Neural Computing and Applications (2020) 32:981–993
https://doi.org/10.1007/s00521-018-3813-6

DEEP LEARNING FOR MUSIC AND AUDIO

Deep learning for music generation: challenges and directions

Jean-Pierre Briot[1] · François Pachet[2]

Received: 2 December 2017 / Accepted: 6 October 2018 / Published online: 16 October 2018

Abstract

In addition to traditional tasks such as prediction, classification and translation, deep learning is receiving growing attention as an approach for music generation, as witnessed by recent research groups such as Magenta at Google and CTRL (Creator Technology Research Lab) at Spotify. The motivation is in using the capacity of deep learning architectures and training techniques to automatically learn musical styles from arbitrary musical corpora and then to generate samples from the estimated distribution. However, a direct application of deep learning to generate content rapidly reaches limits as the generated content tends to mimic the training set without exhibiting true creativity. Moreover, deep learning architectures do not offer direct ways for controlling generation (e.g., imposing some tonality or other arbitrary constraints). Furthermore, deep learning architectures alone are autistic automata which generate music autonomously without human user interaction, far from the objective of interactively assisting musicians to compose and refine music. Issues such as control, structure, creativity and interactivity are the focus of our analysis. In this paper, we select some limitations of a direct application of deep learning to music generation and analyze why the issues are not fulfilled and how to address them by possible approaches. Various examples of recent systems are cited as examples of promising directions.

Keywords Deep learning · Music · Generation · Challenges · Directions · Control · Structure · Creativity · Interactivity

1 Introduction

1.1 Deep learning

Deep learning has become a fast-growing domain and is now used routinely for classification and prediction tasks, such as image and voice recognition, as well as translation. It emerged about 10 years ago, when a deep learning architecture significantly outperformed standard techniques using handcrafted features on an image classification task [21]. We may explain this success and reemergence of artificial neural networks architectures and techniques by the combination of:

1. *technical progress*, such as: convolutions, which provide motif translation invariance [4], and LSTM (Long Short-Term Memory), which resolved inefficient training of recurrent neural networks [22];
2. availability of multiple *datasets*;
3. availability of *efficient and cheap computing power*, e.g., offered by graphics processing units (GPU).

There is no consensual definition of deep learning. It is a repertoire of machine learning (ML) techniques, based on artificial neural networks.[1] The common ground is the term *deep*, which means that there are multiple layers processing multiple levels of abstractions, which are automatically extracted from data, as a way to express complex representations in terms of simpler representations.

Main applications of deep learning are within the two traditional machine learning tasks of *classification* and *prediction*, as a testimony of the initial DNA of neural networks: logistic regression and linear regression. But a growing area of application of deep learning techniques is

✉ Jean-Pierre Briot
Jean-Pierre.Briot@lip6.fr

François Pachet
francois@spotify.com

[1] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, 75005 Paris, France

[2] Spotify Creator Technology Research Lab, Paris, France

[1] With many variants such as convolutional networks, recurrent networks, autoencoders, restricted Boltzmann machines [15].

the *generation* of *content*: text, images and *music*, the focus of this article.

1.2 Deep learning for music generation

The motivation for using deep learning, and more generally machine learning techniques, to generate musical content is its generality. As opposed to handcrafted models for, e.g., grammar-based [37] or rule-based music generation systems [8], a machine learning-based generation system can automatically learn a model, a *style*, from an arbitrary corpus of music. Generation can then take place by using prediction (e.g., to predict the pitch of the next note of a melody) or classification (e.g., to recognize the chord corresponding to a melody), based on the distribution and correlations learnt by the deep model which represent the style of the corpus.

As stated by Fiebrink and Caramiaux [12], benefits are as follows: (1) It can make creation feasible when the desired application is too complex to be described by analytical formulations or manual brute force design and (2) learning algorithms are often less brittle than manually designed rule sets and learned rules are more likely to generalize accurately to new contexts in which inputs may change.

1.3 Challenges

A direct application of deep learning architectures and techniques to generation, although it could produce impressing results,[2] suffers from some limitations. We consider here:[3]

- *Control*, e.g., tonality conformance, maximum number of repeated notes, rhythm,;
- *Structure*, versus wandering music without a sense of direction;
- *Creativity*, versus imitation and risk of plagiarism;
- *Interactivity*, versus automated single-step generation.

1.4 Related work

A comprehensive survey and analysis by Briot et al. of deep learning techniques to generate musical content is available in a book [2]. In [20], Herremans et al. propose a function-oriented taxonomy for various kinds of music generation systems. Examples of surveys about AI-based methods for algorithmic music composition are by Papadopoulos and Wiggins [34] and by Fernández and Vico [11], as well as books by Cope [3] and by Nierhaus [29]. In [17], Graves analyzes the application of recurrent neural networks architectures to generate sequences (text and music). In [12], Fiebrink and Caramiaux address the issue of using machine learning to generate creative music. We are not aware of a comprehensive analysis dedicated to deep learning (and artificial neural networks techniques) that systematically analyzes limitations and challenges, solutions and directions, in other words, that is *problem-oriented* and not just application-oriented.

1.5 Organization

The article is organized as follows: Section 1 (this section) introduces the general context of deep learning-based music generation and lists some important challenges. It also includes a comparison to some related work. The following sections analyze each challenge and some solutions, while illustrating through examples of actual systems: control/Sect. 2, structure/Sect. 3, creativity/Sect. 4 and interactivity/Sect. 5.

2 Control

Musicians usually want to adapt ideas and patterns borrowed from other contexts to their own objective, e.g., transposition to another key, minimizing the number of notes. In practice, this means the ability to control generation by a deep learning architecture.

2.1 Dimensions of control strategies

Such arbitrary control is actually a difficult issue for current deep learning architectures and techniques, because standard neural networks are not designed to be controlled. As opposed to Markov models which have an operational model where one can attach constraints onto their internal operational structure in order to control the generation,[4] neural networks do not offer such an operational entry point. Moreover, the distributed nature of their representation does not provide a direct correspondence to the structure of the content generated. As a result, strategies for controlling deep learning generation that we will analyze have to rely on some *external* intervention at various *entry points* (hooks), such as:

- *Input*;
- *Output*;
- *Encapsulation/reformulation*.

[2] Music difficult to distinguish from the original corpus.

[3] Additional challenges are analyzed in [2].

[4] Two examples are Markov constraints [31] and factor graphs [30].

2.2 Sampling

Sampling a model[5] to generate content may be an entry point for control if we introduce *constraints* on the output generation. (This is called *constraint sampling*.) This is usually implemented by a generate-and-test approach, where valid solutions are picked from a set of generated random samples from the model.[6] As we will see, a key issue is how to guide the sampling process in order to fulfill the objectives (constraints); thus, sampling will be often combined with other strategies.

2.3 Conditioning

The strategy of *conditioning* (sometimes also named *conditional architecture*) is to condition the architecture on some extra conditioning information, which could be arbitrary, e.g., a class label or data from other modalities. Examples are:

- a *bass line* or a *beat structure*, in the rhythm generation system [27];
- a *chord progression*, in the MidiNet architecture [42];
- a *musical genre* or an *instrument*, in the WaveNet architecture [40];
- a set of *positional constraints*, in the Anticipation-RNN architecture [18].

In practice, the conditioning information is usually fed into the architecture as an additional input layer. Conditioning is a way to have some degree of *parameterized control* over the generation process.

2.3.1 Example 1: WaveNet audio speech and music generation

The WaveNet architecture by van der Oord et al. [40] is aimed at generating raw audio waveforms. The architecture is based on a convolutional feedforward network without pooling layer.[7] It has been experimented on generation for three audio domains: multi-speaker, text-to-speech (TTS) and music.

The WaveNet architecture uses conditioning as a way to guide the generation, by adding an additional tag as a conditioning input Two options are considered: *global* conditioning or *local* conditioning, depending on whether the conditioning input is shared for *all* time steps or is specific to *each* time step.

An example of application of conditioning WaveNet for a text-to-speech application domain is to feed linguistic features (e.g., North American English or Mandarin Chinese speakers) in order to generate speech with a better prosody. The authors also report preliminary experiments on conditioning music models to generate music given a set of tags specifying, e.g., genre or instruments.

2.3.2 Example 2: Anticipation-RNN Bach melody generation

Hadjeres and Nielsen propose a system named Anticipation-RNN [18] for generating melodies with unary constraints on notes (to enforce a given note at a given time position to have a given value). The limitation when using a standard note-to-note iterative strategy for generation by a recurrent network is that enforcing the constraint at a certain time step may retrospectively invalidate the distribution of the previously generated items, as shown in [31]. The idea is to condition the recurrent network (RNN) on some information summarizing the set of further (in time) constraints as a way to anticipate oncoming constraints, in order to generate notes with a correct distribution.

Therefore, a second RNN architecture,[8] named Constraint-RNN, is used and it functions backward in time and its outputs are used as additional inputs of the main RNN (named Token-RNN), resulting in the architecture shown at Fig. 1, with:

- c_i being a *positional constraint*;
- o_i being the output at index i (after i iterations) of Constraint-RNN – it summarizes constraint information from step i to final step (end of the sequence) N. It will be concatenated ($\oplus$) to input s_{i-1} of Token-RNN in order to predict next item s_i.

The architecture has been tested on a corpus of melodies taken from J. S. Bach chorales. Three examples of melodies generated with the same set of positional constraints (indicated with notes in green within a rectangle) are shown at Fig. 2. The model is indeed able to anticipate each positional constraint by adjusting its direction toward the target (lower-pitched or higher-pitched note).

[5] The model can be stochastic, such as a restricted Boltzmann machine (RBM) [15], or deterministic, such as a feedforward or a recurrent network. In that latter case, it is common practice to sample from the softmax output in order to introduce *variability* for the generated content [2].

[6] Note that this may be a very costly process and moreover with no guarantee to succeed.

[7] An important specificity of the architecture (not discussed here) is the notion of *dilated convolution*, where convolution filters are incrementally dilated in order to provide very large receptive fields with just a few layers, while preserving input resolution and computational efficiency [40].

[8] Both are two-layer LSTMs [22].

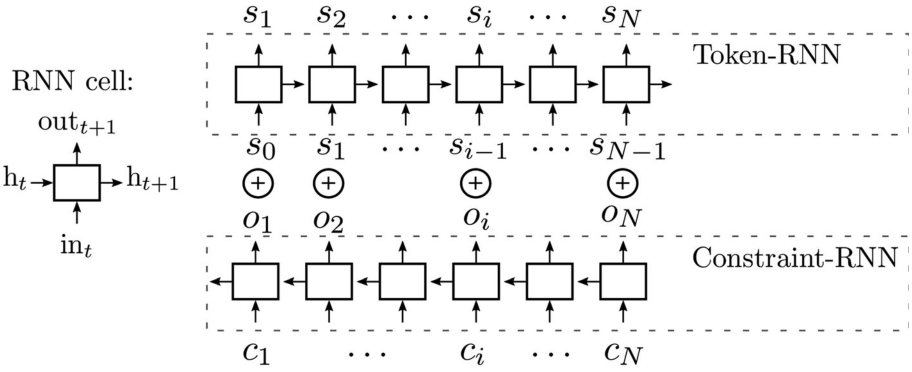

Fig. 1 Anticipation-RNN architecture. Reproduced from [18] with permission of the authors

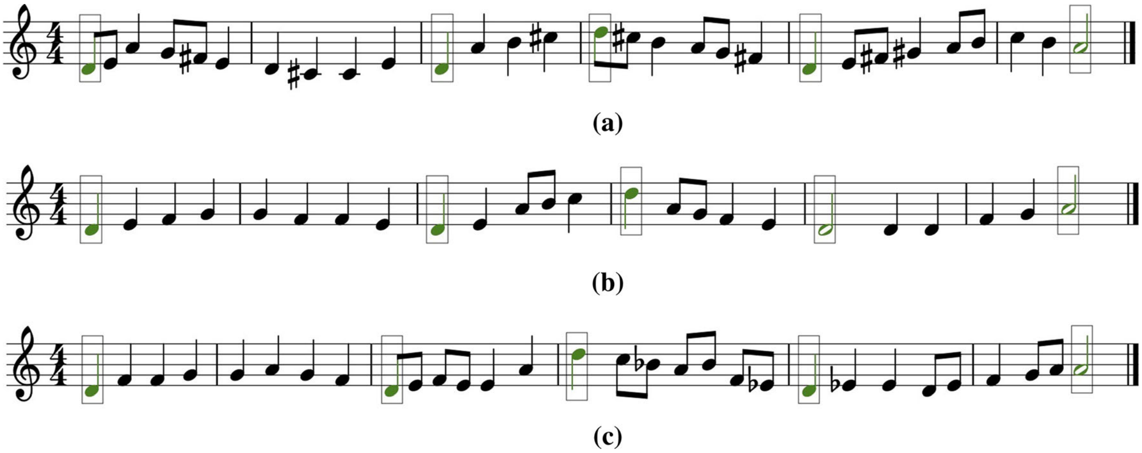

Fig. 2 Examples of melodies generated by Anticipation-RNN. Reproduced from [18] with permission of the authors

2.4 Input manipulation

The strategy of *input manipulation* has been pioneered for images by DeepDream [28]. The idea is that the initial input content, or a brand new (randomly generated) input content, is incrementally manipulated in order to match a *target property*. Note that control of the generation is *indirect*, as it is not being applied to the output but to the *input*, *before generation*. Examples are:

- *maximizing* the *activation* of a specific *unit*, to *exaggerate* some visual element specific to this unit, in DeepDream [28];
- *maximizing* the *similarity* to a given *target*, to create a *consonant melody*, in DeepHear [38];
- *maximizing both* the *content similarity* to some initial image *and* the *style similarity* to a reference style image, to perform *style transfer* [14];
- *maximizing* the *similarity* of *structure* to some reference music, to perform *style imposition* [26].

Interestingly, this is done by reusing standard training mechanisms, namely back-propagation to compute the gradients, as well as gradient descent to minimize the cost.

2.4.1 Example 1: DeepHear ragtime melody accompaniment generation

The DeepHear architecture by Sun [38] is aimed at generating ragtime jazz melodies. The architecture has 4-layer stacked autoencoders (that is 4 hierarchically nested autoencoders), with a decreasing number of hidden units, down to 16 units.

At first, the model is trained[9] on a corpus of 600 measures of Scott Joplin's ragtime music, split into 4-measure long segments. Generation is performed by inputting random data as the seed into the 16 bottleneck hidden-layer units and then by feedforwarding it into the chain of decoders to produce an output (in the same 4-measure long format of the training examples), as shown in Fig. 3.

In addition to the generation of new melodies, DeepHear is used with a different objective: to harmonize a melody, while using the *same* architecture as well as what has

[9] Autoencoders are trained with the same data as input and output and therefore have to discover significative features in order to be able to reconstruct the compressed data.

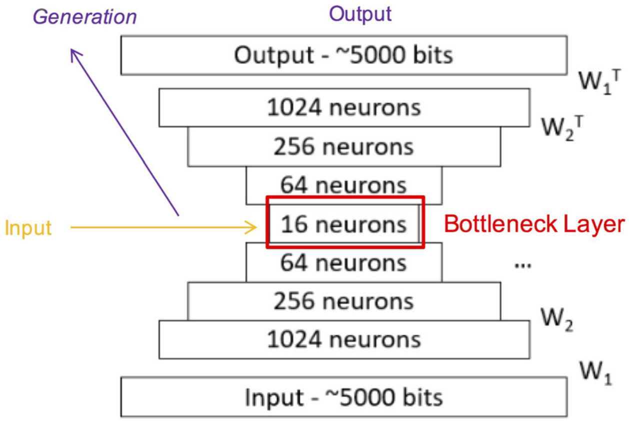

Fig. 3 Generation in DeepHear. Extension of a figure reproduced from [38] with permission of the author

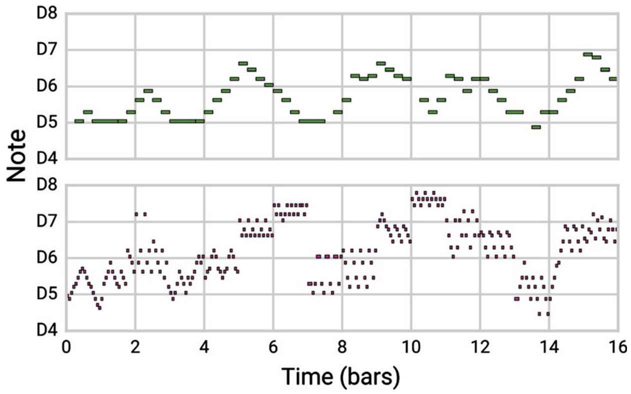

Fig. 4 Example of melody generated (bottom) by MusicVAE by adding a "high note density" attribute vector to the latent space of an existing melody (top). Reproduced from [35] with permission of the authors

already been learnt.[10] The idea is to find a label instance of the set of features, i.e., a set of values for the 16 units of the bottleneck hidden layer of the stacked autoencoders which will result in some decoded output matching as much as possible a given melody. A simple distance function is defined to represent the dissimilarity between two melodies (in practice, the number of not matched notes). Then a gradient descent is conducted onto the variables of the embedding, guided by the gradients corresponding to the distance function until finding a sufficiently similar decoded melody. Although this is not a real counterpoint but rather the generation of a similar (consonant) melody, the results do produce some naive counterpoint with a ragtime flavor.

2.4.2 Example 2: VRAE video game melody generation

Note that input manipulation of the hidden-layer units of an autoencoder (or stacked autoencoders) bears some analogy with variational autoencoders,[11] such as, for instance, the VRAE (variational recurrent autoencoder) architecture of Fabius and van Amersfoort [10]. Indeed in both cases, there is some exploration of possible values for the hidden units (latent variables) in order to generate variations of musical content by the decoder (or the chain of decoders). The important difference is that in the case of variational autoencoders, the exploration of values is *user-directed*, although it could be guided by some principle, for example an interpolation to create a medley of two songs, or the addition or subtraction of an attribute vector capturing a given characteristic (e.g., high density of notes as in Fig. 4). In the case of input manipulation, the exploration of values is automatically guided by the gradient-following mechanism, the user having priorly specified a cost function to be minimized or an objective to be maximized.

2.4.3 Example 3: Image and audio style transfer

Style transfer has been pioneered by Gatys et al. [14] for images. The idea, shown in Fig. 5, is to use a deep learning architecture to independently capture:

- the features of a first image (named the *content*),
- and the *style* (the correlations between features) of a second image (named the *style*),
- and then, to use gradient following to guide the incremental modification of an initially random third image, with the double objective of *matching both* the *content* and the *style* descriptions.[12]

Transposing this style transfer technique to music was a natural direction, and it has been experimented independently for audio, e.g., in [13] and [39], both using a spectrogram (and not a direct wave signal) as input. The result is effective, but not as interesting as in the case of painting style transfer, being somehow more similar to a sound merging of the style and of the content. We believe that this is because of the *anisotropy*[13] of global music content representation.

[10] Note that this is a simple example of *transfer learning* [15], with a same domain and a same training, but for a different task.

[11] A variational autoencoder (VAE) [25] is an autoencoder with the added constraint that the encoded representation (its latent variables) follows some prior probability distribution (usually a Gaussian distribution). Therefore, a variational autoencoder is able to learn a "smooth" latent space mapping to realistic examples.

[12] Note that one may balance between content and style objectives through some α and β parameters in the $\mathcal{L}_{total}$ combined loss function shown at top of Fig. 5.

[13] In the case of an image, the correlations between visual elements (pixels) are equivalent whatever the direction (horizontal axis, vertical axis, diagonal axis or any arbitrary direction); in other words, correlations are *isotropic*. In the case of a global representation of musical content (see, e.g., Fig. 12), where the horizontal dimension represents time and the vertical dimension represents the notes, horizontal correlations represent *temporal* correlations and vertical correlations represent *harmonic* correlations, which have very different nature.

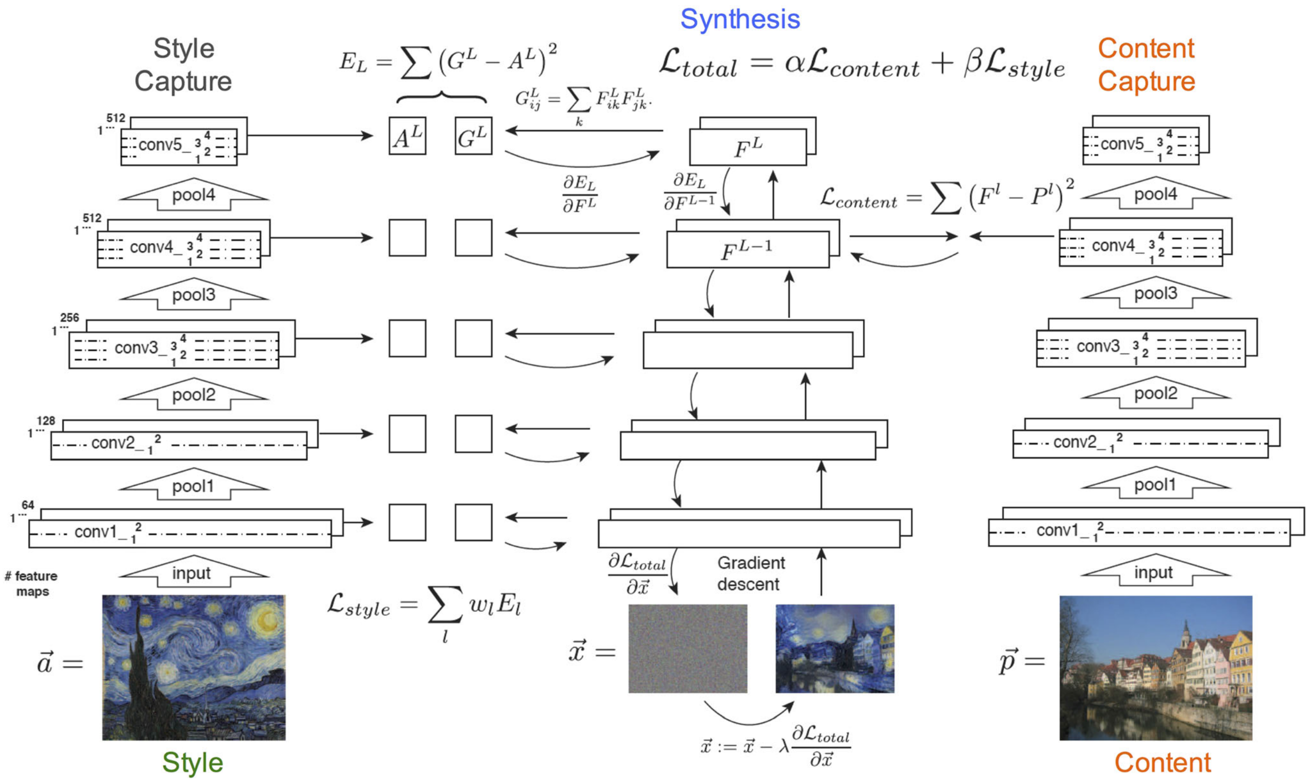
Style Capture
$E_L = \sum (G^L - A^L)^2$
Synthesis
$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$
Content Capture
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}$
A^L
G^L
F^L
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
F^{L-1}
512
conv5_1 2 3 4
pool4
512
conv4_1 2 3 4
pool3
256
conv3_1 2 3 4
pool2
128
conv2_1 2
pool1
64
conv1_1 2
input
# feature maps
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Style
Content

Fig. 5 Style transfer full architecture/process. Reproduced with permission of the authors

2.4.4 Example 4: C-RBM Mozart sonata generation

The C-RBM architecture proposed by Lattner et al. [26] uses a restricted Boltzmann machine (RBM) to learn the *local structure*, seen as the *musical texture*, of a corpus of musical pieces (in practice, Mozart sonatas). 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. The main idea is in imposing by *constraints* onto the generated piece some more *global structure* (form, e.g., AABA, as well as tonality), seen as a *structural template* inspired from the reference of an existing musical piece. This is called *structure imposition*,[14] also coined as *templagiarism* (short for template plagiarism) by Hofstadter [23].

Generation is done by *sampling* from the RBM with 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;
- *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 predefined constraints. The principle of the process (illustrated in Fig. 6) 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 to impose the high-level structure, followed by *p* runs of *selective Gibbs sampling* 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 in variance over solutions.

Results are quite convincing. However, as discussed by the authors, their approach is not exact, as for instance by the Markov constraints approach proposed in [31].

[14] Note that this also some kind of style transfer [5], although of a high-level structure and not a low-level timbre as in Sect. 2.4.3.

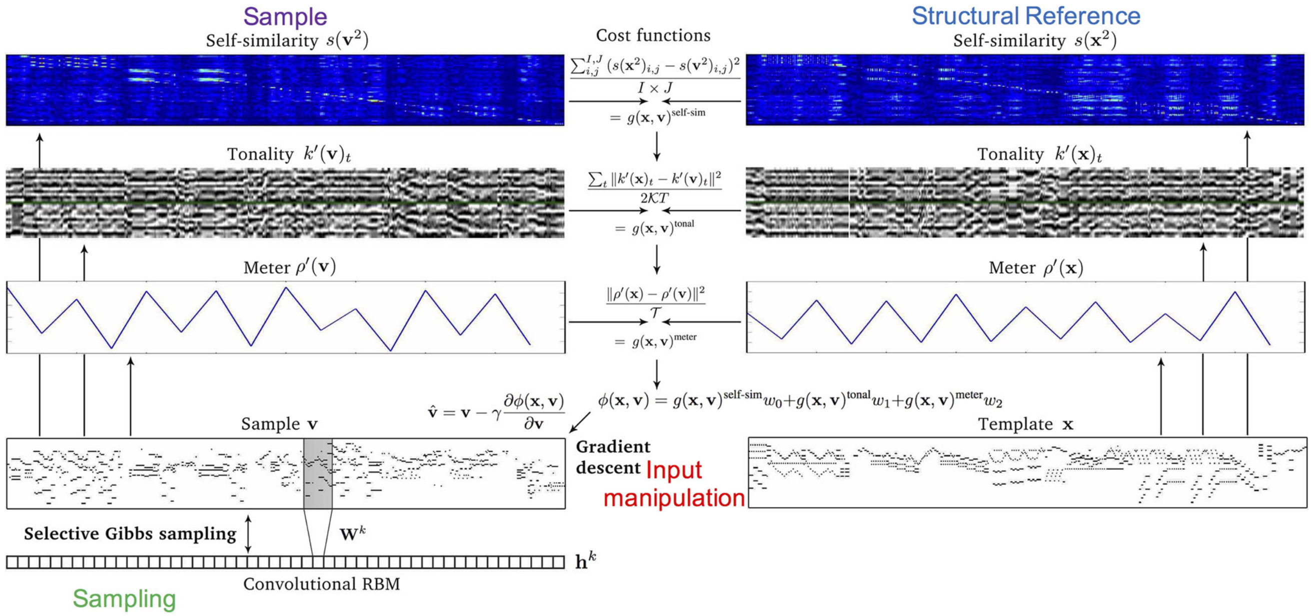
Sample
Self-similarity $s(\mathbf{v}^2)$
Cost functions
$\frac{\sum_{i,j}^{I,J}(s(\mathbf{x}^2)_{i,j}-s(\mathbf{v}^2)_{i,j})^2}{I\times J}$
$= g(\mathbf{x},\mathbf{v})^{\text{self-sim}}$
Structural Reference
Self-similarity $s(\mathbf{x}^2)$
Tonality $k'(\mathbf{v})_t$
$\frac{\sum_t \|k'(\mathbf{x})_t - k'(\mathbf{v})_t\|^2}{2KT}$
$= g(\mathbf{x},\mathbf{v})^{\text{tonal}}$
Tonality $k'(\mathbf{x})_t$
Meter $\rho'(\mathbf{v})$
$\frac{\|\rho'(\mathbf{x})-\rho'(\mathbf{v})\|^2}{T}$
$= g(\mathbf{x},\mathbf{v})^{\text{meter}}$
Meter $\rho'(\mathbf{x})$
$\hat{\mathbf{v}} = \mathbf{v} - \gamma\frac{\partial\phi(\mathbf{x},\mathbf{v})}{\partial\mathbf{v}}$
$\phi(\mathbf{x},\mathbf{v}) = g(\mathbf{x},\mathbf{v})^{\text{self-sim}}w_0 + g(\mathbf{x},\mathbf{v})^{\text{tonal}}w_1 + g(\mathbf{x},\mathbf{v})^{\text{meter}}w_2$
Sample $\mathbf{v}$
Template $\mathbf{x}$
Gradient descent
Input manipulation
Selective Gibbs sampling
$\mathbf{W}^k$
$\mathbf{h}^k$
Sampling
Convolutional RBM

Fig. 6 C-RBM architecture

2.5 Reinforcement

The strategy of *reinforcement* is to *reformulate* the generation of musical content as a *reinforcement learning problem*, while using the output of a trained recurrent network as an *objective* and adding user-defined constraints, e.g., some tonality rules according to music theory, as an *additional objective*.

Let us, at first, quickly remind the basic concepts of reinforcement learning, illustrated in Fig. 7:

- An *agent* sequentially selects and performs *actions* within an *environment*;
- Each action performed brings it to a new *state*,
- with the *feedback* (by the environment) of a *reward* (*reinforcement signal*), which represents some *adequation* of the action to the environment (the situation).
- The objective of *reinforcement learning* is for the agent to learn a near-optimal *policy* (sequence of actions) in order to maximize its *cumulated rewards* (named its *gain*).

Generation of a melody may be formulated as follows (as in Fig. 8): The *state s* represents the musical content (a *partial melody*) generated so far and the *action a* represents the selection of next *note* to be generated.

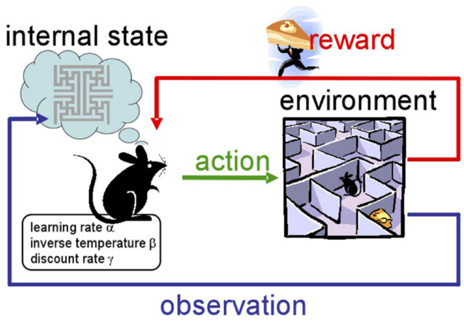

Fig. 7 Reinforcement learning (conceptual model)—reproduced from [7]

2.5.1 Example: RL-tuner melody generation

The *reinforcement strategy* has been pioneered by the *RL-tuner* architecture by Jaques et al. [24]. The architecture, illustrated in Fig. 8, consists in two reinforcement learning architectures, named Q Network and Target Q Network,[15] and two *recurrent network* (RNN) architectures, named Note RNN and Reward RNN.

After training Note RNN on the corpus, a fixed copy named Reward RNN is used as a *reference* for the reinforcement learning architecture. The *reward r* of Q Network is defined as a combination of two objectives:

- Adherence to *what has been learnt*, by measuring the similarity of the action selected (next note to be

[15] They use a deep learning implementation of the Q-learning algorithm. Q Network is trained in parallel to Target Q Network which estimates the value of the gain [41].

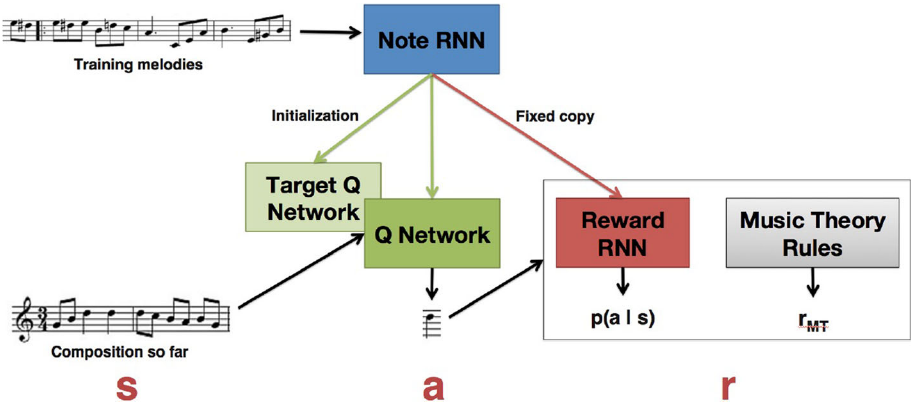

Fig. 8 RL-tuner architecture

generated) to the note predicted by Reward RNN in a similar state (partial melody generated so far);

- Adherence to *user-defined constraints* (e.g., consistency with current tonality, avoidance of excessive repetitions), by measuring how well they are fulfilled.

Although preliminary, results are convincing. Note that this strategy has the potential for adaptive generation by incorporating feedback from the user.

2.6 Unit selection

The *unit selection* strategy relies in querying successive *musical units* (e.g., a melody within a measure) from a database and in *concatenating* them in order to generate some sequence according to some user characteristics.

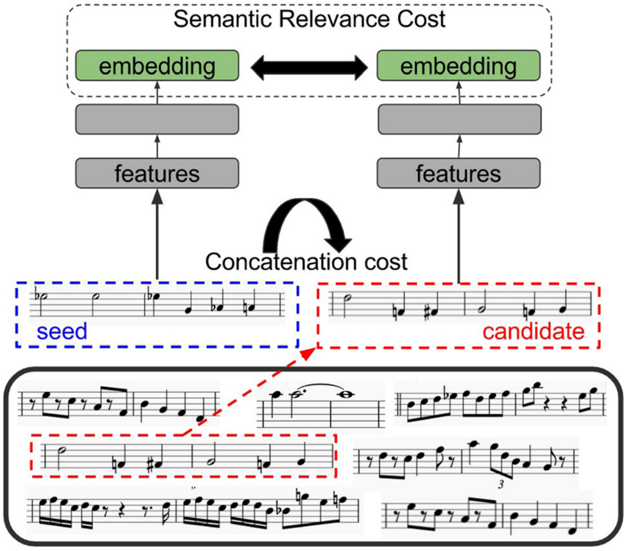

Fig. 9 Unit selection based on semantic cost

2.6.1 Example: Unit selection and concatenation melody generation

This strategy has been pioneered by Bretan et al. [1] and is actually inspired by a technique commonly used in text-to-speech (TTS) systems and adapted in order to generate melodies. (The corpus used is diverse and includes jazz, folk and rock). The key process here is *unit selection* (in general each unit is one measure long), based on two criteria: *semantic relevance* and *concatenation cost*. The architecture includes one *autoencoder* and two LSTM *recurrent networks*.

The first preparation phase is feature extraction of musical units. Ten manually handcrafted features are considered, following a *bag-of-words* (BOW) approach (e.g., counts of a certain pitch class, counts of a certain pitch class rhythm tuple, if first note is tied to previous measure), resulting in 9675 actual features.

The key of the generation is the process of selection of a best (or at least, very good) successor candidate to a given musical unit. Two criteria are considered:

- *Successor semantic relevance* – It is based on a model of transition between units, as learnt by a LSTM recurrent network. In other words, that relevance is based on the distance to the (ideal) next unit as predicted by the model;
- *Concatenation cost* – It is based on another model of transition,[16] this time between the last note of the unit and the first note of the next unit, as learnt by another LSTM recurrent network.

The combination of the two criteria (illustrated in Fig. 9) is handled by a heuristic-based dynamic ranking process. As for a recurrent network, generation is iterated in order to create, unit by unit (measure by measure), an arbitrary length melody.

Note that the unit selection strategy actually provides *entry points* for control, as one may extend the selection

[16] At a more fine-grained level, note-to-note level, than the previous one.

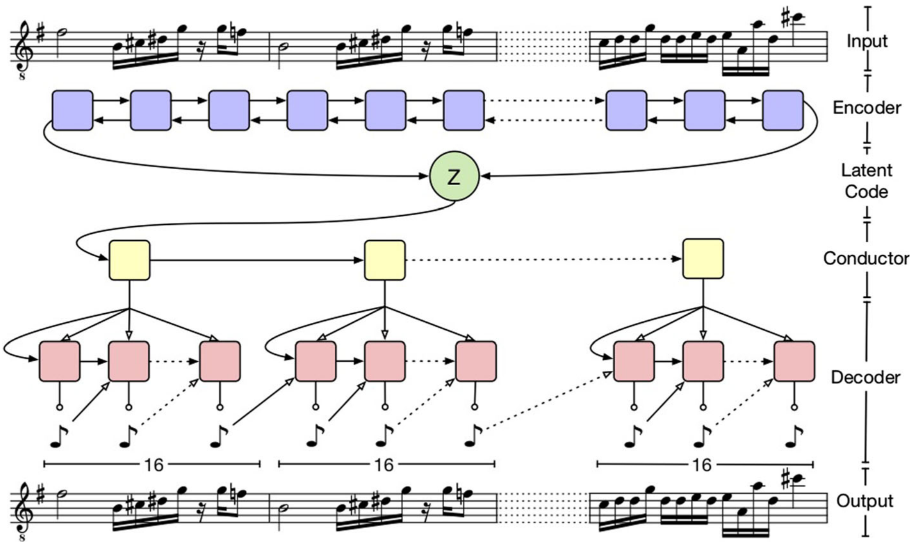

Fig. 10 MusicVAE architecture. Reproduced from [36] with permission of the authors

framework based on two criteria: successor semantic relevance and concatenation cost with user-defined constraints/criteria.

3 Structure

Another challenge is that most of the existing systems have a tendency to generate music with "no sense of direction." In other words, although the style of the generated music corresponds to the corpus learnt, the music lacks some *structure* and appears to wander without some higher organization, as opposed to human-composed music which usually exhibits some global organization (usually named a *form*) and identified components, such as:

- Overture, Allegro, Adagio or Finale for classical music;
- AABA or AAB in Jazz;
- Refrain, Verse or Bridge for songs.

Note that there are various possible levels of structure. For instance, an example of finer grain structure is at the level of melodic patterns that can be repeated, often transposed in order to adapt to a new harmonic structure.

Reinforcement (as used by RL-tuner in Sect. 2.5.1) and structure imposition (as used by C-RBM in Sect. 2.4.4) are approaches to enforce some constraints, possibly high level, onto the generation. An alternative top-down approach is followed by the unit selection strategy (see Sect. 2.6), by incrementally generating an abstract sequence structure and filling it with musical units, although the structure is currently flat. Therefore, a natural direction is to explicitly consider and process different levels (hierarchies) of temporality and of structure.

3.1 Example: MusicVAE multivoice generation

Roberts et al. propose a hierarchical architecture named MusicVAE [36] following the principles of a variational autoencoder encapsulating recurrent networks (RNNs, in practice LSTMs) such as VRAE introduced in Sect. 2.4.2, with two differences:

- the encoder is a bidirectional RNN;
- the decoder is a hierarchical two-level RNN composed of:
    - a high-level RNN named the Conductor producing a sequence of embeddings;
    - a bottom-layer RNN using each embedding as an initial state[17] and also as an additional input concatenated to its previously generated token to produce each subsequence.

The resulting architecture is illustrated in Fig. 10. The authors report that an equivalent "flat" (without hierarchy) architecture, although accurate in modeling the style in the case of 2-measure long examples, turned out inaccurate in the case of 16-measure long examples, with a 27% error increase for the autoencoder reconstruction. Some preliminary evaluation has also been conducted with a comparison by listeners of three versions: flat architecture, hierarchical architecture and real music for three types of music: melody, trio and drums, showing a very significant gain with the hierarchical architecture.

[17] In order to prioritize the Conductor RNN over the bottom-layer RNN, its initial state is reinitialized with the decoder-generated embedding for each new subsequence.

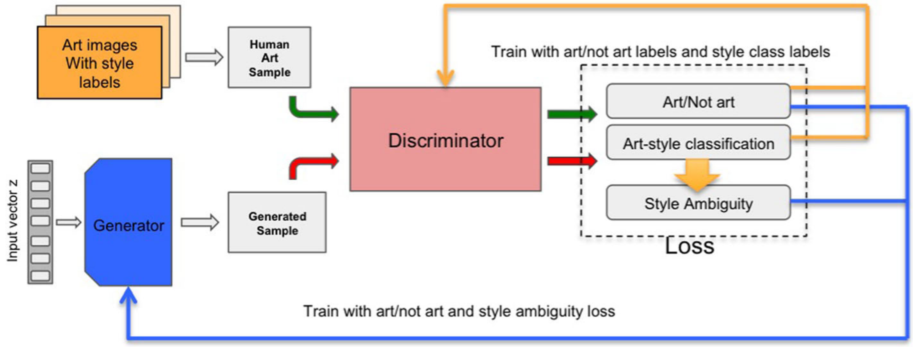

Fig. 11 Creative adversarial networks (CAN) architecture

4 Creativity

The issue of the *creativity* of the music generated is not only an artistic issue, but also an economic one, because it raises a *copyright issue*.[18]

One approach is *a posteriori*, by ensuring that the generated music is not too similar (e.g., in not having recopied a significant amount of notes of a melody) to an existing piece of music. To this aim, existing tools to detect similarities in texts may be used.

Another approach, more systematic but more challenging, is *a priori*, by ensuring that the music generated will not recopy a given portion of music from the training corpus.[19] A solution for music generation from Markov chains has been proposed [32]. It is based on a variable-order Markov model and constraints over the order of the generation through some min-order and max-order constraints, in order to attain some sweet spot between junk and plagiarism. However, there is none yet equivalent solution for deep learning architectures.

4.1 Conditioning

4.1.1 Example: MidiNet melody generation

The MidiNet architecture by Yang et al. [42], inspired by WaveNet (see Sect. 2.3.1), is based on generative adversarial networks (GAN) [16] (see Sect. 4.2). It includes a conditioning mechanism incorporating history information (melody as well as chords) from previous measures. The authors discuss two methods to control creativity:

- by restricting the conditioning by inserting the conditioning data only in the intermediate convolution layers of the generator architecture;
- by decreasing the values of the two control parameters of feature matching regularization, in order to less enforce the distributions of real and generated data to be close.

These experiments are interesting although the approach remains at the level of some ad hoc tuning of some hyper-parameters of the architecture.

4.2 Creative adversarial networks

Another more systematic and conceptual direction is the concept of creative adversarial networks (CAN) proposed by Elgammal et al. [9], as an extension of generative adversarial networks (GAN) architecture, by Goodfellow et al. [16], which trains simultaneously two networks:

- a *Generative model* (or *generator*) G, whose objective is to transform random noise vectors into faked *samples*, which resemble real samples drawn from a distribution of real images; and
- a *Discriminative model* (or *discriminator*) D, that estimates the probability that a sample came from the training data rather than from G.

The generator is then able to produce user-appealing synthetic samples (e.g., images or music) from noise vectors. The discriminator may then be discarded.

Elgammal et al. propose in [9] to extend a GAN architecture into a *creative adversarial networks* (CAN) architecture, shown in Fig. 11, where the generator receives from the discriminator not just one but *two signals*:

- the first signal, analog to the case of the standard GAN, specifies how the discriminator believes that the generated item comes from the training dataset of real art pieces;
- the second signal is about how easily the discriminator can classify the generated item into *established styles*. If there is some strong ambiguity (i.e., the various classes are equiprobable), this means that the generated item is difficult to fit within the existing art styles.

These two signals are thus contradictory forces and push the generator to explore the space for generating items that are at the same time close to the distribution of the existing art

[18] On this issue, see a recent paper [6].

[19] Note that this addresses the issue of avoiding a significant recopy from the training corpus, but it does not prevent *reinventing* an existing music outside of the training corpus.

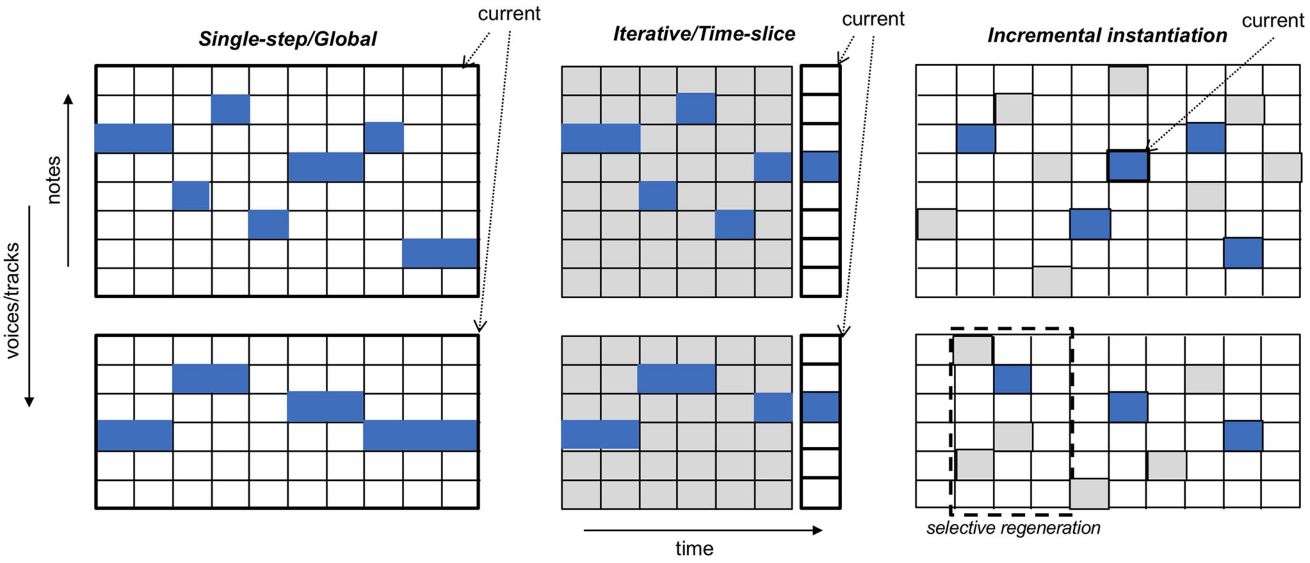

Fig. 12 Strategies for instantiating notes during generation

pieces and with some style originality. Note that this approach assumes the existence of a prior style classification and it also reduces the idea of creativity to exploring new styles (which indeed has some grounding in the art history).

5 Interactivity

In most of the existing systems, the generation is automated, with little or no *interactivity*. As a result, local modification and regeneration of a musical content is usually not supported, the only available option being a whole regeneration (and the loss of previous attempt). This is in contrast to the way a musician works, with successive partial refinement and adaptation of a composition.[20] Therefore, some requisites for interactivity are the incrementality and the locality of the generation, i.e., the way the variables of the musical content are instantiated.

5.1 Instantiation strategies

Let us consider the example of the generation of a melody. The two most common strategies (illustrated in Fig. 12)[21] for instantiating the notes of the melody are:

- *Single-step/global* – A global representation including all time steps is generated in a single step by a feedforward architecture. An example is DeepHear [38] in Sect. 2.4.1.
- *Iterative/time slice* – A time slice representation corresponding to a single time step is iteratively generated by a recurrent architecture (RNN). An example is Anticipation-RNN [18] in Sect. 2.3.2.

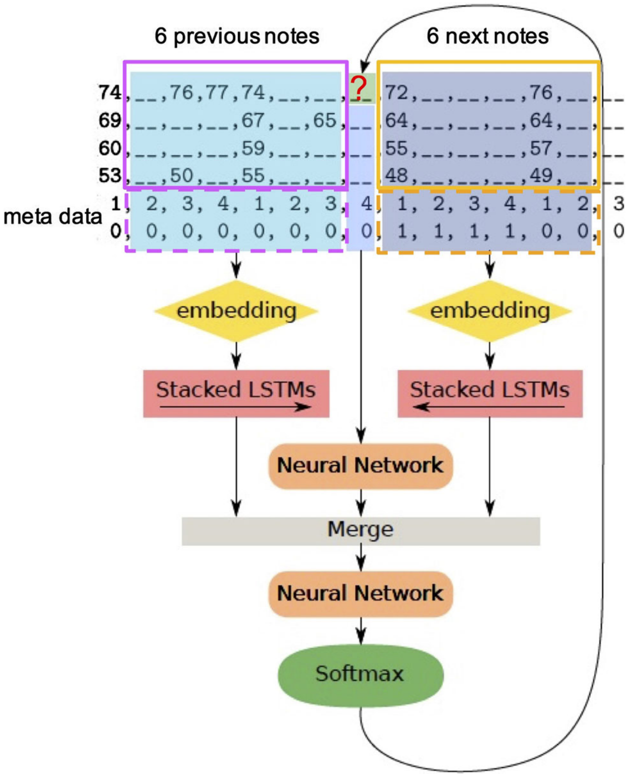

Fig. 13 DeepBach architecture

[20] An example of interactive composition environment is FlowComposer [33]. It is based on various techniques such as Markov models, constraint solving and rules.

[21] The representation shown is of type piano roll with two simultaneous voices (tracks). Parts already processed are in light gray; parts being currently processed have a thick line and are pointed as "current"; notes to be played are in blue.

Fig. 14 DeepBach incremental generation/sampling algorithm

```
Create four lists $V = (V_1; V_2; V_3; V_4)$ of length $L$;
Initialize them with random notes drawn from the ranges of the corresponding voices
for $m$ from 1 to $max\ number\ of\ iterations$ do
    Choose voice $i$ uniformly between 1 and 4;
    Choose time $t$ uniformly between 1 and $L$;
    Re-sample $V_i^t$ from $p_i(V_i^t | V_{\setminus i,t}, \theta_i)$
end for
```

Let us now consider an alternative strategy, *incremental variable instantiation*. It relies on a global representation including all time steps. But, as opposed to single-step/-global generation, generation is done *incrementally* by progressively instantiating and refining values of variables (notes), in a non-deterministic order. Thus, 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 voices* (shown as selective regeneration in Fig. 12), without regenerating the whole content.

5.2 Example: DeepBach chorale generation

This incremental instantiation strategy has been used by Hadjeres et al. in the DeepBach architecture [19] for generation of Bach chorales.[22] The architecture, shown in Fig. 13, 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 *backward* in time. Therefore, two recurrent networks (more precisely, LSTM) are used, one summing up past information and the other 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. The first 4 lines of the example data on top of Fig. 13 correspond to the 4 voices.[23] Actually, this architecture is replicated 4 times, one for each voice (4 in a chorale).

Training, as well as generation, is not done in the conventional way for neural networks. The objective is to predict the value of the current note for a given voice (shown with a red ? on top center of Fig. 13), using as information surrounding contextual notes. The training set is formed online 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 (shown in Fig. 14; see details in [19]) to produce a set of values (each note) of a polyphony, following the distribution that the network has learnt.

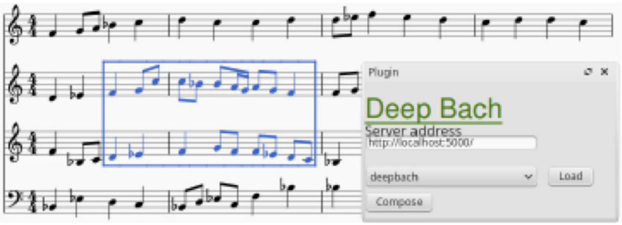

Fig. 15 DeepBach user interface

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 the corresponding counterpoint voices for these measures.

The user interface of DeepBach, shown in Fig. 15, allows the user to interactively select and control global or partial (re)generation of chorales. It opens up new ways of composing Bach-like chorales for non-experts in an interactive manner, similarly to what is proposed by FlowComposer for lead sheets [33]. It is implemented as a plug-in for the MuseScore music editor.

6 Conclusion

The use of deep learning architectures and techniques for the generation of music (as well as other artistic contents) is a growing area of research. However, there remain open challenges such as control, structure, creativity and interactivity, that standard techniques do not directly address. In this article, we have discussed a list of challenges, introduced some strategies to address them and have illustrated them through examples of actual architectures.[24] We hope that the analysis presented in this article will help in better understanding the issues and possible solutions and therefore may contribute to the general research agenda of deep learning-based music generation.

Acknowledgements We thank Gaëtan Hadjeres and Pierre Roy for related discussions. This research was partly conducted within the Flow Machines project which received funding from the European Research Council under the European Union Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 291156.

[22] J. S. Bach chose various given melodies for a soprano and composed the three additional ones (for alto, tenor and bass) in a *counterpoint* manner.

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

[24] A more complete survey and analysis is [2].

References

1. Bretan M, Weinberg G, Heck L (2016) A unit selection methodology for music generation using deep neural networks. arXiv:1612.03789v1
2. Briot JP, Hadjeres G, Pachet F (2018) Deep learning techniques for music generation. Computational synthesis and creative systems, Springer, London
3. Cope D (2000) The algorithmic composer. A-R Editions
4. Cun YL, Bengio Y (1998) Convolutional networks for images, speech, and time-series. In: The handbook of brain theory and neural networks. MIT Press, Cambridge, pp 255–258
5. Dai S, Zhang Z, Xia GG (2018) Music style transfer issues: a position paper. arXiv:1803.06841v1
6. Deltorn JM (2017) Deep creations: intellectual property and the automata. Front Digit Humanit 4:3
7. Doya K, Uchibe E (2005) The cyber rodent project: exploration of adaptive mechanisms for self-preservation and self-reproduction. Adapt Behav 13(2):149–160
8. Ebcioğlu K (1988) An expert system for harmonizing four-part chorales. Comput Music J 12(3):43–51
9. Elgammal A, Liu B, Elhoseiny M, Mazzone M (2017) CAN: creative adversarial networks generating "art" by learning about styles and deviating from style norms. arXiv:1706.07068v1
10. Fabius O, van Amersfoort JR (2015) Variational recurrent autoencoders. arXiv:1412.6581v6
11. Fernández JD, Vico F (2013) AI methods in algorithmic composition: a comprehensive survey. J Artif Intell Res 48:513–582
12. Fiebrink R, Caramiaux B (2016) The machine learning algorithm as creative musical tool. arXiv:1611.00379v1
13. Foote D, Yang D, Rohaninejad M (2016) Audio style transfer: do androids dream of electric beats? https://audiostyletransfer.wordpress.com
14. Gatys LA, Ecker AS, Bethge M (2015) A neural algorithm of artistic style. arXiv:1508.06576v2
15. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
16. Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozairy S, Courville A, Bengio Y (2014) Generative adversarial nets. arXiv:1406.2661v1
17. Graves A (2014) Generating sequences with recurrent neural networks. arXiv:1308.0850v5
18. Hadjeres G, Nielsen F (2017) Interactive music generation with positional constraints using anticipation-RNN. arXiv:1709.06404v1
19. Hadjeres G, Pachet F, Nielsen F (2017) DeepBach: a steerable model for bach chorales generation. arXiv:1612.01010v2
20. Herremans D, Chuan CH, Chew E (2017) A functional taxonomy of music generation systems. ACM Comput Surv 50(5):69
21. Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
22. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
23. Hofstadter D (2001) Staring Emmy straight in the eye—and doing my best not to flinch. In: Cope D (ed) Virtual music—computer synthesis of musical style. MIT Press, Cambridge, pp 33–82
24. Jaques N, Gu S, Turner RE, Eck D (2016) Tuning recurrent neural networks with reinforcement learning. arXiv:1611.02796
25. Kingma DP, Welling M (2014) Auto-encoding variational Bayes. arXiv:1312.6114v10
26. Lattner S, Grachten M, Widmer G (2016) Imposing higher-level structure in polyphonic music generation using convolutional restricted Boltzmann machines and constraints. arXiv: 1612.04742v2
27. Makris D, Kaliakatsos-Papakostas M, Karydis I, Kermanidis KL (2017) Combining LSTM and feed forward neural networks for conditional rhythm composition. In: Boracchi G, Iliadis L, Jayne C, Likas A (eds) Engineering applications of neural networks: 18th international conference, EANN 2017, Athens, Greece, Aug 25–27, 2017, Proceedings, Springer, London, pp 570–582
28. Mordvintsev A, Olah C, Tyka M (2015) Deep dream. https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html
29. Nierhaus G (2009) Algorithmic composition: paradigms of automated music generation. Springer, Berlin
30. Pachet F, Papadopoulos A, Roy P (2017) Sampling variations of sequences for structured music generation. In: Proceedings of the 18th international society for music information retrieval conference (ISMIR 2017), Suzhou, China, Oct 23–27, pp 167–173
31. Pachet F, Roy P, Barbieri G (2011) Finite-length markov processes with constraints. In: Proceedings of the 22nd international joint conference on artificial intelligence (IJCAI 2011). Barcelona, Spain, pp 635–642
32. Papadopoulos A, Roy P, Pachet F (2014) Avoiding plagiarism in Markov sequence generation. In: Proceedings of the 28th AAAI conference on artificial intelligence (AAAI 2014). Québec, PQ, Canada, pp 2731–2737
33. Papadopoulos A, Roy P, Pachet F (2016) Assisted lead sheet composition using FlowComposer. In: Rueher M (ed) Principles and practice of constraint programming: 22nd international conference, CP 2016, Toulouse, France, Sept 5–9, 2016, proceedings. Springer, London, pp 769–785
34. Papadopoulos G, Wiggins G (1999) AI methods for algorithmic composition: a survey, a critical view and future prospects. In: AISB 1999 symposium on musical creativity, pp 110–117 (1999)
35. Roberts A, Engel J, Raffel C, Hawthorne C, Eck D (2018) A hierarchical latent vector model for learning long-term structure in music. arXiv:1803.05428v2
36. Roberts A, Engel J, Raffel C, Hawthorne C, Eck D (2018) 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
37. Steedman M (1984) A generative grammar for Jazz chord sequences. Music Percept 2(1):52–77
38. Sun F (2017) DeepHear—composing and harmonizing music with neural networks. https://fephsun.github.io/2015/09/01/neural-music.html. Accessed 21 Dec 2017
39. Ulyanov D, Lebedev V (2016) Audio texture synthesis and style transfer. https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer/
40. van den Oord A, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A, Kavukcuoglu K (2016) WaveNet: a generative model for raw audio. arXiv:1609.03499v2
41. van Hasselt H, Guez A, Silver D (2015) Deep reinforcement learning with double Q-learning. arXiv:1509.06461v3
42. Yang LC, Chou SY, Yang YH (2017) 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