*, 2009.

[9] Chih-Wei Wu and Alexander Lerch, "Automatic drum transcription using the student-teacher learning paradigm with unlabeled music data," in *Proc. 18th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Suzhou, China, Oct. 2017.

[10] George Tzanetakis, Ajay Kapur, and Richard I McWalter, "Subband-based drum transcription for audio signals," in *Proc. 7th IEEE Workshop on Multimedia Signal Processing*, Shanghai, China, Oct. 2005.

[11] Maximos A. Kaliakatsos-Papakostas, Andreas Floros, Michael N. Vrahatis, and Nikolaos Kanellopoulos, "Real-time drums transcription with characteristic bandpass filtering," in *Proc. Audio Mostly: A Conf. on Interaction with Sound*, Corfu, Greece, 2012.

[12] Olivier Gillet and Gaël Richard, "Supervised and unsupervised sequence modelling for drum transcription," in *Proc. 8th Intl. Conf. on Music Information Retrieval (ISMIR)*, Vienna, Austria, Sept. 2007.

[13] Derry FitzGerald, Bob Lawlor, and Eugene Coyle, "Subband independent subspace analysis for drum transcription," in *Proc. 5th Intl. Conf. on Digital Audio Effects (DAFx)*, Hamburg, Germany, 2002.

[14] Andrio Spich, Massimiliano Zanoni, Augusto Sarti, and Stefano Tubaro, "Drum music transcription using prior subspace analysis and pattern recognition," in *Proc. 13th Intl. Conf. on Digital Audio Effects (DAFx)*, Graz, Austria, 2010.

[15] Christian Dittmar and Christian Uhle, "Further steps towards drum transcription of polyphonic music," in *Proc. 116th Audio Engineering Soc. Conv.*, Berlin, Germany, May 2004.

[16] Chih-Wei Wu and Alexander Lerch, "Drum transcription using partially fixed non-negative matrix factorization with template adaptation," in *Proc. 16th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Málaga, Spain, Oct. 2015.

[17] Christian Dittmar and Daniel Gärtner, "Real-time transcription and separation of drum recordings based on nmf decomposition," in *Proc. 17th Intl. Conf. on Digital Audio Effects (DAFx)*, Erlangen, Germany, Sept. 2014.

[18] Chih-Wei Wu, Christian Dittmar, Carl Southall, Richard Vogl, Gerhard Widmer, Jason Hockman, Meinhard Müller, and Alexander Lerch, "A review of automatic drum transcription," *IEEE Trans. on Audio, Speech and Language Processing*, vol. 26, no. 9, Sept. 2018.

[19] Richard Vogl, Matthias Dorfer, and Peter Knees, "Recurrent neural networks for drum transcription," in *Proc. 17th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, New York, NY, USA, Aug. 2016.

[20] Richard Vogl, Matthias Dorfer, and Peter Knees, "Drum transcription from polyphonic music with recurrent neural networks," in *Proc. 42nd IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA, USA, Mar. 2017.

[21] Sebastian Böck, Florian Krebs, and Gerhard Widmer, "Joint beat and downbeat tracking with recurrent neural networks," in *Proc. 17th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, New York, NY, USA, 2016.

[22] Chih-Wei Wu and Alexander Lerch, "On drum playing technique detection in polyphonic mixtures," in *Proc. 17th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, New York City, United States, August 2016.

[23] Justin Salamon, Rachel M Bittner, Jordi Bonada, Juan José Bosch Vicente, Emilia Gómez Gutiérrez, and Juan Pablo Bello, "An analysis/synthesis framework for automatic f0 annotation of multitrack datasets," in *Proc. 18th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Suzhou, China, Oct. 2017.

[24] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, Oct. 2014.

[25] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, Nov. 1997.

[26] Sebastian Böck and Gerhard Widmer, "Maximum filter vibrato suppression for onset detection," in *Proc 16th Intl Conf on Digital Audio Effects*, Maynooth, Ireland, Sept. 2013.

[27] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[28] Olivier Gillet and Gaël Richard, "Enst-drums: an extensive audio-visual database for drum signals processing," in *Proc. 7th Intl. Conf. on Music Information Retrieval (ISMIR)*, Victoria, BC, Canada, Oct. 2006.

[29] Carl Southall, Chih-Wei Wu, Alexander Lerch, and Jason Hockman, "Mdb drums – an annotated subset of medleydb for automatic drum transcription," in *Late Breaking/Demos, 18th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Suzhou, China, Oct. 2017.

[30] Rachel M Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello, "Medleydb: A multitrack dataset for annotation-intensive mir research.," in *Proc. 15th Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Taipei, Taiwan, Oct. 2014, vol. 14.

8

DRUM TRANSCRIPTION METHODS IN COMPARISON

This chapter covers attempts to evaluate and compare methods for ADT under controlled conditions. To this end methods introduced in this work alongside other state of the art methods published in recent years, are considered. As mentioned in the introduction, a second class of drum transcription methods prevalent in recent related work are NMF-based approaches. Besides them, other deep-learning-based methods using a similar processing pipeline and network architectures are the predominant class of systems being used in ADT publications of recent years. This chapter covers the contents of three works which are part of the list of additional publications of this thesis. The first one is an overview article on drum transcription which also performs a thorough evaluation of NMF-based and RNN-based ADT systems:

Chih-Wei Wu, Christian Dittmar, Carl Southall, Richard Vogl, Gerhard Widmer, Jason Hockman, Meinhard Müller, and Alexander Lerch. "A Review of Automatic Drum Transcription". In: *IEEE Transactions on Audio, Speech and Language Processing* 26.9 (2018).

Furthermore, this chapter covers two MIREX submissions for the drum transcription task from 2017 and 2018. In them, modifications of methods presented in the previous chapters are introduced and the results of those methods for the drum transcription task are presented:

Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees. "MIREX Submission For Drum Transcription 2017". In: *MIREX extended abstracts, 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017.

Richard Vogl and Peter Knees. "MIREX Submission For Drum Transcription 2018". In: *MIREX extended abstracts, 19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018.

The following sections will provide a brief overview of the findings in these publications.

8.1 COMPARISON OF STATE-OF-THE-ART SYSTEMS

This section will focus on experiments and findings presented in the drum transcription review article by Wu et al. [105]. First, this work

presents an detailed introduction to the drum transcription task. Then a detailed related work section follows, providing an overview of all relevant publications dealing with ADT. After discussing different processing steps commonly used in ADT systems, a new classification scheme for drum transcription methods is introduced. The classification scheme is designed to be extensible which allows future approaches using yet unknown technologies to be covered. A large part of the article deals with describing current RNN-based and NMF-based systems. Subsequently, ten different variants of state-of-the-art ADT methods are evaluated and their performance under equal test conditions is compared. To this end, three different evaluation strategies on three well-known publicly available drum transcription datasets are introduced. The three evaluation strategies are: *(i)* a three-fold cross-validation on the natural splits of each dataset, *(ii)* a randomized 70/15/15% train/validation/test split evaluation, and *(iii)* a cross-dataset evaluation.

The results show that, overall, RNN and NMF systems both perform well on the used datasets. However, RNN systems have the tendency to outperform NMF systems on more complex data, while also providing more flexibility for future improvements and adaptations. For example, the necessary relatively large number of columns for the basis matrix of unconstrained NMF systems is a major challenge when trying to adapt NMF-based methods to transcribe more than three drum instruments. On the other hand, NMF-based systems require very little training data compared to data requirements of NN-based systems. Depending on the application, both method classes can be valid choices.

8.2 MIREX CHALLENGE

MIREX [56] is an annually held event which aims at evaluating MIR algorithms and methods on common data under equal conditions. Often, different evaluation strategies and datasets are used in publications which introduce methods to solve MIR tasks. MIREX was introduced to provide the infrastructure for a comparison of these methods in a controlled environment. The MIREX community usually holds its meeting alongside the International Society for Music Information Retrieval (ISMIR) conference, where also results are published.

A MIREX drum transcription task was first introduced in 2005 with the first installation of MIREX. The task was re-run in 2006, using an extended set of data, consisting of three datasets covering a variety of musical genres. The focus was on three drum instruments, namely bass drum, snare drum, and hi-hat, as was customary at that time. For evaluation, F-measure, precision, and recall (as discussed in Section 2.1.3, this is a common and valid strategy), are used.

After not being run for eleven years, the drum transcription task was reestablished in 2017, using two of the three original evaluation

Algorithm	type	fm mean	BD fm	SD fm	HH fm
RV1 [94]	CRNN	**0.71**	**0.82**	**0.70**	**0.53**
RV2 [94]	RNN	0.67	0.78	0.67	0.51
RV3 [94]	CNN	0.68	0.81	0.64	0.51
RV4 [94]	ensemble	0.70	0.81	**0.70**	0.52
CS1 [79]	RNN	0.61	0.79	0.55	0.46
CS2 [79]	RNN	0.63	0.78	0.57	0.49
CS3 [79]	RNN	0.63	0.78	0.58	0.49
CW1 [106]	NMF	0.51	0.68	0.48	0.38
CW2 [106]	NMF	0.55	0.70	0.55	0.40
CW3 [106]	NMF	0.53	0.67	0.46	0.42

Table 8.1: F-measure (fm) results for the 2017 MIREX drum transcription task. RV1-RV4 represent methods introduced in this thesis, and slight variations thereof. The individual columns represent overall mean F-measure results (fm mean), results for bass drum (BD), snare drum (SD), and hi-hat (HH).

datasets alongside three newly created sets. The new evaluation data covers a variety of tracks and shorter excerpts, comprising recorded music pieces of different genres as well as synthetic data, with and without accompaniment. As in the first renditions in 2005 and 2006, the evaluation only targeted bass drum, snare drum, and hi-hat. This was due to the fact that most state-of-the-art systems still only considered these instruments. With the introduction of systems that were trying to deal with a larger variety of drum instruments, in 2018, a second subtask considering a total of eight drum instruments was additionally introduced. To this end, another dataset was added to the pool of evaluation data.

There are two different submission protocols for MIREX: algorithms can be submitted for so-called train/test tasks, or simply as pre-trained models. For train/test tasks, trainable algorithms are submitted which are then trained and subsequently evaluated on separate datasets on the MIREX servers. In case of pre-trained algorithms, systems trained on a public training set are submitted and are then evaluated on a secret test set. For the drum transcription task a pre-trained-model submission protocol was chosen. This was due to the fact that many methods for this task are based on deep learning for which training is computationally expensive, making it difficult to run a train/test task.

The results of the 2017 evaluation are shown in Table 8.1, while results of the 2018 tasks can be found in Table 8.2 and Table 8.3. The complete description of submissions and detailed results can be found on the drum transcription task's results pages, hosted on the MIREX wiki website: `http://www.music-ir.org/mirex/wiki/2017:Drum_Transcription_Results` and `http://www.music-ir.org/mirex/wiki/2018:Drum_Transcription_Results`.

*Algorithm	comment	fm mean	fm sum	KD fm	SD fm	HH fm
RV1 [94]	CRNN	**0.69**	**0.74**	0.78	**0.69**	0.53
RV2 [94]	CNN	0.65	0.72	0.75	0.65	0.50
CS2 [79]	RNN	0.63	0.70	0.78	0.59	0.50
CS4 [79]	RNN	0.62	0.68	0.77	0.58	0.47
JAR1 [40]	CNN	**0.69**	0.72	0.78	0.65	0.53
JAR2 [40]	CNN	0.68	0.73	0.78	0.65	0.53
JAR3 [40]	CNN	**0.69**	0.73	**0.79**	0.65	**0.54**
JAR5 [40]	CNN	0.67	0.71	0.77	0.63	0.53
JS1		0.61	0.62	0.75	0.59	0.43
JS2		0.62	0.64	0.78	0.63	0.40
JS3		0.61	0.62	0.76	0.63	0.36
JS4		0.57	0.58	0.73	0.61	0.32

Table 8.2: F-measure (fm) results for the 2018 MIREX drum transcription task for three drum instrument classes. RV1 and RV2 represent approaches introduced in this work. The individual columns represent overall mean F-measure results (fm mean), results for bass drum (BD), snare drum (SD), and hi-hat (HH).

The results show that research at the moment focuses on NN-based approaches, which seem to outperform NMF-based techniques on diverse data. Furthermore, transcribing more than three instruments seems to be particularly difficult. Nevertheless, a performance boost in the case of eight instrument classes can be observed when using the method proposed in Chapter 7. Overall, methods introduced in this thesis have been the most successful for recent renditions of the MIREX drum transcription tasks.

8.3 CONTRIBUTIONS OF AUTHORS

For the ADT overview article [105], my main contributions were writing the RNN parts together with Carl Southall, writing the challenges section, and running my methods on the data used for evaluation. I additionally helped writing the rest of the article by proofreading, drafting some of the figures, contributing to the related work sections, and helping to design the new classification scheme in discussions and writing.

For both MIREX submissions I prepared the algorithms for submission and wrote the extended abstracts. Matthias Dorfer contributed to the first submission by helping with the work for the original paper [94]. Gerhard Widmer and Peter Knees, acted as supervisors and provided feedback for the extended abstracts.

Additionally, I was responsible for running the ADT MIREX task as one of three task captains for both instances. As such, I sent out the call for submissions and updated the MIREX wiki page, helped collect

and prepare the datasets, collected the submissions and extended abstracts from participants, ran the evaluation, created result tables and updated the MIREX result pages.

*Algorithm	comment	fm mean	fm sum	BD fm	SD fm	TT fm	HH fm	CY fm	RD fm	CB fm	CL fm
RV3 [101]	CRNN	**0.62**	**0.68**	0.78	**0.52**	**0.37**	0.58	0.38	**0.65**	0.82	0.71
CS2 [79]	CNN	0.47	0.54	0.78	0.50	0.15	0.60	0.19	0.14	0.02	0.02
CS4 [79]	CNN	0.50	0.56	**0.79**	0.51	0.14	**0.62**	0.20	0.13	0.02	0.02
JS1		0.55	0.57	0.75	0.49	0.15	0.52	0.20	0.20	0.68	**0.90**
JS2		0.58	0.61	**0.79**	**0.52**	0.19	0.52	0.32	0.41	0.80	**0.90**
JS3		0.57	0.62	0.77	0.51	0.23	0.50	0.41	0.47	**0.82**	**0.90**
JS4		0.54	0.59	0.73	0.48	0.29	0.46	**0.39**	0.50	**0.82**	**0.90**

Table 8.3: F-measure (fm) results for the 2018 MIREX drum transcription task for eight drum instrument classes. RV3 represents the work on multi-instrument transcription introduced in this thesis. The individual columns represent overall *mean* and overall *sum* (compare Section 2.1.3) F-measure results (fm mean, fm sum), results for bass drum (BD), snare drum (SD), tom toms (TT), hi-hat (HH), cymbals (CY), ride (RD), bells (CB), and stick clicks/claves (CL).

Part III

DRUM PATTERN GENERATION

9 SYNOPSIS

The individual chapters of this part introduce the main publications of this thesis that focus on automatic drum pattern generation. Similar as in the previous part, the introduced methods are based on deep learning techniques. While in the first three works the generation engine is based on RBMs, the last work uses GANs to create symbolic drum patterns. The publications also introduce different user interface (UI) prototypes used for testing the drum pattern generation engines. To evaluate these systems, qualitative and quantitative user studies are conducted.

The first work, covered in Chapter 10, introduces an RBM-based drum pattern generation approach. As UI, a simple step-sequencer-based graphical user interface (GUI) controlled with a mouse alongside a hardware MIDI controller is used. The focus of this work is pattern variation, i.e. the starting point for the generative model is a drum pattern provided by the user. This seed pattern is used to implicitly extract rhythmical properties (latent variables learned during training) which are then used to generate patterns in the vicinity of the seed pattern in the latent variable space. As a first attempt of evaluating such a prototype, a small-scale user study is conducted.

In Chapter 11 the RBM generation engine and UI introduced in Chapter 10 is evaluated in detail using two different user studies. To this end, a qualitative user study with eleven experts as well as a large scale web-based survey are conducted. In these studies the RBM-based drum pattern generator is evaluated against a GA-based approach, a database-lookup approach, and patterns generated by an expert. The patterns which were created by an expert act as a baseline. A statistical analysis and a discussion of the interviews reveal that each approach has strengths and weaknesses regarding different investigated properties.

Chapter 12 deals with introducing improvements to the RBM pattern generation engine. In comparison to the method used in Chapter 10 and 11, drum patterns for all instruments are generated simultaneously and a different distance metric is used to measure pattern similarity. Furthermore, based on feedback of the previous user studies, new features are added to the UI while the interaction paradigm is switched from mouse/hardware-controller-based to a touch-interface-based UI. Similarly as in the previous chapter, a qualitative user study consisting of interviews with ten experts was conducted, where the new UI and new features are evaluated. Additionally, a randomized

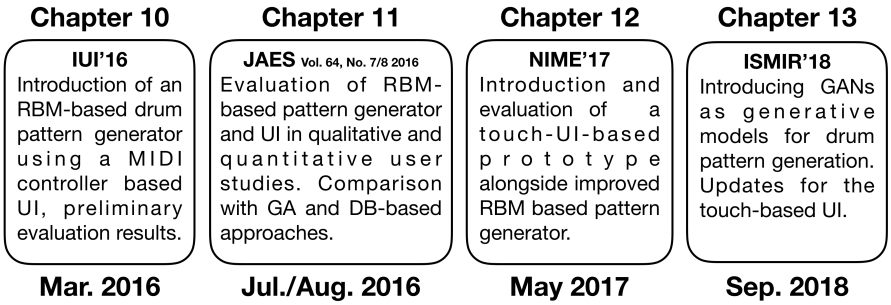


Figure 9.1: Timeline of publications and brief overview of topics covered in them, with corresponding chapters.

A/B comparison between the RBM-based approach used in Chapter 11 and the improved version from this chapter is performed.

The fourth work, which is discussed in Chapter 13, introduces a novel drum pattern generation engine using GANs. Since GANs are notoriously difficult to train, much work was invested to select and fine tune model architecture as well as training approach. To be able to control properties of generated patterns, the GAN is conditioned on genre as well as on features representing complexity and loudness extracted from the drum patterns. As training data for GAN training, two different datasets are used. The first one consists of drum patterns extracted from a large scale synthetic dataset featuring drum annotations. The second one is created by transcribing drum tracks of a large scale genre dataset using the drum transcription approach introduced in Chapter 7. The resulting pattern generation engine is demonstrated in an updated version of the touch UI prototype introduced in Chapter 12.

Figure 9.1 puts the individual publications into perspective and provides an overview of the timeline while indicating the chapters in which those publications can be found.

10 DRUM PATTERN GENERATION WITH RBMs

10.1 OVERVIEW

The contents for this chapter were published in the following paper:

Richard Vogl and Peter Knees. "An Intelligent Musical Rhythm Variation Interface". In: *Companion Publication 21st International Conference on Intelligent User Interfaces*. Sonoma, CA, USA, 2016.

As discussed in Section 2.2.1, a promising approach for drum pattern generation is to use RBMs. In this chapter, such an RBM-based drum pattern variation engine is introduced. The RBM is trained on a dataset derived from example drum loops from Native Instruments' Maschine[1] software. For training, persistent contrastive divergence [86] with additional modifications enforcing latent variable selectivity and sparsity [27], is used. For generating new drum patterns, a seed pattern entered by the user is provided as input for the RBM. Subsequently, Gibbs sampling is performed to generate variations of the seed pattern. Doing so allows the RBM to be primed with a latent variable state corresponding to the desired properties for newly generated drum patterns. The idea is that this way the Gibbs sampling will produce new and interesting drum patterns that exhibit similar properties as the entered seed pattern. Alongside the pattern generation engine, a simple UI prototype is introduced. The prototype consists of a step sequencer interface using a mouse to input patterns, while parameters for pattern generation are modified via a hardware MIDI controller.

For evaluation, a small-scale user study consisting of qualitative expert interviews is used. In the user study ten experts from the field of music production and performance are interviewed while exploring the prototype. The results of this study give indications that RBMs are, to a certain degree, able to produce musically meaningful drum patterns. Furthermore, the findings indicate that while the UI prototype could find application in a studio environment, a live application would be problematic.

1 https://www.native-instruments.com/en/products/maschine/production-systems/maschine/

10.2 CONTRIBUTIONS OF AUTHORS

For this work I was responsible for data preparation, RBM training, UI development, conducting the experiments and interviews, as well as writing the paper.

Peter Knees acted as supervisor and helped with writing the paper by providing valuable feedback.

As indicated by the acknowledgments, the original data used for RBM training was collected by Matthias Leimeister, and the basis on which the UI prototype was built was provided by Michael Hlatky as part of a common interface definition used for the experiments covered in Chapter 11.

An Intelligent Musical Rhythm Variation Interface

Richard Vogl
Department of Computational Perception
Johannes Kepler University
Linz, Austria
richard.vogl@jku.at

Peter Knees
Department of Computational Perception
Johannes Kepler University
Linz, Austria
peter.knees@jku.at

IUI'16 Companion, March 7–10, 2016, Sonoma, CA, USA.
ACM 978-1-4503-4140-0/16/03.
http://dx.doi.org/10.1145/2876456.2879471

Abstract
The drum tracks of electronic dance music are a central and style-defining element. Yet, creating them can be a cumbersome task, mostly due to lack of appropriate tools and input devices. In this work we present an artificial-intelligence-powered software prototype, which supports musicians composing the rhythmic patterns for drum tracks. Starting with a basic pattern (seed pattern), which is provided by the user, a list of variations with varying degree of similarity to the seed pattern is generated. The variations are created using a generative stochastic neural network. The interface visualizes the patterns and provides an intuitive way to browse through them. A user study with ten experts in electronic music production was conducted to evaluate five aspects of the presented prototype. For four of these aspects the feedback was generally positive. Only regarding the use case in live environments some participants showed concerns and requested safety features.

Author Keywords
Rhythm pattern generation; restricted Boltzmann machines; machine learning; neural networks; generative stochastic models.

ACM Classification Keywords
H.5.2 [User Interfaces]: Graphical user interfaces, Input devices and strategies

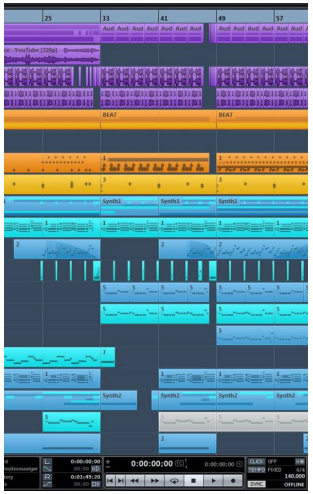


Figure 1: An EDM track being arranged in a DAW software. DAWs are programs used to produce music. The horizontal colored bars represent the tracks for instruments. The orange track contains the basic rhythmic pattern and its variations during build-ups, breaks, and fills.

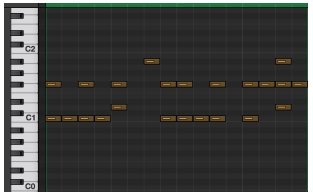


Figure 2: A piano roll editor showing a manually entered rhythm pattern.

Introduction

Nowadays, more than ever before, digital tools for music production play an important role in the workflow of music producers. Such tools cover applications like digital audio workstations (DAWs; see figure 1), integrated hardware/software solutions like grooveboxes, and software tools and plugins like synthesizers and audio effects. The *GiantSteps*[1] project focuses on simplifying the workflow of music producers by developing *intelligent agents* for the usage in electronic dance music (EDM) production and performance.

As yet, drum tracks are built by arranging rhythm patterns from a pattern library, or by creating patterns manually. Using predefined patterns bears the risk of sounding unoriginal, while creating them manually is a time consuming task and requires more musical knowledge. Entering rhythm patterns in a DAW is done using a mouse or MIDI controllers (keyboards and drum pads) to set notes in a piano roll (see figure 2) or similar editor. Step-sequencer-like interfaces are usually a feature of grooveboxes and drum machines and are typically found in setups for live performances.

When it comes to samplers and synthesizers for drums in EDM, a wide variety of commercial products as well as a lively research community exist. However, there are few works on automated drum rhythm variation and creation. In the works of Kaliakatsos–Papakostas et al. [2] and Ó Nuanáin et al. [4] genetic algorithms to generate rhythmic patterns are used. Genetic algorithms tend to produce random variations and the results strongly depend on the used fitness function.

Restricted Boltzmann machines (RBM, introduced in [5]) form a group of generative stochastic neural networks which are well suited for pattern generation. Battenberg et al. [1] use a variant, the conditional RBM, to classify the meter of drum patterns. They mention the capability of the learned model to generate drum patterns similar to the training data, given a seed pattern. Lattner et al. [3] use a similar method to predict the evolution of features for song segmentation.

[1] http://www.giantsteps-project.eu/

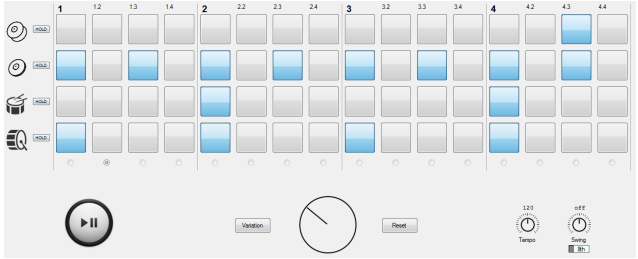


Figure 3: Screenshot of the prototype. The 4 by 16 step sequencer array poses as visualization and input for the drum rhythm patterns. Beneath the array are controls for playback, the pattern variation control to determine the degree of variation, and controls for tempo and swing (all also controllable through a hardware interface).

In this work, we present an intuitive interface for drum pattern generation. Underlying the interface, an RBM, trained on a database of drum patterns is used to create variations of a seed pattern. In addition to presenting the implemented prototype, we report on user feedback we gathered from interviews conducted with experts during hands-on sessions to evaluate the prototype.

UI and Method

The developed prototype aims at supporting the producer of an EDM track creating variations of drum patterns, as well as providing creative input for creating new drum patterns. The visualization and input interface for these pat-

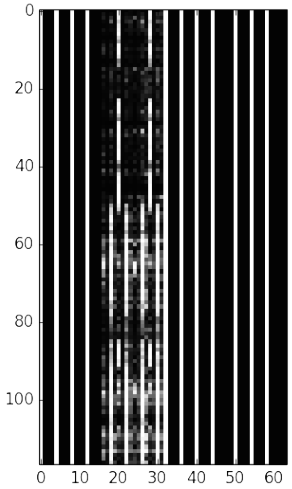


Figure 4: The evolution of the visible nodes of the RBM while creating pattern variations for the snare drum. The x-axis represent the index of the visible node of the RBM. The y-axis represents the number of Gibbs sampling step, starting at the top with the original input pattern and progressing downwards. Active nodes are represented by white, inactive nodes by black pixels.

terns within the prototype employ the well established *step sequencer* user interface (UI) paradigm. The controls for pattern variation are implemented as a dial on which the variations are placed ordered by sparsity and similarity to the seed pattern. Figure 3 shows a screenshot of the prototype's UI.

The output of the prototype is sent via MIDI (Musical Instrument Digital Interface),[2] making the integration into existing setups easy. All UI components, with the exception of the step sequencer array, can be controlled by an external MIDI hardware controller. Musicians and producers are familiar with controlling production software with MIDI controllers – especially in the context of live performances.

To generate meaningful, yet creative patterns, a process which combines obedience to musical rules with elements of surprise and unpredictability is needed. In order to fulfill this requirement, Gibbs sampling of an RBM was chosen as variation method. Apart from being well researched, RBMs feature a technique called *clamping*, which improves the quality of the generated patterns greatly. For details on Gibbs sampling, clamping, and RBM training, the reader is referred to the work by Hinton et al. [6]. RBMs are neural networks and have to be trained on representative training data. As training data a set of 16,513 one-bar drum patterns was used. The patterns were extracted from the sample drum loop library of Native Instrument's *Maschine*[3] software. The library consists of drum patterns for EDM, Hip Hop, and RnB. Since the main focus of this work is EDM, this library was well suited.

To generate variations of the seed pattern, first the seed pattern is entered into the visible layer of the RBM. Then variations for every instrument are generated individually by clamping all other instruments and performing several Gibbs sampling steps. Figure 4 shows the evolution of the visible layer of the RBM performing Gibbs sampling steps. It can be observed how the snare pattern (nodes 16-31) evolves while the other instruments (nodes 0-15 and 32-63) are clamped to their original values. The sorted single-instrument-pattern lists are then combined to full rhythm patterns by using bass drum, snare drum, open, and closed hi-hat patterns at the same indices.

[2] https://en.wikipedia.org/wiki/MIDI

[3] http://www.native-instruments.com/en/products/maschine/production-systems/maschine-studio/

Early Prototype Evaluation

To evaluate the quality of the generated patterns, as well as the interaction with the UI, a user study was conducted of which we report first findings. To this end we interviewed ten experts in EDM creation in a guided, informal way while they were exploring the prototype – see figure 5. Table 1 summarizes the number of positive responses for five evaluated aspects we deemed crucial for the success of such an interface.

Over two thirds of the users (seven of ten) considered the variations to maintain the basic rhythmic idea they entered. Only participants who entered patterns untypical for EDM, complained that the variations did not conserve their basic rhythmic idea. This can easily be explained by the fact that the RBM was trained on EDM patterns and therefore tried to converge on these kind of patterns.

Nine out of ten participants considered the variations produced by the prototype to be musically meaningful. Eight participants commented positively on the way they interact with the prototype (step sequencer, variation dial and hardware controller), as exemplified by these quotes:

> "It works like it should, so I think it is quite user friendly. [...] I also think the scrolling [through the variations] is cool because it is fast and practical."
> JKU-15-05

Figure 5: A study participant using the prototype to explore pattern variations. A MIDI hardware controller is used to enable a more direct interaction.

> "I have tested quite a lot of hardware sequencer things and I think a feature like that would be pretty cool, actually. Especially if it has lots of variations like we had right there." JKU-15-08

Regarding the use of the prototype in live performances, the participants presented themselves cautious. Six out of ten participants stated that they could imagine to use the prototype in a live environment. Some of the participants would use this kind of tool only with the addition of features like a preview function (visually or audible) or the option to limit the degree of variation. The idea of using the prototype in a studio environment was met with enthusiasm. Participants were eager to use the prototype to create variations and get inspiration from it in a production context.

Conclusion

We presented a prototype for an intelligent rhythm agent to assist musicians and producers in the context of EDM production and live performances. A user study was conducted to evaluate both the pattern variation algorithm as well as the UI of the prototype. The study shows that the interaction concept of the prototype is something most participants can imagine working with. It also implies that the acceptance of such a tool in a studio environment would be high, while concerns were raised about precision and reliability when it comes to live performance scenarios. The created patterns were mostly considered musical and in many cases perceived to reflect the basic rhythmic idea of the seed pattern.

aspect	positive comments (out of 10)
seed rhythm is preserved	7
patterns are meaningful	9
prototype interaction	8
would use live	6
would use in studio	9

Table 1: Number of participants giving positive responses wrt. the topics of interest of the user study. The total number of participants (N) was ten.

Acknowledgments

We thank Michael Hlatky for sharing the source code of a simple step sequencer which was used to build the UI and Matthias Leimeister for extracting the *Maschine* drum loop library. This work is supported by the European Union Seventh Framework Programme FP7 / 2007-2013 through the GiantSteps project (grant agreement no. 610591).

REFERENCES

1. E. Battenberg and D. Wessel. 2012. Analyzing Drum Patterns Using Conditional Deep Belief Networks. In *Proc 13th International Society for Music Information Retrieval Conference*.
2. M.A. Kaliakatsos–Papakostas, A. Floros, and M.N. Vrahatis. 2013. evoDrummer: Deriving Rhythmic Patterns through Interactive Genetic Algorithms. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*. Lecture Notes in Computer Science, Vol. 7834. Springer, 25–36.
3. S. Lattner, M. Grachten, K. Agres, and C.E. Cancino Chacón. 2015. Probabilistic Segmentation of Musical Sequences using Restricted Boltzmann Machines. In *Proc 5th Mathematics and Computation in Music Conference*.
4. C. Ó Nuanáin, P. Herrera, and S. Jordà. 2015. Target-Based Rhythmic Pattern Generation and Variation with Genetic Algorithms. In *Proc 12th Sound and Music Computing Conference*.
5. P. Smolensky. 1986. Information processing in dynamical systems: Foundations of harmony theory. *Parallel Dist. Proc.* (1986), 194–281.
6. G. E. Hinton, S. Osindero, and Y. Teh. 2006. A Fast Learning Algorithm for Deep Belief Nets. In *Neural Comput.* Vol. 18. MIT Press, 1527–1554.

11

EVALUATION OF DRUM PATTERN GENERATION METHODS

11.1 OVERVIEW

The contents for this chapter were published in the following article:

Richard Vogl, Matthias Leimeister, Cárthach Ó Nuanáin, Sergi Jordà, Michael Hlatky, and Peter Knees. "An Intelligent Interface for Drum Pattern Variation and Comparative Evaluation of Algorithms". In: *Journal of the Audio Engineering Society* 64.7/8 (2016).

This publication introduces and compares three different drum pattern variation systems: *(i)* a GA-based, *(ii)* an RBM-based, and *(iii)* a database-lookup-based approach. The MIDI-controller-based UI introduced in Chapter 10 is used as a common interface to test these methods. Two different user studies are conducted to evaluate the individual drum pattern generation systems. Data for the first study is collected using an online survey. In the web form of the survey, a visual representation alongside an audio rendering of the drum patterns are presented to the participants. After listening to seed patterns and variations, participants are asked to rate certain aspects of the generated patterns. For this quantitative study additional variations provided by an expert are used as a baseline for evaluation. Additionally, a qualitative study consisting of eleven interviews with music production experts was conducted. The questions of interest covered by the interviews comprise both, aspects of the generated patterns and usability of the UI prototype.

The evaluation reveals that the individual systems have strong points in different areas. While the database-based approach appears to be more conservative, which results in more trust but also less creative patterns, the GA-based approach produces more wild and inspiring patterns which makes it less applicable in live environments. The RBM-based approach provides a compromise between the databased-based and GA-based approach, producing more varying patterns without being too unpredictable, and conserving basic properties of the seed pattern. Participants were also questioned about usability of the user interface. While feedback regarding the interaction with the system was generally positive, five participants deemed the application in a live setting to be problematic. However, most participants found the UI suitable in a studio or production setting.

11.2 CONTRIBUTIONS OF AUTHORS

As first author of this work, I created most of the contents. The RBM based drum pattern generation prototype introduced in [97] was created by myself, I designed and conducted both the web survey as well as most of the interviews. Statistical and qualitative evaluation of the results as well as writing of the larger part of the article was done by me.

Matthias Leimeister contributed the database-based pattern variation system, drafted Section 1 on related work, as well as wrote the basis of Section 3.3 describing his system.

Cárthach Ó Nuanáin contributed the GA based pattern variation system, and wrote the basis for Section 3.5 explaining his system. He also conducted two of the interviews for the qualitative study.

Sergi Jordà acted as supervisor and provided valuable feedback by proofreading the article.

Michael Hlatky defined and provided the common interface for the variation algorithms and also contributed a simple UI on which the UI used for the experiments was based.

Peter Knees acted as supervisor and helped with the design of the statistical evaluation, with writing the article as well as proofreading.

As indicated by the acknowledgments, Stefan Lattner provided help with the Lrn2 framework and support with RBM training.

OPEN ACCESS Freely available online

Journal of the Audio Engineering Society
Vol. 64, No. 7/8, July/August 2016 (© 2016)
DOI: http://dx.doi.org/10.17743/jaes.2016.0016

An Intelligent Interface for Drum Pattern Variation and Comparative Evaluation of Algorithms

RICHARD VOGL[1], MATTHIAS LEIMEISTER[2], CARTHACH Ó NUANAIN[3],
(richard.vogl@jku.at) (matthias.leimeister@native-instruments.de) (carthach.onuanain@upf.edu)

SERGI JORDA[3], MICHAEL HLATKY,[2] *AES Member*, **AND PETER KNEES[1]**

[1]*Department of Computational Perception, Johannes Kepler University Linz, Austria*
[2]*Native Instruments GmbH, Berlin, Germany*
[3]*Music Technology Group, Universitat Pompeu Fabra, Barcelona, Spain*

Drum tracks of electronic dance music pieces are a central and style-defining element. Yet, creating them can be a cumbersome task, mostly due to lack of appropriate tools and input devices. In this work we use a UI prototype that aims at supporting musicians to compose the rhythmic patterns for drum tracks, to compare different algorithms for drum pattern variation. Starting with a basic pattern (seed pattern), which is provided by the user, a list of variations with varying degrees of similarity to the seed pattern is generated. The variations are created using one of the three algorithms compared: (i) a similarity-based lookup method using a rhythm pattern database, (ii) a generative approach based on a stochastic neural network, and (iii) a genetic algorithm using similarity measures as a target function. The interface visualizes the patterns and provides an intuitive way to browse through them. User test sessions with experts in electronic music production were conducted to evaluate aspects of the prototype and algorithms. Additionally a web-based survey was performed to assess perceptual properties of the variations in comparison to baseline patterns created by a human expert. The web survey shows that the algorithms produce musical and interesting variations and that the different algorithms have their strengths in different areas. These findings are further supported by the results of the expert interviews.

0 INTRODUCTION

Nowadays, more than ever before, digital tools for music production play an important role in the workflow of music producers. Such tools cover applications like digital audio workstations (DAW), integrated hardware/software solutions like grooveboxes, and software tools and plugins like synthesizers and audio effects. In the *GiantSteps* project, we focus on simplifying the workflow of music producers by developing *intelligent agents* for the usage in electronic dance music (EDM) production and performance.[1]

Usually, drum tracks are built by arranging rhythm patterns from a pattern library or by creating patterns manually. Using predefined patterns bears the risk of sounding unoriginal, while creating them manually is a time-consuming task and requires more musical knowledge. Entering rhythm patterns in a DAW is typically done using a mouse or MIDI controllers (keyboards and drum pads) to set notes in a piano roll or similar editor. Step-sequencer-like interfaces are usually a feature of grooveboxes and drum machines and are found in many setups for live performances.

The aim of this work is to build a tool that supports musicians in an intuitive way at creating variations or finding inspiration for new drum rhythm patterns. Maintaining the basic style, or rhythmical idea, while providing meaningful and interesting—maybe even surprising—variations is a main goal.

When it comes to samplers and synthesizers for drums in EDM, a wide variety of commercial products as well as a lively research community exist. However, there are few works on automated drum rhythm variation and creation.

1 RELATED WORK

Some commercial products in the field of music production include tools that attempt to automate the creation of a drum track. In Apple's Logic Pro software,[2] the user can

[1] http://www.giantsteps-project.eu/

[2] http://www.apple.com/logic-pro/

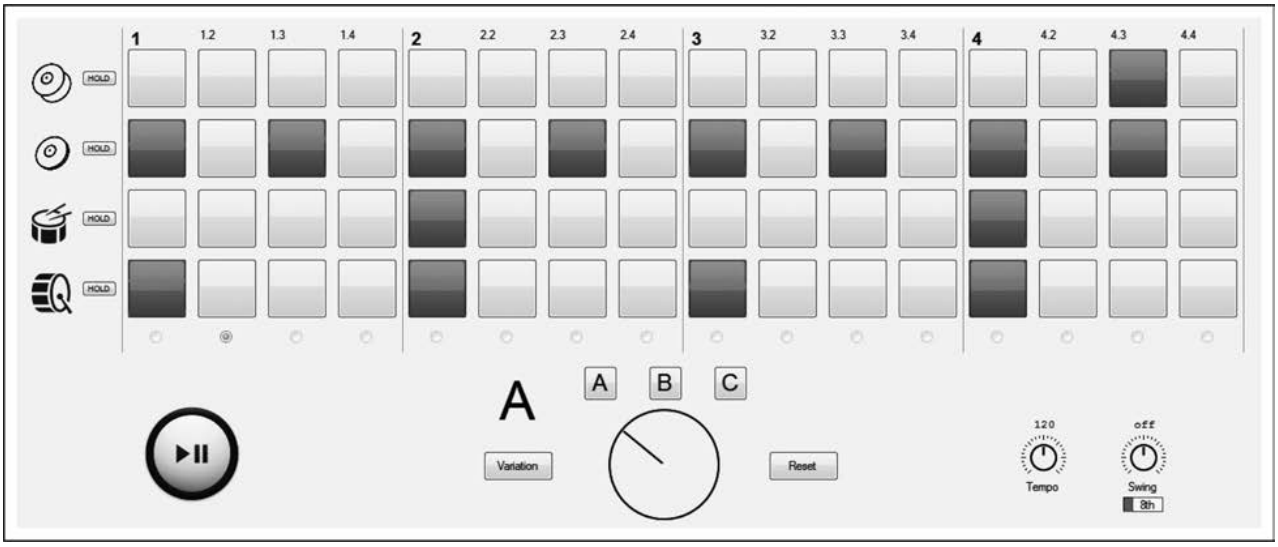


Fig. 1 Screenshot of the prototype. The 4-by-16 step sequencer array acts as visualization and input for the drum rhythm patterns. Beneath the array are controls for playback, the pattern variation control, buttons for algorithm selection, and controls for tempo and swing (all also controllable through a hardware interface).

create a *Drummer* track that contains basic drum patterns that can be changed via an interactive GUI. After choosing a genre-specific drummer (e.g., Rock) and drum kit, the style of the drum track can be varied by moving in a two-dimensional panel that represents the dimensions soft-loud and simple-complex. Steinberg's Groove Agent[3] is a drum plugin that comes with a big selection of drum kits and loops that can be arranged to a full drum track in a DAW project. It includes a mode for performance (*Style Player*), in which the user can choose different styles of patterns based on intensity and complexity. An interesting feature is the automatic mode that varies the complexity of patterns on the fly. EZ Drummer by Toontrack[4] provides a large library of drum patterns together with the option for searching matching patterns based on a seed pattern that the user can record. Since these products are mainly positioned for the production of rock and alternative music, they are only to a certain degree applicable for EDM. Furthermore, professional producers often refrain from using out-of-the-box patterns for fear of sounding unoriginal.

The scientific community mainly focuses on methods and algorithms providing the underlying functionality. Of special interest is the question how the human perception of rhythm, especially the notion of similarity, can be modeled. Toussaint [20] introduces and compares several measures for the similarity of two rhythmic patterns. Among them are the Hamming distance, edit distance, Euclidean distance of inter-onset-interval vectors, and the interval-ratio-distance. To gain insight into similarity of the patterns, phylogenetic trees are built based on the computed distance matrices—a bioinformatics technique that is originally used to visualize the relationship of DNA sequences. In the work of Kaliakatsos-Papakostas et al. [9] an automatic drum rhythm generation tool based on genetic algorithms is described. It generates variations of a base rhythm and allows the user to set parameters of the generation process such as divergence between the base rhythm and the generated ones. Ó Nuanáin et al. [12] use a similar approach for rhythm pattern variation using genetic algorithms. Sioros and Guedes [15] introduce a system that recombines MIDI drum loops in realtime based on a measure for complexity of rhythmic patterns. The complexity is measured by means of syncopation and density. A more detailed discussion of different approaches for measuring syncopation in the same context is presented in [16]. Apart from approaches that compute similarity between rhythmic patterns on a symbolic representation, there exist methods that consider other aspects of rhythmic similarity. Holzapfel and Stylianou [7] use the scale transform to develop a tempo invariant rhythmic similarity measure for music. Jensen et al. [8] as well as Gruhne and Dittmar [5] use logarithmic autocorrelation functions calculated on different forms of onset density functions to obtain tempo invariant rhythmic features. Other works investigate properties of swing and try to quantify the swing-ratios of rhythmic patterns in audio recordings, e.g., [3, 11].

Given training data, machine learning methods can be used to train generative models for rhythm patterns. Paiement et al. [13] introduce a probabilistic model for relative distances between rhythms, which is applied to subsequences of patterns. This is in turn used to predict the continuation of a rhythmic pattern given its past, utilizing a hidden Markov model (HMM). Another group of widely used generative models are restricted Boltzmann machines (RBM) [6]. Boulanger-Lewandowski et al. [2] use an extension of RBMs with recurrent connections to model and generate polyphonic music. Battenberg and Wessel [1] use another variant, the conditional RBM, to analyze drum patterns for their meter. They mention the capability of the learned model to generate drum patterns similar to the training data given a seed pattern. This idea was further

[3] http://www.steinberg.net/en/products/vst/groove_agent/
[4] https://www.toontrack.com/product/ezdrummer-2/

developed in the work by Vogl and Knees [21] to build a drum rhythm variation tool based on an RBM and a step sequencer interface. In this work, this prototype is further extended by incorporating and extensively comparing two more variation algorithms.

2 PATTERN VARIATION USER INTERFACE

To be able to compare and evaluate different algorithms, we extended the interface prototype presented in [21] to accommodate three variation methods that can be selected via the UI. Fig. 1 shows the UI of the prototype.

It resembles a standard drum step sequencer, providing four instruments (bass-drum, snare, open, and closed hi-hat) programmed within one 4/4 measure bar of 16th notes. This type of interface was chosen since it represents one of the prevalent interfaces for drum track creation in EDM. In this context, we focus solely on the variation of the plain symbolic rhythm patterns. Hence, we deliberately decouple the pattern from aspects of accentuation, micro timing, and swing, which are considered independent dimensions and should remain under the control of the artist. For controlling tempo and swing, the presented UI exhibits respective knobs. The latter allows the user to set an additional swing ratio that shifts either 8th or 16th notes to create a "swing feel." These parameters enable the users to recreate a rhythmic style they usually work with, while keeping the UI complexity at the necessary minimum.

A central UI element is the variation browsing knob and the controls to set and reset the pattern for which the variations should be created. After pressing the "variation" button, the selected algorithm generates variations. These variations are ordered ascending regarding the number of active notes. By turning the variation knob to the left, the variations will become more sparse; when turning the knob to the right the variations will become more dense. By pressing the reset button, the variation knob jumps back to the seed pattern. Above the variation knob are buttons to select one of the three variation algorithms. The assignment is randomized after every start of the prototype to avoid any bias introduced by the order. The output of the interface is sent via MIDI. All UI elements can be controlled via MIDI, making it possible to use an external hardware controller, or integrating the prototype into a DAW software.

3 PATTERN VARIATION ALGORITHMS

The aim of the presented algorithms is to support the user in creating interesting drum tracks based on an initial seed pattern. Hence, the variation algorithm receives a one-bar pattern of kick, snare, closed hi-hat, and open hi-hat notes as input. Based on this, a set of 32 new patterns is computed and returned to the UI as suggested variations of the seed pattern. This set is sorted according to density of note events per bar. The seed pattern is placed in the list according to its own density as a starting point for exploring the patterns. Two of the following algorithms are data-driven as they require model training or a database lookup. Therefore, we first describe the collection of rhythm patterns that has been used throughout the experiments as database and for model training.

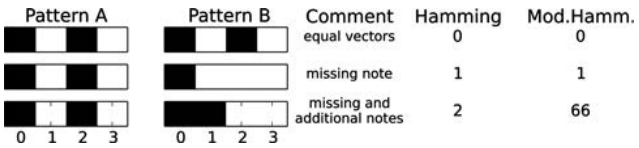


Fig. 2. Rhythm patterns as binary vectors of size four. Depicted are examples for two equal vectors, vectors with missing notes (or additional notes if the right vector is used as reference), and vectors with both missing and additional notes. The columns to the right show the corresponding Hamming and modified Hamming distance.

3.1 Data Set

Maschine[5] is an integrated hardware/software groove box application that is focused on the production of urban and electronic music. The application and its extension packs contain a large library of drum sound samples and examples for drum patterns.

This collection of drum patterns was used to compile a data set for the development of rhythm pattern variation algorithms. Using the MIDI export function of the application, the example drum patterns have been exported to MIDI files and cut into patterns of one bar length. From the exported samples, only the instruments kick, snare, closed hi-hat, and open hi-hat were used. The resulting patterns were checked for exact repetitions and some were removed based on musical constraints. For example, only patterns containing between two and six bass drum notes per bar, between one and five snare drum notes, as well as at least two hi-hat notes were kept. This was done to exempt breaks and very sparse patterns from the database. The final set consists of 2,752 unique patterns.

3.2 Modified Hamming Distance

Both the database-driven approach (Sec. 3.3) as well as the neural-network-based method (Sec. 3.4) use a rhythm pattern similarity function. To calculate the similarity, the rhythm patterns are represented as 64-bit (16 notes, 4 instruments = 64 notes) binary vectors. Then the number of bits with different values between the two vectors is counted (= Hamming distance). An additional offset (64) is added if one vector compared to the other has additional as well as missing notes. See Fig. 2 for examples. This choice was made based on the findings in [20] and [12] as well as the goal to favor patterns that have as much overlap as possible with the seed pattern. It is supposed to create the sensation of a steady evolution when browsing the resulting list of variation patterns, which was sorted using this distance function.

3.3 Database-Driven Approach

The database-driven algorithm (referred to as *DB* in the following) is based on retrieving patterns similar to the seed

[5] http://www.native-instruments.com/products/maschine

pattern from a database and suggesting these as variations. For the rhythm pattern database, the patterns extracted from Maschine (cf., Sec. 3.1) were used. The modified Hamming distance is used to compute the similarity of the seed pattern (query) to every pattern in the database (targets). In case the target pattern contains more notes, but also misses notes from the query, the pattern is modified by adding the missing notes from the query pattern. As a result, generated patterns with more notes than the query always contain all notes from the query pattern. The resulting patterns are sorted according to the computed distance and a list of the 32 most similar patterns is returned.

3.4 Neural Network Based Method

To generate meaningful, yet creative patterns, a process that combines obedience to musical rules with elements of unpredictability is needed. To achieve this, Restricted Boltzmann machines (RBMs, introduced in [17]), which are generative stochastic neural networks, are used. From training, they learn a probability distribution defined by the training data. From this distribution samples can be drawn, which can be used for pattern generation.

RBMs consist of two layers of artificial neurons: the visible layer, which is used as both in- and output and a hidden layer that represents latent variables. The neurons are fully linked only between the two layers (hence the name "restricted"). Fig. 4 depicts an example of a small RBM with four visible nodes and three hidden nodes. The RBM used in this work consists of 64 nodes in the visible layer, which correspond to the notes (16x4) in the step sequencer grid (see Fig. 1). The hidden layer consists of 500 nodes. The training was done using the Lrn2 framework of the Lrn2Cre8 project.[6]

The RBM is trained using the Maschine rhythm pattern data set by means of persistent contrastive divergence (PCD) [19] training. Additionally, latent selectivity and sparsity, as described in [4], as well as drop-out [18] for training are used.

To generate variations of the seed pattern, first, the seed pattern is entered into the visible layer of the RBM. Then, variations for every instrument are generated individually by clamping (nodes fixed to their original input values) all other instruments and performing several Gibbs sampling steps. A Gibbs sampling step consists of (i) calculating the values for the hidden layer by using the input values in the visible layer and the learned network weights, (ii) binarizing the values of the hidden layer by applying the sigmoid function and a random threshold, and (iii) calculating new values for the visible layer by using the values in the hidden layer and the network weights. Fig. 3 shows the evolution of the visible layer of the RBM performing Gibbs sampling steps. It can be observed how the snare pattern (nodes 16–31) evolves while the other instruments (nodes 0–15 and 32–63) are clamped to their original values. For more details on Gibbs sampling, clamping, and RBM training, the reader is referred to the work by Hinton et al. [6] and the other references provided.

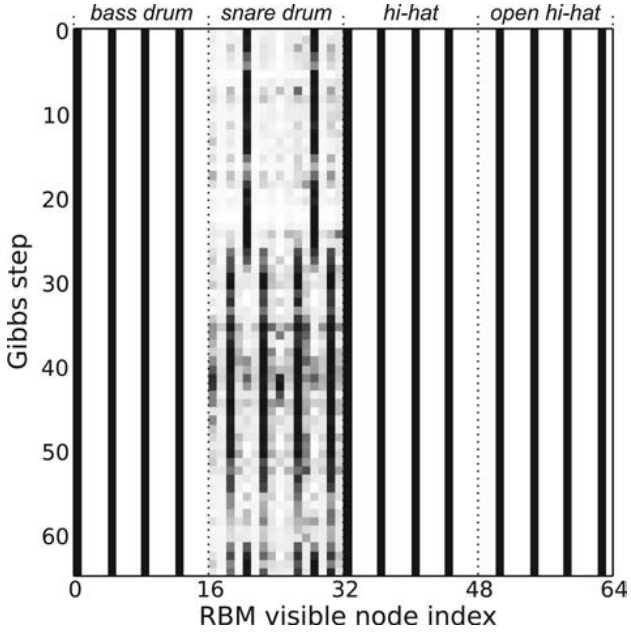


Fig. 3. The evolution of the visible nodes of the RBM while creating pattern variations for the snare drum. The x-axis represent the index of the visible node of the RBM. The y-axis represents the index of the Gibbs sampling step, starting at the top with the original input pattern and progressing downwards. Active nodes are represented by black, inactive nodes by white pixels.

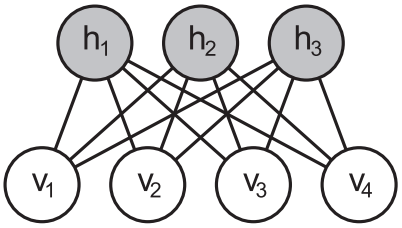


Fig. 4. Structure of a Restricted Boltzmann Machine. White circles represent the visible and gray circle the hidden nodes. Between the two layers, the nodes are fully connected, while nodes within the same layers are not connected.

The generated single-instrument patterns are sorted using our modified Hamming distance. The sorted single-instrument pattern lists are then combined to full rhythm patterns by using bass drum, snare drum, open, and closed hi-hat patterns at the same indices. This approach will subsequently be referred to as *RBM*.

3.5 Genetic Algorithm

Genetic algorithms are frequently used in algorithmic composition and other creative applications in what is often called "generative art." In essence, a genetic algorithm involves successive splicing, pairing, and mutating of data structures in a simulated program of natural selection. At the core of a genetic algorithm lies the fitness function that determines whether individuals fulfill some solution criteria.

[6] https://github.com/OFAI/lrn2

Table 1. Mnemonics used in the article.

Algorithms	*DB*	database based method
	RBM	neural network based method
	GEN	genetic algorithm based method
	EXP	patterns created by an expert
Variations	*sp1*	sparse variation close to seed (–3)
	den1	dense variation close to seed (+3)
	den2	dense variation far from seed (+6)

In [12], a genetic algorithm that generates rhythmic patterns using an automatic fitness function by determining similarity of new patterns to a seed pattern, was presented. In this work an adapted version of this method is used to meet the constraints and requirements of the UI prototype.

The initial population pool is initialized with a uniformly random distributed set of 16x4 binary pattern genomes, conforming to the kick, snare, and hi-hat representation used in this work. The algorithm commences and iterates through successive stages of evolution. During a single stage of evolution, patterns from the population pool are selected and paired for mating. New patterns are generated by selecting two parent patterns for crossover (bits from each are copied to the child) and mutation (a very small portion of the child is randomly modified). The fitness of these new patterns is determined by measuring their rhythmic similarity to the input target pattern using a distance function. The evaluation in [12] revealed that the Hamming distance correlates best with human judgments when dealing with polyphonic patterns. For this reason, as well as to remain consistent with the distance measures used in the other algorithmic approaches, in this work the Hamming distance is used.

Achieving a balance between pattern diversity and reasonable convergence time is one of the main challenges. To this end, two adjustments to the method presented in [12] are made. First, a more conventional roulette selection scheme [14] is adapted: candidates from the population are chosen for pairing and splicing from the fittest 100 members only. Second, a stage of elitism in the selection procedure is introduced: a small fraction of the fittest individuals are simply copied to the next generation. This fraction can be tweaked for drastic decreases in the algorithm's convergence time.

We run the algorithm and store the best member from each generation in a list to get a diverse spread of rhythmic patterns in terms of target similarity. From this list 32 patterns are chosen from equidistantly distributed indexes, to be returned to the UI. The genetic algorithm approach will subsequently be referred to as *GEN*.

Table 1 summarizes the mnemonics of the different algorithms for better comprehension.

4 EVALUATION—EXPERT INTERVIEWS

To evaluate all three algorithms as well as the UI prototype, two separate studies have been carried out. The first study consists of interviews conducted with experts in EDM production and performance. The study's aim was to gather feedback on the quality of the variations produced by the single algorithms as well as on the usability of the UI. Sec. 5 covers the second study. It is a web-based survey in which variations of selected seed patterns were to be rated. The goal of this survey was to obtain quantitative data on various properties of the variation algorithms.

4.1 Method

We conducted interviews with musicians experienced in working with DAWs and producing EDM and/or performing EDM live. The interviews were conducted in a guided and informal way inspired by the "thinking aloud" method [10]. The participants were introduced to the prototype by briefly explaining aim and functionality. Then they were asked to explore the functions and algorithms of the prototype while talking about their experience and thoughts. For every interview, the algorithm button assignment was randomized to avoid experimenter bias. The interviews were recorded and transcribed for later analysis.

The aim was to get answers to five core aspects we deemed crucial for the success of a rhythm pattern variation tool:

1. Is the basic rhythm of the seed pattern preserved in the variations?
2. Are variations musically meaningful?
3. Is the interaction with the prototype intuitive?
4. Would the prototype be useful in a live environment?
5. Would the prototype be useful in a studio/production environment?

Additional comments on feature requests, use-case scenarios, and UI interaction were collected.

4.2 Interview Results

In total, 11 interviews were conducted with musicians in their private studios, at the HAMR'15 hackday in Málaga, and at the Red Bull Music Academy in Paris. After being introduced to the prototype, the participants spent on average 23 minutes (min.: 17, max.: 30 minutes) exploring the behavior of the three pattern variation algorithms (i.e., about 8 minutes per algorithm on average). Five participants used multiple seed patterns (up to three patterns), while the others just used one seed pattern to generate variations. Participants were encouraged to browse through the whole list of variations to get an overall image of how the single algorithms behave. Table 2 summarizes the comments of the interviewees regarding the five aspects of interest identified earlier.

Comments were considered to be positive whenever the interviewees explicitly expressed a positive impression regarding the aspects. For example:

- Rhythm and musicality:
  "That [generating variations] actually worked: It adds stuff on the right, it removes on the left." JKU-15-09

Table 2. Number of participants giving positive responses *wrt.* the topics of interest of the user study.

Aspect	Algo.	Positive comments
rhythm is preserved	DB	10 *91.0%*
	RBM	8 *72.7%*
	GEN	5 *45.5%*
patterns are musical	DB	11 *100%*
	RBM	10 *91.0%*
	GEN	7 *63.6%*
prototype interaction		9 *81.8%*
would use live		6 *54.5%*
would use in studio		10 *91.0%*

Note: The total number of participants (*N*) was eleven.

- Prototype interaction:
  "It works like it should, so I think it is quite user friendly. [. . .] I also think the scrolling [through the variations] is cool because it is fast and practical." JKU-15-05
- Studio and live usage:
  "I would say it would be interesting, in this form, for a studio [. . .]. But it would be very inefficient in a live setting." RBMA-15-04

Most feature requests were uttered in the context of live environments. The most demanded features requested were, among other things: (i) a preview function, visually or audible—mentioned six times, (ii) a way to store or bookmark patterns—mentioned four times, and (iii) an option to make patterns switch only on the downbeat during playback—mentioned twice.

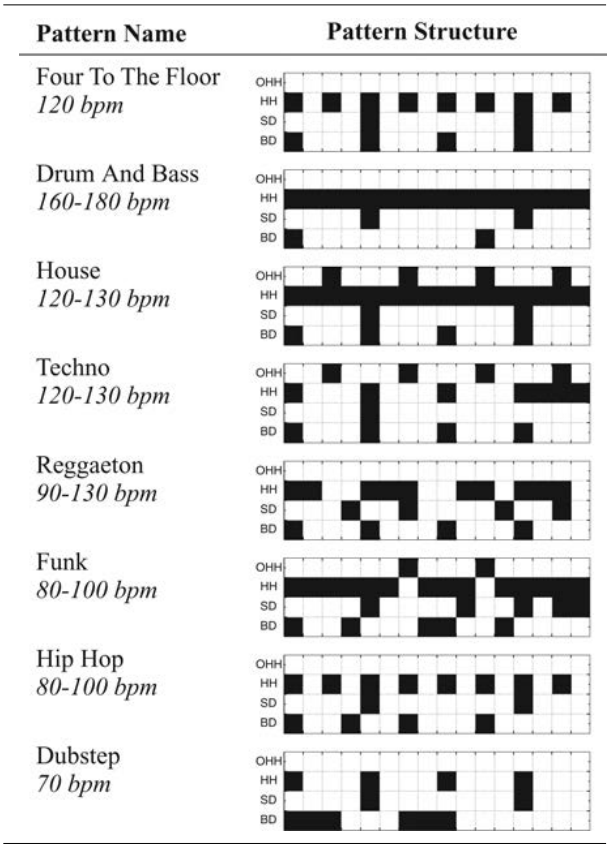
5 EVALUATION—WEB SURVEY

To substantiate the findings from the expert interviews and evaluate properties of the generated rhythm patterns quantitatively, we additionally conducted a web-based survey. Based on the feedback gathered so far, we were specifically interested in the algorithms' generated variations in terms of rhythm preservation, difference in detail, interestingness, suitability as substitution, and suitability as fill (see below). We define four research questions (RQ) covering these properties. These RQs are additionally used to structure the evaluation and results (cf., Sec. 5.2):

- RQ I: Are there significant differences between the individual algorithms for each property?
- RQ II: Do the variations capture the basic rhythm of the seed patterns?
- RQ III: Is the sorting within the list provided by the algorithm reasonable?
- RQ IV: Are the investigated properties independent or are these aspects correlated?

These properties are evaluated against a baseline consisting of pattern variations created by a human expert.

Table 3. List of seed patterns of the web survey.



Note: OHH stands for open hi-hat, HH for hi-hat, SD for snare drum, and BD for bass drum. Beneath the pattern name a typical tempo range for the pattern is provided. As tempo for the audio renderings, the mean of the range was chosen.

5.1 Method

Eight one-bar rhythm patterns (see Table 3), which represent distinctive characteristic styles found in electronic and urban music, were selected as basic seed patterns. For each basic pattern/algorithm combination, three variations were generated. Additionally, posing as a fourth "algorithm" and providing a baseline, a human, i.e., a musician familiar with different EDM styles, created three variations for each seed pattern (referred to as *EXP* in the following). The three variations were taken from the variation lists generated by the algorithms in an automated process. They were selected at the following distances to the seed within the list: –3 (sparse variation; in the following referred to as *sp1*), +3 (dense variation; *den1*), and +6 (*den2*). The mnemonics for the variations can also be found in Table 1. This results in a total number of 12 variations (four algorithms, each three variations) for one seed pattern. The maximum total number of patterns to evaluate per survey participant was, therefore, 96 patterns.

In the survey, we asked five questions to investigate the research questions defined earlier:

- How well does this pattern capture the basic rhythm of the original pattern? (Referred to as *rhythm* in the following)

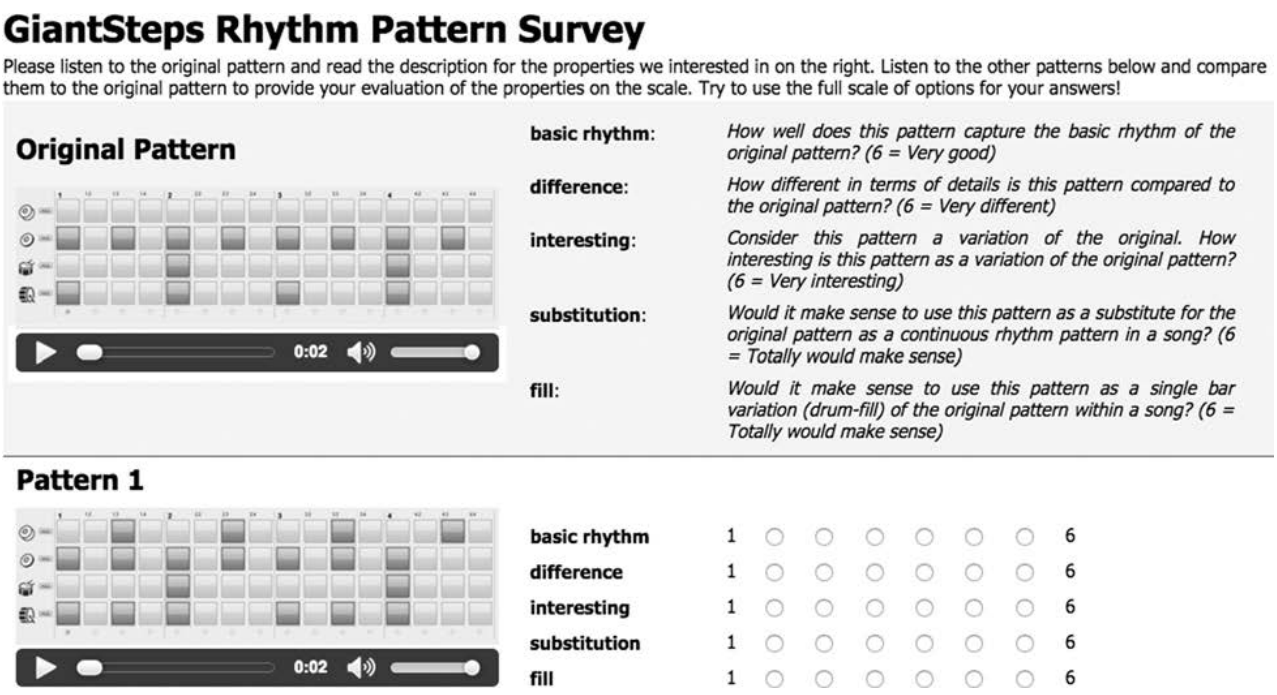


Fig. 5. Evaluation page of the web survey. On the left side the original pattern, and the first variation can be seen. In the top right, the single questions are explained. In the lower right the six-point Likert scales for the five questions are shown.

- How different in terms of details is this pattern compared to the original pattern? (*difference*)
- Consider this pattern a variation of the original. How interesting is this pattern as a variation of the original pattern? (*interesting*)
- Would it make sense to use this pattern as a substitute for the original pattern as a continuous rhythm pattern in a song? (*substitution*)
- Would it make sense to use this pattern as a single bar variation (drum-fill) of the original pattern within a song? (*fill*)

The answers to these questions were collected as the score on a six-point Likert scale (no neutral answer, thus "forced choice").

The survey was conducted by means of a web application. On the first page general information such as age, gender, and questions about musical activity were collected. Subsequently, every seed pattern was represented on a single page where answers to the five questions for the twelve variations had to be provided. Visual aids in the form of images of the rhythm pattern in the step sequencer grid, as well as a rendered audio version of the rhythm patterns using genre specific drum kits were provided. Fig. 5 shows a screenshot of a evaluation page of the web survey. The sequence of the seed patterns (= survey pages) as well as the sequence of the variations was randomized per user to avoid bias caused by a certain order of the variations. It was not mandatory that all seed patterns were processed by every participant. The progress was stored after each page and it was possible to continue the survey at another point in time.

5.2 Survey Results

The web survey was online from October 2015 until January 2016 and was distributed to local contacts and advertised on the Music-IR and the SMC mailing lists. In total, 43 users participated for which 1,536 entries were recorded. Every entry consists of scores for the five survey questions (*rhythm*, *difference*, *interesting*, *substitution*, and *fill*) for one of the three variations (*sp1*, *den1*, and *den2*) of a certain algorithm (one of *DB, RBM, GEN*, or *EXP*) of a specific seed pattern. This results in 384 data points per algorithm and 512 data points per variation. Fig. 6 shows the distribution of data points among the single seed patterns.

The resulting data set has a strong bias towards male participants: Only 2 out of 43 participants (4.7%) are female. Over 90% (39 out of 43) of the participants are able to read sheet music and almost half of them (20 out of 43) play some kind of percussive instrument.

We conducted one-way ANOVA analysis with consecutive Post-Hoc tests to analyze if the mean scores of the answers to the individual survey questions have significant differences, i.e., we test against the null hypothesis that there is no difference in the mean scores of the different algorithms (including the expert user *EXP*). First, the single distributions were tested for homogeneity of variances using Levene's test. In case of homogeneous variances an ANOVA with subsequent Ryan-Einot-Gabriel-Welsch Range Post-Hoc test was used. In case of inhomogeneous variances, additionally a Welch test with subsequent Games-Howell Post-Hoc test was used ($p = 0.05$ for all tests). This setup is used to find answers to the four research questions (RQ) defined earlier:

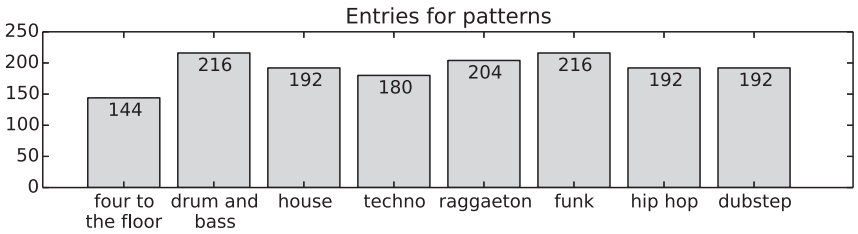


Fig. 6. Distribution of data points among the single seed patterns used in the web-based survey.

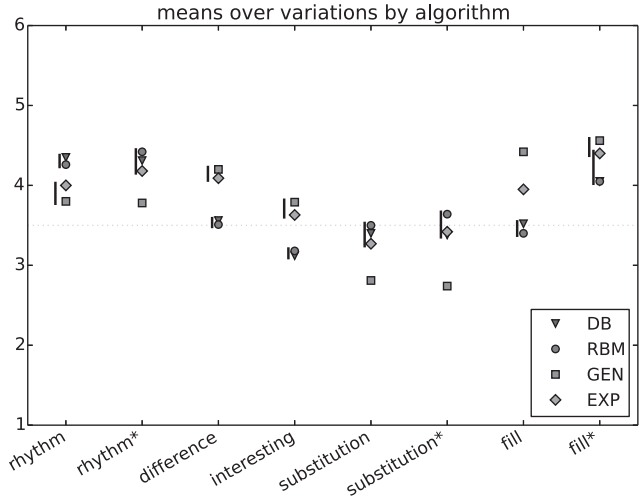


Fig. 7. Mean scores of each algorithm for the aspects evaluated. The black bars indicate homogeneous subsets according to significance analysis. The series with asterisk use only a subset of the full data set: *rhythm** and *substitution** are calculated using only variations *sp1* and *den1*, *fill** was calculated using only variation *den2*.

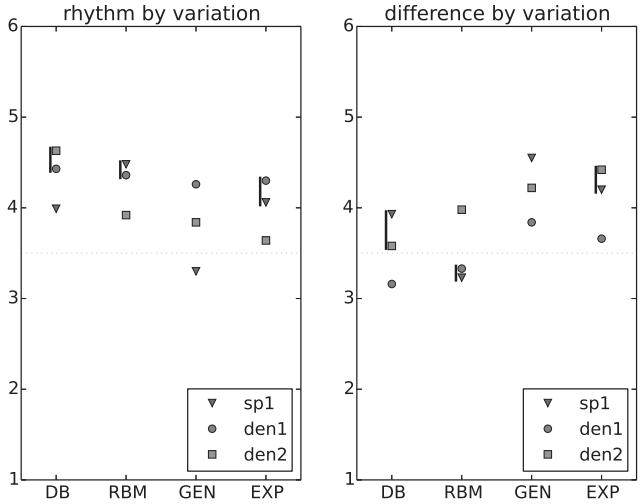


Fig. 8 Mean scores for rhythm (left) and difference (right) for each algorithm split by variation. The black bars indicate homogeneous subsets according to significance analysis.

RQ I: Are there significant differences between the individual algorithms for each property? Fig. 7 shows the evaluation results comparing the means of the individual algorithms for all survey questions. The baseline is represented by the scores for the expert patterns (*EXP*) that are visualized as diamonds in Fig. 7.

For this evaluation, apart from the full data set, two data subsets were additionally used: In the case of *rhythm** and *substitution** all entries with *den2* as variation were excluded. This was done to check if scores for *rhythm* and *substitution* increase if only the similar variations are considered. In the case of *fill** only entries with the more distant variation *den2* were used. This set was used to gain insight if scores for *fill* increase for dense patterns which are more different.

For *rhythm*, all algorithms perform equally well or better than the baseline. Significantly worse than the baseline are: *GEN* in the case of *rhythm**, *DB* and *RBM* in both cases of *difference* and *interesting*, *GEN* in both cases of *substitution* and *substitution**, and *DB* and *RBM* in the case of *fill*. In all other cases the algorithms perform equally well or better than the baseline patterns created by the expert.

The algorithms are able to reproduce the basic rhythm pattern of the seed patterns as good as the expert. While *RBM* and *DB* fail to produce as interesting and different patterns as the expert, *GEN* does not generate substitutes as well as the expert. Since *fill** is more relevant in a real-live scenario (in the interviews users tended to browse to the far right to look for fills), all algorithms can be considered to produce fills equally well as the expert, while *GEN* still performs significantly better than *DB* and *RBM*.

RQ II: Do the variations capture the basic rhythm of the seed patterns? The analysis to *RQ I* shows that for *rhythm*, all algorithms perform equally well or better than the human expert when calculating means over the full data set. For this question the data is split into three subsets for the variations *sp1*, *den1*, and *den2*. This is done in order to check if *rhythm* stays the same regardless of the degree of variation. In the left plot of Fig. 8 it can be observed that *rhythm* does not stay the same for the single variations, not even for *EXP* which was the baseline. In the case of *RBM* and *EXP* only the pattern *den2* scores significantly worse. *DB* performs equally well for *den1* and *den2* but significantly worse for *sp1*. Also *GEN* scores worse for *den2* and even more so for *sp1*.

We can summarize that for *den2* the *rhythm* scores are generally worse than for *den1*. This leads to the assumption that it is difficult for the algorithms (but also for the expert) to reproduce the basic rhythm for more different patterns. The only exception to this is the *DB* approach, which can be explained by the fact that *DB* never changes the basic rhythm pattern when producing patterns with more notes,

as explained in Sec. 3.3. On the other hand *DB* seems to fail to provide sparse patterns with the same basic rhythmic structure. This is probably caused by the fact that the database consists of a limited number of patterns, therefore it is hard to find sparse patterns with the same basic rhythm. *GEN* fails to provide good patterns not only for *den2* but also for *sp1*, which was also observable in the interviews:

> "To the left it [GEN] just goes crazy." JKU-15-01
>
> "If I turn to the right, there were many things which were OK, but on the left side not so much. [. . .] [GEN] went completely crazy on the left side." JKU-15-03

While the results for *RBM* are comparable to the expert baseline (*EXP*), *GEN* and *DB* seem to have problems reproducing sparse patterns that capture the basic rhythm.

RQ III: Is the sorting within the list provided by the algorithm reasonable? To evaluate this question two assumptions regarding the scores for *difference* are tested. The first assumption is, that variations *sp1* and *den1* score equally since they were taken at the same distance to the seed pattern. Second, for variation *den2* the score should raise since the distance was twice as large compared to *den1*. The right plot in Fig. 8 shows the evaluation results for *RQ III*. Only for *RBM* both assumptions are true. The second assumption is true for all algorithms, only the assumption that *sp1* and *den1* score equally has to be rejected for *DB, GEN*, and *EXP*. *sp1* is, in fact, always rated to be more different than *den1*. Since this is even the case for the baseline (*EXP*) one could assume that sparse patterns are generally perceived more different than dense patterns with the same distance.

Apart from the fact that sparse patterns generally seem to be rated more different than dense patterns, the sorting can be considered reasonable.

RQ IV: Are the investigated properties independent or are these aspects correlated? Fig. 9 visualizes the correlation between the answers to the survey questions. Only the pair *rhythm*/*fill* shows no significant correlation, which might imply that for fills it is not important if the basic rhythmic feel is preserved. *Substitution* and *fill* show a weak negative correlation that seems reasonable, since patterns rarely qualify for both categories. *Interesting* shows a slight positive correlation with *rhythm* and *difference*. This could imply that participants only find patterns interesting if they conserve the basic rhythm while introducing change. There is a strong positive correlation between *interesting* and *fill* that implies that participants tend to consider interesting patterns suitable as fills. *Difference* shows a weak positive correlation with *fill* that might imply that fills are supposed to be different from the basic rhythm. *Substitution* and *rhythm* also turn out to correlate strongly positively, which comes as no surprise since substitutes should, in general, be similar to the basic rhythm patterns. The same line of reasoning can be applied to the strong negative correlation between *substitution* and *difference*. Since *rhythm* also correlates strongly negatively with *difference*, the aforementioned correlation might be merely a transient

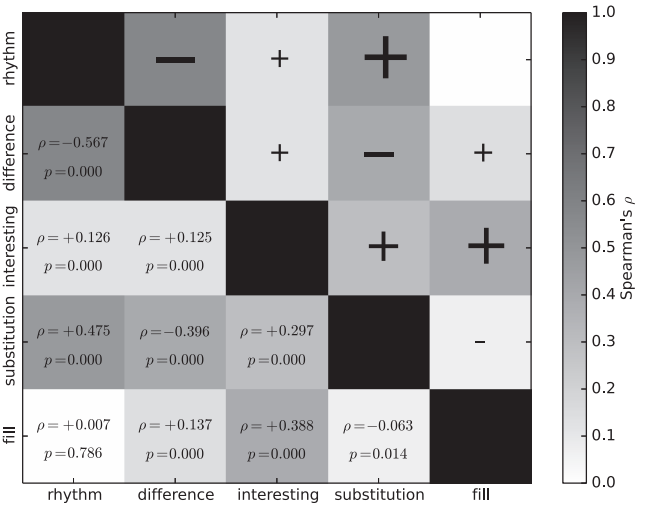


Fig. 9. Symmetric correlation matrix of Spearman's correlation values (ρ) and significance levels (p) for the answers to the survey questions. The upper right half visualizes the value (darker=higher) and direction (negative / positive) of the correlation. The lower left half contains the numeric values.

effect caused by the two very strong correlations of *rhythm* with *difference* (negative) and *substitution* (positive).

6 CONCLUSION

We presented three different algorithms to create variations of one bar drum rhythm patterns as well as the extension of an interface prototype. The aim of the prototype is to support EDM producers and performers to find suitable drum patterns. We used the prototype to test and evaluate the variation algorithms by means of two studies: A series of qualitative expert interviews and a quantitative web-based survey. The expert interviews show that the interaction concept of the prototype is something most participants can imagine working with. It also implies that the acceptance of such a tool in a studio environment would be high, while concerns were raised about precision and reliability when it comes to live performance scenarios. The patterns created by the database-based approach (*DB*) and the neural-network-based method (*RBM*) were mostly considered musical and in many cases perceived to reflect the basic rhythmic idea of the seed pattern. While the genetic algorithm (*GEN*) produced usable patterns in many cases, it was considered more suitable for fills and creative exploration. The web-based study, using an expert-created baseline, allows interesting insights that support the findings of the expert interviews: *GEN* produces patterns suitable for fills that have a tendency to be more different and interesting than the ones produced by *RBM* or *DB*, which in turn are more conservative and suitable as substitute patterns for basic rhythms. The findings of the two studies support each other and shed light on the properties of the compared methods as well as on the perception of rhythm variations of users in general.

Accompanying materials covering the raw survey data, images and audio renderings of the survey patterns,

the UI prototype, a short demo video, and the training configuration for the RBM are available at: https://github.com/GiantSteps/rhythm-pattern-variation-study

7 ACKNOWLEDGMENTS

This work is supported by the European Union's seventh Framework Programme FP7 / 2007-2013 for research, technological development, and demonstration under grant agreement no. 610591 (GiantSteps). We thank Stefan Lattner for providing help with the Lrn2 framework as well as giving hints and support for training the RBM. We gratefully acknowledge the support of NVIDIA Corporation with the donation of one Titan Black GPU used for this research.

8 REFERENCES

[1] E. Battenberg and D. Wessel, "Analyzing Drum Patterns Using Conditional Deep Belief Networks," in *Proc. 13th Intl. Society for Music Information Retrieval Conf.* (2012).

[2] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription," in *Proc. 29th Intl. Conf. on Machine Learning* (2012).

[3] C. Dittmar and M. Pfleiderer, "Automated Estimation of Ride Cymbal Swing Ratios in Jazz Recordings," in *Proc. 16th Intl. Society for Music Information Retrieval Conf.* (2015).

[4] H. Goh, N. Thome, and M. Cord, "Biasing Restricted Boltzmann Machines to Manipulate Latent Selectivity and Sparsity," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* (2010).

[5] M. Gruhne and C. Dittmar, "Improving Rhythmic Pattern Features Based on Logarithmic Preprocessing," presented at the *126th Convention of the Audio Engineering Society* (2009 May), convention paper 7817.

[6] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554 (2006), http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[7] A. Holzapfel and Y. Stylianou, "Scale Transform in Rhythmic Similarity of Music," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 1, pp. 176–185 (2011), http://dx.doi.org/10.1109/TASL.2010.2045782.

[8] J. H. Jensen, M. G. Christensen, and S. H. Jensen, "A Tempo-Insensitive Representation of Rhythmic Patterns," in *Proc. 17th European Signal Processing Conf.* (2009).

[9] M. A. Kaliakatsos-Papakostas, A. Floros, and M. N. Vrahatis, "evoDrummer: Deriving Rhythmic Patterns through Interactive Genetic Algorithms," in *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, vol. 7834 of *LNCS*, pp. 25–36 (Springer, 2013), http://dx.doi.org/10.1007/978-3-642-36955-1_3.

[10] C. Lewis, "Using the "Thinking-Aloud" Method in Cognitive Interface Design," Tech. Rep. RC-9265, IBM TJ Watson Research Center (1982).

[11] U. Marchand and G. Peeters, "Swing Ratio Estimation," in *Proc. 18th Intl. Conf. on Digital Audio Effects* (2015).

[12] C. Ó Nuanáin, P. Herrera, and S. Jordà, "Target-Based Rhythmic Pattern Generation and Variation with Genetic Algorithms," in *Proc. 12th Sound and Music Computing Conf.* (2015).

[13] J.-F. Paiement, Y. Grandvalet, S. Bengio, and D. Eck, "A Generative Model for Rhythms," in *NIPS Workshop on Brain, Music and Cognition* (2007).

[14] D. Shiffman, S. Fry, and Z. Marsh, *The Nature of Code* (Daniel Shiffman, 2012).

[15] G. Soros and C. Guedes, "Complexity Driven Recombination of MIDI Loops," in *Proc. 12th Intl. Society for Music Information Retrieval Conf.* (2011).

[16] G. Sioros, A. Holzapfel, and C. Guedes, "On Measuring Syncopation to Drive an Interactive Music System," in *Proc. 13th Intl. Society for Music Information Retrieval Conf.* (2012).

[17] P. Smolensky, "Information Processing in Dynamical Systems: Foundations of Harmony Theory," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pp. 194–281 (MIT Press, 1986).

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958 (2014).

[19] T. Tieleman and G. Hinton, "Using Fast Weights to Improve Persistent Contrastive Divergence," in *Pro. 26th Intl. Cons. on Machine Learning* (2009), http://dx.doi.org/10.1145/1553374.1553506.

[20] G. Toussaint, "A Comparison of Rhythmic Similarity Measures," in *Proc. 5th Intl. Cons. on Music Information Retrieval* (2004).

[21] R. Vogl and P. Knees, "An Intelligent Musical Rhythm Variation Interface," in *Companion Publication 21st Intl. Cons. on Intelligent User Interfaces* (2016), http://dx.doi.org/10.1145/2876456.2879471.

THE AUTHORS


Richard Vogl


Matthias Leimeister


Cárthach Ó Nuanáin


Sergi Jordà


Michael Hlatky


Peter Knees

Richard Vogl is a Ph.D. researcher at the Dept. of Computational Perception of the Johannes Kepler University Linz in Austria. His research is focused on rhythm and drum analysis and machine learning.

•

Matthias Leimeister is a member of the music information research team at Native Instruments, Berlin. His research interests include rhythm analysis, automatic transcription, and music recommendation.

•

Cárthach Ó Nuanáin is a Ph.D. researcher with the Music and Multimodal Interaction team with the Music Technology Group at Universitat Pompeu Fabra, Barcelona. His research interests includes electronic music, algorithmic composition, interface technology. and interaction.

•

Sergi Jordà holds a B.S. in fundamental physics and a Ph.D. in computer science and digital communication. He is a senior researcher with the Music Technology Group at Universitat Pompeu Fabra in Barcelona, where he directs the Music and Multimodal Interaction team, and an Associate Professor in the same university.

•

Michael Hlatky works as a product designer in the areas of interaction design, user interface design and experience design. Before joining Native Instruments, he worked as a sound designer and DSP developer for Audi AG and Bang&Olufsen a/s.

•

Peter Knees holds a doctorate degree in computer science and is currently assistant professor of the Dept. of Computational Perception of the Johannes Kepler University Linz in Austria. For over a decade, he has been an active member of the music information retrieval research community, branching out to the related areas of multimedia, text IR, recommender systems, and digital media arts.

12 TOUCH UI FOR DRUM PATTERN VARIATION

12.1 OVERVIEW

The contents for this chapter were published as the following paper:

Richard Vogl and Peter Knees. "An Intelligent Drum Machine for Electronic Dance Music Production and Performance". In: *Proceedings of the 17th International Conference on New Interfaces for Musical Expression (NIME)*. Copenhagen, Denmark, 2017.

In this work, a new UI based on a multi-touch interface is presented. The goal is to make manipulation of the step sequencer grid simpler, assuming that interaction with the touch UI is more direct and feels more natural. Additionally, knobs which can be placed on the touch screen and used to control virtual knobs, are tested. The variation engine, which is based on an RBM similar to the one used in Chapter 10, is further improved. Shortcomings and feature requests identified in the user study in Chapter 11 are fixed and incorporated, respectively.

For evaluation, again, a small scale qualitative user study is performed to evaluate *(i)* the new touch-based UI against the old prototype, and *(ii)* the two different RBM variation engines in a randomized A/B setup. The newly added UI features as well as the physical knob extensions to the touch UI are separately evaluated. Findings of the evaluation show that the touch interface and new variation engine was favored, while the physical knobs did not add any additional value for the majority of the participants. In general, participants found the touch interface more practical and usable than the mouse-based prototype—especially for live settings.

12.2 CONTRIBUTIONS OF AUTHORS

As main author I created almost all of the content for this work. I created the new touch-UI based prototype, updated the RBM based pattern variation system, designed the experiments, conducted interviews, evaluated results, and wrote the paper.

Peter Knees acted as supervisor and helped with writing the paper and experiment design.

An Intelligent Drum Machine for Electronic Dance Music Production and Performance

Richard Vogl,[1,2] Peter Knees[1]
[1] Institute of Software Technology & Interactive Systems, Vienna University of Technology, Austria
[2] Department of Computational Perception, Johannes Kepler University, Linz, Austria
richard.vogl@{tuwien.ac.at, jku.at}, peter.knees@tuwien.ac.at

ABSTRACT

An important part of electronic dance music (EDM) is the so-called beat. It is defined by the drum track of the piece and is a style defining element. While producing EDM, creating the drum track tends to be delicate, yet labor intensive work. In this work we present a touch-interface-based prototype with the goal to simplify this task. The prototype aims at supporting musicians to create rhythmic patterns in the context of EDM production and live performances. Starting with a seed pattern which is provided by the user, a list of variations with varying degree of deviation from the seed pattern is generated. The interface provides simple ways to enter, edit, visualize and browse through the patterns. Variations are generated by means of an artificial neural network which is trained on a database of drum rhythm patterns extracted from a commercial drum loop library. To evaluate the user interface as well as the quality of the generated patterns a user study with experts in EDM production was conducted. It was found that participants responded positively to the user interface and the quality of the generated patterns. Furthermore, the experts consider the prototype helpful for both studio production situations and live performances.

Author Keywords

Rhythm pattern generation; restricted Boltzmann machines; machine learning; neural networks; generative stochastic models.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: Graphical user interfaces, Input devices and strategies

1. INTRODUCTION

Electronic dance music (EDM) covers a wide range of genres with common production techniques, heavily utilizing synthesizers, sampling, digital effects, and sequencer software or digital audio workstations (DAWs). While these tools are nowadays also used in rock, pop, and other music production processes, they are more prominently featured in and are the foundation of EDM.

An important stylistic property of most EDM genres is the utilization of style-specific repetitive rhythmic patterns, the so-called beat. For shaping and defining the beat, the drum track of the piece and its rhythmic interaction with other instruments are essential. Creating the drum track of an EDM arrangement is, therefore, of high importance and can be time consuming. In this paper we present an intelligent software prototype — implemented as touch interface on a tablet computer — aiming at helping musicians to accomplish this task. More precisely, the developed prototype supports the musician or producer of an EDM track in adding variation to a drum track by intelligently providing creative input for drum pattern creation.

In the prototype's current implementation, we use a step sequencer representation for drum patterns with four drum instruments (kick, snare, hi-hat, open hi-hat) and 16 steps at which these instruments can be either on or off. As artificial intelligence engine, a generative stochastic neural network, concretely a restricted Boltzmann machine trained on EDM drum patterns, is implemented. It is used to generate stylistically suited variations of a given drum pattern. In this context, using loops from drum libraries is usually undesired because it makes the results predictable, boring, and available to everyone (regardless of skill), putting artistic identity in jeopardy. Ultimately, the prototype aims at supporting musicians to create original and interesting rhythmic patterns in the context of EDM production and live performances. To assess the suitability of the presented approach in these tasks, we performed a qualitative user study with expert users in electronic music production. We discuss the outcomes of this study after reviewing related work and detailing the algorithmic and technical implementation.

2. RELATED WORK

There are only a few commercial products for automated rhythmic pattern variation and creation. With Groove Agent,[1] Steinberg provides a drum plugin covering a wide variety of drum kits and loops. It also features a mode which allows variation of the complexity of patterns on the fly. Apple's DAW, Logic Pro,[2] features an automatic drummer plugin which allows the user to select a certain drum kit sound and music style. The patterns played can be changed by controlling complexity and loudness in a two-dimensional system using an x/y pad. These tools aim at amateur and semi-professional production and recording of alternative and rock tracks, and find therefore little use in EDM productions.

In the work of Kaliakatsos-Papakostas et al. [8], a method for automatic drum rhythm generation based on genetic algorithms is introduced. The method creates variations of a rhythm pattern and allows the user to change parameters

NIME'17, May 15-19, 2017, Aalborg University Copenhagen, Denmark.

[1] http://www.steinberg.net/en/products/vst/groove_agent
[2] http://www.apple.com/logic-pro

such as level of variation between the original and the generated patterns. In the work of Ó Nuanáin et al. [9] a similar method for rhythm pattern variation is presented using genetic algorithms in combination with different, more simple, fitness functions. This approach was one of the methods evaluated in [19] where it has been shown that it is more difficult to generate patterns with consistent style using genetic algorithms.

When working with lists of rhythm patterns and sorting or ranking is required, similarity measures for such patterns are needed. In the work of Toussaint [15] several similarity measures for rhythmic patterns are discussed: The Hamming distance, edit distance, Euclidean distance of inter-onset-interval vectors, and the interval-ratio-distance are compared by building phylogenetic trees based on the computed distance matrices. Holzapfel and Stylianou [6] use audio signals as input and present a tempo invariant rhythmic similarity measure utilizing the scale transform. Other methods which can be applied to audio signals are presented by Jensen et al. [7] as well as Gruhne and Dittmar [4]. Both works obtain tempo invariant rhythmic features by applying logarithmic autocorrelation to different onset density functions. Since this work focuses on the use of symbolic representations of rhythm patterns, similarity measures based on audio signals are unsuitable. Therefore, primarily the methods compared in [15] were relevant.

A group of widely used generative models are restricted Boltzmann machines (RBMs) [11, 5]. Battenberg et al. [1] use a variation, the conditional RBM, to analyze drum patterns and classify their meter. They mention the capability of the learned model to generate drum patterns similar to the training data given a seed pattern. Boulanger-Lewandowski et al. [2] use an extension of an RBM with recurrent connections to model and generate polyphonic music. In the work of Vogl and Knees [18] a drum pattern variation method based on an RBM is demonstrated. In [19] different pattern variation methods for drum rhythm generation are evaluated using two user studies. It shows that RBMs are capable of reasonably generating drum patterns.

In the current work, a system which is able to create meaningful variations of a seed pattern is presented. Weaknesses identified in the interface in [19] are considered and a touch-interface-based prototype is introduced. The RBM-based variation method is further improved and the system is evaluated using a qualitative user study involving ten experts in electronic music production.

3. INTELLIGENT PATTERN VARIATION METHOD

The centerpiece of the prototype is the artificial intelligence driven pattern variation engine. Its task is to create rhythm patterns as variations of a seed pattern utilizing sampling of an RBM. For the training of the RBM a data set of EDM and urban music drum rhythm patterns had to be created. RBMs are two layered neural networks which can be used to generate patters. Fig. 1 shows the basic structure and components of a simple RBM. This pattern generation method was chosen for several reasons. Training and sampling of RBMs is well researched and RBMs have been shown to be applicable for pattern generation in general. Furthermore, sampling of RBMs is computationally efficient and can be performed sufficiently fast also on low-end portable devices to ensure reasonable response times for user interaction. When sampling from RBMs, a seed pattern, which determines the characteristics of the generated patterns, can be provided.

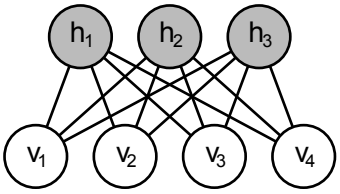


Figure 1: Simplified visualization of the structure of RBMs. The lower layer in white represents the visible nodes (v_n) while the upper layer in gray represents the hidden nodes (h_m). Every connection between two nodes has an assigned weight (w_{mn}) and every node has its own bias value (bv_n and bh_m for visible and hidden nodes respectively – not shown in the diagram).

3.1 Training Data Set

For training, a data set of 2,752 unique one-bar drum patterns containing bass drum, snare drum, and hi-hat onsets was used. The patterns were taken from the sample drum loop library of Native Instrument's *Maschine*[3]. The exported patterns were split into one bar segments, quantized to 16^{th} notes, and converted into a 64 bit (for the 4 by 16 rhythm patterns) binary vector format. Finally, duplicate patterns, as well as patterns which did not meet musical constraints were removed. Only patterns with two to six bass drum notes per bar, one to five snare drum notes, and at least two hi-hat notes were kept. This was done to exclude very sparse breaks as well as too dense fills from the data set. The *Maschine* library contains drum patterns for EDM and other urban music like Hip Hop and RnB. Since the main focus of this work is EDM, this library was well suited.

3.2 Network Training

The used RBM consists of 64 visible nodes, which represent the 16 by 4 drum patterns (16 steps per bar for four instruments), and 500 nodes in the hidden layer. The training for the RBM was performed using the *lrn2* framework of the *lrn2cre8* project[4]. As training algorithm, persistent contrastive divergence (PCD) introduced by Tieleman et al. [14] was used. This method represents an improved version of the well-known contrastive divergence (CD) method introduced by Hinton et al. [5] in 2006. Additionally, latent selectivity and sparsity as described in the work of Goh et al. [3] as well as Drop-out [13] was used to reduce overfitting.

The output of a training run are the weights and biases of the neural network. These are used by the variation algorithm to create the rhythm patterns in the context of the provided seed pattern.

3.3 Pattern Generation

While training of the RBM is similar to [19], the pattern generation was adapted to overcome shortcomings of the method identified in the evaluation of [19].

To use the trained RBM for pattern generation, the seed pattern is first converted to the 64 bit vector format by concatenating the 16 steps of the sequencer grid of each instrument (cf. fig. 2). This vector is then used as input for the visible layer of the RBM. In contrast to [19] no clamping is used and variations are generated for all instruments at

[3] http://www.native-instruments.com/en/products/maschine/production-systems/maschine-studio/
[4] http://lrn2cre8.eu/

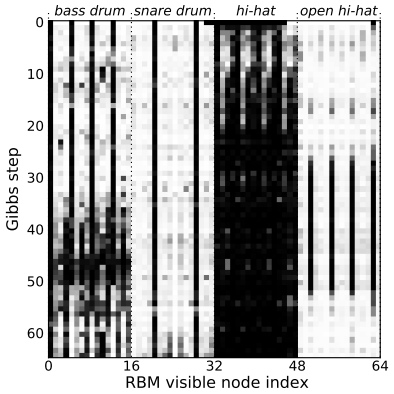


Figure 2: Values of visible layer nodes of the RBM while creating pattern variations. The x-axis represents the index of the visible node. On the y-axis, the number of Gibbs step is indicated starting at the top with the original input pattern and progressing downwards. Black pixels represent 1 and white 0. Only the first 64 iterations of Gibbs sampling are shown.

once using Gibbs sampling.

A Gibbs sampling step consists of: *i.* Calculating the values of the hidden layer (h_m) by multiplying the input layer's values (i_n) with the corresponding weights (w_{nm}) plus bias of the hidden layer (bh_m):

$$h_m = bh_m + \sum_{n=0}^{N} i_n \cdot w_{nm} \tag{1}$$

where N is the total number of nodes in the visible layer. *ii.* Applying the logistic sigmoid function to map the values of the hidden layer into the interval $[0, 1]$ (hs_m) and subsequent binarization with random threshold (sampling):

$$hs_m = \frac{1 + tanh(\frac{h_m}{2})}{2} \tag{2}$$

$$hb_m = \begin{cases} 1, & \text{for } hs_m \geq t \\ 0, & \text{else} \end{cases} \tag{3}$$

where t is a random threshold in the interval $[0, 1]$ and hb_m is the binarized value of h_m. *iii.* Calculating the updated values for the visible layer by multiplying the values of the hidden layer with the corresponding weights (w_{nm}) plus bias of the visible layer (bv_n):

$$i_n = bv_n + \sum_{m=0}^{M} hb_m \cdot w_{nm} \tag{4}$$

where M is the total number of nodes in the hidden layer. To visualize this, fig. 2 shows the values of the nodes in the visible layer during several steps of Gibbs sampling.

In contrast to the pattern variation method used in [19], in this work for every seed pattern 64 variations are generated and only the most suitable (i.e., similar) 32 patterns are presented to the user. This is done to achieve a greater possibility of obtaining an equal number of more dense and more sparse variations. Furthermore, patterns which are very dissimilar to the seed pattern can be discarded.

To arrange the generated patterns in a meaningful way, the patterns are sorted in two steps. First, the patterns are divided into two lists according to the number of active notes in them. One list contains patterns with fewer active notes than the seed pattern (*sparse list*), while the other one contains only pattern with more or equal number of active notes (*dense list*). Second, these lists are sorted according to their similarity to the seed pattern. To build the final list used for the variation dial, the 16 most similar patterns from the *sparse list* are arranged ascending, followed by the seed pattern, followed by the 16 most similar patterns from the *dense list* arranged descending. It should be noted that in rare cases, given a very sparse or dense seed pattern, it may occur that the 64 generated variations do not contain 16 more dense or more sparse patterns. In that case more patterns from the other sub list are used to obtain a final list size of 32 patterns.

3.4 Distance Measure

To sort the pattern lists, a similarity measure for rhythm patterns is required. Although Toussaint [15] observes that the Hamming distance is only moderately suited as a distance measure for rhythmic patterns, it is widely used in the literature (see [9, 10, 19]). Since the requirements in this work are similar to the ones in [19], likewise a modified Hamming distance is implemented. To calculate the standard Hamming distance, simply the differences between the binary vectors of the two rhythm patterns are counted. I.e. for every note in the 16-by-4 grid a check is performed if its state is the same (on/off) in both patterns. If it is not, the distance is increased by one. The modified Hamming distance used in this work weights the individual instruments differently: Differences in the bass drum track contribute with four to the distance, snare drum notes with eight, closed hi-hat notes with one, and open hi-hat notes with four. This is done to take the importance of the drum instruments regarding the perceived differences between rhythm patterns into account. While additional or missing closed hi-hat notes only change the overall rhythmic feel of a pattern very little, additional snare drum or bass drum notes often change the style of the pattern completely. The values for the weighting were determined experimentally and using the results of the web survey in [19].

4. USER INTERFACE

In fig. 3, a screenshot of the prototype's UI is shown. For input and visualization of drum patterns in the UI, the well established *step sequencer* concept is employed. A drum step sequencer, as for example the famous hardware sequencer Roland TR-808, allows the user to activate certain notes for different drum instruments by pressing push buttons in a fixed grid. These patterns are then played back in an endless loop. This concept was used since it is one of the prevalent drum machine interfaces for EDM production. Drum patterns used in this work consist of one bar with notes for bass drum, snare drum, open and closed hi-hat. The time grid resolution is quantized to 16^{th} notes, which is simplification commonly used in step sequencers. The controls for pattern variation are implemented as two buttons (set/reset) and a central dial on which the variations are placed ordered by sparsity and distance to the seed pattern. Variations are generated by pressing the set button. The seed pattern is expected not to be highly complex, nor too simple, to give the variation algorithm enough freedom to find variations in both directions. During exploration of the patterns, the seed pattern can always be quickly accessed by pressing the "reset" button.

Next to the pattern variation controls a start/pause playback button and knobs to control the tempo in beats per minute (BPM) and the ratio of swing for 8^{th} notes can be found. The selected tempo and swing-ratio do not affect the pattern variation but rather provide the possibility to

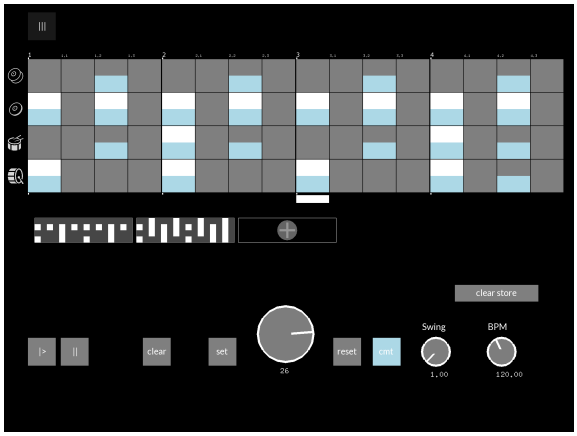


Figure 3: Screenshot of the UI. It consists of a 4 by 16 button array which poses as the visualization and input of the step sequencer. Beneath it are pattern storage, controls for playback, the pattern variation controls, and controls for tempo and swing. The blue blocks in the lower half of a step sequencer block visualize the pattern under preview. It can be activated by pressing the blue commit button in the pattern variation control section of the UI.

tune the rendering of the entered beat to match the musical style desired by the user.

The utilization of a touch-base interface allows the user to enter and change patterns in the step sequencer grid more easily and without the use of a mouse or track pad. This is expected to improve the acceptance of such a tool in live environments where simplicity and robustness of the input methods are obligatory. On the other hand this means that a widespread and accepted input method, namely physical knobs of MIDI controllers are not necessary, since all knobs can be controlled via the touch surface. While it is possible to map the input knobs and buttons to external MIDI controllers, in this work, another input method, namely knobs which can be used on the touch surface were evaluated. Fig. 4 shows the usage of such knobs on an iPad running the software prototype. The concept of physical knobs which can be used on a touch interface is not new (see e.g. ROTOR[5] or Tuna Knobs[6]). Using such knobs allows the combination of two powerful input methods with individual strengths. While the multi-touch interface provides an easy way of manipulating rhythm patterns, physical knobs might provide a greater degree of precision than fingers on a touch interface.

Improvements proposed in [19] cover: *i.* a pattern storage, *ii.* an optional pattern preview system, and *iii.* the possibility to activate new patterns only at the start of a new bar.

The pattern storage which is located beneath the step sequencer grid can be used to store patterns by tapping the plus sign. Stored patterns are displayed as a list of small thumbnails of the sequencer grid and can be brought back to the step sequencer by tapping the patterns. The store can be cleared using the "clear store" button beneath the store grid.

The pattern preview function is integrated into the step sequencer. When the preview is switched on, the active pattern is depicted using white blocks in the sequencer. If the variation dial is used to browse through the variations,

[5] http://reactable.com/rotor/

[6] http://www.tunadjgear.com/

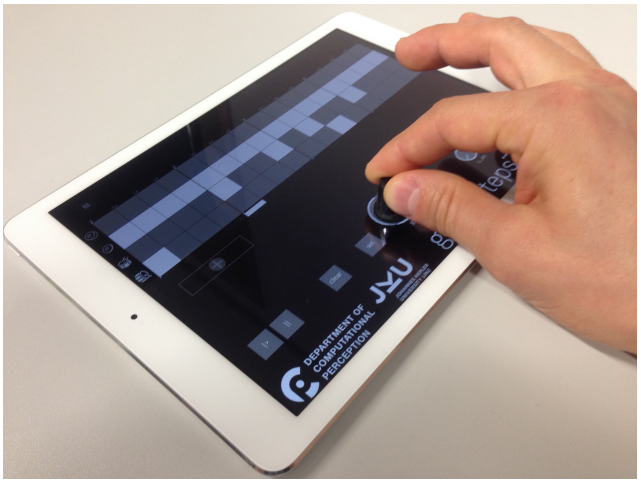


Figure 4: A physical knob used to control the variation dial on the touch surface. While the used knob is an improvised prototype, commercial products already exist for this purpose.

they are not immediately used for playback but rather visualized as blue blocks in the lower half of the step sequencer grid. See fig. 3 which shows a screenshot of a pattern being visualized in preview mode while another pattern is active. To use the currently previewed pattern, the commit button ("cmt") has to be pressed.

The prototype can be synchronized with other instruments using Ableton's Link technology.[7] The output of the prototype is sent via MIDI to be further processed in a DAW or synthesized using an external MIDI instrument. Alternatively, an internal drum synthesizer can be used to render audio playback of the patterns.

5. EVALUATION

To evaluate the interaction with the prototype as well as the quality of the generated patterns compared to the prototype presented in [18], a qualitative user study was conducted. To this end, experts were interviewed using a questionnaire as guideline. The experts were required to have experience in *i.* using DAWs or similar music production software, *ii.* producing or performing electronic music live, and *iii.* using drum step sequencers and/or drum roll editors. The software prototype used in [19] was made available by the authors as accompanying materials of the article[8].

During a session, participants were introduced to the two prototypes and the aim and functionality was explained. They were asked to input rhythm patterns they usually work with and let the prototype generate a list of variations for these patterns. After browsing through the patterns and exploring the features of the systems, users were interviewed about their experience with the prototypes and their preferences. Specifically, they were asked to rate the following properties on five point Likert scales: *i.* The usability of the prototypes, *ii.* the application of such a tool in a live performance or *iii.* in a studio production environment, *iv.* preferred input method (MIDI controller and mouse, touch interface, or touch interface combined with physical knobs), and *v.* usefulness of the additional features in the touch-interface-based prototype.

Additionally to the UI evaluation, the differences between the pattern variation algorithms were also tested. To this

[7] https://www.ableton.com/en/link/

[8] https://github.com/GiantSteps/rhythm-pattern-variation-study

	Vogl et al. [19]	Present work
Consistency	3.7	**3.9**
Musicality	4.2	**4.4**
Difference	3.2	**2.9**
Difference RMSE	0.6	**0.3**
Interestingness	3.8	**4.0**
Substitute	3.8	**4.4**
Fill	**4.0**	3.6

Table 1: Mean values of participant rating for the two algorithms. For *difference* additionally the RMSE to the neutral value (3) is provided.

end, both algorithms were implemented in the touch-interface-based prototype. Participants were asked to browse through variation lists generated by both algorithms for seed patterns of their choice. After that, they were asked to rate both algorithms in the following categories on five point Likert scales: *i.* Consistency of the variations with the seed pattern, *ii.* musicality and meaningfulness of created patterns, *iii.* difference of created patterns to the seed pattern, *iv.* creativity and interestingness of created patterns, *v.* suitability of created patterns for a continuous beat, and *vi.* suitability of patterns for fills or breaks. These categories correspond roughly to the ones used in the web survey in [19]. The order in which the algorithms were tested was randomized to avoid experimenter bias. The Likert scale for the *difference* rating ranged from "too similar" to "too different", therefore the optimal answer "just right" was placed in the middle. This is also reflected in the evaluation section: For the *difference* ratings additionally root mean square errors (RMSE) towards the optimal value (3) are provided.

A more general discussion including topics like the prototype's concept and applicability, positive and negative experiences during the experiment, UI details, missing features, and the participant's usual workflow and preferred tools concluded the sessions.

6. RESULTS AND DISCUSSION

The interviews were conducted during the period between June and October of 2016. In total ten experts participated in the survey. Their mean age is 31.1, the gender distribution is 9 male and one self-identified neither as male nor female. Seven participants had formal musical education whereas three are autodidacts. Eight participants actively play an instrument and all use DAWs on a regular basis, are familiar with step sequencers and have several years of experience in electronic music production.

Tab. 1 shows the mean values for the participants' ratings of the comparison between the variation algorithm used in [19] (top) and the one presented in this work (bottom). Since the number of participants of ten is too low for meaningful statistical significance analysis the numbers merely indicate tendencies. Nevertheless, a Wilcoxon signed ranks test was used to test for significance in the rating differences in aspects of the two UIs, and the two variation algorithms. As expected, the observed improvements are not significant (alpha=.05) due to small sample size, except for assessment of usability, where the touch based UI was considered better usable. The ratings in combination with in depth discussions with the participants show a clear preference towards the UI and variation generation algorithm presented in this work. The only exception being the suitability of the generated patterns for fills. This can be explained by the fact that outlier patterns are discarded by the variation algorithm and therefore it produces patterns more similar to the seed pattern, which may be less suitable for fills. The exact definition of fills depends on the music style, e.g. the "amen-break" is originally a fill but forms the basis of the continuous rhythms in drum-and-bass and breakbeat music. Generally, patterns which greatly differ from the basic rhythms, but somehow fit the given style can be considered as fills and breaks. For the tasks at hand, we relied on the individual understanding and definition of the expert users.

Tendencies regarding the ratings for the UI are similarly consistent. Ratings for usability are significantly higher for the touch interface (mean: 4.7/4.3). The difference is even greater for the suitability in live scenarios (mean: 4.0/3.5). While 50% of the participants uttered concerns about the practicality of using a mouse to enter rhythm patterns on stage, only two participants were concerned that the touch device is not suitable for a live performance. One participant's reservations regarding the touch interface did not concern the way of interaction but rather if the hardware (iPad) would survive the harsh conditions on stage and on the road (heat, mechanical strain, spilled drinks, etc.). The second one raised concerns regarding the touch interface's precision and reliability in a live environment. Regarding the applicability of the prototypes in a studio or production setting, the difference was smaller, but still in favor of the touch based prototype (mean: 4.7/4.6). The comment of one participant nicely summarizes the tenor of the users:

> "Using the touch interface is definitely faster and easier [...] compared to entering patterns with a mouse." Participant03

Regarding the preferences of the input method, a clear tendency towards the touch interface was observable: Six participants preferred the touch interface, three were undecided, and only one voted in favor of the physical controller and mouse system. Regarding the touch-compatible physical knob prototypes, seven participants preferred the touch-only approach, one was undecided, and two preferred using the physical knobs.

The additional features were generally received very positively. Only two participants were unsure if the feature to start new patterns only with a new bar was useful. All other participants were in favor for all three additional features.

In the discussions with the participants several key messages were identified. Three participants considered the arrangement of the patterns in the one-dimensional list of the variation wheel as being unclear or ambiguous:

> "It seems a bit random to me. I can browse through the list [...] but I cannot look for something specific." Participant04

While the idea of a simple one-dimensional variation dial introduced in [18] was well suited to conduct experiments regarding the quality of variation algorithms, it might be an over-simplification for user interaction. After all two different properties (sparseness and similarity) are projected into one dimension. Participants suggested to solve this by adding the option to change the sorting of the list or by using an x/y variation pad similar to the one used for the Drummer plugin of the Logic Pro DAW.

While the visual preview was a well received feature, two participants missed an acoustic preview or "pre-listen" mode. Finding suitable audio material for remixing by listening to it on separate headphones is a common technique used by DJs in live situations.

Almost all participants (8/10) mentioned that they use a drum roll editor within their DAW to produce drum rhythm patterns. One explicitly stated that he tries to avoid it:

> "I use the piano roll editor in Cubase if I have to, but it is a real pain." Participant04

7. CONCLUSION

In this work we presented a touch-interface-based prototype to assist musicians and producers in the context of EDM production with creating the drum track. This is accomplished by providing variations of a basic rhythmic pattern entered by the user. The variations are created utilizing Gibbs sampling of an RBM trained on appropriate rhythm patterns. The implemented method builds on established algorithms and improves them. Accessing and browsing through the patterns is accomplished using a simple interface built around a central dial.

A user study was conducted to evaluated both the pattern variation algorithm as well as the user interface of the prototype. The results of the study show that musicians and producers consider the interface intuitive. It is also shown that acceptance of the system in a live scenario is higher than for the compared prototype while acceptance in a production setting is still given. The pattern variation algorithm was considered to produce patterns more consistent with the seed pattern than the compared system, but only at the expense of the capability to create patterns suitable for fills and breaks.

The UI incorporates additional features requested by participants of a similar study. These features cover a preview for generated patterns, a pattern storage, and the ability to start patterns only at bar changes. These features were received positively by the participants. While the idea of using physical knobs on a touch interface was interesting for many participants, most participants preferred using a simple touch interface without additional knobs, especially in live settings.

Considering the feedback of the expert users, it can be concluded that such a system could find acceptance in the area of EDM production and performance. To achieve this, the prototype will still have to be improved in regard of the UI's visual design as well as the arrangement and browsing metaphor of the drum patterns.

To support more drum instruments prevalent in EDM production a larger and more diverse training data set for the variation method will be necessary. To obtain such a data set, automatic drum transcription methods (e.g., [17, 12, 16, 20]) could be utilized. Using such an approach would also allow to expand this method to be applicable on other music genres outside of EDM. However, to gain acceptance in the community of producers and musicians of other music genres an entirely different approach for pattern visualization and input might be required, since the used step sequencer representation is, at the moment, prominently tied to the field of EDM production. Nonetheless, from the results obtained in this study, we are confident that the use of generative models provides a valuable addition to the current paradigms of music production practice and helps in building intelligent and therefore more intuitive and effective interfaces.

8. ACKNOWLEDGMENTS

We thank Matthias Leimeister for extracting the MIDI and text mapping for the *Maschine* sample drum loop library. This work was supported by the European Union FP7 through the GiantSteps project (grant agreement no. 610591) and the Austrian FFG under the BRIDGE 1 project SmarterJam (858514).

9. REFERENCES

[1] E. Battenberg and D. Wessel. Analyzing drum patterns using conditional deep belief networks. In *Proc 13th ISMIR*, 2012.

[2] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proc 29th ICML*, 2012.

[3] H. Goh, N. Thome, and M. Cord. Biasing restricted boltzmann machines to manipulate latent selectivity and sparsity. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[4] M. Gruhne and C. Dittmar. Improving rhythmic pattern features based on logarithmic preprocessing. In *Proc 126th AES Convention*, 2009.

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006.

[6] A. Holzapfel and Y. Stylianou. Scale transform in rhythmic similarity of music. *IEEE TASLP*, 19(1):176–185, 2011.

[7] J. H. Jensen, M. G. Christensen, and S. H. Jensen. A tempo-insensitive representation of rhythmic patterns. In *Proc 17th EUSIPCO*, 2009.

[8] M. A. Kaliakatsos-Papakostas, A. Floros, and M. N. Vrahatis. evodrummer: Deriving rhythmic patterns through interactive genetic algorithms. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, *LNCS* vol 7834, 2013.

[9] C. Ó Nuanáin, P. Herrera, and S. Jordà. Target-based rhythmic pattern generation and variation with genetic algorithms. In *Proc 12th Sound and Music Computing Conference*, 2015.

[10] J.-F. Paiement, Y. Grandvalet, S. Bengio, and D. Eck. A generative model for rhythms. In *NIPS Workshop on Brain, Music and Cognition*, 2007.

[11] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pages 194–281. MIT Press, 1986.

[12] C. Southall, R. Stables, and J. Hockman. Automatic drum transcription using bidirectional recurrent neural networks. In *Proc 17th ISMIR*, 2016.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, June 2014.

[14] T. Tieleman and G. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proc 26th ICML*, 2009.

[15] G. Toussaint. A comparison of rhythmic similarity measures. In *Proc 5th ISMIR*, 2004.

[16] R. Vogl, M. Dorfer, and P. Knees. Recurrent neural networks for drum transcription. In *Proc 17th ISMIR*, 2016.

[17] R. Vogl, M. Dorfer, and P. Knees. Drum transcription from polyphonic music with recurrent neural networks. In *Proc 42nd ICASSP*, 2017.

[18] R. Vogl and P. Knees. An intelligent musical rhythm variation interface. In *IUI Companion*, 2016.

[19] R. Vogl, M. Leimeister, C. Ó. Nuanáin, S. Jordà, M. Hlatky, and P. Knees. An intelligent interface for drum pattern variation and comparative evaluation of algorithms. *JAES*, 64(7/8):503–513, 2016.

[20] C.-W. Wu and A. Lerch. Drum transcription using partially Fixed Non-Negative Matrix Factorization with Template Adaptation. In *Proc 16th ISMIR*, 2015.

13 DRUM PATTERN GENERATION WITH GANS

13.1 OVERVIEW

This chapter is an extended version of the following demonstration paper:

Hamid Eghbal-Zadeh et al. "A GAN based Drum Pattern Generation UI Prototype". In: *Late Breaking/Demos, 19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018.

In this chapter a novel GAN-based drum pattern generation method is introduced. Using the new model, the number of instruments is increased to eight and the number of time steps per bar is increased to 32 (32^{nd} notes). The used GAN architecture consists of a convolutional recurrent generator and discriminator. The recurrent layers operate on a sequence of musical bars, while the convolutions are responsible to model onsets within one bar. This enables this model to be rained on a sequence of bars, and also to generate an arbitrary number of bars as a consistent sequence of bars. In this context, it is important to use sequences of bars of drum patterns which are representative of drum tracks used in real music. Thus, as training data for this model, two different datasets containing drum tracks from real music are used. While the first set consists of symbolic drum patterns extracted from MIDI files, the second one comprises drum patterns extracted from real songs (audio) using a drum transcription system. The GUI used in previous user studies was updated to fit the needs of the new generative model: First, the step sequencer grid was adapted to be able to represent the 8-by-32 patterns. Second, the UI can now handle patterns consisting of several bars. Third, the variation controls were adapted, now featuring an additional x/y pad which controls the continuous latent variables used for conditioning the generative model.

Section 13.3 will additionally provide details on the technical implementation of the pattern generation system as unpublished original work.

A GAN BASED DRUM PATTERN GENERATION UI PROTOTYPE

Hamid Eghbal-zadeh*[1] **Richard Vogl***[1,2] **Gerhard Widmer**[1] **Peter Knees**[2]
* Equal contribution
[1] Institute of Computational Perception, JKU Linz, Austria
[2] Faculty of Informatics, TU Wien, Vienna, Austria
hamid.eghbal-zadeh@jku.at, richard.vogl@tuwien.ac.at

ABSTRACT

Tools which support artists with the generation of drum tracks for music productions have gained popularity in recent years. Especially when utilizing digital audio workstations, using e.g. predefined drum loops can speed up drafting a musical idea. However, regarding variation and originality of drum patterns, methods based on generative models would be more suited for this task. In this work we present a drum pattern generation prototype based on a step sequencer interface which allows to explore the suitability of generative adversarial networks for this task.

1. INTRODUCTION

Automatic music generation has been an active field of research for several years. While early attempts mostly use rule-based and probabilistic models, recent systems focus on machine learning and artificial intelligence based methods [6, 14]. These methods can be divided into two groups: *i)* methods focusing on generating audio signals and *ii)* methods that create symbolic music. Generative systems that focus on full music tracks may find application in the context of media arts or for automatic sound track generation e.g. for video games. In this work we focus on generating only drum tracks which is a relevant task in the context of digital music production. Generating a symbolic drum track can be a labor intensive task, while involving repetitive steps. To draft musical ideas quickly, it is often desirable to have a simple and fast method to create a basic drum track. Some digital audio workstations (DAWs) and drum sequencers thus provide predefined drum loops to enable such a workflow. Apple's Logic Pro X DAW features a so called *Drummer* plugin to create drum tracks, which allows to interactively vary loudness and complexity, besides other parameters. While for the early stages of writing music, these approaches help to speed up the process, musicians and producers often refrain from using these technologies for the end product. This is due to the fact that out-of-the-box patterns bear the danger of sounding unoriginal.

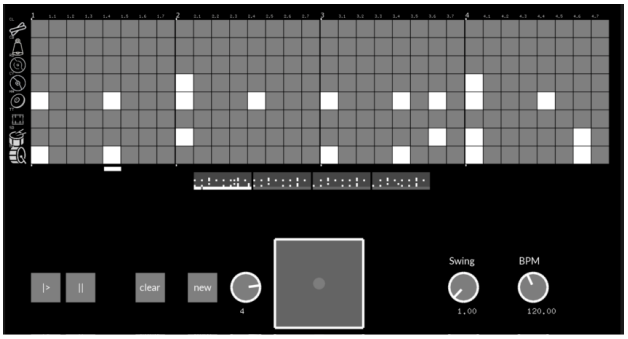


Figure 1. UI of the proposed prototype.

To incorporate more natural variation and thus overcome these downsides, generative approaches for drum tracks can be used. Early methods for symbolic drum track generation built on genetic algorithms (GAs) [5, 7, 8, 11]. Vogl et al. [12] use restricted Boltzmann machines (RBMs) to achieve this. Recent works show that generative adversarial networks (GANs [4]) can be used for music generation; both for symbolic music [3, 14] as well as audio [2, 10]. In this work, GANs are used to directly generate symbolic drum tracks, parameterizable and controllable in a similar fashion as the *Drummer* plugin in Logic Pro X.

2. METHOD

The drum pattern variation engine is implemented using a GAN. The network is trained on sets of 8-by-32 matrix representations of drum patterns. Each of these matrices represent a bar of music, while the eight rows represent different drum instruments and the 32 columns individual discrete time steps (32nd notes). For this work dynamics (changes in volume for each onset) are ignored as a simplification.

2.1 Network Structure and Training

The structure of the GAN is based on a convolutional recurrent design. Four deconvolution layers with 3x3 filters are used to generate a bar (8x32 matrix), while recurrent connections over bars are used to model the temporal evolution of patterns. This way, a varying number of bars can

be considered during training or inference.

As loss for GAN training a Wasserstein loss [1] in combination with additional conditioning on genre, and two features extracted from the one-bar drum patterns (complexity and loudness) are used.

2.2 Training Data

A MIDI dataset of popular songs published as part of [13] serves as training data for the GAN. First, drum tracks were extracted from the full MIDI files and then converted to the 8-by-32 matrix per bar representation. The genre tags for this data were created by parsing the MIDI file names to extract artist name and utilizing an artist to genre lookup (*Spotify* API [1]). Additionally a large scale genre dataset of two-minute electronic dance music (EDM) samples was used [9]. To extract symbolic drum tracks, a state-of-the-art drum transcription system [13] was applied.

3. UI

Figure 1 shows the basic user interface of the prototype. The main area features a eight-by-32 step-sequencer grid. Step sequencers have been shown to be an effective input method for drum pattern creation, however, they only allow a discrete time grid. Beneath the step-sequencer grid, an x/y pad allows control of complexity and loudness for pattern creation. Additionally controls for playback control, genre, tempo, and swing ratio allow customization of the drum patterns.

4. CONCLUSION

In this work we present a GAN based drum pattern generation prototype, trained using large scale drum pattern datasets. These datasets were extracted from MIDI songs and two minute audio excerpts utilizing a drum transcription system. The UI follows well-known interface paradigms used for drum pattern generation.

5. ACKNOWLEDGMENTS

This work has been partly funded by the Austrian FFG under the BRIDGE 1 project *SmarterJam* (858514), as well as by the Austrian Ministries BMVIT and BMWFW, and the Province of Upper Austria, via the COMET Center SCCH.

6. REFERENCES

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, 2017.

[2] Chris Donahue, Julian McAuley, and Miller Puckette. Synthesizing audio with generative adversarial networks. *arXiv preprint arXiv:1802.04208*, 2018.

[3] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Demonstration of a convolutional gan based model for generating multi-track piano-rolls. In *Late Breaking/Demos, 18th International Society for Music Information Retrieval Conference (ISMIR)*, 2017.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[5] Damon Horowitz. Generating rhythms with genetic algorithms. In *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 94, page 1459, 1994.

[6] Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Andrew Dai, Matt Hoffman, Curtis Hawthorne, and Douglas Eck. Generating structured music through self-attention. In *The 2018 Joint Workshop on Machine Learning for Music*, 2018.

[7] Maximos A. Kaliakatsos-Papakostas, Andreas Floros, Nikolaos Kanellopoulos, and Michael N. Vrahatis. Genetic evolution of l and fl-systems for the production of rhythmic sequences. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, pages 461–468, 2012.

[8] Maximos A. Kaliakatsos-Papakostas, Andreas Floros, and Michael N. Vrahatis. evodrummer: Deriving rhythmic patterns through interactive genetic algorithms. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, volume 7834 of *Lecture Notes in Computer Science*, pages 25–36. 2013.

[9] Peter Knees, Angel Faraldo, Perfecto Herrera, Richard Vogl, Sebastian Böck, Florian Hörschläger, and Mickael Le Goff. Two data sets for tempo estimation and key detection in electronic dance music annotated from user corrections. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, 2015.

[10] Narain Krishnamurthy. Beatgan - generating drum loops via gans. `https://github.com/NarainKrishnamurthy/BeatGAN2.0`, 2018.

[11] Cárthach Ó Nuanáin, Perfecto Herrera, and Sergi Jordà. Target-based rhythmic pattern generation and variation with genetic algorithms. In *Proceedings of the 12th Sound and Music Computing Conference*, 2015.

[12] Richard Vogl and Peter Knees. An intelligent drum machine for electronic dance music production and performance. In *Proceedings of the 17th International Conference on New Interfaces for Musical Expression (NIME)*, Copenhagen, 2017.

[13] Richard Vogl, Gerhard Widmer, and Peter Knees. Towards multi-instrument drum transcription. In *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx-18)*, 2018.

[14] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.

[1] `https://developer.spotify.com/console`

13.3 METHOD

As mentioned in the introduction to this chapter, for this prototype a GAN [28] is used as pattern generation method. GANs consist of a generator and a discriminator network. These two networks can be seen as opponents in a min/max game, where the generator tries to produce patterns which are indistinguishable from real patterns sampled from the training dataset; while the discriminator tries to identify generated patterns. In training these two networks simultaneously, both the generator and discriminator improve. For a more detailed introduction to GANs, we refer the reader back to Section 2.3.7.

A goal of this work is to add additional inputs to the generator which allow to control certain qualities of the generated drum patterns. To this end, genre tags and two features extracted from the symbolic drum patterns are used. In case of the pattern features two continuous inputs are used while the genre tag is represented by a one-hot encoded binary vector of size 26 (one bit for each used genre). These values are used as additional inputs for the generator next to the uniform random noise vector. Additionally, these values are also used as additional outputs for discriminator training. This setup forces the discriminator to extract these values from input patterns and, as a consequence, the generator to embed these features within the generated patterns. Note that while the discriminator and generator compete in terms of the discriminator output (real v.s. generated pattern class) they have to cooperate in terms of the additional features. Training the generator with these additional inputs makes the learned pattern distribution of the generator dependent on the input variables and allows a certain degree of control of the generator during inference. The next subsections will cover details of the network architecture, training data, and training strategy.

13.3.1 *Network Architecture*

For the GAN used in this work, both generator and discriminator feature a convolutional recurrent architecture. The idea behind this is that while convolutions process a single bar to model local patterns and structures, the recurrent layers ensure that a repetitive structure and dependencies in the sequence of bars can be modeled.

Another advantage of using a recurrent layer to model sequences of bars is that variable length sequences can be used for both, training and inference. This makes the model more flexible in the context of available training data and possible applications. Figure 13.1 visualizes an example of a sequence of single-bar patterns and how convolutions and the recurrent layer are involved as part of the generator network.

The detailed structure of both the generator and discriminator networks are provided in Figure 13.2. In the case of the generator, the

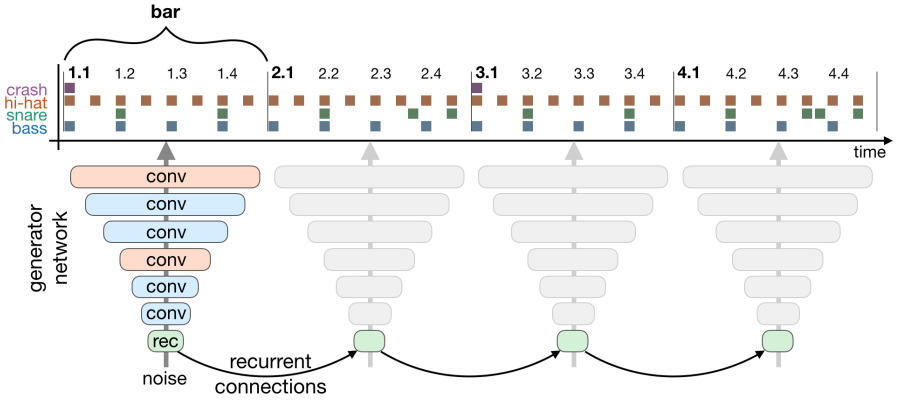


Figure 13.1: A sequence of one-bar patterns as output of the generator. Note that the recurrent layer models dependencies between bars.

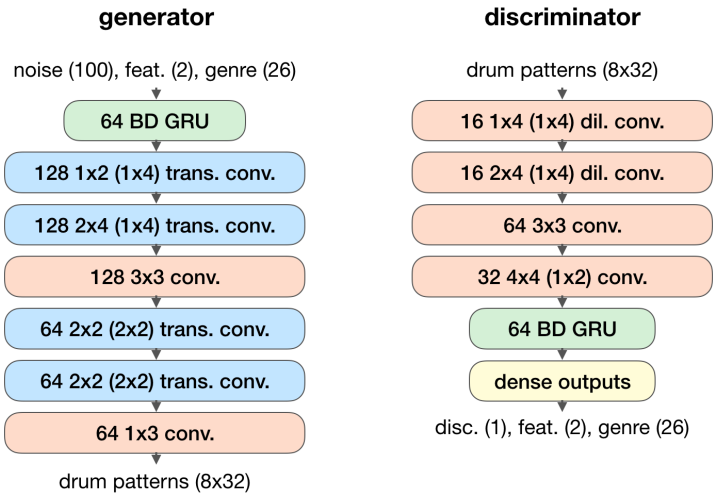


Figure 13.2: Architecture of the used GAN model. Note that while for the discriminator convolutional layers are used, in the case of the generator transposed convolutions in combination with normal convolutions are implemented. The numbers for each layer designate: *(i)* number of filters, *(ii)* filter size, and *(iii)* in parentheses strides for transposed convolutions and dilation ratio for dilated convolutions. The recurrent layers are implemented as bidirectional GRUs. As input for the generator *(i)* a uniformly distributed noise vector of size 100, *(ii)* the vector with the two features (complexity and loudness of drum patterns), and *(iii)* the one-hot encoded genre vector with size 26 (26 different genres) are used. The discriminator's outputs comprise *(i)* the discriminator output (real or generated pattern), *(ii)* output for predicted feature (same as for the generator's input: complexity and loudness), and *(iii)* output vector for one-hot encoded predicted genre (same as for the generator's input).

task of the network is to generate an $N_i \times N_t \times N_b$ tensor from an $(N_n + N_c + N_d) \times N_b$ matrix sampled from the latent space, where N_i is the number of instruments (8 for the implemented prototype), N_t is the number of time steps per bar (32), N_b the number of bars (e.g. 4), N_n the size of input noise for the generator (100), N_c the size of continuous conditional inputs (2, complexity and loudness), and N_d the size of discrete one-hot conditional input (26, number of genres). This is achieved by a combination of convolutions and transposed convolutions. The individual choices for number of layers, filter sizes, and strides are optimized to achieved the correct output size. For training and inference, the input variables are copied for each bar to be generated. This makes the recurrent connections the only source of variation for consecutive patterns in a sequence of bars, ensuring consistence. Of course for other applications, e.g. to generate a continuous cross-fading between different styles, different choices can be viable.

The primary role of the discriminator network is to decide if the $N_i \times N_t \times N_b$ patterns are real or generated. This is modeled using a single sigmoid output. Additionally, the discriminator is tasked to reproduce the discrete one-hot genre label (softmax output) as well as the two continuous features (sigmoid outputs). All this is achieved by using a stack of dilated and normal convolutions, compare Figure 13.2. The reasoning behind using 1×4 dilated filters for the first two layers is based on the time resolution of the drum patterns: In the context of this work, one bar is modeled by 32 discrete time steps, and we only consider patterns in $4/4$ time signature. The main rhythmical structure is, therefore, based on quarter notes, and a typical rhythmical pattern is usually built from integer number subdivisions. Thus, the 1×4 dilations allow the network to easily focus patterns based on 8^{th} and quarter notes ($32/4 = 8$).

13.3.2 *Training Data*

Since the goal is to generate realistic sequences of drum patterns by using a recurrent GAN architecture, it is imperative to use suitable examples for training. To this end, two different pattern datasets are created. The first set is extracted from the large scale synthetic MIDI drum transcription dataset introduced in Chapter 7. Using the drum annotations and additional beat annotations extracted from the MIDI files, sequences of one-bar patterns are generated. Artist names are extracted using the track titles to generate genre lists for each track by using the *Spotify*[1] application programming interface (API). By filtering out tracks without artist name, and artist names that cannot be matched with artists in the *Spotify* API, a final set of 2583 tracks is obtained. The tracks have an average length of around 100 bars. The

1 https://developer.spotify.com/console

genre	count	genre	count	genre	count
classical	3	dance	466	metal	114
pop	140	rap	6	blues	248
folk	357	trance	1	indie	39
ska	1	house	11	electro	10
grunge	309	jazz	41	swing	6
break	5	funk	46	country	119
emo	1	punk	95	soul	39
disco	50	hip hop	11	rock	465

Table 13.1: Resulting genre tags, and number of tracks for the MIDI subset.

genre	count	genre	count
house	1000	dubstep	999
funk-r-and-b	1000	progressive-house	1000
hip-hop	999	tech-house	1000
reggae-dub	1000	dj-tools	998
drum-and-bass	998	glitch-hop	999
electronica	1000	trance	1000
indie-dance-nu-disco	996	minimal	999
deep-house	1000	pop-rock	999
hard-dance	998	chill-out	1000
breaks	999	techno	1000
hardcore-hard-techno	998	psy-trance	999
electro-house	1000		

Table 13.2: The 23 genre tags and number of tracks for the *Beatport* subset.

detailed genre tags provided by the *Spotify* API are simplified using the genre list shown in Table 13.1.

The second set consists of drum patterns extracted from an audio dataset. This is done by utilizing the drum transcription model for eight instrument classes published in [101]. The audio material is obtained by downloading a set of around 23 000 two-minute samples alongside genre annotations from the *Beatport* website. This audio file collection is a superset of the GiantSteps datasets [47] and an artifact of creating these sets. In the case of this dataset, the original genre annotations as provided on the *Beatport* platform, displayed in Table 13.2, are used. Note that the *dj-tools* tracks are not used for training, since they mostly do not contain music, but sound effects and other audio material. The tracks of this set contain approximately 63 bars on average.

Using the symbolic drum patterns, two features used to condition the generator during training are extracted for each bar. Note that the calculation for both is arbitrary and their validity is to be evaluated in user studies. The first feature is a measure of complexity

of the pattern; in this context we mainly aim at identifying patterns used as fills. Thus, only instruments often used in fills contribute to this complexity features: bass drum, snare drum, and tom-toms. The complexity measure is now defined as the number of onsets of these instruments within one bar.

The second feature is designed to represent the perceived loudness of a pattern. To this end, the number of onsets of instruments that are more dominant (snare drum, cymbal, cowbell) and the number of onsets of instruments with a soft sound (bass drum, tom-tom, ride) are counted. The loudness measure is defined as the ratio between loud onsets and soft onsets.

Both the values for the complexity and loudness measure are normalized to the range 0–1 for both datasets.

13.3.3 *Training Strategy*

Two individual models are trained using the two different datasets. For training, an adam [46] optimizer is used.

For the first experiments, a simple binary cross-entropy for sigmoid outputs and a categorical cross-entropy loss for the one-hot genre output was used. However, empirical testing showed that better results are achieved using a Wasserstein loss [1, 2] with gradient penalty [30]. To be able to implement a Wasserstein loss, it is necessary to remove all non-linearities from the discriminator's outputs (i.e. all outputs are now linear). This also implicates that the outputs of the discriminator can no longer be interpreted as originally intended, but rather act as (Lipschitz continuous) functions measuring the similarity between the training data distribution and the generator's output data distribution.

13.4 FUTURE WORK

As already done for other drum pattern generation methods presented in this thesis, a qualitative evaluation using expert interviews will be conducted to evaluate the pattern generation method, the user experience of the UI, and suitability of control modalities for the generative model. The feedback from this user study will again be incorporated into future iterations of the prototype. Besides these, the next steps for the generative model will be to investigate options to add velocity/dynamics as well as micro timing, whenever applicable.

13.5 CONTRIBUTIONS OF AUTHORS

This work consists of equal contributions by Hamid Eghbal-zadeh and myself.

I was responsible for collecting and creating the datasets, as well as preparing the data for GAN training. I also designed the batch iterator

for network training, updated the UI to work with the new pattern variation engine, created some of the network architectures, including the CRNN architecture which was used in the end for both models. I was involved in creating the code basis for GAN training, helped fixing bugs and optimizing loss functions, etc. for GAN training. For the demo submission I wrote most of the text for the paper and created the figure of the UI prototype. I wrote all of the original contend on this topic for this thesis.

Hamid Eghbal-zadeh introduced me to GANs and created the basis of the code for GAN training, modifications for WGAN training, as well as the basis for conditional GAN training. He helped with running the training and with debugging the code. Additionally, he proofread the extended abstract for the demo submission.

Gerhard Widmer and Peter Knees acted as supervisors and provided valuable feedback throughout the process.

Part IV

DATASETS AND CONCLUSION

14 DATASETS

This chapter summarizes the datasets that were created alongside the publications which make up this thesis. Datasets are a valuable resource for the research community, since collecting data and creating annotations is a very time consuming task.

14.1 LARGE SCALE EDM DATASET

The dataset discussed in this section is related to the following publication:

Peter Knees, Angel Faraldo, Perfecto Herrera, Richard Vogl, Sebastian Böck, Florian Hörschläger, and Mickael Le Goff. "Two Data Sets for Tempo Estimation and Key Detection in Electronic Dance Music Annotated from User Corrections". In: *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*. Malaga, Spain, 2015

This work is part of the list of additional publications of this thesis. In the paper, two datasets consisting of electronic dance music (EDM) tracks were introduced. The music in these datasets consists of the two minute audio excerpts provided by the *Beatport* platform[1]. In the process of creating these datasets, a total of 22981 tracks was downloaded from the *Beatport* platform. This number results from downloading approximately 1000 tracks for each of the 23 genres tags which are used on the *Beatport* platform. While only a subset of these tracks have been used to create the key and tempo datasets published in the original publication [47], genre annotations for all tracks are provided by the artists. Annotations and an audio download tool of the key and tempo datasets can be found at:
`https://github.com/GiantSteps/giantsteps-tempo-dataset` and
`https://github.com/GiantSteps/giantsteps-key-dataset`.

The full dataset consisting of 22981 tracks with genre annotations has been used as basis for training a drum pattern generation GAN in Chapter 13. This was done by using the ADT approach introduced in Chapter 7 and 6 to extract large amounts of drum patterns alongside the genre annotations of this dataset. For more details on this dataset, see Chapter 13.

1 `https://www.beatport.com/`

14.2 LARGE SCALE SYNTHETIC DATASET

In Chapter 7 a publicly available large scale synthetic drum transcription dataset was created. The dataset consists of music synthesized from MIDI songs downloaded from a MIDI song database website containing mainly pop and rock songs. Songs are available as drum solo tracks as well as mixed tracks with accompaniment. Drums were synthesized using a variety of manually double checked and corrected drum sound fonts. Additionally, the dataset is available in balanced variants, normalizing the number of onsets for each drum instrument. This is achieved by switching the notes of certain drum instruments for individual tracks, for example replacing all hi-hat notes with ride notes. By doing this in a systematic way, the notes for each instrument under observation can be balanced almost perfectly. Parts of this set have been used for the 2018 MIREX eight-class drum transcription task. The dataset is available for download at `http://ifs.tuwien.ac.at/~vogl/dafx2018/` For more details on this dataset see Chapter 7.

14.3 RBMA MULTI-TASK DATASET

In the course of the GiantSteps[2] project, a dataset consisting of freely available music from the Red Bull Music Academy[3] (RBMA) was created. While this dataset is currently not yet published in its entirety, parts of it have been published [94] and been used in the MIREX drum transcription task. For a description of the already published parts, see Chapter 6. The full dataset not only contains detailed drum instrument annotations, but also key, chords, tempo, as well as beat and downbeat annotations. Annotations for drums, beats, key, and chords were manually created by several annotators. It provides annotations for tracks from the 2011[4] and 2013[5] RBMA Various Assets samplers. The music covers a variety of electronically produced and recorded songs of diverse genres.

14.4 CONTRIBUTIONS OF AUTHORS

In case of the EDM datasets created using the *Beatport* platform [47], the main work was conducted by Peter Knees. I was responsible for creating the scripts for downloading and organizing the full set of audio tracks, extracting the genre part of the dataset, and helped writing the paper. Creating the key and tempo annotations was done by Peter Knees, Ángel Faraldo, and Perfecto Herrera. Sebastian Böck,

2 `http://www.giantsteps-project.eu`
3 `http://www.redbullmusicacademy.com/`
4 `http://daily.redbullmusicacademy.com/2012/02/various-assets`
5 `http://daily.redbullmusicacademy.com/2014/01/various-assets-nyc-2013`

Florian Hörschläger, and Mickael Le Goff ran the evaluation of state-of-the-art methods to create the baseline results for the datasets. For the use of the dataset for GAN training in the context of drum pattern generation I ran the drum transcription and beat tracking on the full dataset and created the drum patterns.

The synthetic MIDI datasets used in Chapter 7 was completely created by myself. It is published alongside the work introducing methods for multi-instrument drum transcription [101].

For the RBMA dataset, I was responsible to organize and lead the annotation effort partly done by external annotators. Because the quality requirements, especially for drum annotations, was quite high, I double checked all drum annotations and manually corrected large parts of the annotations. Sebastian Böck helped with creating the beat and downbeat annotations.

15 CONCLUSION

To conclude the thesis, this chapter will start with a critical view on the presented works, discuss future work, and review overall findings in a broader context. In Section 15.1, the publications of this thesis are analyzed from a critical standpoint and possible minor shortcomings and chances for improvement are identified. After that, open problems in the context of ADT and drum pattern generation as well as directions to tackle them will be pointed out. The last section of this chapter and thesis will then discuss choices, findings, and results in a broader context and also discuss overall implications of this work. For a more detailed discussion of the individual methods and findings, the reader is referred to the publications in corresponding chapters.

15.1 A CRITICAL VIEW

The work covered in the publications included in this thesis was designed, written, and published over the course of four years. The quality of conducted experiments and scientific writing reflect the author's growth of knowledge, experience, repertoire of evaluation methods, and scientific writing skills within this time. This can especially be observed by looking at the implemented deep learning techniques throughout the different works (especially in the context of ADT), since I started my work on deep learning more or less at the same time. In the following, the individual publications shall be revisited with a critical eye from a current standpoint, identifying minor shortcomings and room for improvement.

[92] Richard Vogl, Matthias Dorfer, and Peter Knees. "Recurrent neural networks for drum transcription". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*. New York, NY, USA, 2016.

From a current standpoint, in this work, a rather naïve implementation of neural networks is used for drum transcription. As we know now, plain RNNs have drawbacks with respect to modeling long-term structural relations. Nevertheless, the evaluation performed in this work does not suffer from that fact, since the focus is more on local properties of the onsets. This is also the only ADT work in which no three-fold cross-validation on natural splits of the datasets is employed. Rather, a randomized 70/15/15% train/validation/test evaluation, as well as a cross-dataset evaluation is performed. In general, this strategy will yield better performance results than the three-fold cross-

validation on natural splits, since train and test data are more similar. In this work this strategy was used, because such a randomized evaluation is common practice in the context of deep learning. Furthermore, it can be argued that it is a fair way of comparing to the results in [14], which is used as a baseline. This is because in [14] also sound samples of the test set are used for basis vector priming. A similar evaluation strategy is included in [93] to compare results with [14], which is reasonable.

[93] Richard Vogl, Matthias Dorfer, and Peter Knees. "Drum Transcription from Polyphonic Music with Recurrent Neural Networks". In: *Proceedings of the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. New Orleans, LA, USA, 2017.

This work already improves on most of the minor shortcomings of the previous work. One could claim that the transformations used for data-augmentation are overfitted to the dataset, since specifically the different pitched bass drums are a major challenge of the ENST-drums dataset. However, the experiments in [105] show that the model generalizes well across different datasets, which validates the chosen data-augmentation transformations.

[94] Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees. "Drum Transcription via Joint Beat and Drum Modeling using Convolutional Recurrent Neural Networks". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017.

From a current standpoint, the training sequence lengths chosen in this work might be suboptimal in the context of beat tracking. Note that beat tracking was not the main focus. The 100 and 400 frame sequences represent one and four seconds of audio, where four seconds fit approximately two bars of music at an average tempo of 120 beats per minute (BPM). Originally, this choice was made to keep batch size equal for all models, while using the larger CRNN model. Especially in the context of beat tracking, four seconds is relatively short, while one second generally does not provide enough context to detect beats. This explains the comparably low performance in the beat tracking evaluation, which might ultimately be a reason why no further performance improvement can be achieved for multi-task training in the case of the large CRNN. However, the findings in the context of beat tracking in this work are still relevant since they provide a lower bound for training sequence length. Furthermore, later, unpublished experiments show that batch size does not have much impact in the context of beat tracking. By using training sequences of up to 3200 samples (32 seconds of audio) while decreasing batch size accordingly, beat tracking performance further improves. The limiting factor is

graphical processing unit (GPU) memory—to be able to increase the sequence length, either batch size, spectral context, or model size must be decreased.

[101] Richard Vogl, Gerhard Widmer, and Peter Knees. "Towards Multi-Instrument Drum Transcription". In: *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx)*. Aveiro, Portugal, 2018.

This work provides a comprehensive evaluation of different approaches for increasing the number of instruments under observation, while also including a new dataset for this task. When put into context with the work by Cartwright and Bello [12], one could argue that using the instrument weighting introduced in [93] and also reused in [12] could have been evaluated alongside. This approach is dismissed in the beginning of the work, since we knew that for very imbalanced classes there is no improvement to be expected. The question remains if using weighting for loss functions of the individual instruments could improve the performance in the case of the large-scale dataset. While some instrument classes in this dataset are still relatively sparsely populated there might be sufficiently many onsets for the class weighting to be effective.

[97] Richard Vogl and Peter Knees. "An Intelligent Musical Rhythm Variation Interface". In: *Companion Publication 21st International Conference on Intelligent User Interfaces*. Sonoma, CA, USA, 2016.

This work briefly introduces the pattern variation method and UI also used in follow-up works. Due to the format of the submission, many details are omitted. Details were, however, discussed in person with interested people at the poster presentation and later published in the follow-up article:

[100] Richard Vogl, Matthias Leimeister, Cárthach Ó Nuanáin, Sergi Jordà, Michael Hlatky, and Peter Knees. "An Intelligent Interface for Drum Pattern Variation and Comparative Evaluation of Algorithms". In: *Journal of the Audio Engineering Society* 64.7/8 (2016).

This work builds on a comprehensive qualitative and quantitative evaluation. Conducting qualitative interviews in a way to not bias the participants constitutes a challenge. While most participants were quite open and blunt when verbalizing their opinion of the prototype, it can not be ruled out that to some degree interviewees tried to meet expectations of the interviewer. However, results of the qualitative evaluation correlate with the findings from the interviews.

[98] Richard Vogl and Peter Knees. "An Intelligent Drum Machine for Electronic Dance Music Production and Performance". In: *Proceedings of the 17th International Conference on New Interfaces for Musical Expression (NIME)*. Copenhagen, Denmark, 2017.

In this work, it was not possible to make the A/B comparison for the UI completely blind, since it was quite obvious which was the *old* and which one the *new* iteration of the UI. This is, however, true for most A/B UI testing. Furthermore, the fact that participants were able to identify the *old* interface is an indicator that the redesign was successful, by itself.

[20] Hamid Eghbal-Zadeh, Richard Vogl, Gerhard Widmer, and Peter Knees. "A GAN based Drum Pattern Generation UI Prototype". In: *Late Breaking/Demos, 19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018.

This work is a preliminary publication of work-in-progress as a demonstration paper. Due to the format of the publication, there is not much room for methodical details. This is, however, compensated with extra sections providing more details in this thesis. A general problem with GANs is that many hyperparameters are chosen and often tuned arbitrarily to achieve successful training results. In this work, an attempt was made to choose an architecture that reflects domain knowledge on properties of rhythmic patterns, which was also inspired by findings from ADT research.

15.2 FUTURE DIRECTIONS

Since humans still outperform ADT methods, further improvement of transcription performance would be desirable. Building on the proposed deep-learning-based methods and utilizing new findings from the deep learning community, the introduced system has the potential to achieve this.

As discussed in Section 2.1.4, there are still properties of drum instrument onsets which are not considered in current state-of-the-art methods. These comprise, for example, dynamics/onset velocities and classification of playing techniques. The main challenge with these details is to collect appropriate training data. It is practically impossible to create sufficient data for these tasks for methods used in this work by means of manual annotations, due to the large amount of parameters and required accuracy. Therefore, it will be necessary to either use synthetic data, automate data annotation, or employ other learning paradigms, for example: un-/semi-supervised techniques, zero/one-shot learning, et cetera.

Similar approaches could help to better model large numbers of different drum instruments and improve performance for multi-instrument

transcription. An issue that needs further investigation is the tendency of humans to label drum instruments according to their function within a song, rather than strictly using their timbre and sound as indicator. For example, humans tend to label instruments playing the *backbeat* (notes on beats 2 and 4) as snare drum, even if they sound different than a typical snare drum. This is due to the fact that, especially in rock and pop music, the snare drum usually accentuates the backbeat. This might be a cause for quite diverse sounding instruments within certain instrument classes in ADT datasets. To investigate this issue further, it could help to create embeddings and/or cluster drum instrument sounds and compare them to the used labels within songs.

Another direction in the context of data generation is the use of data augmentation and on-the-fly data creation. With the availability of large sets of symbolic data for drum transcription, it could make sense to generate the audio on-the-fly during training. This would allow to easily employ data augmentation while immensely reducing physical size of the training data. While such an approach would be computationally more expensive, it would help generate sufficient training data when working on detailed transcripts incorporating dynamics and playing techniques.

Finally, regarding evaluation practices, while f-measure, precision, and recall have been successfully used as evaluation metrics for a long time, it might be worth investigating better suited evaluation strategies for drum patterns.

In the context of drum pattern generation, the main goal for future iterations should be to further improve pattern generation capabilities. Using GANs seems to be a promising approach, and there is a plethora of deep learning techniques that could help improve the quality of generated patterns.

To be able to better evaluate implemented pattern generation methods, further improvements of the UI prototype will be necessary. A reasonable approach would be to build a plugin which can be used within a DAW, to allow easier integration into the usual workflow of survey participants. With such tools at hand, presenting a working prototype to a wider audience and conduct further user studies would be feasible.

Another direction to explore in the context of GAN training is to try to evaluate the quality of the generated patterns without user feedback. This could be achieved by comparing statistical metrics calculated for both, the training dataset and a set of samples created by the generator.

15.3 DISCUSSION

In this thesis deep-learning-based methods for ADT and drum pattern generation are introduced. In the works on ADT covered by this thesis,

first RNN-based systems are introduced. Later it is found that CNNs can perform similarly well, and further improvements are achieved by combining convolutional with recurrent layers. The results indicate that CRNN systems have the ability to learn and leverage knowledge about the structure of typical drum patterns, given that sufficiently long training sequences are used. This seems also to be confirmed when evaluating on real data while training on the class-balanced datasets in Chapter 7. In this experiment, no performance improvements can be observed although it would have been expected. Under the assumption that CRNNs are able to learn typical drum pattern structures, a possible explanation for this behavior is that the artificial drum patterns (created by balancing the classes of the synthetic dataset) used for training, do not provide any useful knowledge about the structure of real drum patterns.

This finding further motivates the use of CRNNs for pattern generation in the experiments in Chapter 13. While the method based on RBMs introduced in Chapter 10 is promising and could still be improved by using more complex network architectures alongside other modifications, the use of GANs allows to leverage findings from the work on ADT more easily.

The experiments and user studies on drum pattern generation indicate that users are willing to incorporate artificial intelligence into their workflow, as long as they feel in control. They also hint that the UI has a large impact on the experience. Discussing possible applications for drum pattern generation methods, and talking about expectations on how these should work with participants of the user studies was very interesting. The interviews indicate that evaluating pattern generation quality is not only subjective, but also depends on the musical context. While some generative approaches might appear to be too unpredictable for electronic dance music live performances, artists in the area of experimental music might appreciate unpredictable behavior.

A core problem of music transcription and music generation is that humans usually are not aware or cannot exactly express how they tackle these tasks themselves. This makes it difficult (maybe even impossible) for humans to manually craft algorithms to solve these problems. Deep learning and more generally machine learning, on the other hand, provide methods to circumvent the explicit programming of these algorithms. However, this comes at the price of requiring large amounts of training data, which can be costly to create—depending on the task.

While a certain amount of diverse training data is indispensable, the mantra that more data will always improve performance should be taken with a grain of salt (see Chapter 7). Regularization and techniques to deal with imbalanced data have improved training for small sized datasets, and it can be expected that this trend will continue with the emergence of better regularization techniques.

Another downside of deep learning methods often mentioned is the fact that it is hard to interpret what the models actually learned. This is often a problem, because it has been shown that machine learning methods tend to be *lazy* and abuse imperfections in the training data to solve the task at hand [69, 72][1]. This is generally problematic since it usually leads to the model not generalizing well on other, unseen data. To approach this issue, efforts have been made to analyze trained neural networks (see e.g. saliency maps [74]) beyond looking at filter maps and activations. An interesting and promising direction of research in this context are so-called adversarial attacks [51].

While for some applications understanding and formally validating the inner workings of a neural network would be desirable, it is not a strict requirement in the context of MIR, as long as it can be shown that the methods reliably generalize sufficiently well. Due to the artistic context of many MIR tasks, which makes it hard to define them precisely, one cannot expect that machines will be able to solve these tasks perfectly. However, the ability to understand the inner workings of a trained neural network would provide insight how these tasks can be solved. The question remains if these insights could be used to understand how humans solve these tasks.

1 An example of such a *lazy* behavior is demonstrated for singing voice detection on this accompanying website for [69]: `https://jobim.ofai.at/singinghorse/`

BIBLIOGRAPHY

[1] Martin Arjovsky and Léon Bottou. "Towards principled methods for training generative adversarial networks". In: *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. Toulon, France, 2017.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. Sydney, Australia: PMLR, 2017, pp. 214–223.

[3] Eric Battenberg, Victor Huang, and David Wessel. "Live Drum Separation Using Probabilistic Spectral Clustering Based on the Itakura-Saito Divergence". In: *Proceedings of the Audio Engineering Society Conference on Time-Frequency Processing in Audio (AES)*. Helsinki, Finland, 2012.

[4] Eric Battenberg and David Wessel. "Analyzing Drum Patterns Using Conditional Deep Belief Networks". In: *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR)*. Porto, Portugal, 2012.

[5] Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. "A tutorial on onset detection in music signals". In: *IEEE Transactions on speech and audio processing* 13.5 (2005), pp. 1035–1047.

[6] Gilberto Bernardes, Caros Guedes, and Bruce Pennycook. "Style emulation of drum patterns by means of evolutionary methods and statistical analysis". In: *Proceedings of the 7th Sound and Music Computing Conference (SMC)*. Barcelona, Spain, 2010, pp. 1–4.

[7] Sebastian Böck, Florian Krebs, and Gerhard Widmer. "Joint Beat and Downbeat Tracking with Recurrent Neural Networks". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*. New York, NY, USA, 2016.

[8] Sebastian Böck and Markus Schedl. "Enhanced beat tracking with context-aware neural networks". In: *Proceedings of the 14th Conference on Digital Audio Effects (DAFx)*. Paris, France, 2011.

[9] Sebastian Böck and Markus Schedl. "Polyphonic piano note transcription with recurrent neural networks". In: *Proceedings of the 37th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Kyoto, Japan, 2012, pp. 121–124. ISBN: 978-1-4673-0045-2.

[10] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription". In: *Proceedings of the 29th International Conference on Machine Learning (ICML)*. Ed. by John Langford and Joelle Pineau. ICML '12. Edinburgh, Scotland, GB: Omnipress, 2012, pp. 1159–1166. ISBN: 978-1-4503-1285-1.

[11] Juan José Burred. *Detailed derivation of multiplicative update rules for NMF*. `https://www.jjburred.com/research/pdf/jjburred_nmf_updates.pdf`. [Online; accessed 16-August-2018]. 2014.

[12] Mark Cartwright and Juan P. Bello. "Increasing Drum Transcription Vocabulary Using Data Synthesis". In: *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx)*. Aveiro, Portugal, 2018, pp. 72–79.

[13] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar, 2014.

[14] Christian Dittmar and Daniel Gärtner. "Real-time transcription and separation of drum recordings based on NMF decomposition". In: *Proceedings of the 17th International Conference on Digital Audio Effects (DAFx)*. Erlangen, Germany, 2014.

[15] Chris Donahue, Julian McAuley, and Miller Puckette. "Synthesizing Audio with Generative Adversarial Networks". In: *arXiv preprint arXiv:1802.04208* (2018).

[16] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. "MuseGAN: Demonstration of a convolutional GAN based model for generating multi-track piano-rolls". In: *Late Breaking/Demos, 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017.

[17] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.7 (2011), pp. 2121–2159.

[18] Simon Durand, Juan P Bello, Bertrand David, and Gaël Richard. "Downbeat tracking with multiple features and deep neural networks". In: *Proceedings of the 40th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. South Brisbane, Australia, 2015, pp. 409–413.

[19] Simon Durand, Juan P Bello, Bertrand David, and Gaël Richard. "Feature adapted convolutional neural networks for downbeat tracking". In: *Proceedings of the 41st IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. Shanghai, China, 2016, pp. 296–300.

[20] Hamid Eghbal-Zadeh, Richard Vogl, Gerhard Widmer, and Peter Knees. "A GAN based Drum Pattern Generation UI Prototype". In: *Late Breaking/Demos, 19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018.

[21] Derry FitzGerald, Bob Lawlor, and Eugene Coyle. "Drum transcription in the presence of pitched instruments using prior subspace analysis". In: *Proceedings of the Irish Signals Systems Conference*. Limerick, Ireland, 2003.

[22] Derry FitzGerald, Robert Lawlor, and Eugene Coyle. "Prior subspace analysis for drum transcription". In: *Proceedings of the 114th Audio Engineering Society Conference*. Amsterdam, Netherlands, 2003.

[23] Derry FitzGerald and Jouni Paulus. "Unpitched percussion transcription". In: *Signal Processing Methods for Music Transcription*. Ed. by Anssi Klapuri and Manuel Davy. Springer, 2006, pp. 131–162.

[24] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks". In: *Journal of machine learning research* 3.8 (2002), pp. 115–143.

[25] Olivier Gillet and Gaël Richard. "Automatic transcription of drum loops". In: *Proceedings of the 29th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vol. 4. Montreal, QC, Canada, 2004, pp. iv-269–iv-272. ISBN: 0-7803-8484-9.

[26] Olivier Gillet and Gaël Richard. "Transcription and separation of drum signals from polyphonic music". In: *IEEE Transactions on Audio, Speech, and Language Processing* 16.3 (2008), pp. 529–540.

[27] Hanlin Goh, Nicolas Thome, and Matthieu Cord. "Biasing Restricted Boltzmann Machines to Manipulate Latent Selectivity and Sparsity". In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. Whistler, BC, Canada, 2010.

[28] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems (NIPS)*. 2014, pp. 2672–2680.

[29] Matthias Gruhne and Christian Dittmar. "Improving Rhythmic Pattern Features Based on Logarithmic Preprocessing". In: *Proceedings of the 126th Audio Engineering Society Convention*. Munich, Germany, 2009.

[30] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. "Improved training of wasserstein gans". In: *Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. 5767–5777.

[31] Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse Engel, Sageev Oore, and Douglas Eck. "Onsets and Frames: Dual-Objective Piano Transcription". In: *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018.

[32] Perfecto Herrera, Amaury Dehamel, and Fabien Gouyon. "Automatic Labeling of Unpitched Percussion Sounds". In: *Audio Engineering Society Convention 114*. 2003.

[33] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18.7 (2006), pp. 1527–1554. ISSN: 0899-7667.

[34] Jordan Hochenbaum and A Kapur. "Drum Stroke Computing: Multimodal Signal Processing for Drum Stroke Identification and Performance Metrics". In: *Proceedings of the 11th International Conference on New Interfaces for Musical Expression (NIME)*. 2011.

[35] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[36] André Holzapfel and Yannis Stylianou. "Scale Transform in Rhythmic Similarity of Music". In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.1 (2011), pp. 176–185.

[37] Damon Horowitz. "Generating rhythms with genetic algorithms". In: *Proceedings of the 12th National Conference on Artificial Intelligence*. Vol. 94. Cambridge, MA. Seattle, WA, USA: The AAAI Press, Menlo Park, California, 1994, p. 1459.

[38] Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Andrew Dai, Matt Hoffman, Curtis Hawthorne, and Douglas Eck. "Generating Structured Music Through Self-Attention". In: *The 2018 Joint Workshop on Machine Learning for Music*. 2018.

[39] Sergey Ioffe and Christian Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. `http://arxiv.org/abs/1502.03167`. 2015. eprint: `arXiv:1502.03167`.

[40] Celine Jacques and Axel Röbel. "Automatic Drum Transcription with Convolutional Neural Networks". In: *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx)*. Aveiro, Portugal, 2018, pp. 80–86.

[41] Jesper Højvang Jensen, Mads Græsbøll Christensen, and Søren Holdt Jensen. "A Tempo-insensitive Representation of Rhythmic Patterns". In: *Proceedings of the 17th European Signal Processing Conference (EUSIPCO)*. Glasgow, Scotland, 2009, pp. 1509–1512.

[42] Maximos A. Kaliakatsos-Papakostas, Andreas Floros, Nikolaos Kanellopoulos, and Michael N. Vrahatis. "Genetic Evolution of L and FL-systems for the Production of Rhythmic Sequences". In: *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '12. Philadelphia, Pennsylvania, USA: ACM, 2012, pp. 461–468. ISBN: 978-1-4503-1178-6.

[43] Maximos A. Kaliakatsos-Papakostas, Andreas Floros, and Michael N. Vrahatis. "evoDrummer: Deriving Rhythmic Patterns through Interactive Genetic Algorithms". In: *Evolutionary and Biologically Inspired Music, Sound, Art and Design*. Vol. 7834. Lecture Notes in Computer Science. Springer, 2013, pp. 25–36. ISBN: 978-3-642-36954-4.

[44] Maximos A. Kaliakatsos-Papakostas, Andreas Floros, Michael N. Vrahatis, and Nikolaos Kanellopoulos. "Real-time drums transcription with characteristic bandpass filtering". In: *Proceedings of Audio Mostly: A Conference on Interaction with Sound*. Corfu, Greece, 2012.

[45] Rainer Kelz, Matthias Dorfer, Filip Korzeniowski, Sebastian Böck, Andreas Arzt, and Gerhard Widmer. "On the potential of simple framewise approaches to piano transcription". In: (2016).

[46] Diederik P. Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[47] Peter Knees, Angel Faraldo, Perfecto Herrera, Richard Vogl, Sebastian Böck, Florian Hörschläger, and Mickael Le Goff. "Two Data Sets for Tempo Estimation and Key Detection in Electronic Dance Music Annotated from User Corrections". In: *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*. Malaga, Spain, 2015, pp. 364–370.

[48] Filip Korzeniowski and Gerhard Widmer. "End-to-End Musical Key Estimation Using a Convolutional Neural Network". In: *Proceedings of the 25th European Signal Processing Conference (EUSIPCO)*. 2017, pp. 996–1000.

[49] Florian Krebs, Sebastian Böck, Matthias Dorfer, and Gerhard Widmer. "Downbeat Tracking using Beat-Synchronous Features and Recurrent Neural Networks". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*. New York, NY, USA, 2016, pp. 129–135.

[50] Narain Krishnamurthy. *BeatGAN - Generating Drum Loops via GANs*. `https://github.com/NarainKrishnamurthy/BeatGAN2.0`. 2018.

[51] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *CoRR* abs/1607.02533 (2016). arXiv: `1607.02533`.

[52] Clément Laroche, Hélène Papadopoulos, Matthieu Kowalski, and Gaël Richard. "Drum Extraction in Single Channel Audio Signals using Multi-Layer Non Negative Matrix Factor Deconvolution". In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. New Orleans, LA, USA, 2017, pp. 46–50.

[53] Daniel D Lee and H Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755 (1999), p. 788.

[54] Daniel D. Lee and H. Sebastian Seung. "Algorithms for Non-negative Matrix Factorization". In: *Advances in Neural Information Processing Systems (NIPS)*. Denver, CO, USA, 2000, pp. 556–562.

[55] Henry Lindsay-Smith, Skot McDonald, and Mark Sandler. "Drumkit Transcription via Convolutive NMF". In: *Proceedings of the International Conference on Digital Audio Effects (DAFx)*. York, UK, 2012.

[56] *MIREX website*. `http://www.music-ir.org/`. [Online; accessed 16-August-2018]. 2018.

[57] Marius Miron, Matthew EP Davies, and Fabien Gouyon. "An open-source drum transcription system for Pure Data and Max MSP". In: *Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Vancouver, BC, Canada, 2013, pp. 221–225.

[58] David E. Moriarty and Risto Mikkulainen. "Efficient reinforcement learning through symbiotic evolution". In: *Machine learning* 22.1-3 (1996), pp. 11–32.

[59] Yurii Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence O (1/k2)". In: *Doklady an SSSR*. Vol. 269. 3. 1983, pp. 543–547.

[60] Cárthach Ó Nuanáin, Perfecto Herrera, and Sergi Jordà. "Target-Based Rhythmic Pattern Generation and Variation with Genetic Algorithms". In: *Proceedings of the 12th Sound and Music Computing Conference (SMC)*. Maynooth, Ireland, 2015.

[61] Christopher Olah. *Understanding LSTMs*. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. [Online; accessed 16-August-2018]. 2015.

[62] Pentti Paatero and Unto Tapper. "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values". In: *Environmetrics* 5.2 (1994), pp. 111–126.

[63] Jean-François Paiement, Yves Grandvalet, Samy Bengio, and Douglas Eck. "A Generative Model for Rhythms". In: *NIPS Workshop on Brain, Music and Cognition*. Whistler, BC, Canada, 2007.

[64] Jouni Paulus and Anssi Klapuri. "Drum sound detection in polyphonic music with hidden Markov models". In: *EURASIP Journal on Audio, Speech, and Music Processing* 14 (2009).

[65] Matthew Prockup, Erik M. Schmidt, Jeffrey Scott, and Youngmoo E. Kim. "Toward Understanding Expressive Percussion Through Content Based Analysis". In: *Proceedings of the 14th Society for Music Information Retrieval Conference (ISMIR)*. 2013.

[66] Emmanuel Ravelli, Juan Bello, and Mark Sandler. "Automatic Rhythm Modification of Drum Loops". In: 14 (2007), pp. 228–231.

[67] Axel Röbel, Jordi Pons, Marco Liuni, and Mathieu Lagrange. "On Automatic Drum Transcription using Non-Negative Matrix Deconvolution and Itakura Saito Divergence". In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Brisbane, Australia, 2015, pp. 414–418.

[68] W. Andrew Schloss. "On the Automatic Transcription of Percussive Music - From Acoustic Signal to High-Level Analysis". PhD thesis. Stanford University, 1985.

[69] Jan Schlüter. "Learning to Pinpoint Singing Voice from Weakly Labeled Examples". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*. New York, USA, 2016.

[70] Jan Schlüter and Sebastian Böck. "Improved musical onset detection with convolutional neural networks". In: *Proceedings of the 39th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. Florence, Italy, 2014, pp. 6979–6983.

[71] Jan Schlüter and Thomas Grill. "Exploring data augmentation for improved singing voice detection with neural networks". In: *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*. Málaga, Spain, 2015.

[72] Jan Schlüter and Bernhard Lehner. "Zero-Mean Convolutions for Level-Invariant Singing Voice Detection". In: *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018.

[73] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[74] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. Banff, AB, Canada, 2014. eprint: 1312.6034.

[75] Paris Smaragdis. "Non-negative Matrix Factor Deconvolution; Extraction of Multiple Sound Sources from Monophonic Inputs". In: *Proceedings of the International Conference on Independent Component Analysis and Blind Signal Separation (ICA)*. Grenada, Spain, 2004, pp. 494–499.

[76] Paris Smaragdis. "Non-negative Matrix Factor Deconvolution; Extraction of Multiple Sound Sources from Monophonic Inputs". In: *Proceedings of the 5th International Conference on Independent Component Analysis and Blind Signal Separation (ICA)*. Vol. 3195. Lecture Notes in Computer Science. Granada, Spain: Springer, 2004, pp. 494–499. ISBN: 978-3-540-23056-4.

[77] Paul Smolensky. "Information Processing in Dynamical Systems: Foundations of Harmony Theory". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*. MIT Press, 1986, pp. 194–281.

[78] Carl Southall, Ryan Stables, and Jason Hockman. "Automatic Drum Transcription using Bidirectional Recurrent Neural Networks". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*. New York, NY, USA, 2016.

[79] Carl Southall, Ryan Stables, and Jason Hockman. "Automatic Drum Transcription for Polyphonic Recordings using Soft Attention Mechanisms and Convolutional Neural Networks". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017, pp. 606–612.

[80] Vinícius M. A. Souza, Gustavo E. A. P. A. Batista, and Nilson E. Souza-Filho. "Automatic classification of drum sounds with indefinite pitch". In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. Killarney, Ireland, 2015, pp. 1–8.

[81] Andrio Spich, Massimiliano Zanoni, Augusto Sarti, and Stefano Tubaro. "Drum music transcription using prior subspace analysis and pattern recognition". In: *Proceedings of the 13th International Conference of Digital Audio Effects (DAFx)*. Graz, Austria, 2010.

[82] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[83] Richard S. Sutton. "Two problems with backpropagation and other steepest-descent learning procedures for networks". In: *Proceedings of the 8th annual Conference of the Cognitive Science Society*. Erlbaum. 1986, pp. 823–831.

[84] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. "Training neural networks without gradients: A scalable admm approach". In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. 2016.

[85] Lucas Theis, Aäron van den Oord, and Matthias Bethge. "A note on the evaluation of generative models". In: *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. San Juan, Puerto Rico, 2016.

[86] Tijmen Tieleman and Geoffrey Hinton. "Using Fast Weights to Improve Persistent Contrastive Divergence". In: *Proceedings of the 26th International Conference on Machine Learning (ICML)*. ACM. Montreal, QC, Canada, 2009, pp. 1033–1040.

[87] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural Networks for Machine Learning*. 2012.

[88] Adam Tindale, Ajay Kapur, George Tzanetakis, and Ichiro Fujinaga. "Retrieval of percussion gestures using timbre classification techniques". In: *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*. Barcelona, Spain, 2004.

[89] Godfried Toussaint. "A Comparison of Rhythmic Similarity Measures". In: *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*. `http://ismir2004.ismir.net/proceedings/p045-page-242-paper134.pdf`. Barcelona, Spain, 2004.

[90] Godfried Toussaint. "Generating "Good" Musical Rhythms Algorithmically". In: *Proceedings of the 8th International Conference on Arts and Humanities*. Honolulu, HI, USA, 2010, pp. 774–791.

[91] George Tzanetakis, Ajay Kapur, and Richard I McWalter. "Subband-based drum transcription for audio signals". In: *Proceedings of the 7th IEEE Workshop on Multimedia Signal Processing*. Shanghai, China, 2005. ISBN: 0-7803-9288-4.

[92] Richard Vogl, Matthias Dorfer, and Peter Knees. "Recurrent neural networks for drum transcription". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*. New York, NY, USA, 2016.

[93] Richard Vogl, Matthias Dorfer, and Peter Knees. "Drum Transcription from Polyphonic Music with Recurrent Neural Networks". In: *Proceedings of the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. New Orleans, LA, USA, 2017.

[94] Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees. "Drum Transcription via Joint Beat and Drum Modeling using Convolutional Recurrent Neural Networks". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017, pp. 150–157.

[95] Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees. "MIREX Submission For Drum Transcription 2017". In: *MIREX extended abstracts, 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017.

[96] Richard Vogl, Hamid Eghbal-Zadeh, Gerhard Widmer, and Peter Knees. "GANs and Poses: An Interactive Generative Music Installation Controlled by Dance Moves". In: *Interactive Machine-Learning for Music @Exhibition at ISMIR*. Paris, France, 2018.

[97] Richard Vogl and Peter Knees. "An Intelligent Musical Rhythm Variation Interface". In: *Companion Publication 21st International Conference on Intelligent User Interfaces*. Sonoma, CA, USA, 2016.

[98] Richard Vogl and Peter Knees. "An Intelligent Drum Machine for Electronic Dance Music Production and Performance". In: *Proceedings of the 17th International Conference on New Interfaces for Musical Expression (NIME)*. Copenhagen, Denmark, 2017, pp. 251–256.

[99] Richard Vogl and Peter Knees. "MIREX Submission For Drum Transcription 2018". In: *MIREX extended abstracts, 19th International Society for Music Information Retrieval Conference (ISMIR)*. Paris, France, 2018.

[100] Richard Vogl, Matthias Leimeister, Cárthach Ó Nuanáin, Sergi Jordà, Michael Hlatky, and Peter Knees. "An Intelligent Interface for Drum Pattern Variation and Comparative Evaluation of Algorithms". In: *Journal of the Audio Engineering Society* 64.7/8 (2016), pp. 503–513.

[101] Richard Vogl, Gerhard Widmer, and Peter Knees. "Towards Multi-Instrument Drum Transcription". In: *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx)*. Aveiro, Portugal, 2018, pp. 57–64.

[102] Paul J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[103] Wikipedia contributors. *Drum kit — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Drum_kit&oldid=854392691`. [Online; accessed 16-August-2018]. 2018.

[104] Wikipedia contributors. *Precision and recall — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=853202943`. [Online; accessed 16-August-2018]. 2018.

[105] Chih-Wei Wu, Christian Dittmar, Carl Southall, Richard Vogl, Gerhard Widmer, Jason Hockman, Meinhard Müller, and Alexander Lerch. "A Review of Automatic Drum Transcription". In: *IEEE Transactions on Audio, Speech and Language Processing* 26.9 (2018), pp. 1457–1483.

[106] Chih-Wei Wu and Alexander Lerch. "Drum transcription using partially fixed non-negative matrix factorization with template adaptation". In: *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*. Málaga, Spain, 2015, pp. 257–263.

[107] Chih-Wei Wu and Alexander Lerch. "On Drum Playing Technique Detection in Polyphonic Mixtures". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*. New York City, United States, 2016, pp. 218–224.

[108] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation". In: *arXiv preprint arXiv:1703.10847* (2017).

[109] Kazuyoshi Yoshii, Masataka Goto, and Hiroshi G. Okuno. "Automatic Drum Sound Description for Real-World Music Using Template Adaptation and Matching Methods". In: *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*. Barcelona, Spain, 2004, pp. 184–191.

[110] Kazuyoshi Yoshii, Masataka Goto, and Hiroshi G Okuno. "Drum sound recognition for polyphonic audio signals by adaptation and matching of spectrogram templates with harmonic structure suppression". In: *IEEE Transactions on Audio, Speech, and Language Processing* 15.1 (2007), pp. 333–345.

[111] Fisher Yu and Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". In: *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. San Juan, Puerto Rico, 2016. eprint: 1511.07122.

[112] Matthew D. Zeiler. *ADADELTA: an adaptive learning rate method*. https://arxiv.org/abs/1212.5701. 2012. eprint: arXiv:1212.5701.

[113] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.