

Music Generation by Deep Learning – Challenges and Directions*

Jean-Pierre Briot[†]

François Pachet[‡]

[†] Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6, Paris, France
Jean-Pierre.Briot@lip6.fr

[‡] Spotify Creator Technology Research Lab, Paris, France
francois@spotify.com

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, 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.

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 [HOT06]. 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 [CB98], and LSTM (Long Short-Term Memory), which resolved inefficient training of recurrent neural networks [HS97];
2. availability of multiple *data sets*;
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 networks¹. 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 the *generation of content*: text, images, and *music*, the focus of this article.

*To appear in Special Issue on Deep learning for music and audio, Neural Computing & Applications, Springer Nature, 2018.

¹With many variants such as convolutional networks, recurrent networks, autoencoders, restricted Boltzmann machines, etc. [GBC16].

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 [Ste84] or rule-based music generation systems [Ebc88], 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 in [FC16], benefits are: 1) it can make creation feasible when the desired application is too complex to be described by analytical formulations or manual brute force design; 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 results², suffers from some limitations. We consider here³:

- *Control*, e.g., tonality conformance, maximum number of repeated notes, rhythm, etc.;
- *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 [BHP18]. In [HCC17], Herremans *et al.* propose a function-oriented taxonomy for various kinds of music generation systems. Examples of surveys about of AI-based methods for algorithmic music composition are by Papadopoulos and Wiggins [PW99] and by Fernández and Vico [FV13], as well as books by Cope [Cop00] and by Nierhaus [Nie09]. In [Gra14], Graves analyses the application of recurrent neural networks architectures to generate sequences (text and music). In [FC16], 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/section 2, structure/section 3, creativity/section 4 and interactivity/section 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 generation⁴, 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:

²Music difficult to distinguish from the original corpus.

³Additional challenges are analyzed in [BHP18].

⁴Two examples are Markov constraints [PRB11] and factor graphs [PPR17].

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

2.2 Sampling

Sampling a model⁵ 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 model⁶. 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 [MKPKK17];
- a *chord progression*, in the MidiNet architecture [YCY17];
- a *musical genre* or an *instrument*, in the WaveNet architecture [vdODZ⁺16];
- a set of *positional constraints*, in the Anticipation-RNN architecture [HN17].

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.* [vdODZ⁺16] is aimed at generating raw audio waveforms. The architecture is based on a convolutional feedforward network without pooling layer⁷. 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 if 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 [HN17] 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 [PRB11]. 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 architecture⁸, 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 Figure 1, with:

⁵The model can be stochastic, such as a restricted Boltzmann machine (RBM) [GBC16], 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 [BHP18].

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

⁷An 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 [vdODZ⁺16].

⁸Both are 2-layer LSTMs [HS97].

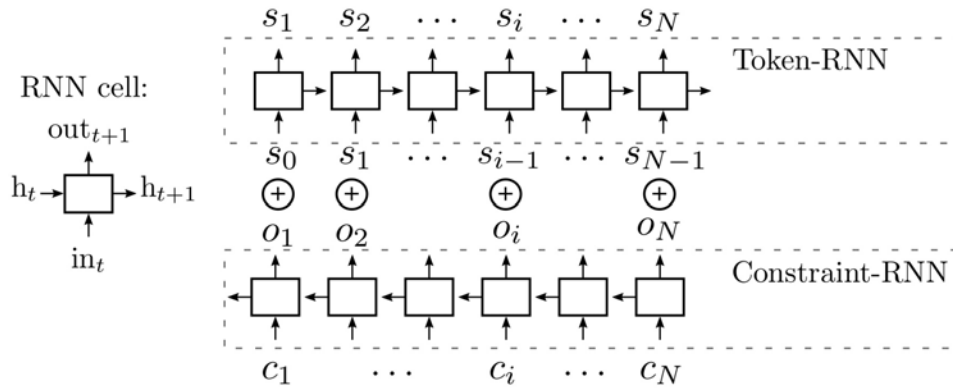


Figure 1: Anticipation-RNN architecture. Reproduced from [HN17] with permission of the authors



Figure 2: Examples of melodies generated by Anticipation-RNN. Reproduced from [HN17] with permission of the authors

- c_i is a *positional constraint*;
- o_i is the output at index i (after i iterations) of Constraint-RNN – it summarizes constraint informations 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 Figure 2. The model is indeed able to anticipate each positional constraint by adjusting its direction towards the target (lower-pitched or higher-pitched note).

2.4 Input Manipulation

The strategy of *input manipulation* has been pioneered for images by DeepDream [MOT15]. 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 [MOT15];
- *maximizing* the *similarity* to a given *target*, to create a *consonant melody*, in DeepHear [Sun17];
- *maximizing both* the *content similarity* to some initial image *and* the *style similarity* to a reference style image, to perform *style transfer* [GEB15];
- *maximizing* the *similarity of structure* to some reference music, to perform *style imposition* [LGW16].

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.

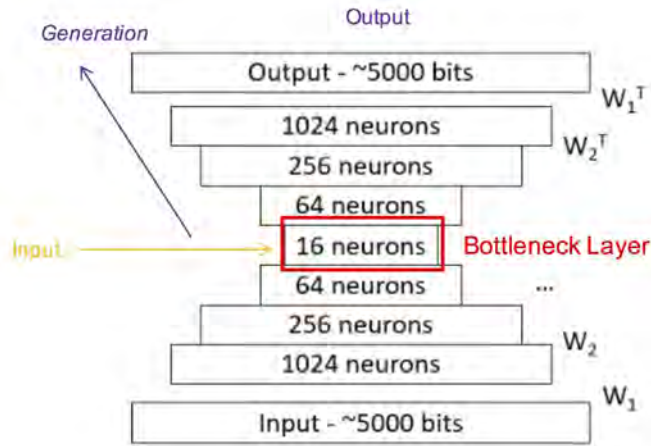


Figure 3: Generation in DeepHear. Extension of a figure reproduced from [Sun17] with permission of the author

2.4.1 Example 1: DeepHear Ragtime Melody Accompaniment Generation

The DeepHear architecture by Sun [Sun17] is aimed at generating ragtime jazz melodies. The architecture is a 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 trained⁹ 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 at Figure 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 already been learnt¹⁰. 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 autoencoders¹¹, such as for instance the VRAE (Variational Recurrent Auto-Encoder) architecture of Fabius and van Amersfoort [FvA15]. 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 Figure 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.* [GEB15] for images. The idea, summarized at Figure 5, is to use a deep learning architecture to independently capture:

- the features of a first image (named the *content*),

⁹Autoencoders 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.

¹⁰Note that this is a simple example of *transfer learning* [GBC16], with a same domain and a same training, but for a different task.

¹¹A variational autoencoder (VAE) [KW14] 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.

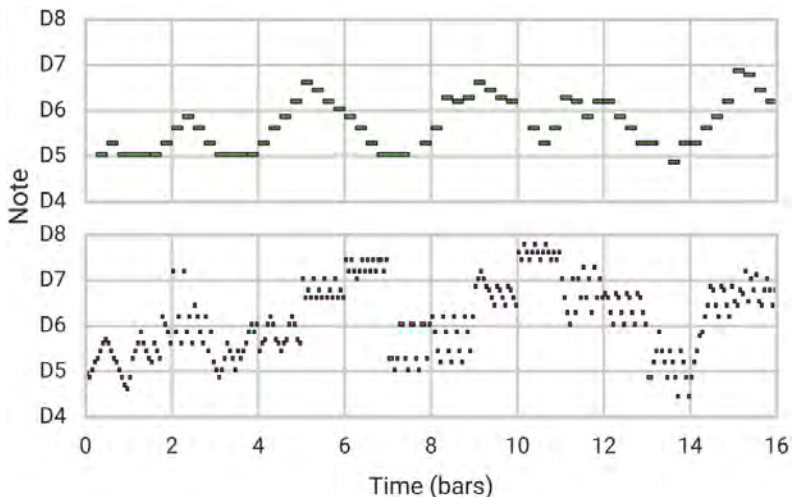


Figure 4: Example of melody generated (bottom) by MusicVAE by adding a “high note density” attribute vector to the latent space of an existing melody (top). Reproduced from [RER⁺18a] with permission of the authors

- 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* descriptions¹².

Transposing this style transfer technique to music was a natural direction and it has been experimented independently for audio, e.g., in [UL16] and [FYR16], 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.

2.4.4 Example 4: C-RBM Mozart Sonata Generation

The C-RBM architecture proposed by Lattner *et al.* [LGW16] 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 [Hof01].

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.

¹²Note 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 Figure 5.

¹³In 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., Figure 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.

¹⁴Note that this also some kind of style transfer [DZX18], although of a high-level structure and not a low-level timbre as in Section 2.4.3.

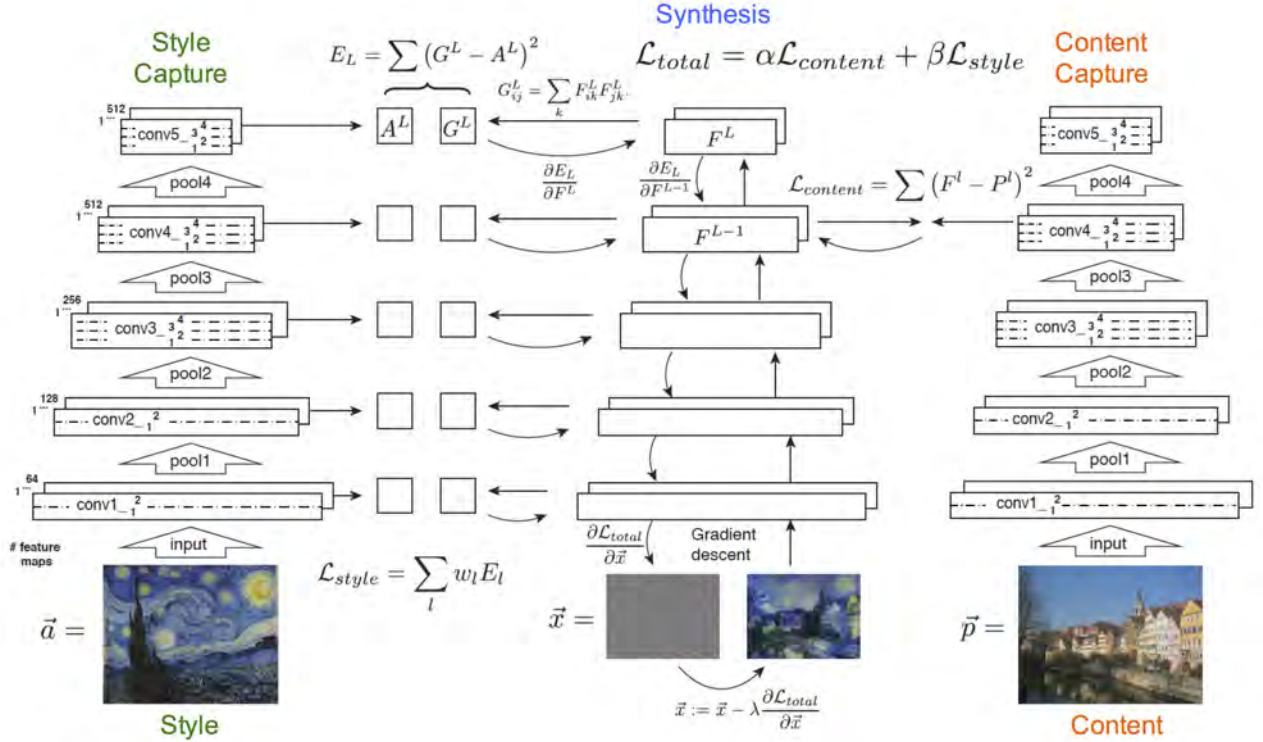


Figure 5: Style transfer full architecture/process. Reproduced with permission of the authors

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 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 of 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 [PRB11].

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 at Figure 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 Figure 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.

2.5.1 Example: RL-Tuner Melody Generation

The *reinforcement strategy* has been pioneered by the *RL-Tuner* architecture by Jaques *et al.* [JGTE16]. The architecture, illustrated at Figure 8, consists in two reinforcement learning architectures, named Q Network and

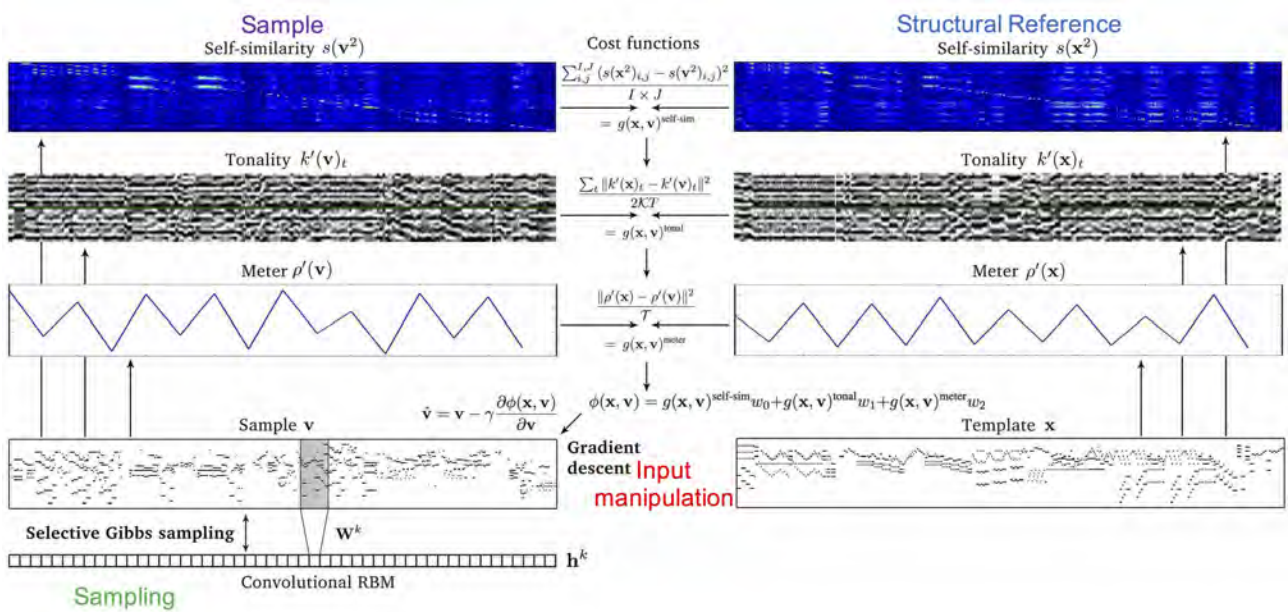


Figure 6: C-RBM Architecture

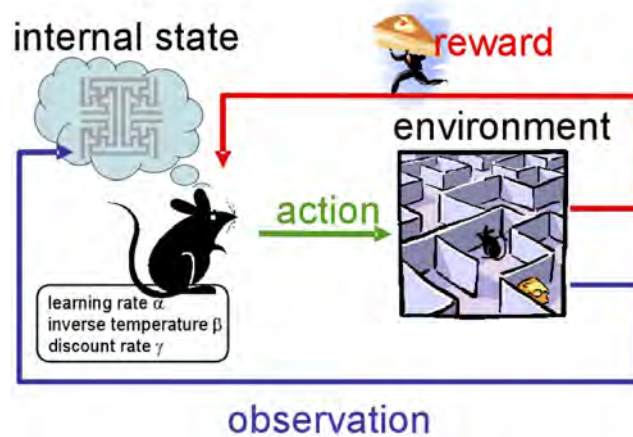


Figure 7: Reinforcement learning (Conceptual model) – Reproduced from [DU05]

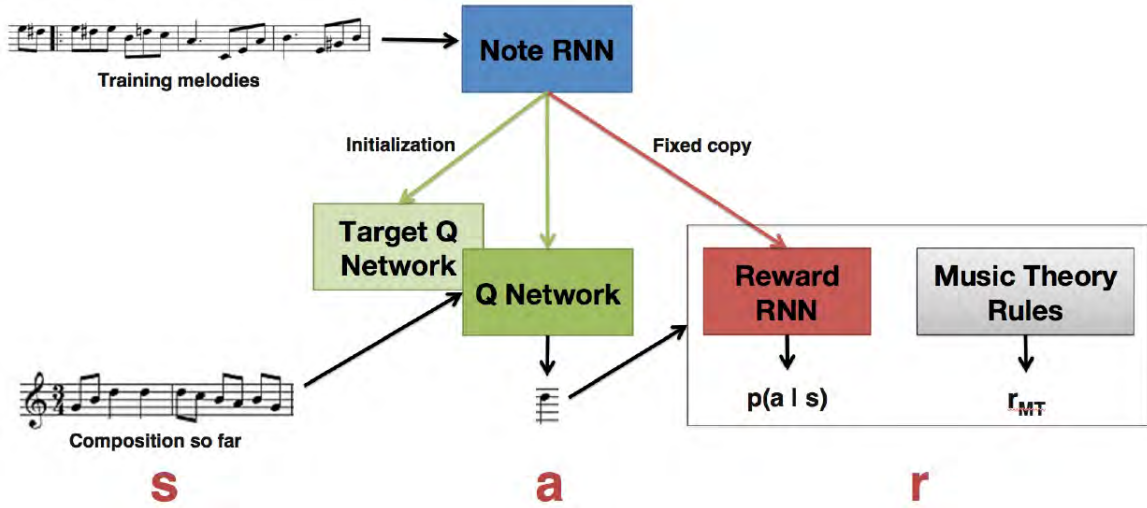


Figure 8: RL-Tuner architecture

Target Q Network¹⁵ 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 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 data base and in *concatenating* them in order to generate some sequence according to some user characteristics.

2.6.1 Example: Unit Selection and Concatenation Melody Generation

This strategy has been pioneered by Bretan *et al.* [BWH16] 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. 10 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, etc.), resulting in 9,675 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 transition¹⁶, 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 at Figure 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.

¹⁵They 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 [vHGS15].

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

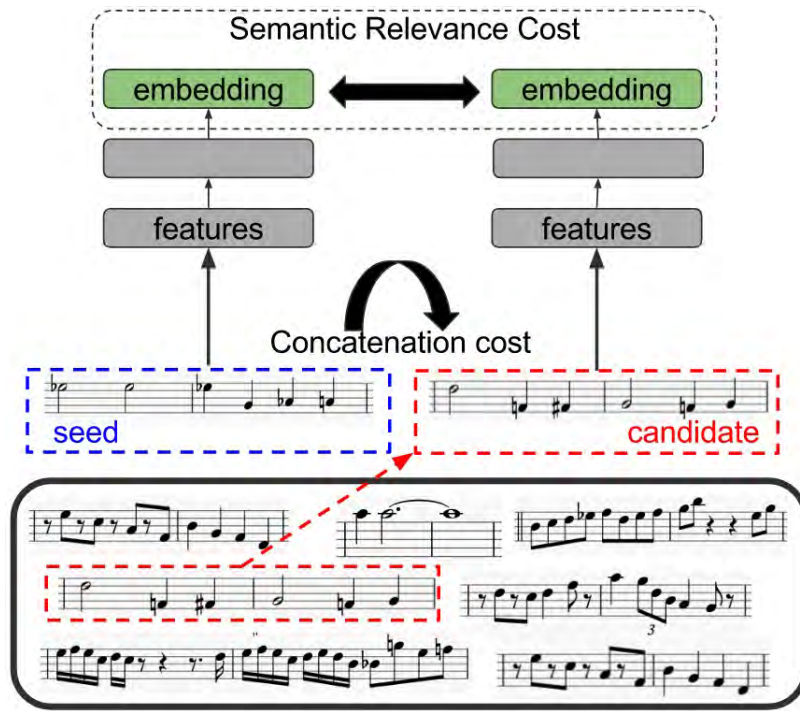


Figure 9: Unit selection based on semantic cost

Note that the unit selection strategy actually provides *entry points* for control, as one may extend the selection framework based on two criteria: successor semantic relevance and concatenation cost with user defined constraints/criteria.

3 Structure

Another challenge is that most 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 at Section 2.5.1) and structure imposition (as used by C-RBM at Section 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 Section 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 [RER⁺18b] following the principles of a variational autoencoder encapsulating recurrent networks (RNNs, in practice LSTMs) such as VRAE introduced at Section 2.4.2, with two differences:

- the encoder is a bidirectional RNN;
- the decoder is a hierarchical 2-level RNN composed of:

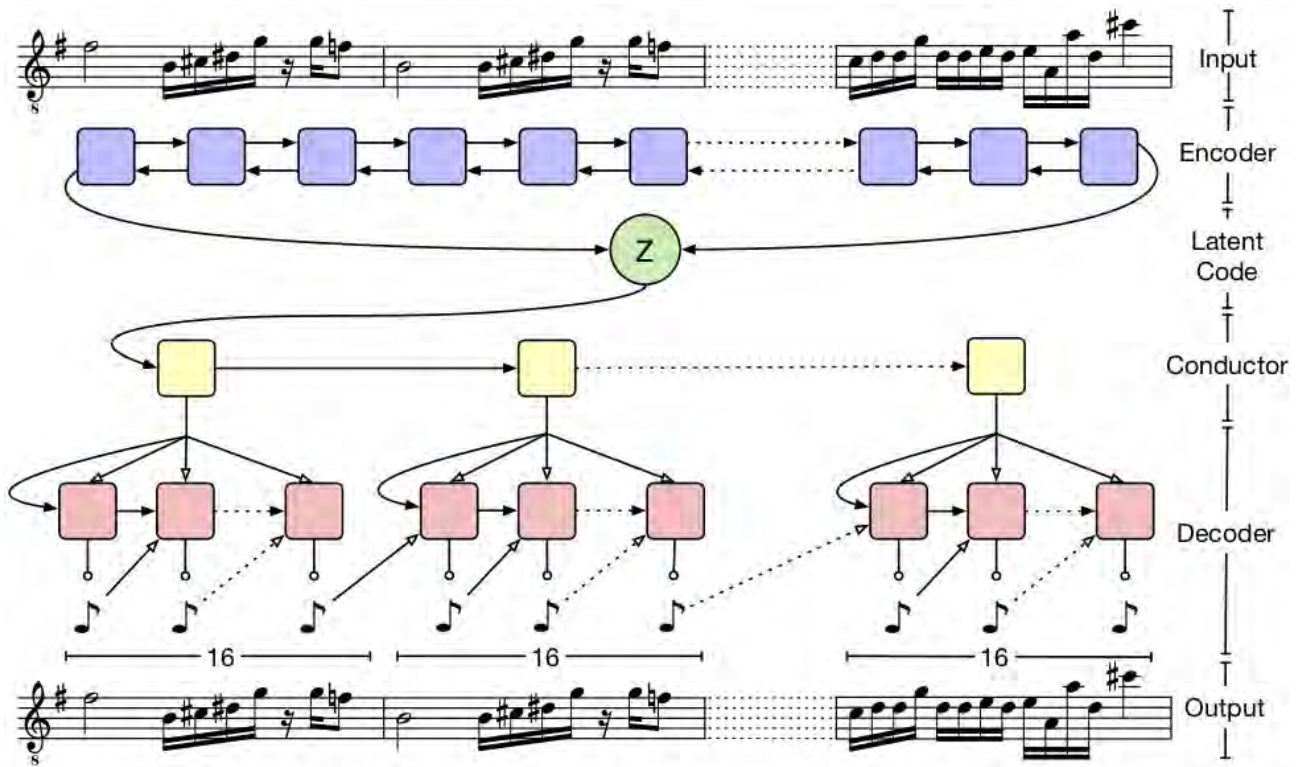


Figure 10: MusicVAE architecture. Reproduced from [RER⁺18b] with permission of the authors

- a high-level RNN named the Conductor producing a sequence of embeddings;
- a bottom-layer RNN using each embedding as an initial state¹⁷ and also as an additional input concatenated to its previously generated token to produce each subsequence.

The resulting architecture is illustrated at Figure 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.

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 corpus¹⁹. A solution for music generation from Markov chains has been proposed [PRP14]. 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.

¹⁷In 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.

¹⁸On this issue, see a recent paper [Del17].

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

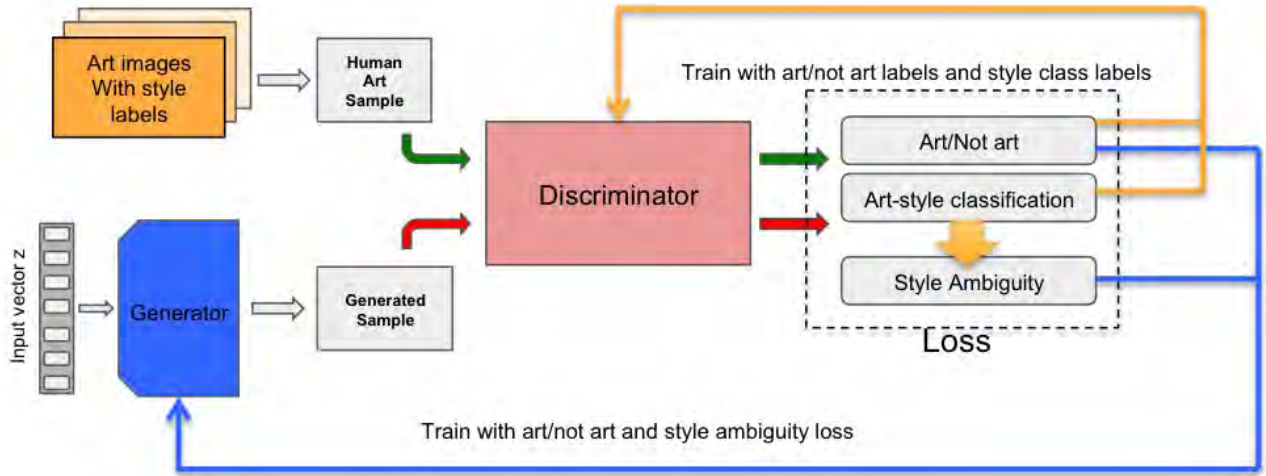


Figure 11: Creative adversarial networks (CAN) architecture

4.1 Conditioning

4.1.1 Example: MidiNet Melody Generation

The MidiNet architecture by Yang *et al.* [YCY17], inspired by WaveNet (see Section 2.3.1), is based on generative adversarial networks (GAN) [GPAM⁺14] (see Section 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 El Gammal *et al.* [ELEM17], as an extension of generative adversarial networks (GAN) architecture, by Goodfellow *et al.* [GPAM⁺14] 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 [ELEM17] to extend a GAN architecture into a *creative adversarial networks* (CAN) architecture, shown at Figure 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 existing art 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).

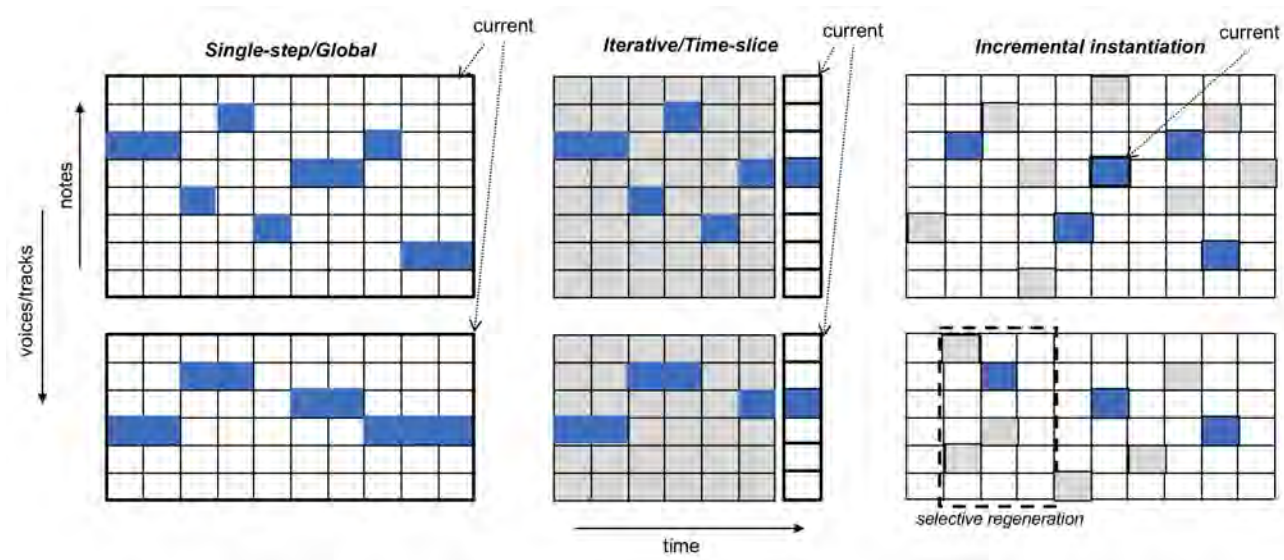


Figure 12: Strategies for instantiating notes during generation

5 Interactivity

In most of 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 composition²⁰. 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 at Figure 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 [Sun17] at Section 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 [HN17] at Section 2.3.2.

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 Figure 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 [HPN17] for generation of Bach chorales²². The architecture, shown at Figure 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 *backwards* in time. Therefore, two 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

²⁰An example of interactive composition environment is FlowComposer [PRP16]. It is based on various techniques such as Markov models, constraint solving and rules.

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

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

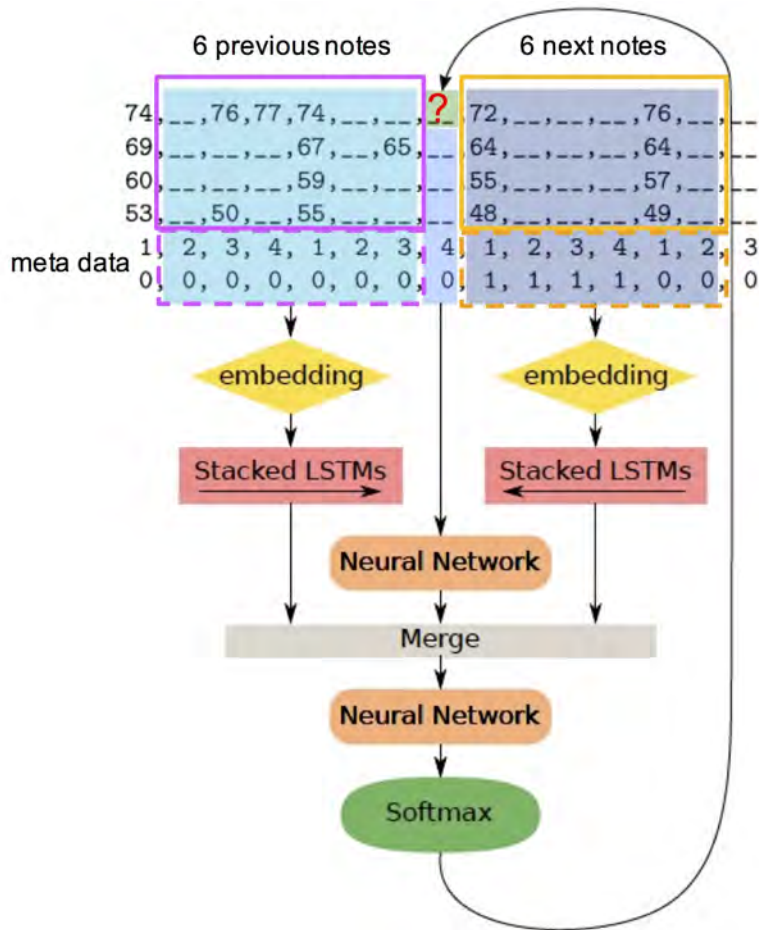


Figure 13: DeepBach architecture

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

Figure 14: DeepBach incremental generation/sampling algorithm

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 the Figure 13 correspond to the 4 voices²³. 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 current note for a given voice (shown with a red ? on top center of Figure 13), 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 (shown in Figure 14, see details in [HPN17]) 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 the corresponding counterpoint voices for these measures.

The user interface of DeepBach, shown at Figure 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 [PRP16]. It is implemented as a plugin for the MuseScore music editor.

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



Figure 15: DeepBach user interface

6 Conclusion

The use of deep learning architectures and techniques for the generation of music (as well as other artistic content) 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 architectures²⁴. We hope that the analysis presented in this article will help at a better understanding of 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 n. 291156.

References

- [BHP18] Jean-Pierre Briot, Gaëtan Hadjeres, and François Pachet. *Deep Learning Techniques for Music Generation*. Computational Synthesis and Creative Systems. Springer Nature, 2018.
- [BWH16] Mason Bretan, Gil Weinberg, and Larry Heck. A unit selection methodology for music generation using deep neural networks, December 2016. arXiv:1612.03789v1.
- [CB98] Yann Le Cun and Yoshua Bengio. Convolutional networks for images, speech, and time-series. In *The handbook of brain theory and neural networks*, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.
- [Cop00] David Cope. *The Algorithmic Composer*. A-R Editions, 2000.
- [Del17] Jean-Marc Deltorn. Deep creations: Intellectual property and the automata. *Frontiers in Digital Humanities*, 4, February 2017. Article 3.
- [DU05] Kenji Doya and Eiji Uchibe. The Cyber Rodent project: Exploration of adaptive mechanisms for self-preservation and self-reproduction. *Adaptive Behavior*, 13(2):149–160, 2005.
- [DZX18] Shuqi Dai, Zheng Zhang, and Gus Guangyu Xia. Music style transfer issues: A position paper, March 2018. arXiv:1803.06841v1.
- [Ebc88] Kemal Ebcioglu. An expert system for harmonizing four-part chorales. *Computer Music Journal (CMJ)*, 12(3):43–51, Autumn 1988.
- [ELEM17] Ahmed Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. CAN: Creative adversarial networks generating “art” by learning about styles and deviating from style norms, June 2017. arXiv:1706.07068v1.
- [FC16] Rebecca Fiebrink and Baptiste Caramiaux. The machine learning algorithm as creative musical tool, November 2016. arXiv:1611.00379v1.

²⁴A more complete survey and analysis is [BHP18].

- [FV13] Jose David Fernández and Francisco Vico. AI methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research (JAIR)*, (48):513–582, 2013.
- [FvA15] Otto Fabius and Joost R. van Amersfoort. Variational Recurrent Auto-Encoders, June 2015. arXiv:1412.6581v6.
- [FYR16] Davis Foote, Daylen Yang, and Mostafa Rohaninejad. Audio style transfer – Do androids dream of electric beats?, December 2016. <https://audiostyletransfer.wordpress.com>.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [GEB15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, September 2015. arXiv:1508.06576v2.
- [GPAM⁺14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozairy, Aaron Courville, and Yoshua Bengio. Generative adversarial nets, June 2014. arXiv:1406.2661v1.
- [Gra14] Alex Graves. Generating sequences with recurrent neural networks, June 2014. arXiv:1308.0850v5.
- [HCC17] Dorien Herremans, Ching-Hua Chuan, and Elaine Chew. A functional taxonomy of music generation systems. *ACM Computing Surveys (CSUR)*, 50(5), September 2017.
- [HN17] Gaëtan Hadjeres and Frank Nielsen. Interactive music generation with positional constraints using Anticipation-RNN, September 2017. arXiv:1709.06404v1.
- [Hof01] Douglas Hofstadter. Staring Emmy straight in the eye—and doing my best not to flinch. In David Cope, editor, *Virtual Music – Computer Synthesis of Musical Style*, pages 33–82. MIT Press, 2001.
- [HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006.
- [HPN17] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. DeepBach: a steerable model for Bach chorales generation, June 2017. arXiv:1612.01010v2.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [JGTE16] Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. Tuning recurrent neural networks with reinforcement learning, November 2016. arXiv:1611.02796.
- [KW14] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes, May 2014. arXiv:1312.6114v10.
- [LGW16] Stefan Lattner, Maarten Grachten, and Gerhard Widmer. Imposing higher-level structure in polyphonic music generation using convolutional restricted Boltzmann machines and constraints, December 2016. arXiv:1612.04742v2.
- [MKPKK17] Dimos Makris, Maximos Kaliakatsos-Papakostas, Ioannis Karydis, and Katia Lida Kermanidis. Combining LSTM and feed forward neural networks for conditional rhythm composition. In Giacomo Boracchi, Lazaros Iliadis, Chrisina Jayne, and Aristidis Likas, editors, *Engineering Applications of Neural Networks: 18th International Conference, EANN 2017, Athens, Greece, August 25–27, 2017, Proceedings*, pages 570–582. Springer Nature, 2017.
- [MOT15] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Deep Dream, 2015. <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>.
- [Nie09] Gerhard Nierhaus. *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer Nature, 2009.
- [PPR17] François Pachet, Alexandre Papadopoulos, and Pierre Roy. Sampling variations of sequences for structured music generation. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR 2017), Suzhou, China, October 23–27, 2017*, pages 167–173, 2017.
- [PRB11] François Pachet, Pierre Roy, and Gabriele Barbieri. Finite-length markov processes with constraints. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJ-CAI 2011)*, pages 635–642, Barcelona, Spain, July 2011.

- [PRP14] Alexandre Papadopoulos, Pierre Roy, and François Pachet. Avoiding plagiarism in Markov sequence generation. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*, pages 2731–2737, Québec, PQ, Canada, July 2014.
- [PRP16] Alexandre Papadopoulos, Pierre Roy, and François Pachet. Assisted lead sheet composition using FlowComposer. In Michel Rueher, editor, *Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, pages 769–785. Springer Nature, 2016.
- [PW99] George Papadopoulos and Geraint Wiggins. AI methods for algorithmic composition: A survey, a critical view and future prospects. In *AISB 1999 Symposium on Musical Creativity*, pages 110–117, April 1999.
- [RER⁺18a] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music, June 2018. arXiv:1803.05428v2.
- [RER⁺18b] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. 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, July 2018.
- [Ste84] Mark Steedman. A generative grammar for Jazz chord sequences. *Music Perception*, 2(1):52–77, 1984.
- [Sun17] Felix Sun. DeepHear – Composing and harmonizing music with neural networks, Accessed on 21/12/2017. <https://fephsun.github.io/2015/09/01/neural-music.html>.
- [UL16] Dmitry Ulyanov and Vadim Lebedev. Audio texture synthesis and style transfer, December 2016. <https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer/>.
- [vdODZ⁺16] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio, December 2016. arXiv:1609.03499v2.
- [vHGS15] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning, December 2015. arXiv:1509.06461v3.
- [YCY17] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. 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, October 2017.