Neural Computing and Applications (2020) 32:995–1005
https://doi.org/10.1007/s00521-018-3868-4

DEEP LEARNING FOR MUSIC AND AUDIO

# Anticipation-RNN: enforcing unary constraints in sequence generation, with application to interactive music generation

Gaëtan Hadjeres[1] · Frank Nielsen[1,2]

Received: 17 November 2017 / Accepted: 9 November 2018 / Published online: 20 November 2018

**Abstract**

Recurrent neural networks (RNNs) are now widely used on sequence generation tasks due to their ability to learn long-range dependencies and to generate sequences of arbitrary length. However, their left-to-right generation procedure only allows a limited control from a potential user which makes them unsuitable for interactive and creative usages such as interactive music generation. This article introduces a novel architecture called *anticipation-RNN* which possesses the assets of the RNN-based generative models while allowing to enforce user-defined unary constraints. We demonstrate its efficiency on the task of generating melodies satisfying unary constraints in the style of the soprano parts of the *J.S. Bach chorale harmonizations*. Sampling using the anticipation-RNN is of the same order of complexity than sampling from the traditional RNN model. This fast and interactive generation of musical sequences opens ways to devise real-time systems that could be used for creative purposes.

**Keywords** Automatic symbolic music generation · Recurrent neural networks · Interactive models · Unary constraints

## 1 Introduction

Recently, a number of powerful generative models on symbolic music have been proposed [14]. If they now perform well on a variety of different musical datasets, from monophonic folk music [29] to polyphonic Bach chorales [19], these models tend to face similar limitations: they do not provide musically interesting ways for a user to interact with them. Most of the time, only an input seed can be specified in order to condition the model upon: once the generation is finished, the user can only accept the result or regenerate another musical content. We believe that this restriction hinders creativity since the user does not play an active part in the music creation process.

Generation in these generative models is often performed from left to right; recurrent neural networks (RNNs) [10] are generally used to estimate the probability of generating the next musical event, and generation is done by iteratively sampling one musical event after another. This left-to-right modeling seems natural since music unfolds through time and this holds both for monophonic [7, 29] and polyphonic [5, 19] music generation tasks. However, this does not match real compositional principles since composition is mostly done in an iterative and non-sequential way [2]. As a simple example, one may want to generate a melody that ends on a specific note, but generating such melodies while staying in the learned style (the melodies are sampled with the correct probabilities) is in general a non-trivial problem when generation is performed from left to right. This problem has been solved when the generative model is a Markov model [22] but remains hard when considering arbitrary RNNs.

In order to solve issues raised by the left-to-right sampling scheme, approaches based on MCMC methods have been proposed, in the context of monophonic sequences with shallow models [25] or on polyphonic musical pieces using deeper models [12, 13]. If these MCMC methods allow to generate musically convincing sequences while enforcing many user-defined constraints, the generation process is generally order of magnitudes longer than the

✉ Gaëtan Hadjeres
gaetan.hadjeres@sony.com

[1] Sony CSL, Paris, France

[2] École Polytechnique, Palaiseau, France

simpler left-to-right generation scheme. This can prevent for instance using these models in real-time settings.

Another related approach is the one proposed in [18] where the authors address the problem of enforcing deterministic constraints on the output sequences. Their approach relies on performing a gradient descent on a regularized objective that takes into account the amount of constraints that are violated in the output sequence. They start from a pre-trained unconstrained model and then “nudge” its weights until it produces a valid sequence. If their model is able to handle a wide range of constraints (such as requiring the output sequence to belong to a context-free language), it enforces these constraints using a costly procedure, namely stochastic gradient descent (SGD). Sequences are generated using the deterministic argmax decoding procedure while our sampling scheme is non-deterministic, which we believe can be a desired feature in the context of interactive music generation. The approach of [17] is similar to the latter approach in the sense that the authors enforce constraints via gradient descent. However, since they rely on convolutional restricted Boltzmann machines, their sampling scheme is no longer deterministic. Their method is a way to sample polyphonic music having some imposed high-level structure (repetitions, patterns) which is imposed through the prescription of some predefined autocorrelation matrix.

The particularity of our approach is that it focuses on a smaller subset of constraints, namely unary constraints, which allows our sampling scheme to be faster since the proposed model takes into account the set of constraints during the training phase instead of the generation phase.

Except from the approaches cited above, the problem of generating sequences while enforcing user-defined constraints is rarely considered in the general machine learning literature but it is of crucial importance when devising interactive generative models. In this article, we propose a neural network architecture called *anticipation-RNN* which is capable of generating in the style learned from a database while enforcing user-defined unary constraints. Our model relies on two stacked RNNs, a *constraint-RNN* going from right to left whose aim is to take into account future constraints and a *token-RNN* going from left to right that generates the final output sequence. This architecture is very general and works with any RNN implementation. Furthermore, the generation process is fast as it only requires two neural network calls per musical event.

Even if the proposed architecture is composed of two RNNs going in opposite directions, it has not to be confused with the bidirectional-RNNs (BRNNs) architectures [26] which are commonly used to either summarize an entire sequence as in [24] or in the context of supervised learning [11]. Even if there have been attempts to use BRNNs in an unsupervised setting [4], these methods are intrinsically based on a MCMC sampling procedure which makes them much slower than our proposed method. The idea of integrating future information to improve left-to-right generation using RNNs has been considered in the Variational Bi-LSTM architecture [28] or in the Twin Networks architecture [27]. The aim of these architectures is to regularize the hidden states of the RNNs so that they better model the data distribution. If ideas could appear to be similar to the ones developed in this paper, these two approaches do not consider the generation of sequences under constraints but are a method to improve the existing RNNs architectures.

The plan for this article is the following: in Sect. 2, we precisely state the problem we consider and Sect. 3 describes the proposed architecture together with an adapted training procedure. Finally, we demonstrate experimentally the efficiency of our approach on the dataset of the *chorale melodies by J.S. Bach* in Sect. 4. In Sect. 5, we discuss about the generality of our approach and about future developments.

Code is available at https://github.com/Ghadjeres/Anticipation-RNN, and the musical examples presented in this article can be listened to on the accompanying Web site: https://sites.google.com/view/anticipation-rnn-examples/accueil.

## 2 Statement of the problem

We consider an i.i.d. dataset $\mathscr{D} := \{s = (s_1, \ldots, s_N) \in \mathscr{A}^N\}$ of sequences of tokens $s_t \in \mathscr{A}$ of arbitrary length $N$ over a vocabulary $\mathscr{A}$. We are interested in probabilistic models over sequences $p(s)$ such that

$$p(s) = \prod_t p(s_t | s_{<t}), \tag{1}$$

where

$$s_{<t} = \begin{cases} (s_1, \ldots, s_{t-1}) & \text{for t > 0} \\ \emptyset, & \text{if t = 0.} \end{cases} \tag{2}$$

This means that the generative model $p(s)$ over sequences is defined using the conditional probabilities $p(s_t|s_{<t})$ only. Generation with this generative model is performed iteratively by sampling $s_t$ from $p(s_t|s_{<t})$ for $t = 1\ldots N$ where $N$ is arbitrary. Due to their simplicity and their efficiency, recurrent neural networks (RNNs) are used to model the conditional probability distributions $p(s_t|s_{<t})$: they allow to reuse the same neural network over the different time steps by introducing a hidden state vector in order to summarize the previous observations we condition on. More precisely, by writing $f$ the RNN, $\mathrm{in}_t$ its input, $\mathrm{out}_{t+1}$ its output and $h_t$ its hidden state at time $t$, we have

$$\mathrm{out}_{t+1}, h_{t+1} = f(\mathrm{in_t}, \mathrm{h_t}) \tag{3}$$

for all time indices $t$. When $\mathrm{in}_t = s_t$, the vector $\mathrm{out}_{t+1}$ is used to define $p(s_{t+1}|s_{<t+1})$ for all time indices $t$ without the need to take as an input the entire sequence history $s_{<t+1}$.

If this approach is successful on many applications, such a model can only be conditioned on the past which prevents some possible creative use for these models: we can easily fix the beginning $s_{<t}$ of a sequence and generate a continuation $s_{\geq t} = (s_t, \ldots, s_N)$ but it becomes more intricate to fix the end $s_{\geq t}$ of a sequence and ask the model to generate a beginning sequence.

We now write $p-(s)$ the probability of a sequence $s$ when no constraint is set. For simplicity of notation, we will suppose that we only generate sequences of fixed length $N$ and denote by $\mathscr{S} := \mathscr{A}^N$ the set of all sequences over $\mathscr{A}$. The aim of this article is to be able to enforce any set $\mathscr{C}$ of unary constraints given by:

$$\mathscr{C} = \{(i, c_i)\}_{i \in I}, \tag{4}$$

where $I$ is the set of constrained time indexes and $c_i \in \mathscr{A}$ the value of the constrained note at time index $i$. Ideally, we want to sample constrained sequences

$$\mathscr{S}_+(\mathscr{C}) := \{s \in \mathscr{S},\ s_i = c_i\, \forall (i, c_i) \in \mathscr{C}\} \tag{5}$$

with the "correct" probabilities. If we denote by $p_+(s|\mathscr{C})$ the probability of a sequence $s$ in the constrained model conditioned on a set of constraints $\mathscr{C}$, this means that we want for all set of constraints $\mathscr{C}$:

$$p_+(s|\mathscr{C}) = 0, \quad \forall s \notin \mathscr{S}_+(\mathscr{C}), \tag{6}$$

and

$$p_+(s|\mathscr{C}) = \frac{1}{\alpha} p_-(s), \quad \forall s \in \mathscr{S}_+(\mathscr{C}), \tag{7}$$

where

$$\alpha := \sum_{s \in \mathscr{S}_+(\mathscr{C})} p_-(s).$$

To put it in words, each set of constraints $\mathscr{C}$ defines a subset $\mathscr{S}_+(\mathscr{C})$ of $\mathscr{S}$ from which we want to sample from using the probabilities (up to a normalization factor) given by $p_-$. However, sampling from $\mathscr{S}_+(\mathscr{C})$ using the acceptance-rejection sampling method is not efficient due to the arbitrary number of constraints. Exact sampling from $\mathscr{S}_+(\mathscr{C})$ is possible when the conditional probability distributions are modeled using models such as Markov models but is intractable in general. This problem in the case of Markov models can in fact be exactly solved when considering more complex constraints on the space of sequences such as imposing the equality or the difference between two sequences symbols $s_i$ and $s_j$. Generalizations of this problem to other types of constraints are discussed in Sect. 5.

## 3 The model

The problem when trying to enforce a constraint $c := (i, c_i)$ is that imposing such a constraint on time index $i$ "twists" the conditional probability distributions $p_-(s_t|s_{<t})$ for $t < i$. However, the direct computation of $p_-(s_t|s_{<t}, s_i = c_i)$ (using Bayes rule when only $p_-(s_t|s_{<t})$ is known) is computationally expensive.

The idea to overcome this issue is to introduce a neural network in order to summarize the set of constraints $\mathscr{C}$. To this end, we introduce an additional token NC (no constraint) to $\mathscr{A}$ indicating that no unary constraint is set at a given position. By doing this, we can rewrite the set $\mathscr{C}$ as a sequence $c = (c_1, \ldots, c_N)$ where $c_i \in \mathscr{A} \cup \{\mathtt{NC}\}$. We then introduce a RNN called *constraint-RNN* in order to summarize the sequence of all constraints. This RNN goes backward (from $c_N$ to $c_1$), and *all* its outputs are used to condition a second RNN called *token-RNN*.

This architecture, called *anticipation-RNN* since the token-RNN is conditioned on what may come next, is depicted in Fig. 1. We notated by $(o_1, \ldots, o_N)$ the output sequence of the constraint-RNN (for notational simplicity, we reversed the sequence numbering: the first output of the constraint-RNN is $o_N$ in our notation). The aim of the output vector $o_t$ is to summarize all information about constraints from time $t$ up to the end of the sequence. This vector is then concatenated to the input $s_{t-1}$ of the token-RNN at time index $t$ whose aim is to predict $s_t$.

Basically, this amounts to modeling the conditional probability distribution $p_+(s|c)$ using the following factorization:

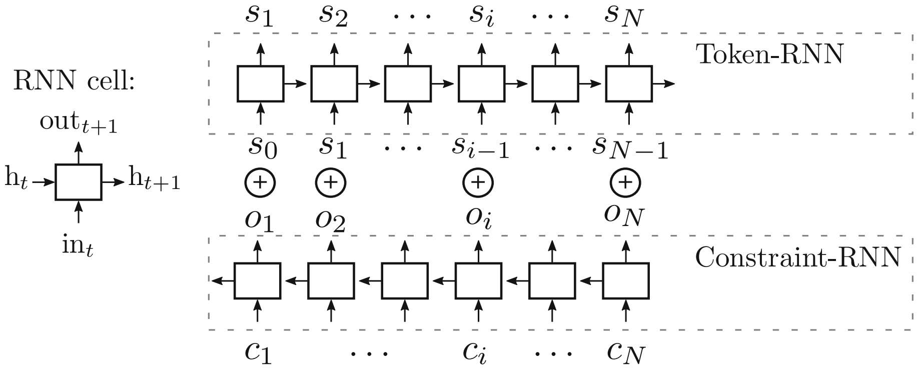
$$p_+(s|c) = \prod_t p_+(s_t|s_{<t}, c_{\geq t}) = \prod_t p_+(s_t|s_{<t}, o_t), \tag{8}$$

where $c_{\geq t}$ is defined similarly as in (2).

Our approach differs from the approaches using Markov models in the sense that we directly train the conditional probability distribution (8) rather than trying to sample sequences in $\mathscr{S}_+(\mathscr{C})$ using $p_-$: we want our probabilistic model to be able to directly enforce hard constraints.

The anticipation-RNN thus takes as an input both a sequence of tokens $(s_0, \ldots, s_{N-1})$ and a sequence of constraints $(c_1, \ldots, c_N)$ and has to predict the shifted sequence $(s_1, \ldots, s_N)$. The only requirement here is that the constraints have to be coherent with the sequence: $c_i = s_i$ if $c_i \neq \mathtt{NC}$. Since we want our model to be able to deal with any unary constraints, we consider the dataset of couples of token sequences and constraint sequences $\mathscr{D}_+$ such that

**Fig. 1** Anticipation-RNN architecture. The aim is to predict $(s_1, \ldots, s_N)$ given $(c_1, \ldots, c_N)$ and $(s_0, \ldots, s_{N-1})$


$s_1$ $s_2$ $\cdots$ $s_i$ $\cdots$ $s_N$
Token-RNN
RNN cell:
$\text{out}_{t+1}$
$\text{h}_t$
$\text{h}_{t+1}$
$\text{in}_t$
$s_0$ $s_1$ $\cdots$ $s_{i-1}$ $\cdots$ $s_{N-1}$
$\oplus$
$o_1$ $o_2$ $o_i$ $o_N$
Constraint-RNN
$c_1$ $\cdots$ $c_i$ $\cdots$ $c_N$

$$\mathscr{D}_+ := \left\{(s, m(s)),\ \forall s \in \mathscr{D}, \forall m \in \{0,1\}^N\right\}, \qquad (9)$$

where $\{0,1\}^N$ is the set of all binary masks: the sequence of constraints $m(s)$ is then defined as the sequence $(c_1, \ldots, c_N)$ where $c_i = s_i$ if $m_i = 1$ and $c_i = \texttt{NC}$ otherwise.

It is important to note that this model not only is able to handle unary constraints, but can also include additional metadata information about the sequence of tokens whose changes we have to anticipate. Indeed, by including such temporal information in the $c$ variables, this model can then learn to anticipate how to generate the tokens that will lead to a sequence complying with the provided metadata in a smooth way. These metadata can be musically relevant features such as the current key or mode, or the position of the cadences as it is done in [13].

This sampling procedure is fast since it only needs two RNN passes on the sequence. This modeling is thus particularly well suited for the real-time interactive generation of music. Furthermore, once the output of the constraint-RNN $o$ is computed, sampling techniques usually applied in sequence generation tasks such as beam search [6, 30] can be used without additional computing costs.

## 4 Experimental results

### 4.1 Dataset preprocessing

We evaluated our architecture on the dataset of the melodies from the four-part chorale harmonizations by J.S. Bach. This dataset is available in the music21 Python package [8], and we extracted the soprano parts from all 402 chorales that are in 4/4. In order to encode these monophonic sequences, we used the melodico-rhythmic encoding described in [13]. In this encoding, time is quantized using a sixteenth note as the smallest subdivision (each beat is divided into four equal parts). On each of these subdivisions, the real name of the note is used as a token if it is the subdivision on which the note is played; otherwise, an additional token denoted as "__" is used in order to indicate that the current note is held. A "rest" token is also used in order to handle rests. An example of an encoded melody using this encoding is displayed in Fig. 2.

The advantage of using such an encoding is that it allows to encode a monophonic musical sequence using only one sequence of tokens. Furthermore, it does not rely on the traditional MIDI pitch encoding but on the real note names: among other benefits, this allows to generate music sheets which are immediately readable and understandable by a musician and with no spelling mistakes. From a machine learning perspective, this has the effect of implicitly taking into account the current key and not throwing away this important piece of information. The model is thus more capable of generating coherent musical phrases. A simple example for this is that this encoding helps to distinguish between a $E\#$ and a $F$ by considering them as two different notes. Indeed, these two notes would appear in contexts that are in different keys (in $C\#$ major or $F\#$ minor in the first case, in $C$ major or $F$ major in the second case for instance).

We also perform data augmentation by transposing all sequences in all possible keys as long as the transposed sequence lies within the original voice range. We end up with an alphabet of tokens $\mathscr{A}$ of size 125.

### 4.2 Implementation details

We used a two-layer stacked LSTM [15] for both the constraint-RNN and the token-RNN using the PyTorch [23] deep learning framework. Both LSTM networks have 256 units, and the constraints tokens $c_i$ and the input tokens $s_i$ are embedded using the same embedding of size 20. Sequences are padded with START and END symbols so that the model can learn when to start and when to finish. We

```
D4 __ E4 __ A4 __ __ __ G4 __ F#4 __ E4 __ __ __
```

**Fig. 2** Melodico-rhythmic encoding of the first bar of the melody of Fig. 8a. Each note name such as $D4$ or $F\#4$ is considered as a single token

add dropout on the input and between the LSTM layers and discuss the effect of the choice of these hyperparameters in Sect. 4.3. We found that adding input on the input is crucial and set this value to 20%.

We fixed the length of the training subsequences to be 20-beat long which means that using the encoding described in Sect. 4.1, we consider sequences of tokens of size 80. The network is trained to minimize the categorical cross-entropy between the true token at position 40 and its prediction. For each training sequence, we sample the binary masks $m(s)$ of (9) by uniformly sampling a masking ratio $p \in [0, 1]$ and then setting each unary constraint with probability $p$.

We perform stochastic gradient descent using the Adam algorithm [16] using the default settings provided by PyTorch for ten epochs with a batch size of 256. In this setting, our best model achieves a validation accuracy of 92.9% with a validation loss of 0.22. These figures are of course highly dependent on our modeling choices such as the number of subdivisions per beat, the preprocessing of our corpus as well as the way we sampled the binary masks.

The sampling procedure is then done iteratively from left to right by sampling the token at time $t$ according to the probabilities given by $p_+(s_t|s_{<t}, o_t)$, where $s_{<t}$ is the sequence of previously generated tokens and $o_t$ the output of the constraint-RNN at position $t$.

## 4.3 Enforcing the constraints

We first check that the proposed architecture is able to enforce unary constraints; namely, that it fulfills the requirement (6).

In order to evaluate this property, we compute the amount of constraints that are enforced for various sets of constraints $\mathscr{C}$. We chose for these sets of constraints different "kinds" of constraints, from constraints that are in the "style of the corpus" to constraints that are totally "out-of-style." More precisely, we considered:

- $\mathscr{C}^1$: the beginning and the ending of an existing chorale melody (first five bars of the chorale melody "Wer nur den lieben Gott läßt walten" with two ablated bars),
- $\mathscr{C}^2$: the beginning and the ending of the same chorale melody, but where the ending has been transposed to a distant key (from $G$ minor to $C\#$ minor),
- $\mathscr{C}^3$: constraints forcing the model to make "big" leaps (chorale melodies tend to be composed of conjunct melodic motions),
- $\mathscr{C}^4$: a chromatic ascending scale,
- $\mathscr{C}^5$: random notes every eighth note,
- $\mathscr{C}^6$: the same random notes as above, but every quarter note.

These sets of unary constraints are displayed in Fig. 3.

We measure the influence of the amount of the dropout that we use in our models (dropout between the LSTM layers) on the following task: for each set of constraints $\mathscr{C}^i$ and for each model, we generate 1000 sequences using $p_+(.|\mathscr{C}^i)$ and compute the percentage of constrained notes that are sampled correctly. We report the results in Table 1.

These results show that for all sets of constraints that define a "possibly-in-style" musical constraint (constraint sets $\mathscr{C}^1$ to $\mathscr{C}^4$), the model manages to enforce the constraints efficiently: even if such constraints could not be encountered in the original dataset (constraint sets $\mathscr{C}^2$ and $\mathscr{C}^4$) . On the contrary, for truly out-of-style constraints (constraint sets $\mathscr{C}^5$ and $\mathscr{C}^6$), the model performs poorly on the task of enforcing these constraints. We do not think that it is a drawback of the model since its aim is to generate melodies in the style of the corpus which is made impossible when constrained with these incoherent constraints.

Table 1 also reveals the non-trivial effects of the choice of the amount of dropout of the models upon their performance on this task.

## 4.4 Anticipation capabilities

If the preceding section demonstrated that the anticipation-RNN architecture is able to enforce a wide variety of sets of unary constraints, we will explore in this section the role of the constraint-RNN and in particular how it is able to learn how to "propagate" the constraints backward, making the token-RNN able to anticipate what will come next.

For this, we will evaluate how the constrained model deviates from the unconstrained model. We compare the constrained model $p_+$ on the same set of constraints $\mathscr{C}$ with its unconstrained counterpart $p_-$. The latter is obtained by conditioning the model of Fig. 1 on a sequence of constraints in the special case where no constraint is set: the sequence of constraints is $(\texttt{NC}, \ldots, \texttt{NC})$.

More precisely, for a set of constraints $\mathscr{C}$, we quantify how the probability distributions $p_+(.|s_{<t}, \mathscr{C})$ differ from the probability distributions $p_-(.|s_{<t})$ by computing how dissimilar they are. We chose as a measure of dissimilarity [3, 9] the Jensen–Shannon divergence [1, 20] which is defined by:

$$\mathrm{JS}(p||q) := \frac{1}{2}\mathrm{KL}(p||m) + \frac{1}{2}\mathrm{KL}(q||m), \tag{10}$$

where $m = \frac{p+q}{2}$, with KL denoting the Kullback–Leibler divergence

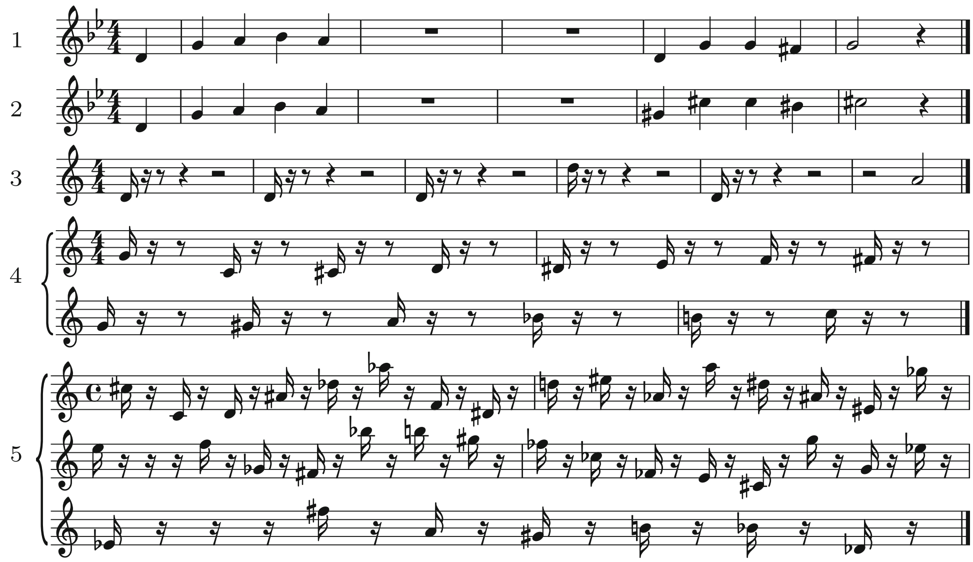


**Fig. 3** Sets of constraints $\mathscr{C}^i$ described in Sect. 4.3, for $i$ ranging from 1 to 5. In this particular figure, rests denote the absence of constraints

**Table 1** Percentage of correctly sampled constrained notes for different models $p_+$ differing only by the amount of dropout they use and constrained on different sets of constraints

| | $\mathscr{C}^1$ | $\mathscr{C}^2$ | $\mathscr{C}^3$ | $\mathscr{C}^4$ | $\mathscr{C}^5$ | $\mathscr{C}^6$ |
|---|---|---|---|---|---|---|
| LSTM dropout = 0.2, dropout on input = 0.2 | 99.78 | 99.73 | 98.78 | 99.77 | 41.28 | 57.06 |
| LSTM dropout = 0.5, dropout on input = 0.2 | 99.90 | 99.01 | 99.32 | 98.68 | 43.83 | 62.08 |

$$\mathrm{KL}(p||q) := \sum_i p_i \log\left(\frac{p_i}{q_i}\right). \tag{11}$$

The Jensen–Shannon divergence has the property of being symmetric, bounded (and thus always definite), and its square root satisfies the triangle inequality [21] which is an important feature compared to other divergences.

In Fig. 4, we plot the evolution of the Jensen-Shannon divergence between the two distributions $p_+(.|s_{<t}, \mathscr{C})$ and $p_-(.|s_{<t})$ during generation for different sets of constraints $\mathscr{C}$. We generated 1000 sequences using the constrained model and computed the average Jensen–Shannon divergence between the two models for each time step. We then averaged the values over each beat in order not to take into account the intra-beat variations. Indeed, due to encoding we chose as well as to the singularity of the musical data we considered, patterns of oscillations appear. Indeed, both models agree in putting much of their probability mass on

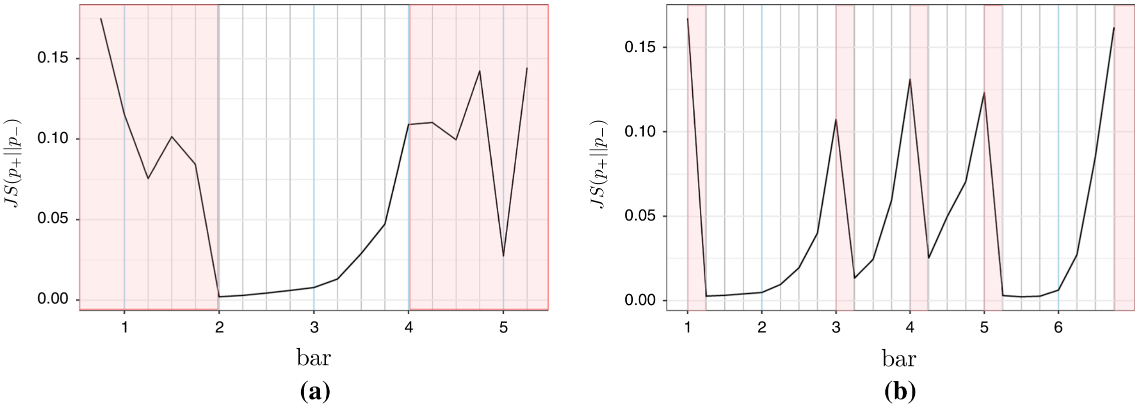


**Fig. 4** Plot of the evolution of the Jensen–Shannon divergence JS $(p_+(s_t|s_{i<t}, \mathscr{C}^i)||p_-(s_t|s_{i<t}))$ between the constrained model $p_+$ and the unconstrained model $p_-$ during generation for two sets of constraints $\mathscr{C}^1$ and $\mathscr{C}^3$. Each point represents the average value of the divergence over one beat. The location of the constraints has been highlighted in red. **a** Set of constraints $\mathscr{C}^1$. **b** Set of constraints $\mathscr{C}^3$ (color figure online)

the hold symbol "__" on the second sixteenth note of each beat since the soprano parts in Bach chorales are mostly composed of half notes, quarter notes and eighth notes. This is independent of the presence or absence of constraints so the constrained and unconstrained models make similar predictions on these time steps resulting in a low divergence.

This plot confirms that the constraints are propagated backward in time and the token-RNN is not only able to enforce constraints but also able to anticipate how to do so.

We now illustrate this feature on a specific example. Figure 5 shows the evolution of $p_+(s_t|s_{<t}, \mathscr{C}^3)$ and $p_-(s_t|s_{<t})$ during generation. It is interesting to note that the conditional probability distributions returned by $p_+(s_t|s_{<t}, \mathscr{C}^3)$ are more concentrated on specific values than the ones returned by $p_-(s_t|s_{<t})$. The concentration of the all probability mass of $p_+(s_t|s_{<t}, \mathscr{C}^3)$ on constrained notes confirms, on this specific example, that the proposed architecture has learned to enforce hard unary constraints.

In order to understand the effect of the constraints, we plot the difference between the two distributions of Fig. 5 for each time step in Fig. 6. This highlights the fact that the probability mass distribution of $p_+$ is "shifted upward" few beats in advance when the next unary constraint is higher than the current note and "downward" in the opposite case.

### 4.5 Sampling with the correct probabilities

We now evaluate that the sampling using $p_+$ fulfills the requirement (7). This means that for any set of constraints $\mathscr{C}$, the ratio between the probabilities of two sequences in $\mathscr{S}_+(\mathscr{C})$ is identical if probabilities are computed using the unconstrained model $p_-(.)$ or if they are computed using the constrained model $p_+(.|\mathscr{C})$. We introduce the set of constraints $\mathscr{C}^0$ consisting of a single constrained note.

For a given set of constraints $\mathscr{C}$, we generated 500 sequences and verified that the requirement (6) is fulfilled for all of these sequences (i.e. all constraints are enforced). In order to check the fulfillment of the requirement (7), we plot for each sequence $s$ its probability in the constrained model $p_+(s)$ (defined as in Eq. 1) as a function of $p_-(s)$ in logarithmic space. We compute these probabilities using (8), but only keep the time steps on which notes are not constrained. The resulting plots are shown in Fig. 7. Table 2 quantifies to which amount the two distributions are proportional on the subsets $\mathscr{S}_+(\mathscr{C}^i)$ for different sets of constraints $\mathscr{C}^i$ and for different models.

The translation in logarithmic space indicates the proportionality between the two distributions as desired.

The conclusion is that our model is able to correctly enforce all constraints for sets of constraints that are plausible with respect to the training dataset (Sect. 4.3) and that on these specific sets of constraints, our sampling procedure respects the relative probabilities between the sequences. In other words, the anticipation-RNN is able to sample with the correct probabilities a subset of sequences defined by a set of unary constraints.

### 4.6 Musical examples

We end this section with the discussion over some generated constrained sequences. Figure 8 shows examples of the enforcement and the propagation of the constraints for the set of constraints $\mathscr{C}^3$: even if generation is done from left to right, the model is able to generate compelling musical phrases while enforcing the constraints. In particular, we see that the model is able to "anticipate" the moment when it has to "go" from a low-pitched note to a high-pitched one and vice versa. The use of the melodico-rhythmic encoding allows to only impose that a note should be played at a given time, without specifying its rhythm. It is interesting to note that such a wide melodic contour (going from a D4 to a D5 and then going back to a D4 in only two bars) is unusual for a chorale melody. Nonetheless, the proposed model is able to generate a convincing Bach-like chorale melody. The three displayed examples show that there is a great variability in the generated solutions: even when constrained on the same set of constraints, the generated melodies have distinct characteristics such as, for example, the key they are in or where cadences could be.

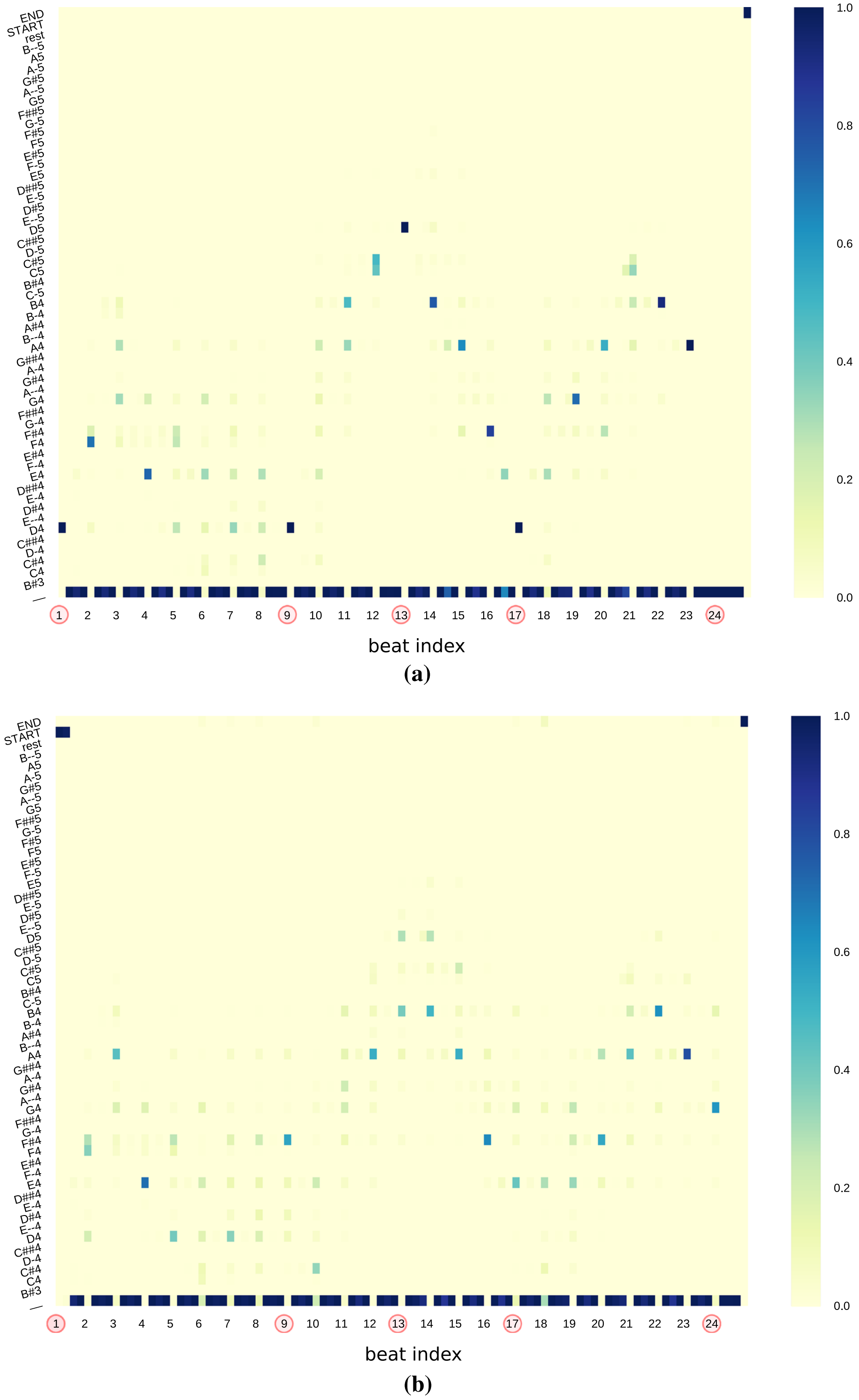
Similarly to [13], we provide a plug-in for the MuseScore music score editor which allows to call the anticipation-RNN in an intuitive way.

## 5 Conclusion

We presented the anticipation-RNN, a simple but efficient way to generate sequences in a learned style while enforcing unary constraints. This method is general and can be used to improve many existing RNN-based generative models. Contrary to other approaches, we teach the model to learn to enforce hard constraints at training time. We believe that this approach is a first step toward the generation of musical sequences subjected to more complex constraints.

The constrained generation procedure is fast since it requires only $2N$ RNN calls, where $N$ is the length of the generated sequence; as it does not require extensive computational resources and provides an interesting user–machine interaction, we think that this architecture paves the way to the development of creative real-time composition software. We also think that this fast sampling could be

**Fig. 5** Plot of $p(s_t|s_{<t})$ as a function of $t$ during the generation of the melody displayed in Fig. 8a in the constrained and unconstrained cases. Beats on which a constraint is set are circled. **a** Constrained case: $p = p_+(.|\mathscr{C}^3)$. **b** Unconstrained case: $p = p_-$



**(a)**

**(b)**

used jointly with MCMC methods in order to provide fast initializations.

Our approach can be seen as a general way to condition RNN models on time-dependent metadata. Indeed, the variable $c$ in (8) is not only restricted to the value of the

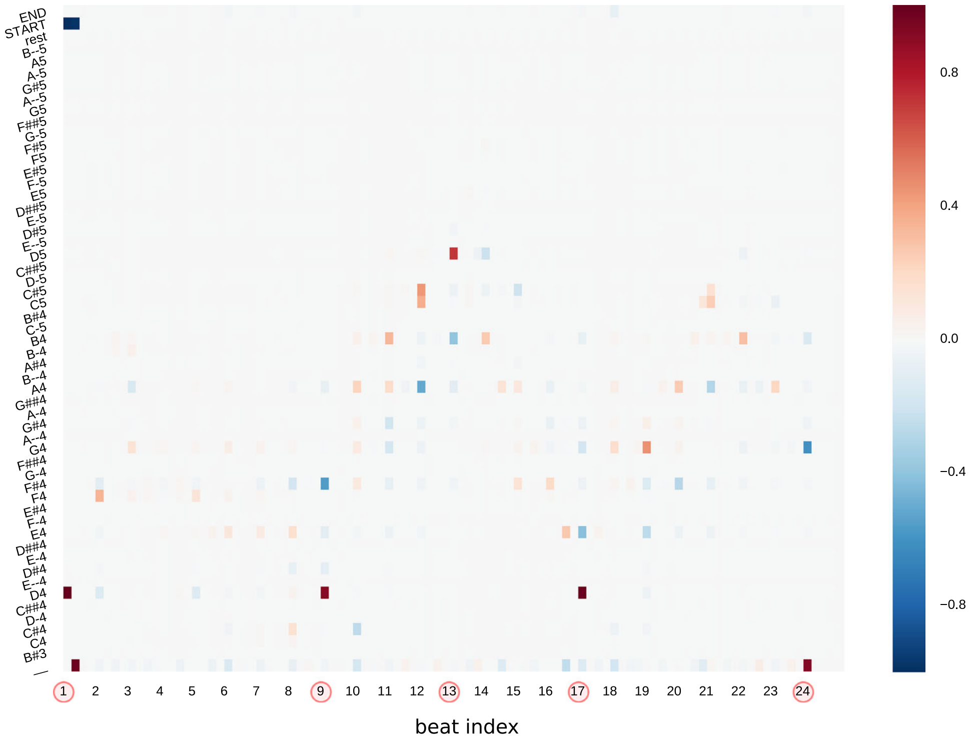


**Fig. 6** Difference between $p_+(s_t|s_{<t})$ and $p_-(s_t|s_{<t})$ as a function of $t$ during the generation of the melody displayed in Fig. 8a. Beats on which a constraint is set are circled. We see that between beats 9–13, the probability mass of the constrained model $p_+$ is shifted upward (compared to the probability distribution given by the unconstrained model $p_-$) in order to enforce the unary constraint $D5$ set at beat 13. From beats 13–17, the situation is reversed: the probability mass of the constrained model $p_+$ is shifted downward in order to enforce the unary constraint $D4$ on beat 17

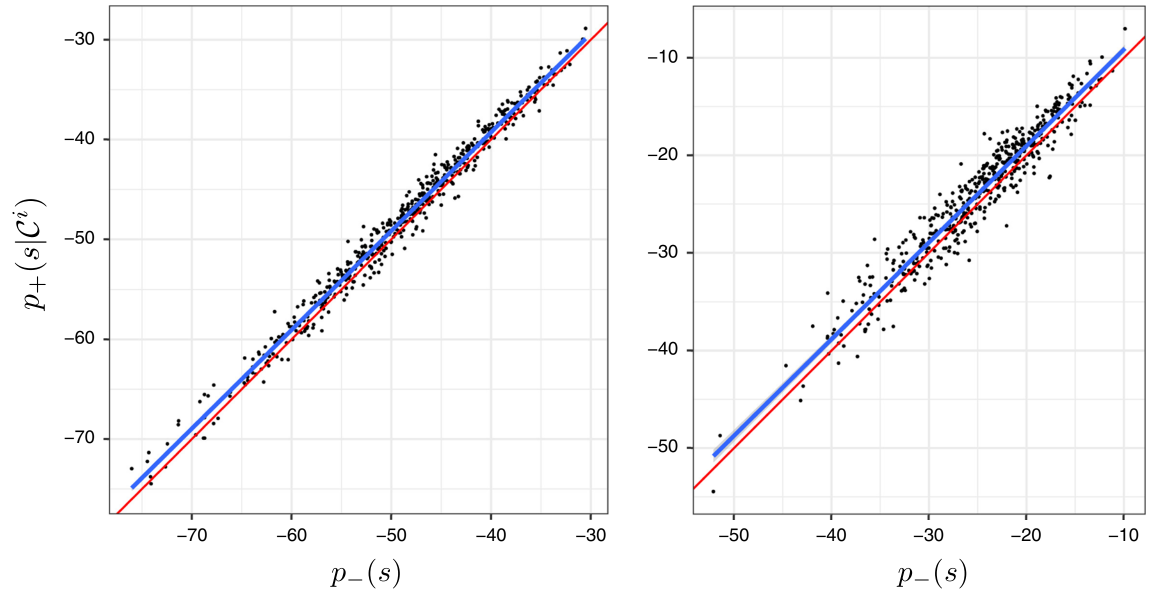


**Fig. 7** Point plots in logarithmic scale of $\ln p_+(s|\mathscr{C}^i)$ ($y$-axis) versus $p_-(s)$ ($x$-axis) on a set of 500 sequences generated using $p_+(s|\mathscr{C}^i)$, for $\mathscr{C}^0$ and $\mathscr{C}^3$. The identity map is displayed in red and the linear regression of the data points in blue. The lines are closed to being parallel indicating the proportionality between the two distributions (color figure online)

**Table 2** Slopes of the linear interpolations displayed in Fig. 7 for different models and different sets of constraints $\mathscr{C}^i$

| | $\mathscr{C}^0$ | $\mathscr{C}^1$ | $\mathscr{C}^2$ | $\mathscr{C}^3$ |
|---|---|---|---|---|
| LSTM dropout = 0.2, dropout on input = 0.2 | 0.99 | 0.99 | 1.05 | 0.96 |
| LSTM dropout = 0.5, dropout on input = 0.2 | 0.99 | 0.93 | 1.00 | 0.92 |

The closer the values are to one, the better the requirement (7) is achieved

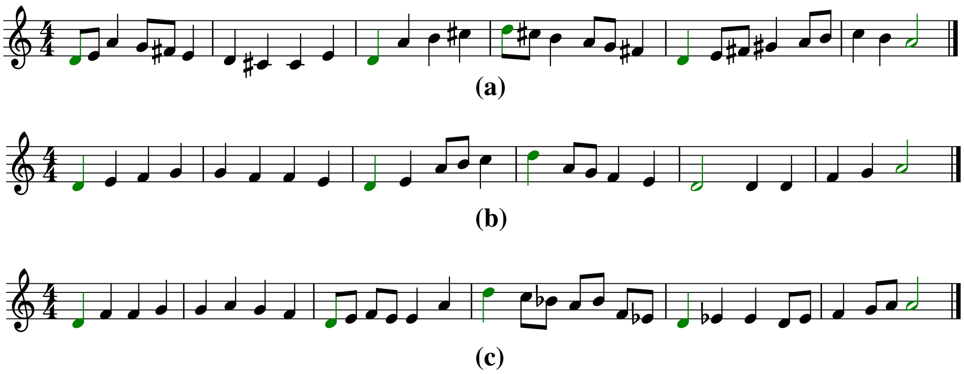


**Fig. 8** Examples of generated sequences in the style of the soprano parts of the J.S. Bach chorales. All examples are subject to the same set of unary constraints $\mathcal{C}^3$ which is indicated using green notes (color figure online)

unary constraints, but can contain more information such as the location of the cadences or the current key. We successfully applied the anticipation-RNN in this setting and report that it manages to enforce these interesting and natural musical constraints in a smooth way while staying in the style of the training corpus.

Future work will aim at handling other types of constraints (imposing the rhythm of the sequences, enforcing the equality between two notes or introducing soft constraints) and developing responsive user interfaces so that all the possibilities offered by this architecture can be used by a wide audience.

**Acknowledgements** This work was initiated while the first author was funded by a Ph.D. scholarship of École Polytechnique (AMX).

## Compliance with ethical standards

**Conflict of interest** The authors declare that they have no conflict of interest.

## References

1. Amari Si, Nagaoka H (2007) Methods of information geometry, vol 191. American Mathematical Society, Providence
2. Balmer Y, Lacôte T, Murray CB (2016) Messiaen the borrower: recomposing Debussy through the deforming prism. J Am Musicol Soc 69(3):699–791. https://doi.org/10.1525/jams.2016.69.3.699
3. Basseville M (1989) Distance measures for signal processing and pattern recognition. Signal Process 18(4):349–369
4. Berglund M, Raiko T, Honkala M, Kärkkäinen L, Vetek A, Karhunen J (2015) Bidirectional recurrent neural networks as generative models—reconstructing gaps in time series. ArXiv e-prints 1504:01575
5. Boulanger-lewandowski N, Bengio Y, Vincent P (2012) Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription. In: Proceedings of the 29th international conference on machine learning (ICML-12), pp 1159–1166
6. Boulanger-Lewandowski N, Bengio Y, Vincent P (2012) Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription. In: Proceedings of the 29th international conference on machine learning (ICML-12), Edinburgh, Scotland, UK, pp 1159–1166
7. Colombo F, Seeholzer A, Gerstner W (2017) Deep artificial composer: a creative neural network model for automated melody generation. In: Computational intelligence in music, sound, art and design: 6th international conference, EvoMUSART 2017, Amsterdam, The Netherlands, April 19–21, 2017, Proceedings. Springer, Cham, pp 81–96. https://doi.org/10.1007/978-3-319-55750-2-6
8. Cuthbert MS, Ariza C (2010) music21: a toolkit for computer-aided musicology and symbolic music data. International Society for Music Information Retrieval
9. Deza MM, Deza E (2009) Encyclopedia of distances. In: Encyclopedia of distances. Springer, Berlin, pp 1–583
10. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, London
11. Graves A (2012) Supervised sequence labelling. In: Supervised sequence labelling with recurrent neural networks. Springer, Berlin, pp 5–13
12. Hadjeres G, Sakellariou J, Pachet F (2016) Style imitation and chord invention in polyphonic music with exponential families. CoRR arxiv: abs/1609.05152
13. Hadjeres G, Pachet F, Nielsen F (2017) DeepBach: a steerable model for Bach chorales generation. In: Proceedings of the 34th international conference on machine learning. ICML 2017, Sydney, NSW, Australia, 6–11 August 2017, pp 1362–1371. http://proceedings.mlr.press/v70/hadjeres17a.html
14. Herremans D, Chuan CH, Chew E (2017) A functional taxonomy of music generation systems. ACM Comput Surv (CSUR) 50(5):69
15. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
16. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:14126980
17. 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
18. Lee JY, Wick M, Tristan J, Carbonell JG (2017) Enforcing constraints on outputs with unconstrained inference. CoRR arxiv: abs/1707.08608

19. Liang F (2016) BachBot: automatic composition in the style of Bach chorales
20. Nielsen F, Boltz S (2011) The Burbea–Rao and Bhattacharyya centroids. IEEE Trans Inf Theory 57(8):5455–5466
21. Österreicher F, Vajda I (2003) A new class of metric divergences on probability spaces and its applicability in statistics. Ann Inst Stat Math 55(3):639–653. https://doi.org/10.1007/BF02517812
22. Papadopoulos A, Pachet F, Roy P, Sakellariou J (2015) Exact sampling for regular and Markov constraints with belief propagation. In: International conference on principles and practice of constraint programming. Springer, pp 341–350
23. Pytorch (2016) Pytorch. https://github.com/pytorch/pytorch
24. Roberts A, Engel J, Raffel C, Hawthorne C, Eck D (2018) A hierarchical latent vector model for learning long-term structure in music. ArXiv e-prints arXiv:1803.05428
25. Sakellariou J, Tria F, Loreto V, Pachet F (2017) Maximum entropy models capture melodic styles. Sci Rep 7(1):9172. https://doi.org/10.1038/s41598-017-08028-4
26. Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE Trans Signal Process 45(11):2673–2681
27. Serdyuk D, Ke NR, Sordoni A, Pal C, Bengio Y (2017) Twin networks: using the future as a regularizer. CoRR arxiv: abs/1708.06742
28. Shabanian S, Arpit D, Trischler A, Bengio Y (2017) Variational Bi-LSTMs. ArXiv e-prints arXiv:1711.05717
29. Sturm BL, Santos JF, Ben-Tal O, Korshunova I (2016) Music transcription modelling and composition using deep learning. ArXiv:1604.08723v1
30. Walder C, Kim D (2016) Computer assisted composition with recurrent neural networks. arXiv preprint arXiv:161200092