From Artificial Neural Networks to Deep Learning for Music Generation - History, Concepts and Trends

Jean-Pierre Briot

From Artificial Neural Networks to Deep Learning for Music Generation – History, Concepts and Trends*

Jean-Pierre Briot†‡

† Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
‡ UNIRIO, Rio de Janeiro, RJ 22290-250, Brazil

Jean-Pierre.Briot@lip6.fr

Abstract: The current tsunami of deep learning (the hyper-vitamined return of artificial neural networks) applies not only to traditional statistical machine learning tasks: prediction and classification (e.g., for weather prediction and pattern recognition), but has already conquered other areas, such as translation. A growing area of application is the generation of creative content: in particular the case of music, the topic of this paper. The motivation is in using the capacity of modern deep learning techniques to automatically learn musical styles from arbitrary musical corpora and then to generate musical samples from the estimated distribution, with some degree of control over the generation. This article provides a survey of music generation based on deep learning techniques. After a short introduction to the topic illustrated by a recent exemple, the article analyses some early works from the late 1980s using artificial neural networks for music generation and how their pioneering contributions foreshadowed current techniques. Then, we introduce some conceptual framework to analyze the various concepts and dimensions involved. Various examples of recent systems are introduced and analyzed to illustrate the variety of concerns and of techniques.

1 Introduction

Since the mid 2010s[1], deep learning has been producing striking successes and is now used routinely for classification and prediction tasks, such as image recognition, voice recognition or translation. It continues conquering new domains, for instance source separation[2] [8] and text-to-speech synthesis [44].

A growing area of application of deep learning techniques is the generation of content. Content can be of various kinds: images, text and music, which is the focus of this article. The motivation is in using now widely available various corpora to automatically learn musical styles and to generate new musical content based on this. Since a few years, there is a large number of scientific papers about deep learning architectures and experiments to generate music, as witnessed in [2]. The objective of this article is to explain some fundamentals as well as various achievements of this stream of research.

1.1 Related Work and Organization

This article takes some inspiration from the recent comprehensive survey and analysis in [2], but has a quite different organization and material and includes an original historical retrospective analysis. Article [3] is an analysis focusing on and driven only by challenges. In [19], Herremans *et al.* propose a function-oriented taxonomy for various kinds of music generation systems. Some more general surveys about of AI-based methods for algorithmic music composition are by Papadopoulos and Wiggins [46] and by Fernández and Vico [9], as well as books by Cope [5] and by Nierhaus [43]. In [15], Graves analyses the application of recurrent neural networks architectures to generate sequences (text and music). In [10], Fiebrink and Caramiaux address the issue of using machine learning to generate creative music. In [47], Pons presents a short historical analysis of the use of neural networks for various types of music applications (that we expand in depth).

*Invited and under evaluation for a Special Issue on Arts in a Neural networks journal.

[1] In 2012, an image recognition competition (the ImageNet Large Scale Visual Recognition Challenge) was won by a deep neural network algorithm named AlexNet [29], with a stunning margin over the other algorithms which were using handcrafted features. This striking victory was the event which ended the prevalent opinion that neural networks with many hidden layers could not be efficiently trained and which started the deep learning wave.

[2] Audio source separation, often coined as the cocktail party effect, has been known for a long time to be a very difficult problem, see the original article in [4].

This article is organized as follows. Section 1 (this section) introduces the general context of deep learning-based music generation and includes a comparison to some related work. Section 2 introduces the principles and the various ways of generating music from models. Section 3 presents a recent example to illustrate the domain of study. Section 4 analyzes in depth some pioneering works from the late 1980s in neural networks-based music generation and their recent impact. Section 5 presents some conceptual framework, in order to organize the various types of current deep learning-based music generation systems, following the model introduced in [2]. We analyze possibles types of representation in Section 6. Then, we analyze successively, basic types of architectures and strategies in Section 7, various ways to construct compound architectures in Section 8 and some more refined architectures and strategies in Section 9, before concluding this article.

2 Music Generation

In this article, we will focus on computer-based music composition and not on computer-based sound generation. This is often also named *algorithmic music composition* [43, 5], in other words, using a formal process, including steps (algorithm) and components, to compose music.

2.1 Brief History

One of the first documented case of algorithmic composition, long before computers, is the Musikalisches Wurfelspiel (Dice Music), attributed to Mozart. A musical piece is generated by concatenating randomly selected (by throwing dices) predefined music segments composed in a given style (Austrian waltz in a given key).

The first musics generated by computer appeared in the late 1950s, shortly after the invention of the first computers. The Illiac Suite is the first score composed by a computer [20] and was an early example of algorithmic music composition, making use of stochastic models (Markov chains) for generation, as well as rules to filter generated material according to desired properties. Note that, as opposed to the previous case which consists in rearranging predefined material, abstract models (transitions and constraints) are used to guide the generation.

One important limitation is that the specification of such abstract models, being rules, grammar, or automata, is difficult (reserved to experts) and error prone. With the advent of machine learning techniques, it became natural to apply them to learn models from a corpus of existing music. In addition, the method becomes, in principle, independent of a specific musical style[3] (e.g., classical, jazz, blues, serial).

2.2 Human Participation and Evaluation

We may consider two main approaches regarding human participation to a computer-based music composition process:

- *autonomous generation* – Some recent examples are Amper, AIVA or Jukedeck systems/companies, based on various techniques (e.g., rules, reinforcement learning, deep learning) aimed at the creation of original music for commercials and documentaries. In such systems, generation is *automated* with the user being restricted to a role of *parametrization* of the system though a set of characteristics (style, emotion targeted, tempo, etc.).

- *composition assistance* – An example is the FlowComposer environment[4] [45]. In such highly interactive systems, the user is composing incrementally with the help (suggestion, completion, complementation, etc.) of the environment.

Most current works using deep learning to generate music are autonomous and the way to evaluate them is a musical Turing test, i.e. presenting to various human evaluators (beginners or experts) original music (of a given style of a known compositor, e.g., Bach[5]) mixed with music generated after having learnt that style. As we will see in the following, deep learning techniques turn out to be very efficient at succeeding in such tests, due to their capacity to learn very well musical style from a given corpus and to generate new music that fits into this style.

As pointed out, e.g., in [3], most current neural networks/deep learning-based systems are black-box autonomous generators, with low capacity for incrementality and interactivity[6]. However, expert users may use them as generators of primary components (e.g., melodies, chord sequences, or/and rhythm loops) and assemble

[3] Actually, the style is defined *extensively* by (and learnt from) the various examples of music selected as the training examples.

[4] Using various techniques such as Markov models, constraints and rules, and not (yet) deep learning techniques.

[5] The fact that Bach music is often used for such experiments may not be only because of his wide availability, but also because his music is actually easier to automate, as Bach himself was somehow an algorithmic music composer. An example is the way he was composing chorales by designing and applying (with talent) counterpoint rules to existing melodies.

[6] Two exceptions will be introduced in Section 9.5.

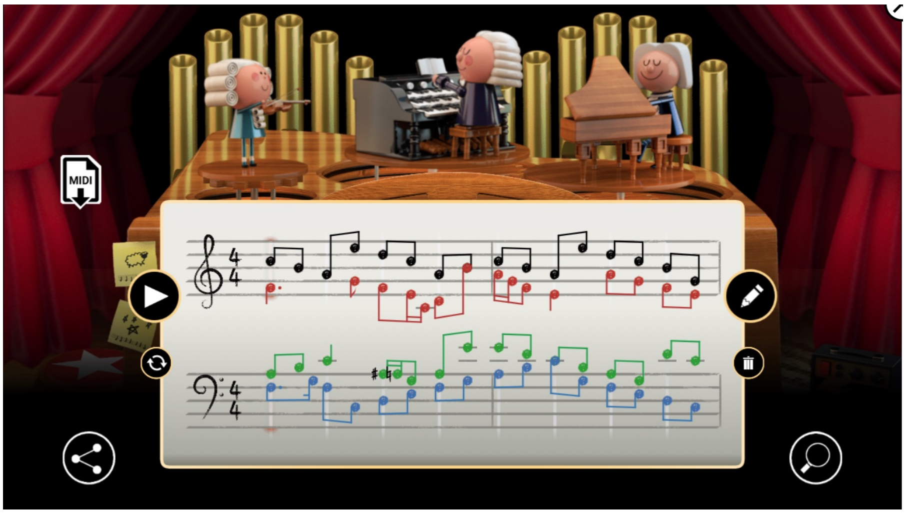

Figure 1: Example of chorale generation by Bach Doodle. The original soprano melody is in black and the generated counterpoint melodies in color (alto in red, tenor in green and bass in blue). ©2019 Google LLC, used with permission

and orchestrate them by hand. An example is the experiment conducted by the YACHT dance music band with the MusicVAE architecture[7] from the Google Magenta Project [36].

3 A First Example

On the 21st of March of 2019, for the anniversary of Bach's birthday, Google presented an interactive Doodle generating some Bach's style counterpoint for a melody entered interactively by the user. In practice the system generated three matching parts, corresponding to alto, tenor and bass voices, as shown in Figure 1. The underlying architecture, named Coconet [24], has been trained on a dataset of 306 Bach chorales. It will be analyzed in Section 9.5, but as it is too much sophisticated for an introduction, we will at first consider a more straightforward architecture, named MiniBach[8] [2, Section 6.2.2].

As a further simplification, we consider only 4 measures long excerpts from the corpus. Therefore, the dataset is constructed by extracting all possible 4 measures long excerpts from the original 352 chorales, also transposed in all possible keys. Once trained on this dataset, the system may be used to generate three counterpoint voices corresponding to an arbitrary 4 measures long melody provided as an input. Somehow, it does capture the practice of Bach, who chose various melodies for a soprano and composed the three additional voices melodies (for alto, tenor and bass) in a counterpoint manner.

The input as well output representations are symbolic, of the piano roll type, with a direct encoding into one-hot vectors[9]. Time quantization (the value of the time step) is set at the sixteenth note, which is the minimal note duration used in the corpus, i.e., there are 16 time steps for each 4/4 measure. The resulting input representation, which corresponds to the soprano melody, has as its size: 21 possible notes $\times$ 16 time steps $\times$ 4 measures = 1,344. The output representation, which corresponds to the concatenation of the three generated counterpoint melodies, has as its size: $(21 + 21 + 28) \times 16 \times 4 = 4{,}480$.

The architecture, shown in Figure 2, is feedforward (the *vanilla* type of artificial neural network architecture) for a multiple classification task: to find out the most likely note for each time slice of the three counterpoint melodies There is a single hidden layer with 200 units[10]. Successive melody time slices are encoded into successive one-hot vectors which are concatenated and directly mapped to the input nodes. In Figure 2, each blackened vector element, as well as each corresponding blackened input node element, illustrate the specific encoding (one-hot vector index) of a specific note time slice, depending on its actual pitch (or a note hold in the case of a longer note, shown with a bracket). The dual process happens at the output. Each grey output node element illustrates the chosen note (the one with the highest probability), leading to a corresponding one-hot index, leading ultimately to a sequence of notes for each counterpoint voice. (For more details, see [2, Section 6.2.2].)

After training on several examples, generation can take place, with an example of chorale counterpoint generated from a soprano melody shown in Figure 3.

[7] To be introduced in Section 9.3.
[8] MiniBach is an over simplification of DeepBach [18], to be analyzed in Section 9.5.
[9] *Piano roll format* and *one-hot encoding* will be explained in Section 6.
[10] This is an arbitrary choice.

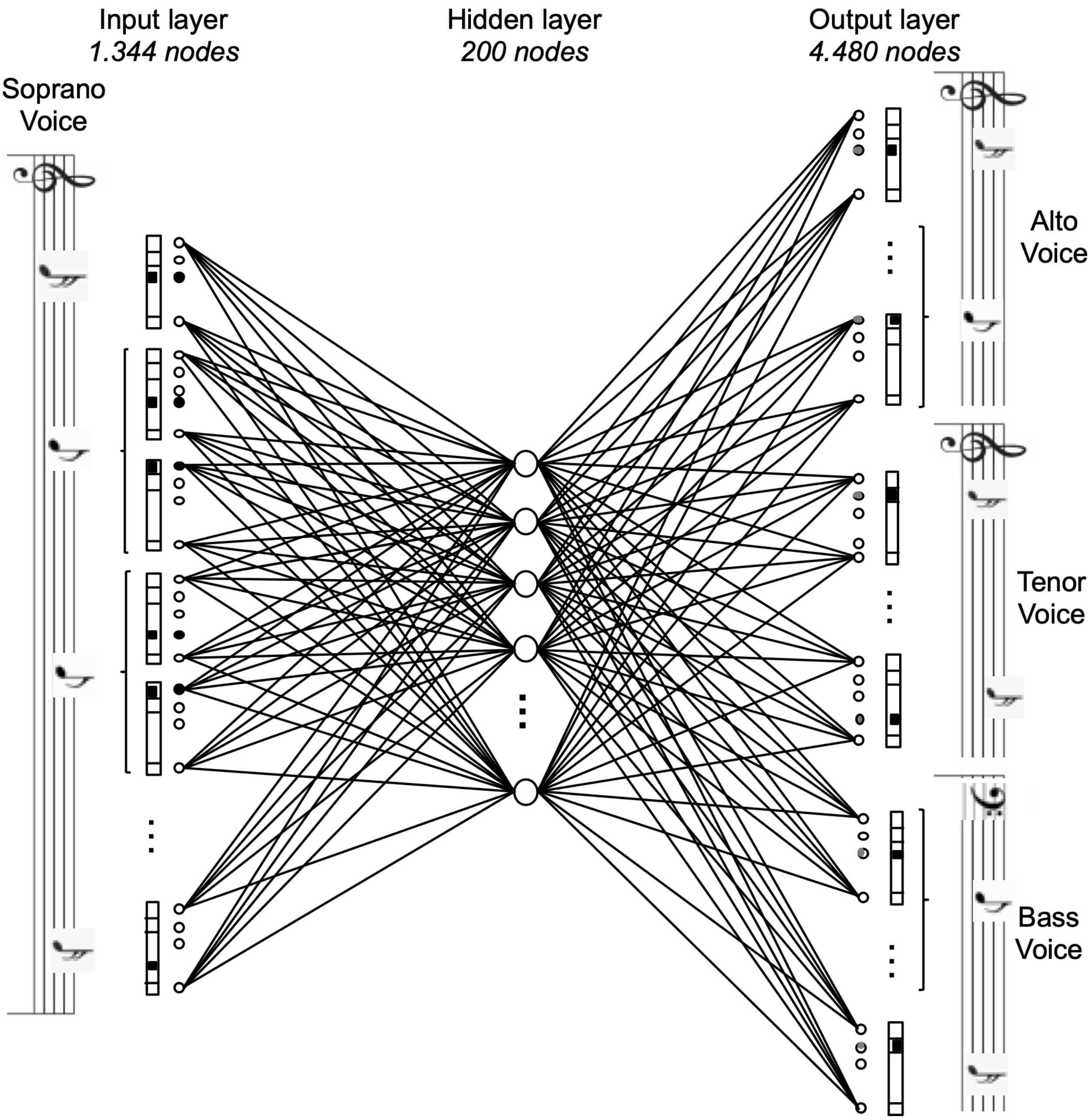

Figure 2: MiniBach architecture and encoding

Figure 3: Example of a chorale counterpoint generated from a soprano melody by MiniBach

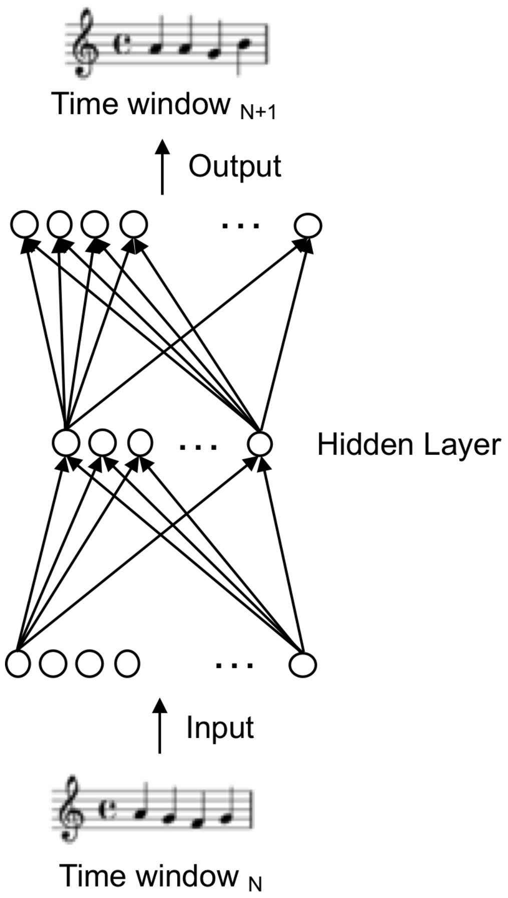

Figure 4: Time-Windowed architecture. Inspired from [55]

4 Pioneering Works and their Offsprings

As pointed out by Pons in [47], a first wave of applications of artificial neural networks to music appeared in the late 1980s[11]. This corresponds to the second wave of the artificial neural networks movement[12] [13, Section 1.2], with the innovation of hidden layers and backpropagation [51, 37].

4.1 Todd's Time-Windowed and Conditioned Recurrent Architectures

The experiments by Todd in [55] were one of the very first attempts at exploring how to use artificial neural networks to generate music. Although the architectures he proposed are not directly used nowadays, his experiments and discussion were pioneering and are still an important source of information.

Todd's objective was to generate a monophonic melody in some iterative way. He named his first design the Time-Windowed architecture, shown in Figure 4, where a sliding window of successive time-periods of fixed size is considered (in practice, one measure long). Generation is conducted iteratively melody segment by segment (and recursively, as current output segment is entered as the next input segment and so on). Note that, although the network will learn the pairwise correlations between two successive melody segments[13], there is no explicit memory for learning long term correlations.

His third design is named Sequential and is shown in Figure 5. The input layer is divided in two parts, named the *context* and the *plan*. The context is the actual memory (of the melody generated so far) and consists in units corresponding to each note (D_4 to C_6), plus a unit about the note begin information (notated as "nb" in Figure 5)[14]. Therefore, it receives information from the output layer which produces next note, with a reentering connexion corresponding to each unit[15]. In addition, as Todd explains it: "A memory of more than just the single previous output (note) is kept by having a self-feedback connection on each individual context

[11] A collection of such early papers is [57].

[12] After the early stop of the first wave due to the critique of the limitation of the Perceptron [39].

[13] In that respect, the Time-Windowed model is analog to an order 1 Markov model (considering only the previous state) at the level of a melody measure.

[14] As a way to distinguish a longer note from a repeated note.

[15] Note that the output layer is isomorphic to the context layer.

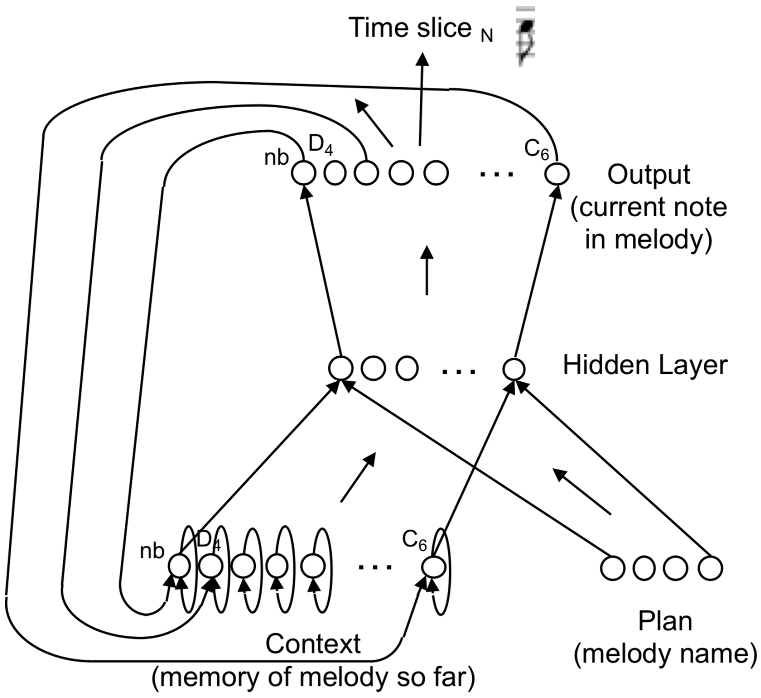

Figure 5: Sequential architecture. Inspired from [55]

unit."[16] The plan is a way to name[17] a particular melody (among many) that the network has learnt.

Training is done by selecting a plan (melody) to be learnt. The activations of the context units are initialized to 0 in order to begin with a clean empty context. The network is then feedforwarded and its output, corresponding to the first time step note, is compared to the first time step note of the melody to be learnt, resulting in the adjustment of the weights. The output values[18] are passed back to the current context. And then, the network is feedforwarded again, leading to the next time step note, again compared to the melody target, and so on until the last time step of the melody. This process is then repeated for various plans (melodies).

Generation of new melodies is conducted by feedforwarding the network with a new plan, corresponding to a new melody (not part of the training plans/melodies). The activations of the context units are initialized to 0 in order to begin with a clean empty context. The generation takes place iteratively, time step after time step. Note that, as opposed to the now more common recursive generation strategy (to be detailed in Section 7.2.1), in which the output is explicitly reentered (recursively) into the input of the architecture, in Todd's Sequential architecture the reentrance is *implicit* because of the specific nature of the recurrent connexions: the output is reentered into the context units while the input – the plan melody – is constant.

After having trained the network on a plan melody, various melodies may be generated by extrapolation by inputing new plans, or by interpolation between several (two or more) plans melodies that have been learnt. An example of interpolation is shown in Figure 6.

4.1.1 Influence

Todd's Sequential architecture is one of the first examples of using a recurrent architecture and an iterative strategy[19] for music generation. Moreover, note that introducing an extra input, named plan, which represents a melody that the network has learnt, could be seen as a precursor of *conditioning* architectures, where a specific input is used to condition (parametrize) the training of the architecture[20].

Furthermore, in the added Addendum of the republication of his initial paper [56, 190–194], Todd mentions some issues and directions:

[16] This is a peculiar characteristic of this architecture, as in recent standard recurrent network architecture recurrent connexions are encapsulated within the hidden layer (as we will see in Section 7.2). The argument by Todd in [55] is that context units are more interpretable than hidden units: "Since the hidden units typically compute some complicated, often uninterpretable function of their inputs, the memory kept in the context units will likely also be uninterpretable. This is in contrast to [this] design, where, as described earlier, each context unit keeps a memory of its corresponding output unit, which is interpretable."

[17] In practice, it is a scalar real value, e.g., 0.7, but Todd discusses his experiments with other possible encodings [55].

[18] Actually, as an optimization, Todd proposes in the following of his description to pass back the target values and not the output values.

[19] These and other types of architectures and generation strategies will be more systematically analyzed in Sections 5 and 7.

[20] An example is to condition the generation of a melody on a chord progression, in the MidiNet architecture [61] to be described in Section 9.4.

Figure 6: Examples of melodies generated by the Sequential architecture. (o_A and o_B) Original plan melodies learnt. (i_1) Melody generated by interpolating between o_A plan and o_B plan melodies. Inspired from [55]

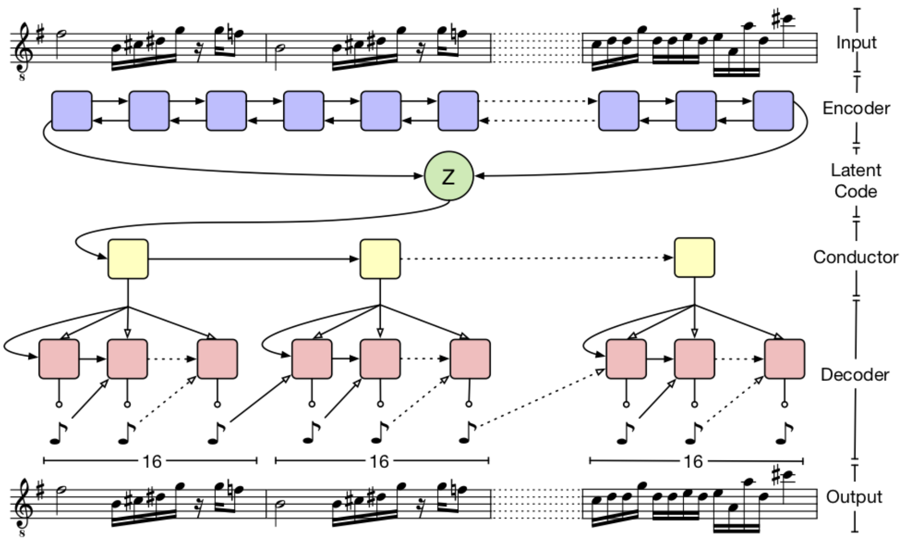

Figure 7: MusicVAE architecture. Reproduced from [49] with permission of the authors

- *structure and hierarchy* – "One of the largest problems with this sequential network approach is the limited length of sequences that can be learned and the corresponding lack of global structure that new compositions exhibit. Hierarchically organized and connected sets of sequential networks hold promise for addressing these difficulties. (...) One solution to these problems is first to take the sequence to be learned and divide it up intro appropriate chunks (...)."
- *multiple time/clocks* – "Of course, one way to present this subsequence-generating network with the appropriate sequence of plans is to generate *those* by another sequential network, operating at a slower time scale."

Thus, these early designs may be seen as precursors of some recent proposals:

- hierarchical architectures, such as MusicVAE [49] (shown in Figure 7 and analyzed in Section 9.3); and
- architectures with multiple time/clocks, such as Clockwork RNN [28] (shown in Figure 8) and SampleRNN [38].

4.2 Lewis' Creation by Refinement

In [32], Lewis introduced a novel way of creating melodies, that he named *creation by refinement (CBR)*, by "reverting" the standard way of using gradient descent for standard task – adjust the connexion *weights* to *minimize* the *classification error* –, into a very different task – adjust the *input* in order to make the *classification value* turn *positive*.

In his described initial experiment [32], the architecture is a conventional feedforward neural network architecture used for binary classification, to classify "well-formed" melodies. The input is a 5-note melody, each note being among the 7 notes (from C to B, without alteration).

For the training phase, Lewis manually constructed 30 examples of what he meant by "well-formed" melodies: using only the following intervals between notes: unison, 3rd and 5th; and also following some scale degree

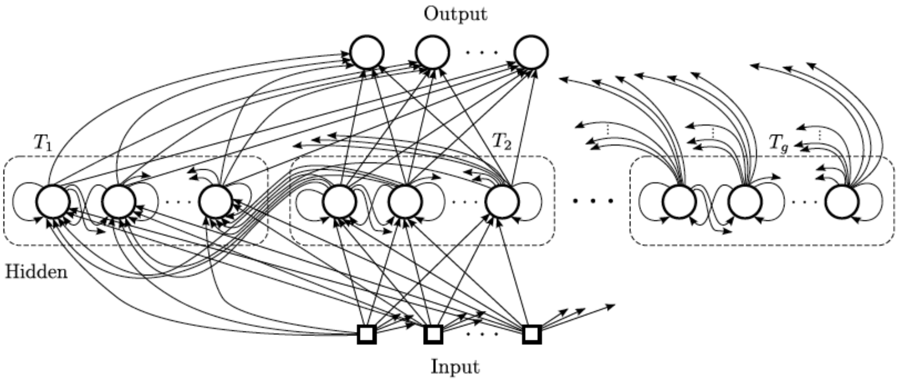

Figure 8: Clockwork RNN architecture. The RNN-like hidden layer is partitioned into several modules each with its own clock rate. Neurons in faster module i are connected to neurons in a slower module j only if a clock period $T_i < T_j$. Reproduced from [28] with permission of the authors

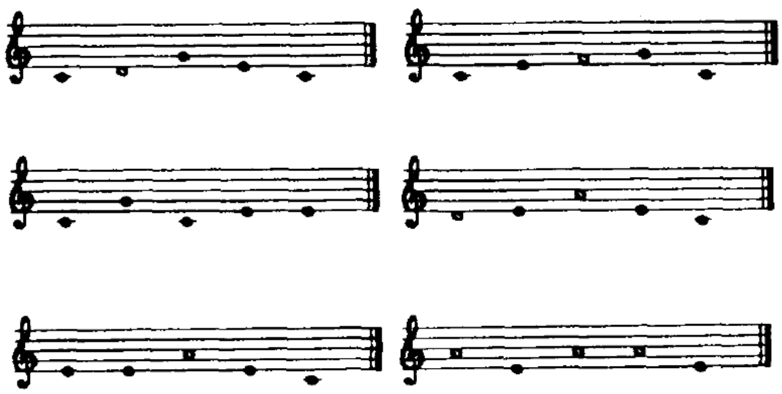

Figure 9: Creation by refinement. (left) Some "well formed" training examples. (right) Some examples of melodies generated Reproduced from [32] with permission of the author ©1988 IEEE

stepwise motion (some training examples are shown in the left part of the Figure 9). He also constructed examples of poorly-formed melodies, not respecting the principles above. The training phase of the network is therefore conventional, by training it with the positive (well-formed) and negative examples that have been constructed.

For the creation by refinement phase, a vector of random values is produced, as values of the input nodes of the network. Then, a gradient descent optimization is applied iteratively to *refine* these values[21] in order to maximize a positive classification (as shown in Figure 10). This will create a new melody which is classified as well-formed. The process may be done again, generating a new set of random values, and controlling their adjustment in order to create a new well-formed melody. Right part of Figure 9 shows some examples of generated melodies. Lewis interpretes the resulting creations as the fact that the network learned some preference for stepwise and triadic motion.

4.2.1 Influence

The approach of creation by refinement by Lewis in 1988 can be seen as the precursor of various approaches for controlling the creation of a content by *maximizing some target property*. Examples of target properties are:

- maximizing a *positive classification* (as a well-formed melody), in Lewis' original creation by refinement proposal [32];
- maximizing the *similarity* to a given *target*, in order to create a consonant melody, as in DeepHear [54];
- maximizing the *activation* of a specific *unit*, to amplify some visual element associated to this unit, as in Deep Dream [42];
- maximizing the *content similarity* to some initial image *and* the *style similarity* to a reference style image, to perform *style transfer* [12];

[21] Actually, in his article, Lewis does not detail the exact representation he uses (if he is using a one-hot encoding for each note) and the exact nature of refinement, i.e., adjustment of the values.

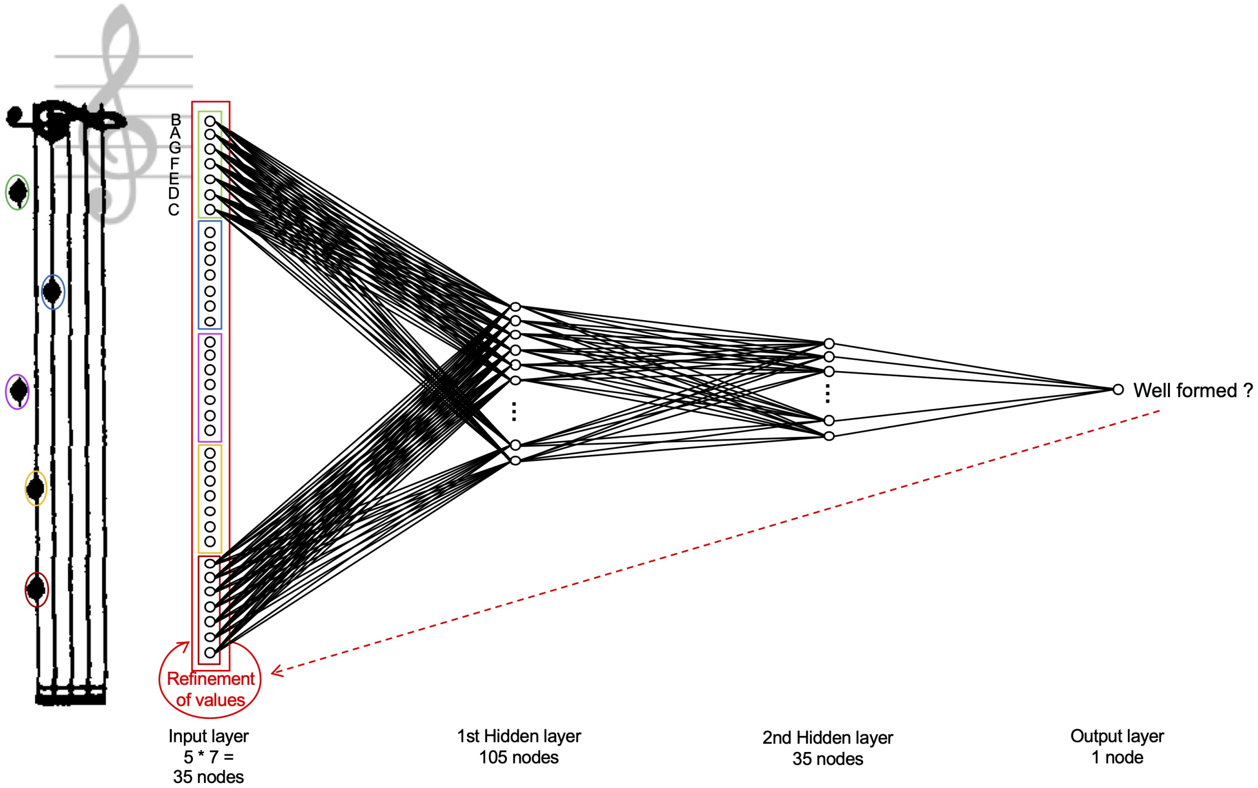

Figure 10: Creation by refinement – Architecture and strategy

- maximizing the *similarity* of the *structure* to some reference music, to perform *style imposition* [31], as will be detailed in Section 9.6.

Interestingly, this is done by reusing standard training mechanisms, namely *backpropagation* to compute the gradients, as well as *gradient descent* (or ascent) to minimize the cost (or to maximize the objective).

Furthermore, in his extended article [33], Lewis proposed a mechanism of *attention* and also of *hierarchy*: "In order to partition a large problem into manageable subproblems, we need to provide both an attention mechanism to select subproblems to present to the network and a context mechanism to tie the resulting subpatterns together into a coherent whole." and "The author's experiments have employed *hierarchical* CBR. In this approach, a developing pattern is recursively filled in using a scheme somewhat analogous to a formal grammar rule such as $ABC \to AxByC$. which expands the string without modifying existing tokens."

The idea of an *attention mechanism*, although not yet very developed, may be seen as a precursor of attention mechanisms in deep learning architectures: at first as an *additional* mechanism to focus on elements of an input sequence during the training phase [13, Section 12.4.5.1], notably for translation applications; until being proposed as the *fundamental and unique* mechanism (as a full alternative to recurrence or convolution) in the Transformer architecture [58], with its application to music generation, named MusicTransformer [25].

4.3 From Neural Networks to Deep Learning

With the third wave of artificial neural networks, named deep learning, experiments on music generation benefited from huge processing power, highly optimized implementations and availability of data, therefore allowing experiments at large or even very large scale. But, as we will see in Section 9, novel types of architectures have also been proposed. Before that, we will introduce a conceptual framework in order to help at organizing, analyzing and classifying the various types of architectures, as well as the various usages of artificial neural networks for music generation.

5 Conceptual Framework

This conceptual framework (initially proposed in [2]) is aimed at helping the analysis of the various perspectives (and elements) leading to the design of different deep learning-based music generation systems[22]. It includes: five main *dimensions* (and their facets) to characterize different ways of applying deep learning techniques to generate musical content, and the associated *typologies* for each dimension. In this article, we will simplify the presentation and focus on the most important aspects.

5.1 The 5 Dimensions

- *Objective*: the nature of the musical content to be generated, as well at its destination and use. Examples are: melody, polyphony, accompaniment; in the form of a musical score to be performed by some human musician(s) or an audio file to be played.
- *Representation*: the nature, format and encoding of the information (examples of music) used to train and to generate musical content. Examples are: signal, transformed signal (e.g., a spectrum, via a Fourier transform), piano roll, MIDI, text; encoded in scalar variables or/and in one-hot vectors.
- *Architecture*: the nature of the assemblage of processing units (the artificial neurons) and their connexions. Examples are: feedforward, recurrent, autoencoder, generative adversarial networks.
- *Requirement*: one of the qualities that may be desired for music generation. Some are easier to achieve, e.g., content or length variability, and some other ones are deeper challenges [3], e.g., control, creativity or structure.
- *Strategy*: the way the architecture will process representations in order to generate[23] the objective while matching desired requirements. Examples are: single-step feedforward, iterative feedforward, decoder feedforward, sampling, input manipulation.

Note that these five dimensions are not orthogonal. The exploration of these five different dimensions and of their interplay is actually at the core of our analysis.

5.2 The Basic Generation Steps

The basic steps for generating music, according to the *objective*, are as follows:

1. select (curate) a *corpus* (a set of *training examples*, representative of the *style* to be learnt);
2. select a type of *representation* and a type(s) of *encoding* and apply them to the *examples*;
3. select a type(s) of *architecture* and *configurate* it;
4. *train* the *architecture* with the *examples*;
5. select a type(s) of *strategy* for *generation* and apply it to *generate* one or various musical contents, and *decode* them into music;
6. select the *preferred* one(s) among the musics *generated*.

6 Representation

The choice of representation and its encoding is tightly connected to the configuration of the input and the output of the architecture, i.e. the number of input and output nodes (variables). Although a deep learning architecture can automatically extract significant features from the data, the choice of representation may be significant for the accuracy of the learning and for the quality of the generated content.

[22] *Systems* refers to the various proposals (architectures, systems and experiments) about deep learning-based music generation surveyed from the literature in [2].

[23] It is important to highlight that, in this conceptual framework, by strategy we only consider the *generation strategy*, i.e., the strategy to generate musical content. A strategy for training an architecture could be quite different and is out of direct concern in this classification.

6.1 Phases and Types of Data

Before getting into the choices of representation to be processed by a deep learning architecture, it is important to identify the main types of data to be considered, depending on the phase (training or generation):

- *training data*, the examples used as input for the training;
- *generation (input) data*, used as input for the generation (e.g., a melody for which an accompaniment will be generated, as in the first example in Section 3); and
- *generated (output) data*, produced by the generation (e.g., the accompaniment generated), as specified by the objective.

Depending on the objective, these two types of data may be equal or different, e.g., in Section 3, the generation data is a melody and the generated data is a set of (3) melodies.

6.2 Format

The format is the nature of the representation of a piece of music to be interpreted by a computer. A big divide in terms of the choice of representation is *audio* versus *symbolic*. This corresponds to the divide between *continuous* and *discrete* variables. Their respective raw material is very different in nature, as are the types of techniques for possible processing and transformation of the initial representation[24]. However, the actual processing of these two main types of representation by a deep learning architecture is basically the *same*[25].

6.2.1 Audio

The main audio formats used are:

- *signal waveform*,
- *spectrum*, obtained via a *Fourier transform*[26].

The advantage of waveform is in considering the raw material untransformed, with its full initial resolution. Architectures that process the raw signal are sometimes named *end-to-end* architectures. The disadvantage is in the computational load: low level raw signal is demanding in terms of both memory and processing. The WaveNet architecture [44], used for speech generation for the Google assistants, was the first to prove the feasibility of such architectures.

6.2.2 Symbolic

The main symbolic formats used are:

- *MIDI*[27] – It it is a technical standard that describes a protocol based on events, a digital interface and connectors for interoperability between various electronic musical instruments, softwares and devices [40]. Two types of MIDI event messages are considered for expressing note occurrence: `Note on` and `Note off`, to indicate, respectively, the start and the end of a note played. The MIDI *note number*, indicates the note *pitch*, specified by an integer within 0 and 127. Each note event is embedded into a data structure containing a delta-time value which also specifies the timing information, specified as a *relative time* (number of periodic ticks from the beginning – for musical scores) or as an *absolute time* (in the case of *real performances*[28]).
- *Piano roll* – It is inspired from automated mechanical pianos with a continuous roll of paper with perforations (holes) punched into it. It is a two dimensional table with the x axis representing the successive time steps and the y axis the pitch, as shown in Figure 11.

[24] In fact, they correspond to different scientific and technical communities, namely *signal processing* and *knowledge representation*.

[25] Indeed, at the level of processing by a deep network architecture, the initial distinction between audio and symbolic representation boils down, as only *numerical* values and operations are considered.

[26] The objective of the Fourier transform (which could be continuous or discrete) is the decomposition of an arbitrary signal into its elementary components (sinusoidal waveforms). As well as compressing the information, its role is fundamental for musical purposes as it reveals the *harmonic* components of the signal.

[27] Acronym of Musical Instrument Digital Interface.

[28] The volume may also be specified.

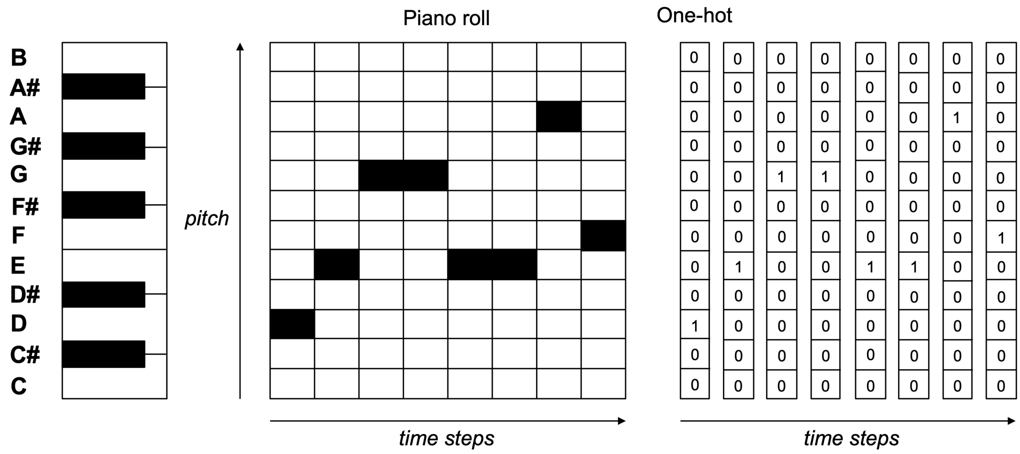

Figure 11: Example of piano roll and corresponding one-hot encoding

```
X: 1
T: A Cup Of Tea
R: reel
M: 4/4
L: 1/8
K: Amix
|:eA (3AAA g2 fg|eA (3AAA BGGf|eA (3AAA g2 fg|1afge d2 gf:
|2afge d2 cd|| |:eaag efgf|eaag edBd|eaag efge|afge dgfg:|
```

Figure 12: ABC notation of "A Cup of Tea". The first six lines are the header and represent *metadata*: T(itle), M(eter), default note L(ength), K(ey), etc. Reproduced from The Session [26] with permission of the manager

- *Text* – A significant example is the ABC notation [59], a *de facto* standard for folk and traditional music[29]. Each note is encoded as a token, the pitch class of a note being encoded as the letter corresponding to its English notation (e.g., `A` for A or La), with extra notations for the octave (e.g., `a'` means two octaves up) and for the duration (e.g., `A2` means a double duration). Measures are separated by "`|`" (bars), as in conventional scores. An example of ABC score is shown in Figure 12.

Note that in these three cases, except for the case of real performances recorded in MIDI, a *global time step* has to be fixed and usually corresponds, as stated by Todd in [55], to the greatest common factor of the durations of all the notes to be learned.

Note that each format has its pros and cons. The ABC notation is very compact but can only represent monophonic melodies. In [86], Huang and Hu claim that one drawback of encoding MIDI messages directly is that it does not effectively preserve the notion of multiple notes being played at once through the use of multiple tracks. In practice, the piano roll is one of the most commonly used representations, although it has some limitations. An important one, compared to MIDI representation, is that there is no note off information. As a result, there is no way to distinguish between a long note and a repeated short note[30]. Main possible approaches for resolving this are:

- to introduce a *hold/replay* representation, as a dual representation of the sequence of notes (used in the DeepJ system [34]);
- to divide the size of the time step by two and always mark a *note ending* with a special tag (used in [7]);
- to divide the size of the time step as before but instead mark a *new note beginning* (used by Todd in [55], see Section 4.1); and
- to use a special *hold* symbol "__" in place of a note to specify when the previous note is held (used in DeepBach [18], see Section 9.5).

The last solution, considering the hold symbol as a note, is simple and uniform but it only applies to the case of a monophonic melody.

[29] Note that the ABC notation has been designed *independently* of computer music and machine learning concerns.

[30] Actually, in the original mechanical paper piano roll, the distinction is made: two holes are different from a longer single hole. The end of the hole is the encoding of the end of the note.

6.3 Encoding

Once the format of a representation has been chosen, the issue still remains of how to *encode* this representation. The *encoding* of a representation (of a musical content) consists in the *mapping* of the representation (composed of a set of *variables*, e.g., pitch or dynamics) into a set of *inputs* (also named *input nodes* or *input variables*) for the neural network architecture.

There are two basic approaches:

- *value-encoding* – A continuous, discrete or boolean variable is directly encoded as a *scalar*; and
- *one-hot-encoding* – A discrete or a categorical variable is encoded as a *categorical variable* through a vector with the number of all possible elements as its length. Then, to represent a given element, the corresponding element of the *one-hot vector*[31] is set to 1 and all other elements to 0.

For instance, the pitch of a note could be represented as a *real number* (its frequency in Hertz), an *integer number* (its MIDI note number), or a *one-hot vector* (actually the most common strategy), as shown in the right part of Figure 11[32]. The advantage of value encoding is its compact representation, at the cost of sensibility because of numerical operations (approximations). The advantage of one-hot encoding is its robustness against numerical operations approximations (discrete versus analog), at the cost of a high cardinality and therefore a potentially large number of nodes for the architecture.

7 Main Basic Architectures and Strategies

For reasons of space limitation, we will now jointly introduce architectures and strategies[33]. For an alternative analysis guided by requirements (challenges), please see [3].

7.1 Feedforward Architecture

The *feedforward architecture*[34] is the *vanilla* type (most basic and very common) of artificial neural networks architectures.

7.1.1 Feedforward Strategy

An example of use has been detailed in Section 3. The generation strategy used in this example is also the *vanilla type* of strategy, as it consists in *feedforwarding* within a single step the input data into the input layer, through successive hidden layers, until the output layer. Therefore we name it the *single-step feedforward strategy*, abbreviated as *feedforward strategy.*

7.1.2 Iterative Strategy

Although feedforward architecture and single-step feedforward strategy are naturally associated, in Todd's Time-Windowed architecture in Section 4.1, generation is processed iteratively by feedforwarding current melody segment in order to obtain next one, and so on. Therefore we name it the *iterative feedforward strategy*, abbreviated as *iterative strategy.*

7.2 Recurrent Architecture

A *recurrent neural network* (RNN) is a feedforward neural network extended with *recurrent connexions* in order to learn series of items (e.g., a melody as a sequence of notes). Todd's Sequential architecture in Section 4.1 is an example although not of a common type. As pointed out in Section 4.1, in modern recurrent architectures, recurrent connexions are encapsulated within the hidden layer, which allows an arbitrary number of recurrent layers (as shown in Figure 13).

[31] The name comes from digital circuits, *one-hot* referring to a group of bits among which the only legal (possible) combinations of values are those with a single *high* (hot!) (1) bit, all the others being *low* (0).

[32] The Figure also illustrates that a piano roll could be straightforwardly encoded as a sequence of one-hot vectors to construct the input representation of an architecture, as, e.g., has been shown in Figure 2.

[33] As a reminder from Section 5.1, we only consider here *generation strategies.*

[34] Also named *multilayer neural network* (MLP).

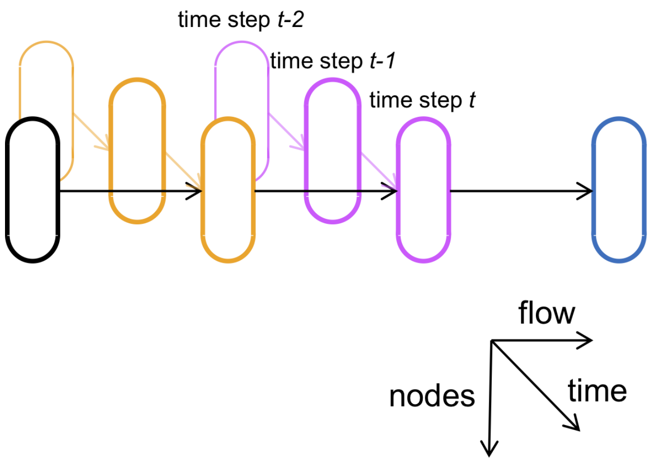

Figure 13: Recurrent neural network. Each successive layer (along the flow of computation) is represented as an oblong design (hiding the detail of its nodes). The diagonal axis represents the time dimension, with the previous step value of each layer in thinner and lighter color

7.2.1 Recursive Strategy

The first music generation experiment using current state of the art of recurrent architectures, the LSTM (Long Short-Term Memory [22]) architecture, is the generation of blues chord (and melody) sequences by Eck and Schmidhuber in [7]. Another interesting example is the architecture by Sturm *et al.* to generate Celtic melodies [53]. It is trained on examples selected from the folk music repository named The Session [26] and uses text (the ABC notation [59], see Section 6.2.2) as the representation format. Generation (an example is shown in Figure 14) is done using a *recursive strategy*, a special case of iterative strategy, for generating a sequence of notes (or/and chords), as initially described for text generation by Graves in [15]:

- select some *seed* information as the *first* item (e.g., the first note of a melody);
- *feedforward* it into the recurrent network in order to produce the *next* item (e.g., next note);
- use this next item as the next input to produce the *next next* item; and
- repeat this process iteratively until a *sequence* (e.g., of notes, i.e. a melody) of the desired length is produced[35].

7.2.2 Sample Strategy

A limitation of applying straightforwardly the iterative feedforward strategy on a recurrent network is that generation is *deterministic*[36]. As a consequence, feedforwarding the *same input* will always produce the *same output*. As the generation of the next note, the next next note, etc., is deterministic, the *same* seed note will lead to the *same* generated series of notes[37]. Moreover, as there are only 12 possible input values (the 12 pitch classes, disregarding the possible octaves), there are only 12 possible melodies.

Fortunately, the solution is quite simple. The assumption is that generation is modeled as a classification task, i.e., the output representation of the melody is one-hot encoded and the output layer activation function is softmax. See an example in Figure 15, where $P(\mathrm{x}_t = \mathrm{C}|\mathrm{x}_{<t})$ represents the conditional probability for the element (pitch of the note) x_t at step t to be a C given the previous elements $\mathrm{x}_{<t}$ (the melody generated so far). The default *deterministic* strategy consists in choosing the pitch with the highest probability, i.e. $\mathrm{argmax}_{\mathrm{x}_t} P(\mathrm{x}_t|\mathrm{x}_{<t})$ (that is G♯ in Figure 15). We can then easily switch to a *nondeterministic* strategy, by *sampling*[38] the output which corresponds (through the softmax function) to a probability distribution between possible pitches. By sampling a pitch following the distribution generated recursively by the architecture[39], we introduce stochasticity in the process of generation and thus *content variability* in the generation.

[35] Note that, as opposed to feedforward strategy (and decoder feedforward strategy, to be introduced in Section 9.1.1), iterative and recursive strategies allow the generation of musical content of *arbitrary length*.

[36] Indeed, most artificial neural networks are deterministic. There are stochastic versions of artificial neural networks – the Restricted Boltzmann Machine (RBM) [21] is an example – but they are not mainstream. An example of use of RBM will be introduced in Section 9.6.

[37] The actual length of the melody generated depending on the number of iterations.

[38] Sampling is the action of generating an element (a *sample*) from a *stochastic* model according to a *probability distribution*.

[39] The chance of sampling a given pitch is its corresponding probability. In the example shown in Figure 15, G♯ has around one chance in two of being selected and A♯ one chance in four.

Figure 14: Score of "The Mal's Copporim" automatically generated. Reproduced from [53] with permission of the authors

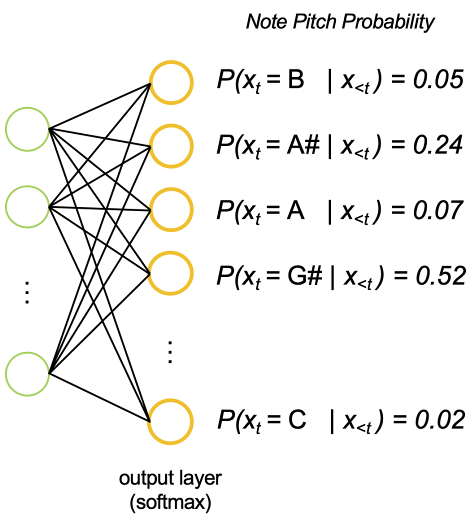

Figure 15: A softmax output layer computes the probability for each pitch

8 Compound Architectures

For more sophisticated objectives and requirements, *compound* architectures may be used. We will see that, from an architectural point of view, various types of composition[40] may be used:

8.1 Composition

Several architectures, of the same type or of different types, are combined, e.g.:

- a bidirectional RNN, combining two RNNs, forward and backward in time, e.g., as used in the C-RNN-GAN [41] (see Figure 16) and the MusicVAE [49] (see Figure 7 and Section 9.3) architectures; and
- the RNN-RBM architecture [1], combining an RNN architecture and an RBM architecture.

8.2 Refinement

One architecture is refined and specialized through some additional constraint(s), e.g.:

- an autoencoder architecture (to be introduced in Section 9.1), which is a feedforward architecture with one hidden layer with the same cardinality (number of nodes) for the input layer and the output layer; and
- a variational autoencoder (VAE) architecture, which is an autoencoder with an additional constraint on the distribution of the variables of the hidden layer (see Section 9.2), e.g., the GLSR-VAE architecture [17].

8.3 Nesting

An architecture is nested into the other one, e.g.:

- a stacked autoencoder architecture[41], e.g., the DeepHear architecture [54]; and
- a recurrent autoencoder architecture (Section 9.3), where an RNN architecture is nested within an autoencoder[42], e.g., the MusicVAE architecture [49] (see Section 9.3).

8.4 Pattern

An architectural pattern is instantiated onto a given architecture(s)[43], e.g.:

- the Anticipation-RNN architecture [16], that instantiates the *conditioning* pattern[44] onto an RNN with the output of another RNN as the conditioning input; and
- the C-RNN-GAN architecture [41], where the *GAN (Generative Adversarial Networks)* pattern (to be introduced in Section 9.4) is instantiated onto two RNN architectures, the second one (discriminator) being bidirectional (see Figure 16); and
- the MidiNet architecture [61] (see Section 9.4), where the *GAN* pattern is instantiated onto two convolutional[45] feedforward architectures, on which a *conditional* pattern is instantiated.

8.5 Classification

Figure 17 illustrates various examples of compound architectures[46] and of actual music generation systems.

[40] We are taking inspiration from concepts and terminology in programming languages and software architectures [52], such as *refinement*, *instantiation*, *nesting* and *pattern* [11].

[41] A *stacked autoencoder* is a hierarchical nesting of autoencoders with decreasing number of hidden layer units, as shown in right part of Figure 18.

[42] More precisely, an RNN is nested within the encoder and another RNN within the decoder. Therefore, it is also named an RNN Encoder-Decoder architecture.

[43] Note that we limit here the scope of a pattern to the *external enfolding* of an existing architecture. Additionally, we could have considered convolutional, autoencoder and even recurrent architectures as an *internal* architectural pattern.

[44] Such as introduced by Todd in his Sequential architecture conditioned by a plan in Section 4.1.

[45] Convolutional architectures are actually an important component of the current success of deep learning and they recently emerged as an alternative, more efficient to train, to recurrent architectures [2, Section 8.2]. A *convolutional* architecture is composed of a succession of feature maps and pooling layers [13, Section 9][2, Section 5.9]. (We could have considered convolutional as an internal architectural pattern, as has just been remarked in a previous footnote.) However, we do not detail convolutional architectures here, because of space limitation and of non specificity to music generation applications.

[46] GAN (Generative Adversarial Networks) pattern and VRAE (Variational Recurrent Autoencoder) architecture will be introduced in Sections 9.4 and 9.3, respectively.

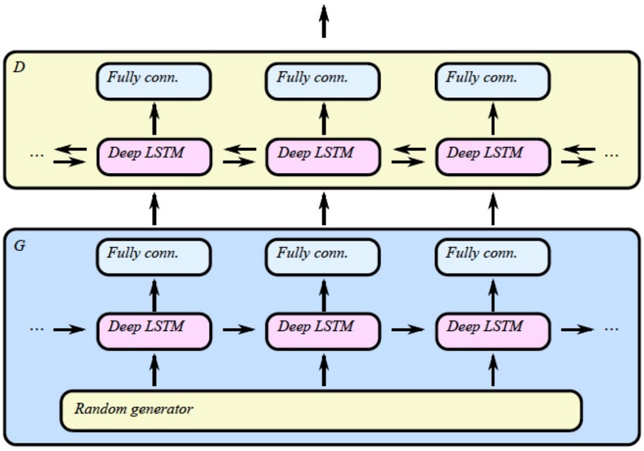

Figure 16: C-RNN-GAN architecture with the D(iscriminator) GAN component being a bidirectional RNN (LSTM). Reproduced from [41] with permission of the authors

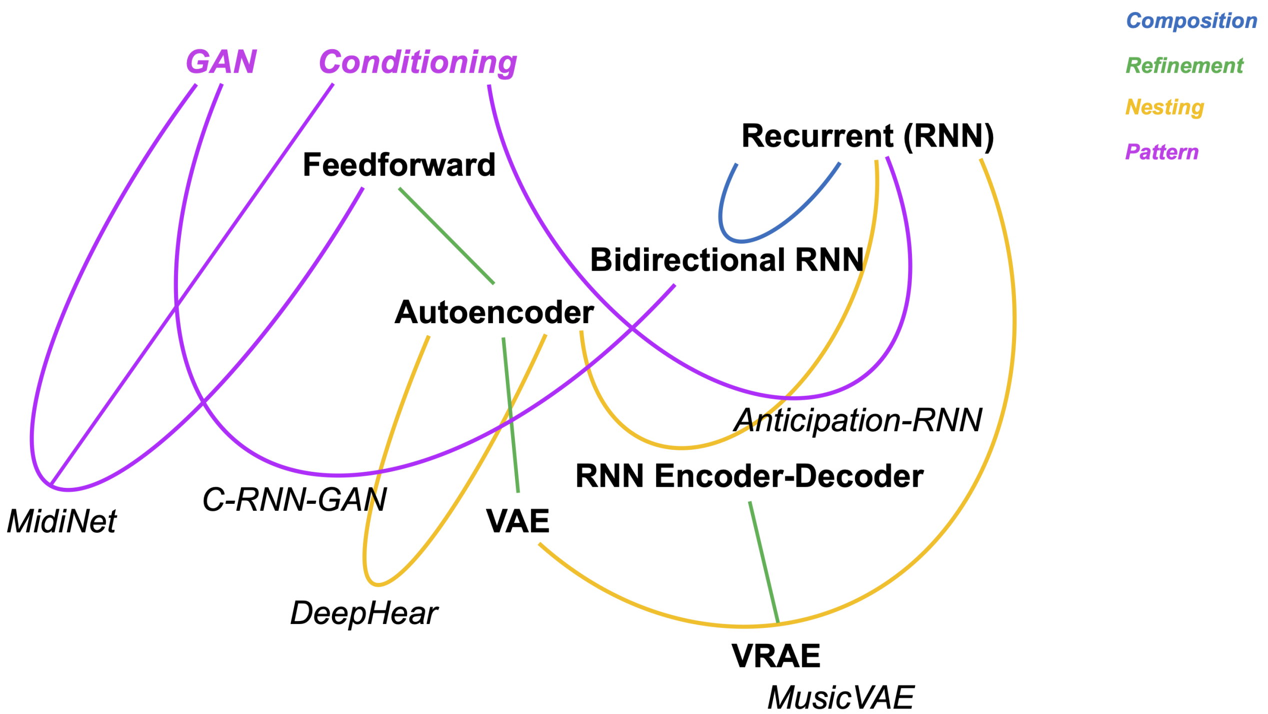

Figure 17: A tentative illustration of various examples of compound architectures and systems (actual music generation systems are in italics)

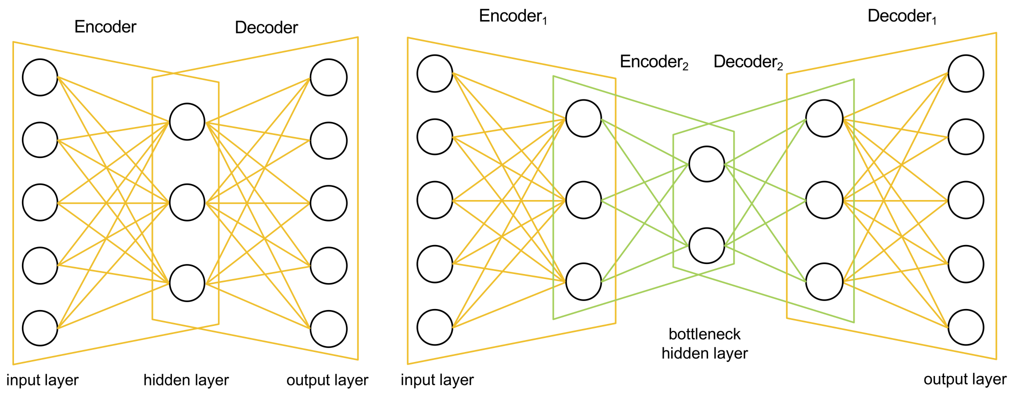

Figure 18: (left) Autoencoder architecture. (right) Stacked autoencoder (order-2) architecture

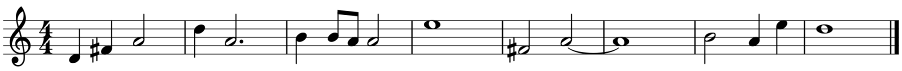

Figure 19: Example of melody generated by an autoencoder trained on a Celtic melodies corpus

8.6 Combined Strategies

Note that the strategies for generation can be combined too, although not in the same way as the architectures: they are actually used simultaneously on different components of the architecture. In the examples discussed in Section 7.2.2, the *recursive strategy* is used by recursively feedforwarding current note into the architecture in order to produce next note and so on, while the *sampling strategy* is used at the output of the architecture to sample the actual note (pitch) from the possible notes with their respective probabilities.

9 Examples of Refined Architectures and Strategies

9.1 Autoencoder Architecture

An *autoencoder* is a feedforward neural network with one hidden layer and with an additional constraint: the number of output nodes is equal to the number of input nodes. The output layer actually *mirrors* the input layer, creating its peculiar symmetric diabolo (or sand-timer) shape aspect, as shown in the left part of Figure 18.

An autoencoder is trained with each of the examples as the input and as the output. Thus, the autoencoder tries to learn the identity function. As the hidden layer usually has fewer nodes than the input layer, the *encoder* component must *compress* information[47], while the *decoder* has to *reconstruct*, as accurately as possible, the initial information. This forces the autoencoder to *discover* significant (discriminating) *features* to encode useful information into the hidden layer nodes (also named the *latent variables*).

9.1.1 Decoder Feedforward Strategy

The latent variables of an autoencoder constitute a compact representation of the common features of the learnt examples. By instantiating these latent variables and decoding them (by feedforwarding them into the decoder), we can generate a new musical content corresponding to the values of the latent variables, in the same format as the training examples. We name this strategy the *decoder feedforward strategy*. An example generated after training an autoencoder on a set of Celtic melodies (selected from the folk music repository The Session [26], introduced in Section 7.2.1) is shown in Figure 19. An early example of this strategy is the use of the DeepHear nested (stacked) autoencoder architecture to generate ragtime music according to the style learnt [54].

9.2 Variational Autoencoder Architecture

A *variational autoencoder* (VAE) [27] is a refinement of an autoencoder, with the added constraint that the encoded representation (the latent variables) follow some prior probability distribution, usually a Gaussian distribution. This constraint is implemented by adding a specific term to the cost function, by computing the cross-entropy between the values of the latent variables and the prior distribution[48].

[47] Compared to traditional dimension reduction algorithms, such as principal component analysis (PCA), feature extraction is nonlinear, but it does not ensure orthogonality of the dimensions, as we will see in Section 9.2.2.

[48] The actual implementation is more refined: the encoder actually generates a mean vector and a standard deviation vector, from which latent variables are sampled [27].

As with an autoencoder, a VAE will learn the identity function, but furthermore the decoder will learn the relation between the prior (Gaussian) distribution of the latent variables and the learnt examples. A very interesting characteristic for generation purposes is in the *meaningful exploration* of the latent space, as a variational autoencoder is able to learn a "smooth"[49] latent space *mapping* to realistic examples.

9.2.1 Variational Generation

Examples of possible dimensions captured by latent variables learnt by the VAE are the note duration range (the distance between shortest and longest note) and the note pitch range (the distance between lowest and highest pitch). This latent representation (vector of latent variables) can be used to explore the latent space with various operations to control/vary the generation of content. Some examples of operations on the latent space (as summarized in [49]) are:

- *translation*;
- *interpolation*[50];
- *averaging*;
- *attribute vector arithmetics*, by addition or subtraction of an attribute vector capturing a given characteristic[51].

9.2.2 Disentanglement

One limitation of using a variational autoencoder is that the dimensions (captured by latent variables) are not independent (orthogonal), as in the case of Principal component analysis (PCA). However, various techniques are being recently proposed to improve the disentanglement of the dimensions (see, e.g., [35]).

Another issue is that the semantics (meaning) of the dimensions captured by the latent variables is automatically "chosen" by the VAE architecture in function of the training examples and the configuration and thus can only be interpreted *a posteriori*. However, some recent approaches propose to "force" the meaning of latent variables, by splitting the decoder into various components and training them onto a specific dimension (e.g., rhythm or pitch melody) [62].

9.3 Variational Recurrent Autoencoder (VRAE) Architecture

An interesting example of nested architecture (see Section 8.3) is a variational recurrent autoencoder (VRAE). The motivation is to combine:

- the *variational* property of the VAE architecture for controlling the generation; and
- the *arbitrary length* property of the RNN architecture used with the recursive strategy.

An example (also hierarchical) is the MusicVAE architecture [49] (shown in Figure 7, with an example of controlled generation in Figure 21).

9.4 Generative Adversarial Networks (GAN) Architecture

An interesting example of architectural pattern is the concept of Generative Adversarial Networks (GAN) [14], as illustrated in Figure 22. The idea is to simultaneously train two neural networks:

- a *generative model* (or *generator*) G, whose objective is to transform a random noise vector into a synthetic (faked) *sample*, which resembles real samples drawn from a distribution of real content (images, melodies...); and
- a *discriminative model* (or *discriminator*) D, which estimates the probability that a sample came from the real data rather than from the generator G.

The generator is then able to produce user-appealing synthetic samples from noise vectors.

An example of the use of GAN for generating music is the MidiNet system [61] aimed at generation of single or multitrack pop music melodies. The architecture, illustrated in Figure 23, follows two patterns: adversarial (GAN) and conditional (on history and on chords to condition melody generation)[52]. It is also convolutional

[49] That is, a small change in the latent space will correspond to a small change in the generated examples, without any discontinuity or jump (see more details in [60]).

[50] The interpolation in the latent space produces more meaningful and interesting melodies than the interpolation in the data space (which basically just varies the ratio of notes from the two melodies) [50], as shown in Figure 20.

[51] This attribute vector is computed as the average latent vector for a collection of examples sharing that attribute (characteristic), e.g., high density of notes (see an example in Figure 21), rapid change, high register, etc.

[52] Please refer to [61] or [2, Section 6.10.3.3] for more details about this sophisticated architecture.

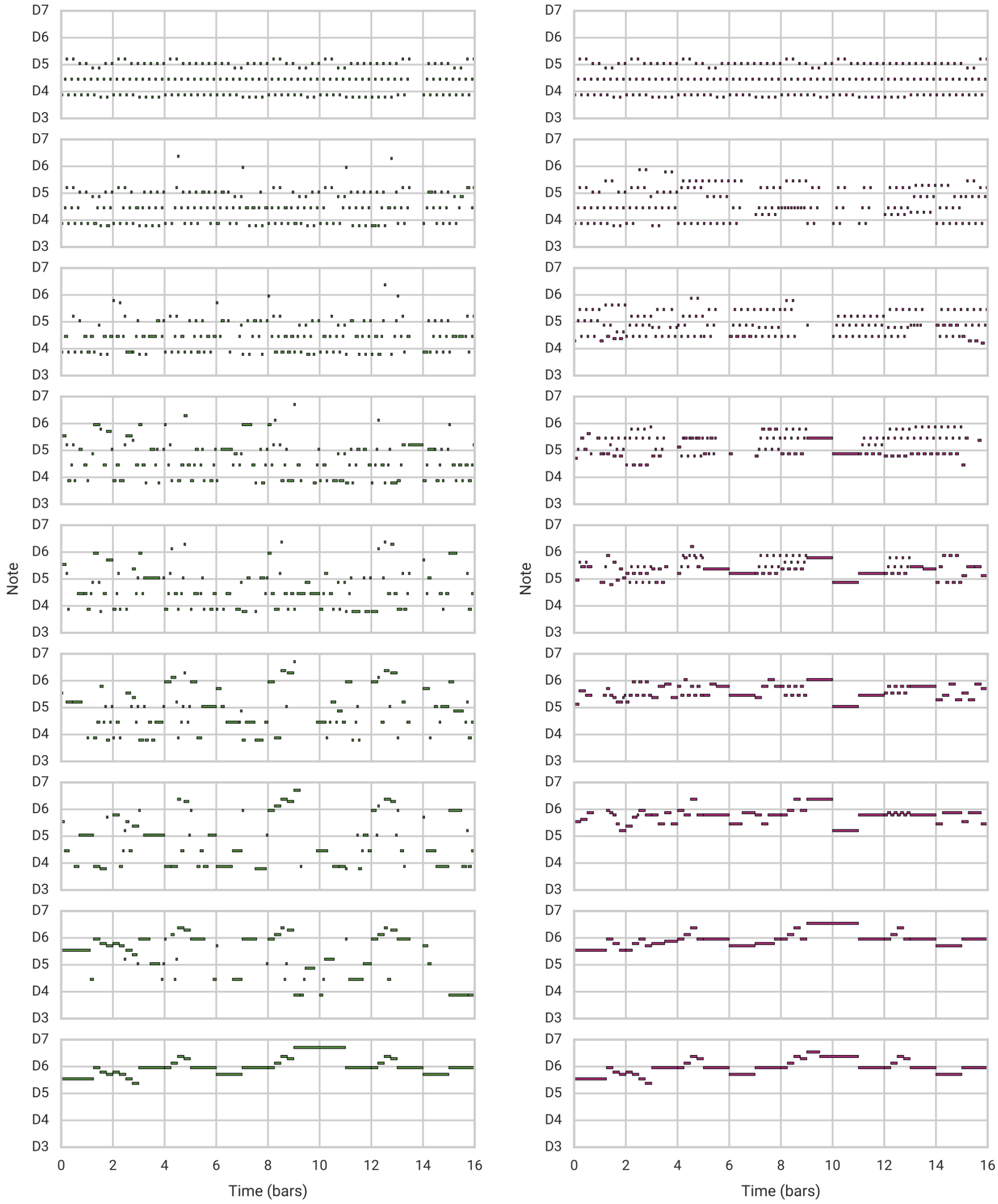

Figure 20: Comparison of interpolations between the top and the bottom melodies by (left) interpolating in the data (melody) space and (right) interpolating in the latent space and decoding it into melodies. Reproduced from [50] with permission of the authors

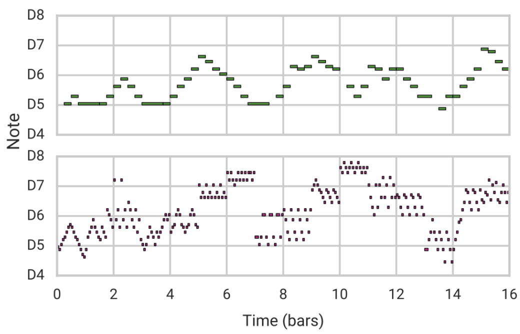

Figure 21: Example of a melody generated (bottom) by MusicVAE by adding a "high note density" attribute vector to the latent space of an existing melody (top). Reproduced from [50] with permission of the authors

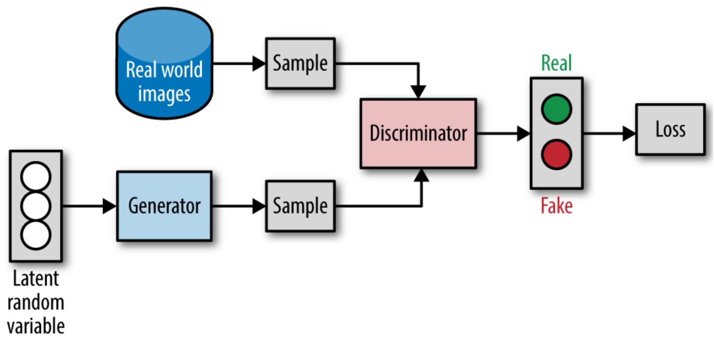

Figure 22: Generative adversarial networks (GAN) architecture. Reproduced from [48] with permission of O'Reilly Media

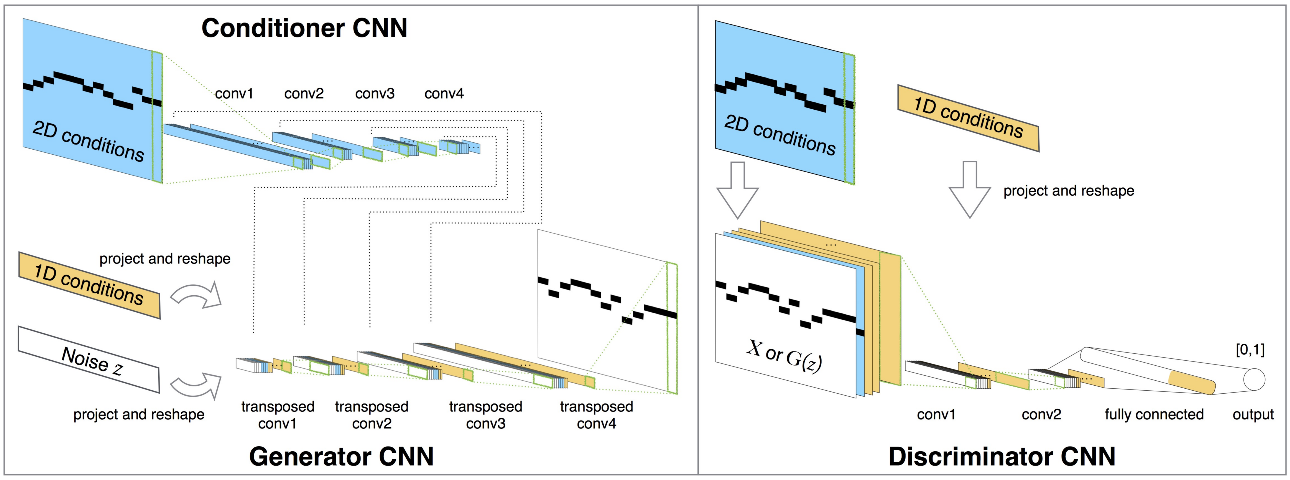

Figure 23: MidiNet architecture. Reproduced from [61] with permission of the authors

(both the generator and the discriminator are convolutional networks). The representation chosen is obtained by transforming each channel of MIDI files into a one-hot encoding of 8 measures long piano roll representations. Generation takes place following an iterative strategy, by sampling one measure after one measure until reaching 8 measures. An example of generation is shown in Figure 24.

9.5 Sampling Strategy

We have discussed in the Section 7.2.2 the use of sampling at the output of a recurrent network in order to ensure content variability. But sampling could also be used as the principal strategy for generation, as we will see in the two following examples. The idea is to consider *incremental variable instantiation*, where a global representation is incrementally instantiated by progressively refining values of variables (e.g., pitch and duration of notes). The main advantage is that it is possible to generate or to *regenerate* only an *arbitrary part* of the musical content, for a specific *time interval* and/or for a specific *subset of tracks/voices*, without regenerating the whole content.

This incremental instantiation strategy has been used in the DeepBach architecture [18] for generation of Bach chorales. The compound architecture[53], shown at Figure 25, combines two recurrent and two feedforward networks. As opposed to standard use of recurrent networks, where a single time direction is considered, DeepBach architecture considers the two directions *forward* in time and *backwards* in time. Therefore, two

[53] Actually this architecture is replicated 4 times, one for each voice (4 in a chorale).

Figure 24: Example of melody and chords generated by MidiNet. Reproduced from [61] with permission of the authors

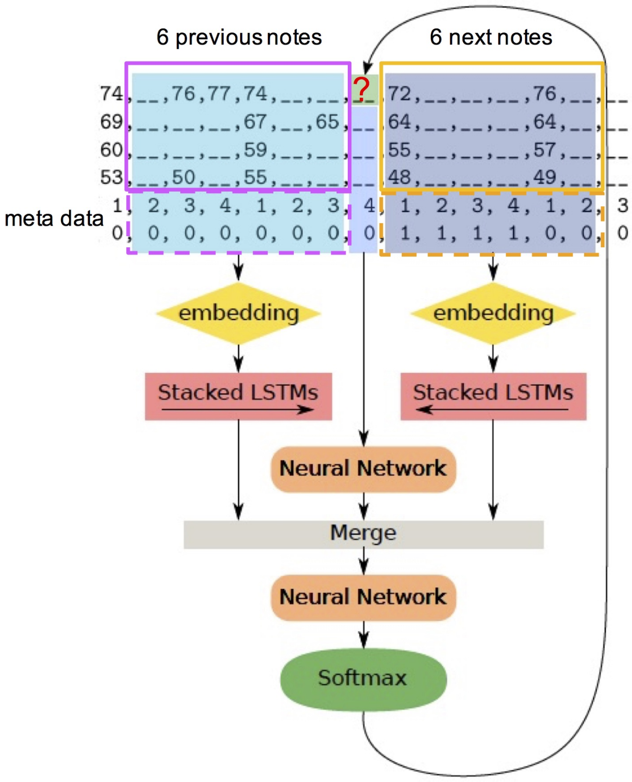

Figure 25: DeepBach architecture for the soprano voice prediction

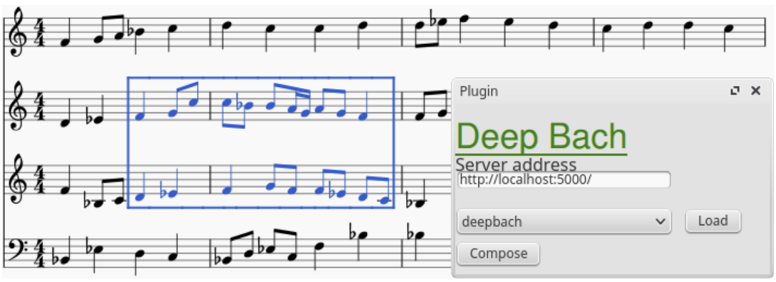

Figure 26: DeepBach user interface. Selecting an interval and two voices (alto and tenor, to be the only ones) to be regenerated

recurrent networks (more precisely, LSTM) are used, one summing up past information and another summing up information coming from the future, together with a non recurrent network for notes occurring at the same time. Their three outputs are merged and passed as the input of a final feedforward neural network whose output is the estimated distribution for all notes time slices for a given voice. The first 4 lines[54] of the example data on top of the Figure 25 correspond to the 4 voices.

Training, as well as generation, is not done in the conventional way for neural networks. The objective is to predict the value of current note for a a given voice (shown with a red ? on top center of Figure 25), using as information surrounding contextual notes. The training set is formed on-line by repeatedly randomly selecting a note in a voice from an example of the corpus and its surrounding context. Generation is done by sampling, using a pseudo-Gibbs sampling incremental and iterative algorithm (see details in [18]) to produce a set of values (each note) of a polyphony, following the distribution that the network has learnt.

The advantage of this method is that generation may be tailored. For example, if the user changes only one or two measures of the soprano voice, he can resample only some corresponding counterpoint voices for these measures, as shown in Figure 26.

Coconet [24], the architecture used for implementing the Bach Doodle (introduced in Section 3), is another example of this approach. It uses a Block Gibbs sampling algorithm for generation and a different architecture (using masks to indicate for each time slice whether the pitch for that voice is known, see Figure 27). Please refer to [24] and [23] for details. An example of counterpoint accompaniment generation has been shown in

[54]The two bottom lines correspond to metadata (fermata and beat information), not detailed here.

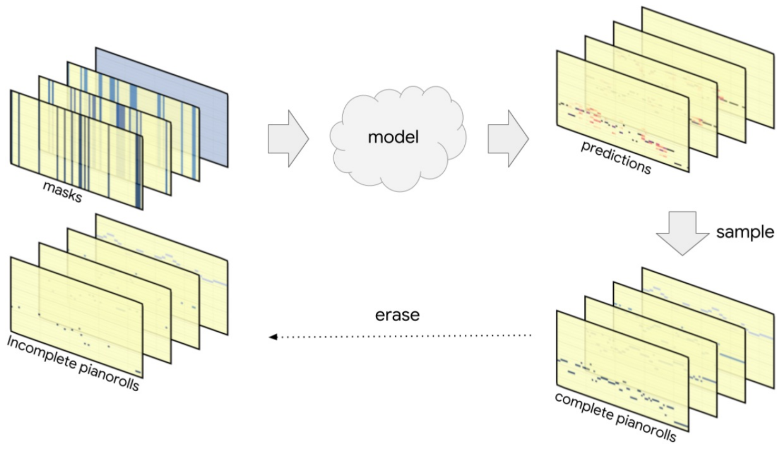

Figure 27: Coconet Architecture. Reproduced from [23] with permission of the authors

Figure 1.

9.6 Creation by Refinement Strategy

Lewis' creation by refinement strategy has been introduced in Section 4.2. It has been reinvented by various systems such as Deep Dream and DeepHear, as discussed in Section 4.2.1.

An example of application to music is the generation algorithm for the C-RBM architecture [30]. The architecture is a refined (convolutional[55]) restricted Boltzmann machine (RBM[56]). It is trained to learn the *local structure* (musical texture/style) of a corpus of music (in this case, Mozart sonatas). The main idea is to impose onto a generated musical content some *global structure* seen as a *structural template* from an existing reference musical piece[57]. The global structure is expressed through three types of constraints:

- *self-similarity*, to specify a *global structure* (e.g., AABA) in the generated music piece. This is modeled by minimizing the distance between the self-similarity matrices of the reference target and of the intermediate solution;
- *tonality constraint*, to specify a *key* (tonality). To estimate the key in a given temporal window, the distribution of pitch classes is compared with the key profiles of the reference; and
- *meter constraint*, to impose a specific *meter* (also named a *time signature*, e.g., 4/4) and its related rhythmic pattern (e.g., accent on the third beat). The relative occurrence of note onsets within a measure is constrained to follow that of the reference.

Generation is performed via *constrained sampling*, a mechanism to restrict the set of possible solutions in the sampling process according to some pre-defined constraints. The principle of the process (illustrated at Figure 28) is as follows. At first, a sample is randomly initialized, following the standard uniform distribution. A step of constrained sampling is composed of n runs of gradient descent (GD) to impose the high-level structure, followed by p runs of *selective Gibbs sampling* (GS) to selectively realign the sample onto the learnt distribution. A simulated annealing algorithm is applied in order to decrease exploration in relation to a decrease of variance over solutions. Figure 29 shows an example of a generated sample in piano roll format.

9.7 Other Architectures and Strategies

Researchers in the domain of deep learning techniques for music generation are designing and experimenting with various architectures and strategies[58], in most cases combinations or refinements of existing ones, or sometimes with novel types, as, e.g., in the case of MusicTransformer [25]. However, there is no guarantee that combining a maximal variety of types will make a sound and accurate architecture[59]. Therefore, it is important to continue

[55]The architecture is convolutional (only) on the time dimension, in order to model temporally invariant motives, but not pitch invariant motives which would break the notion of tonality.

[56]Because of space limitation, and the fact that RBMs are not mainstream, we do not detail here the characteristics of RBM (see, e.g., [13, Section 20.2] or [2, Section 5.7] for details). In a first approximation for this article, we may consider an RBM as analog to an autoencoder, except with two differences: the input and output layers are merged (and named the visible layer) and the model is stochastic.

[57]This is named *structure imposition*, with the same basic approach that of style transfer [6], except that of a high-level structure.

[58]This article is obviously not exhaustive. Interested readers may refer, e.g., to [2] for additional examples and details.

[59]As in the case of a good cook, whose aim is not to simply mix *all* possible ingredients but to discover original successful combinations.

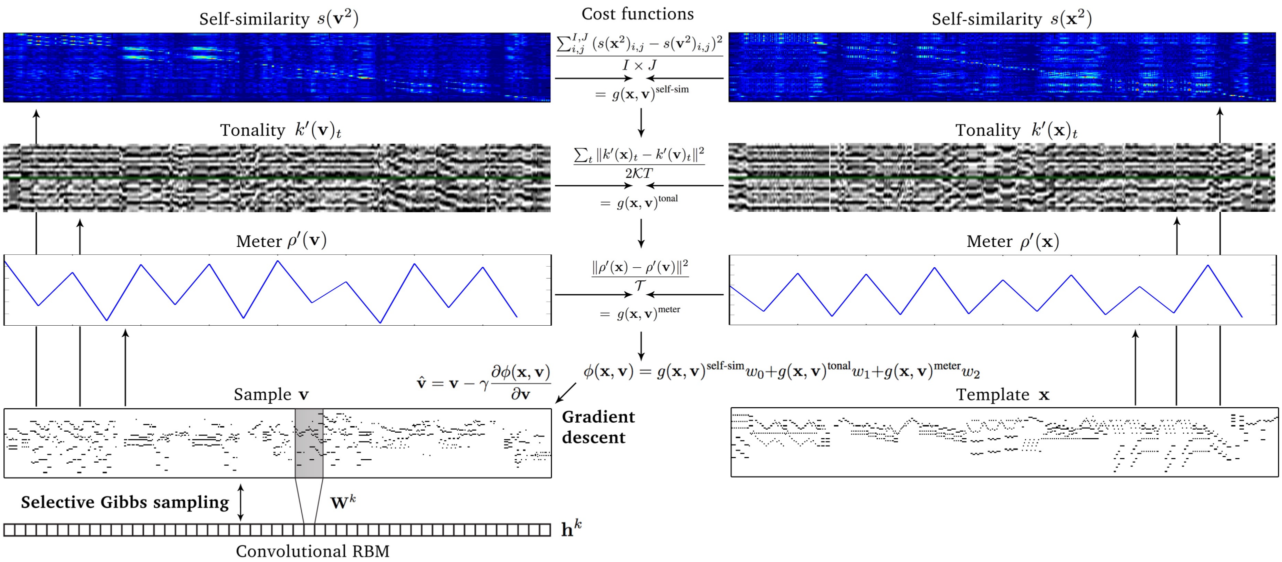

Figure 28: C-RBM Architecture generation algorithm

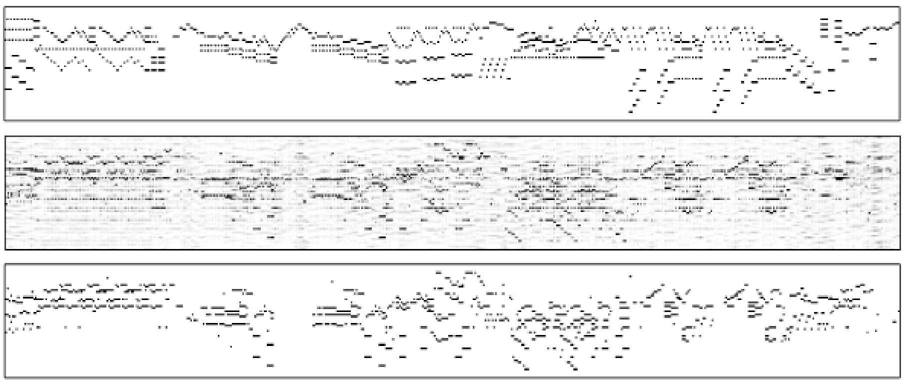

Figure 29: Illustration of constrained sampling. Piano roll representation of: (top) Template piece, (middle) Intermediate sample after the GD phase, (bottom) Sample after the GS phase. Reproduced with permission of the authors

to deepen our understanding and to explore solutions as well as their possible articulations and combinations. We hope that this survey could contribute to that objective.

10 Conclusion

The use of artificial neural networks and deep learning architectures and techniques for the generation of music (as well as other artistic contents) is a very active area of research. In this article, we have introduced the domain, analyzed early and pioneering proposals, introduced a conceptual framework to help at analyzing and classifying the large diversity of systems and experiments described in the literature, illustrated by various examples. We hope that this survey will help in better understanding the domain and trends of deep learning-based music generation.

Acknowledgements

We thank François Pachet and Gaëtan Hadjeres for their participation to the book [2] which has been a significant initial input for this article; CNRS, LIP6 and UNIRIO for their support; and the various participants of our course on the topic[60] for their feedback during the course.

References

[1] Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: 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-12), pp. 1159–1166. Edinburgh, Scotland, U.K. (2012)

[2] Briot, J.P., Hadjeres, G., Pachet, F.D.: Deep Learning Techniques for Music Generation. Computational Synthesis and Creative Systems. Springer International Publishing (2019)

[3] Briot, J.P., Pachet, F.: Music generation by deep learning: challenges and directions. Neural Computing and Applications (NCAA) **32**(4), 981–993 (2020)

[4] Cherry, E.C.: Some experiments on the recognition of speech, with one and two ears. The Journal of the Acoustical Society of America **25**(5), 975–979 (1953)

[5] Cope, D.: The Algorithmic Composer. A-R Editions (2000)

[6] Dai, S., Zhang, Z., Xia, G.G.: Music style transfer issues: A position paper (2018). ArXiv:1803.06841v1

[7] Eck, D., Schmidhuber, J.: A first look at music composition using LSTM recurrent neural networks. Tech. rep., IDSIA/USI-SUPSI, Manno, Switzerland (2002). No. IDSIA-07-02

[8] Emerging Technology from the arXiv: Deep learning machine solves the cocktail party problem. MIT Technology Review (2015). https://www.technologyreview.com/s/537101/deep-learning-machine-solves-the-cocktail-party-problem/

[9] Fernández, J.D., Vico, F.: AI methods in algorithmic composition: A comprehensive survey. Journal of Artificial Intelligence Research (JAIR) (48), 513–582 (2013)

[10] Fiebrink, R., Caramiaux, B.: The machine learning algorithm as creative musical tool (2016). ArXiv:1611.00379v1

[11] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Professional Computing Series. Addison-Wesley (1995)

[12] Gatys, L.A., Ecker, A.S., Bethge, M.: A neural algorithm of artistic style (2015). ArXiv:1508.06576v2

[13] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)

[14] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozairy, S., Courville, A., Bengio, Y.: Generative adversarial nets (2014). ArXiv:1406.2661v1

[15] Graves, A.: Generating sequences with recurrent neural networks (2014). ArXiv:1308.0850v5

[60] Course material is available online at: `http://www-desir.lip6.fr/~briot/cours/unirio3/`.

[16] Hadjeres, G., Nielsen, F.: Interactive music generation with positional constraints using Anticipation-RNN (2017). ArXiv:1709.06404v1

[17] Hadjeres, G., Nielsen, F., Pachet, F.: GLSR-VAE: Geodesic latent space regularization for variational autoencoder architectures (2017). ArXiv:1707.04588v1

[18] Hadjeres, G., Pachet, F., Nielsen, F.: DeepBach: a steerable model for Bach chorales generation (2017). ArXiv:1612.01010v2

[19] Herremans, D., Chuan, C.H., Chew, E.: A functional taxonomy of music generation systems. ACM Computing Surveys (CSUR) **50**(5) (2017)

[20] Hiller, L.A., Isaacson, L.M.: Experimental Music: Composition with an Electronic Computer. McGraw-Hill (1959)

[21] Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)

[22] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)

[23] Huang, C.Z.A., Cooijmans, T., Dinculescu, M., Roberts, A., Hawthorne, C.: Coconet: the ML model behind today's Bach Doodle (2019). https://magenta.tensorflow.org/coconet

[24] Huang, C.Z.A., Cooijmans, T., Roberts, A., Courville, A., Eck, D.: Counterpoint by convolution. In: X. Hu, S.J. Cunningham, D. Turnbull, Z. Duan (eds.) Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR 2017), pp. 211–218. ISMIR, Suzhou, China (2017)

[25] Huang, C.Z.A., Vaswani, A., Uszkoreit, J., Shazeer, N., Hawthorne, I.S.C., Dai, A.M., Hoffman, M.D., Dinculescu, M., Eck, D.: Music transformer: Generating music with long-term structure generating music with long-term structure (2018). ArXiv:1809.04281v3

[26] Keith, J.: The Session (Accessed on 21/12/2016). https://thesession.org

[27] Kingma, D.P., Welling, M.: Auto-encoding variational Bayes (2014). ArXiv:1312.6114v10

[28] Koutník, J., Greff, K., Gomez, F., Schmidhuber, J.: A clockwork RNN (2014). ArXiv:1402.3511v1

[29] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems, *NIPS 2012*, vol. 1, pp. 1097–1105. Curran Associates Inc., Lake Tahoe, NE, USA (2012)

[30] Lattner, S., Grachten, M., Widmer, G.: Imposing higher-level structure in polyphonic music generation using convolutional restricted Boltzmann machines and constraints (2016). ArXiv:1612.04742v2

[31] Lattner, S., Grachten, M., Widmer, G.: Imposing higher-level structure in polyphonic music generation using convolutional restricted Boltzmann machines and constraints. Journal of Creative Music Systems (JCMS) **2**(2) (2018)

[32] Lewis, J.P.: Creation by refinement: A creativity paradigm for gradient descent learning networks. In: IEEE International Conference on Neural Networks, vol. II, pp. 229–233. San Diego, CA, USA (1988)

[33] Lewis, J.P.: Creation by refinement and the problem of algorithmic music composition. In: P.M. Todd, D.G. Loy (eds.) Music and Connectionism, pp. 212–228. MIT Press (1991)

[34] Mao, H.H., Shin, T., Cottrell, G.W.: DeepJ: Style-specific music generation (2018). ArXiv:1801.00887v1

[35] Mathieu, E., Rainforth, T., Siddharth, N., Teh, Y.W.: Disentangling disentanglement in variational autoencoders (2019). ArXiv:1812.02833v3

[36] Matisse, N.: How YACHT fed their old music to the machine and got a killer new album. ArsTechnica (2019). https://arstechnica.com/gaming/2019/08/yachts-chain-tripping-is-a-new-landmark-for-ai-music-an-album-that-doesnt-suck/

[37] McClelland, J.L., Rumelhart, D.E., Group, P.R.: Parallel Distributed Processing – Explorations in the Microstructure of Cognition: Volume 2 Psychological and Biological Models. MIT Press (1986)

[38] Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., Bengio, Y.: SampleRNN: An unconditional end-to-end neural audio generation model (2017). ArXiv:1612.07837v2

[39] Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. MIT Press (1969)

[40] (MMA), M.M.A.: MIDI Specifications (Accessed on 14/04/2017). https://www.midi.org/specifications

[41] Mogren, O.: C-RNN-GAN: Continuous recurrent neural networks with adversarial training (2016). ArXiv:1611.09904v1

[42] Mordvintsev, A., Olah, C., Tyka, M.: Deep Dream (2015). https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

[43] Nierhaus, G.: Algorithmic Composition: Paradigms of Automated Music Generation. Springer Nature (2009)

[44] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: WaveNet: A generative model for raw audio (2016). ArXiv:1609.03499v2

[45] Papadopoulos, A., Roy, P., Pachet, F.: Assisted lead sheet composition using FlowComposer. In: M. Rueher (ed.) Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings, pp. 769–785. Springer Nature (2016)

[46] Papadopoulos, G., Wiggins, G.: AI methods for algorithmic composition: A survey, a critical view and future prospects. In: AISB 1999 Symposium on Musical Creativity, pp. 110–117 (1999)

[47] Pons, J.: Neural networks for music: A journey through its history (2018). https://towardsdatascience.com/neural-networks-for-music-a-journey-through-its-history-91f93c3459fb

[48] Ramsundar, B., Zadeh, R.B.: TensorFlow for Deep Learning. O'Reilly Media (2018)

[49] Roberts, A., Engel, J., Raffel, C., Hawthorne, C., Eck, D.: A hierarchical latent vector model for learning long-term structure in music. In: Proceedings of the 35th International Conference on Machine Learning (ICML 2018). ACM, Montréal, PQ, Canada (2018)

[50] Roberts, A., Engel, J., Raffel, C., Hawthorne, C., Eck, D.: A hierarchical latent vector model for learning long-term structure in music (2018). ArXiv:1803.05428v2

[51] Rumelhart, D.E., McClelland, J.L., Group, P.R.: Parallel Distributed Processing – Explorations in the Microstructure of Cognition: Volume 1 Foundations. MIT Press (1986)

[52] Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall (1996)

[53] Sturm, B.L., Santos, J.F., Ben-Tal, O., Korshunova, I.: Music transcription modelling and composition using deep learning. In: Proceedings of the 1st Conference on Computer Simulation of Musical Creativity (CSCM 16). Huddersfield, U.K. (2016)

[54] Sun, F.: DeepHear – Composing and harmonizing music with neural networks (Accessed on 21/12/2017). https://fephsun.github.io/2015/09/01/neural-music.html

[55] Todd, P.M.: A connectionist approach to algorithmic composition. Computer Music Journal (CMJ) **13**(4), 27–43 (1989)

[56] Todd, P.M.: A connectionist approach to algorithmic composition. In: P.M. Todd, D.G. Loy (eds.) Music and Connectionism, pp. 190–194. MIT Press (1991)

[57] Todd, P.M., Loy, D.G.: Music and Connectionism. MIT Press (1991)

[58] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2017). ArXiv:1706.03762v5

[59] Walshaw, C.: abc notation home page (Accessed on 21/12/2016). http://abcnotation.com

[60] Wild, C.M.: What a disentangled net we weave: Representation learning in VAEs (Pt. 1) (2018). https://towardsdatascience.com/what-a-disentangled-net-we-weave-representation-learning-in-vaes-pt-1-9e5dbc205bd1

[61] Yang, L.C., Chou, S.Y., Yang, Y.H.: MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. In: Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR 2017). Suzhou, China (2017)

[62] Yang, R., Wang, D., Wang, Z., Chen, T., Jiang, J., Xia, G.: Deep music analogy via latent representation disentanglement (2019). ArXiv:1906.03626v4