Generating Polyphonic Music Using Tied Parallel Networks

Daniel D. Johnson(✉)

Harvey Mudd College, Claremont, CA 91711, USA
ddjohnson@hmc.edu

Abstract. We describe a neural network architecture which enables prediction and composition of polyphonic music in a manner that preserves translation-invariance of the dataset. Specifically, we demonstrate training a probabilistic model of polyphonic music using a set of parallel, tied-weight recurrent networks, inspired by the structure of convolutional neural networks. This model is designed to be invariant to transpositions, but otherwise is intentionally given minimal information about the musical domain, and tasked with discovering patterns present in the source dataset. We present two versions of the model, denoted TP-LSTM-NADE and BALSTM, and also give methods for training the network and for generating novel music. This approach attains high performance at a musical prediction task and successfully creates note sequences which possess measure-level musical structure.

1 Introduction

There have been many attempts to generate music algorithmically, including Markov models, generative grammars, genetic algorithms, and neural networks; for a survey of these approaches, see Nierhaus [17] and Fernández and Vico [5]. Neural network models are particularly flexible because they can be trained based on the complex patterns in an existing musical dataset, and a wide variety of neural-network-based music composition models have been proposed [1,7,14,16,22].

One particularly interesting approach to music composition is training a probabilistic model of polyphonic music. Such an approach attempts to model music as a probability distribution, where individual sequences are assigned probabilities based on how likely they are to occur in a musical piece. Importantly, instead of specifying particular composition rules, we can train such a model based on a large corpus of music, and allow it to discover patterns from that dataset, similarly to someone learning to compose music by studying existing pieces. Once trained, the model can be used to generate new music based on the training dataset by sampling from the resulting probability distribution.

Training this type of model is complicated by the fact that polyphonic music has complex patterns along multiple axes: there are both sequential patterns between timesteps and harmonic intervals between simultaneous notes. Furthermore, almost all musical structures exhibit transposition invariance. When music is written in a particular key, the notes are interpreted not based on their absolute

J. Correia et al. (Eds.): EvoMUSART 2017, LNCS 10198, pp. 128–143, 2017.
DOI: 10.1007/978-3-319-55750-2_9

position but instead relative to that particular key, and chords are also often classified based on their position in the key (e.g. using Roman numeral notation). Transposition, in which all notes and chords are shifted into a different key, changes the absolute position of the notes but does not change any of these musical relationships. As such, it is important for a musical model to be able to generalize to different transpositions.

Recurrent neural networks (RNN), especially long short-term memory networks (LSTM) [8], have been shown to be extremely effective at modeling single-dimensional temporal patterns. It is thus reasonable to consider using them to model polyphonic music. One simple approach is to treat all of the notes played at any given timestep as a single input vector, and train an LSTM network to output a vector of probabilities of playing each note in the next timestep [4]. This essentially models each note as an independent event. While this may be appropriate for simple inputs, real-world polyphonic music contains complex harmonic relationships that would be better described using a joint probability distribution. To this end, a more effective approach combines RNNs and restricted Boltzmann machines to model the joint probability distribution of notes at each timestep [3].

Although both of the above approaches enable a network to generate music in a sequential manner, neither are transposition-invariant. In both, each note is represented as a separate element in a vector, and thus there is no way for the network to generalize intervals and chords: any relationship between, say, a G and a B, must be learned independently from the relationship between a G♭ and a B♭. To capture the structure of chords and intervals in a transposition-invariant way, a neural network architecture would ideally consider *relative* positions of notes, as opposed to absolute positions.

Convolutional neural networks, another type of network architecture, have proven to be very adept at feature detection in image recognition; see Krizhevsky et al. [12] for one example. Importantly, image features are also multidimensional patterns which are invariant over shifts along multiple axes, the x and y axes of the image. Convolutional networks enable invariant feature detection by training the weights of a convolution kernel, and then convolving the image with the kernel.

Combining recurrent neural networks with convolutional structure has shown promise in other multidimensional tasks. For instance, Kalchbrenner et al. [10] describe an architecture involving LSTMs with simultaneous recurrent connections along multiple dimensions, some of which may have tied weights. Additionally, Kaiser and Sutskever [9] present a multi-layer architecture using a series of "convolutional gated recurrent units". Both of these architectures have had success in tasks such as digit-by-digit multiplication and language modeling.

In the current work, we describe two variants of a recurrent network architecture inspired by convolution that attain transposition-invariance and produce joint probability distributions over a musical sequence. These variations are referred to as Tied Parallel LSTM-NADE (TP-LSTM-NADE) and Biaxial LSTM (BALSTM). We demonstrate that these models enable efficient encoding of both temporal and pitch patterns by using them to predict and generate musical compositions.

1.1 LSTM

Long Short-Term Memory (LSTM) is a sophisticated architecture that has been shown to be able to learn long-term temporal sequences [8]. LSTM is designed to obtain constant error flow over long time periods by using *Constant Error Carousels* (CECs), which have fixed-weight recurrent connections to prevent exploding or vanishing gradients. These CECs are connected to a set of nonlinear units that allow them to interface with the rest of the network: an *input gate* determines how to change the memory cells, an *output gate* determines how strongly the memory cells are expressed, and a *forget gate* allows the memory cells to forget irrelevant values. The formulas for the activation of a single LSTM block with inputs $\mathbf{x}_t$ and hidden recurrent activations $\mathbf{h}_t$ at timestep t are given below:

$$\begin{aligned}
\mathbf{z}_t &= \tanh(W_{xz}\mathbf{x}_t + W_{hz}\mathbf{h}_{t-1} + \mathbf{b}_z) && \text{block input} \\
\mathbf{i}_t &= \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i) && \text{input gate} \\
\mathbf{f}_t &= \sigma(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f) && \text{forget gate} \\
\mathbf{c}_t &= \mathbf{i}_t \odot \mathbf{z}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1} && \text{cell state} \\
\mathbf{o}_t &= \sigma(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o) && \text{output gate} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) && \text{output}
\end{aligned}$$

where W denotes a weight matrix, b denotes a bias vector, $\odot$ denotes elementwise vector multiplication, and σ and tanh represent the logistic sigmoid and hyperbolic tangent elementwise activation functions, respectively. We are omitting so-called "peephole" connections, which use the contents of the memory cells as inputs to the gates. These connections have been shown not to have a significant impact on the network performance [6]. Like traditional RNN, LSTM networks can be trained by backpropagation through time (BPTT), or by truncated BPTT. Figure 1 gives a schematic of an LSTM block.

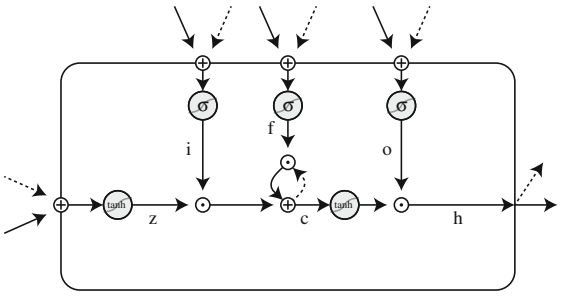

Fig. 1. Schematic of a LSTM block. Dashed lines represent a one-timestep delay. Solid arrow inputs represent $\mathbf{x}_t$, and dashed arrow inputs represent $\mathbf{h}_{t-1}$, each of which are scaled by learned weights (not shown). $\oplus$ indicates a sum, and $\odot$ indicates elementwise multiplication.

1.2 RNN-NADE

The RNN-RBM architecture, as well as the closely related RNN-NADE architecture, are attempts to model the joint distribution of a multidimensional sequence [3]. Specifically, the RNN-RBM combines recurrent neural networks (RNNs), which can capture temporal interactions, and restricted Boltzmann machines (RBMs), which model conditional distributions.

RBMs have the disadvantage of having a gradient that is untractable to compute: the gradients of the loss with respect to the model parameters must be estimated by using a method such as contrastive divergence or Gibbs sampling. To obtain a tractable gradient, the RNN-NADE architecture replaces the RBM with a neural autoregressive distribution estimator (NADE) [13], which calculates the joint probability of a vector of binary variables $\mathbf{v} = [v_1, v_2, \cdots, v_n]$ (here used to represent the set of notes that are being played simultaneously) using a series of conditional distributions:

$$p(\mathbf{v}) = \prod_{i=1}^{|\mathbf{v}|} p(v_i|\mathbf{v}_{<i})$$

with each conditional distribution given by

$$\begin{aligned} \mathbf{h_i} &= \sigma(\mathbf{b}_h + W_{:,<i}\mathbf{v}_{<i}) \\ p(v_i = 1|\mathbf{v}_{<i}) &= \sigma(\mathbf{b}_{v_i} + V_{i,:}\mathbf{h_i}) \\ p(v_i = 0|\mathbf{v}_{<i}) &= 1 - p(v_i = 1|\mathbf{v}_{<i}) \end{aligned}$$

where $\mathbf{b}_v$ and $\mathbf{b}_h$ are bias vectors and W and V are weight matrices. Note that $\mathbf{v}_{<i}$ denotes the vector composed of the first $i-1$ elements of $\mathbf{v}$, $W_{:,<i}$ denotes the matrix composed of all rows and the first $i-1$ columns of W, and $V_{i,:}$ denotes the ith row of V. Under this distribution, the loss has a tractable gradient, so no gradient estimation is necessary.

In the RNN-NADE, the bias parameters $\mathbf{b}_v$ and $\mathbf{b}_h$ at each timestep are calculated using the hidden activations $\hat{\mathbf{h}}$ of an RNN, which takes as input the output $\mathbf{v}$ of the network at each timestep:

$$\begin{aligned} \hat{\mathbf{h}}^{(t)} &= \sigma(W_{v\hat{h}}\mathbf{v}^{(t)} + W_{\hat{h}\hat{h}}\hat{\mathbf{h}}^{(t-1)} + \mathbf{b}_{\hat{h}}) \\ \mathbf{b}_v^{(t)} &= \mathbf{b}_v + W_{\hat{h}b_v}\hat{\mathbf{h}}^{(t)} \\ \mathbf{b}_h^{(t)} &= \mathbf{b}_h + W_{\hat{h}b_h}\hat{\mathbf{h}}^{(t)} \end{aligned}$$

The RNN parameters $W_{v\hat{h}}$, $W_{\hat{h}\hat{h}}$, $\mathbf{b}_v$, $W_{\hat{h}b_v}$, $\mathbf{b}_h$, $W_{\hat{h}b_h}$, as well as the NADE parameters W and V are all trained using stochastic gradient descent.

2 Translation Invariance and Tied Parallel Networks

One disadvantage of distribution estimators such as RBM and NADE is that they cannot easily capture relative relationships between inputs. Although they

can readily learn relationships between any set of particular notes, they are not structured to allow generalization to a transposition of those notes into a different key. This is problematic for the task of music prediction and generation because the consonance or dissonance of a set of notes remains the same regardless of their absolute position.

As one example of this, if we represent notes as a one-dimensional binary vector, where a 1 represents a note being played, a 0 represents a note not being played, and each adjacent number represents a semitone (half-step) increase, a major chord can be represented as

$$\ldots 001000100100 \ldots$$

This pattern is still a major chord no matter where it appears in the input sequence, so

$$1000100100000,$$

$$0010001001000,$$

$$0000010001001,$$

all represent major chords, a property known as *translation invariance*. However, if the input is simply presented to a distribution estimator such as NADE, each transposed representation would have to be learned separately.

Convolutional neural networks address the invariance problem for images by convolving or cross-correlating the input with a set of learned kernels. Each kernel learns to recognize a particular local type of feature. The cross-correlation of two one-dimensional sequences $\mathbf{u}$ and $\mathbf{v}$ is given by

$$(\mathbf{u} \star \mathbf{v})_n = \sum_{m=-\infty}^{\infty} \mathbf{u}_m \mathbf{v}_{m+n}.$$

Crucially, if one of the inputs (say $\mathbf{u}$) is shifted by some offset δ, the output $(\mathbf{u} \star \mathbf{v})_n$ is also shifted by δ, but otherwise does not change. This makes the operation ideal for detecting local features for which the relevant relationships are relative, not absolute.

For our music generation task, we can obtain transposition invariance by designing our model to behave like a cross-correlation: if we have a vector of notes $\mathbf{v}^{(t)}$ at timestep t and a candidate vector of notes $\hat{\mathbf{v}}^{(t+1)}$ for the next timestep, and we construct shifted vectors $\mathbf{w}^{(t)}$ and $\hat{\mathbf{w}}^{(t+1)}$ such that $\mathbf{w}_i^{(t)} = \mathbf{v}_{i+\delta}^{(t)}$ and $\hat{\mathbf{w}}_i^{(t+1)} = \hat{\mathbf{v}}_{i+\delta}^{(t+1)}$, then we want the output of our model to satisfy

$$p(\hat{\mathbf{w}}^{(t+1)} | \mathbf{w}^{(t)}) = p(\hat{\mathbf{v}}^{(t+1)} | \mathbf{v}^{(t)}).$$

2.1 Tied Parallel LSTM-NADE

In order to achieve the above form of transposition invariance, but also handle complex temporal sequences and jointly-distributed output, we propose dividing

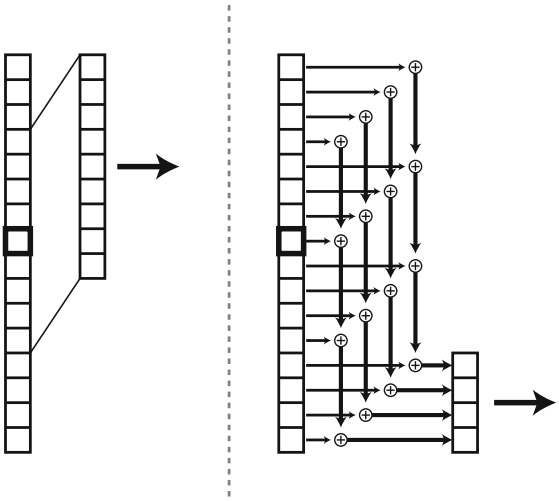

Fig. 2. Illustration of the windowing and binning operations. The thick-outlined box represents the current note. On the left, a local window around the note is extracted, and on the right, notes from each octave are binned together according to their pitch-class. For clarity, an octave is represented here by four notes; in the actual implementation octaves are of size 12.

the music prediction task into a set of *tied parallel networks*. Each network instance will be responsible for a single note, and will have tied weights with every other network instance. In this way, we ensure translation invariance: since each instance uses the same procedure to calculate its output, if we shift the inputs up by some amount δ, the output will also be shifted by δ. Our task is thus to divide the RNN-NADE architecture into multiple networks in this fashion, while still maintaining the ability to model notes conditionally on past output.

In the original RNN-NADE architecture, the RNN received the entire note vector as input. However, since each note now has a network instance that operates relative to that note, it is no longer feasible to give the entire note vector $\mathbf{v}^{(t)}$ as input to each network instance. Instead, we will feed the instance two input vectors, a local window $\mathbf{w}^{(n,t)}$ and a set of bins $\mathbf{z}^{(n,t)}$. The local window contains a slice of the note vector $\mathbf{v}^{(t)}$ such that $\mathbf{w}_i^{(n,t)} = \mathbf{v}_{n-13+i}^{(t)}$ where $1 \leq i \leq 25$ (giving the window a span of one octave above and below the note). If the window extends past the bounds of $\mathbf{v}$, those values are instead set to 0, as no notes are played above or below the bounds of $\mathbf{v}$. The content of each bin $\mathbf{z}_i$ is the number of notes that are played at the offset i from the current note across all octaves:

$$\mathbf{z}_i^{(n,t)} = \sum_{m=-\infty}^{\infty} \mathbf{v}_{i+n+12m}^{(t)}$$

where, again, $\mathbf{v}$ is assumed to be 0 above and below its bounds. This is equivalent to collecting all notes in each pitchclass, measured relative to the current note.

For instance, if the current note has pitchclass D, then bin $\mathbf{z}_2$ will contain the number of played notes with pitchclass E across all octaves. The windowing and binning operations are illustrated in Fig. 2.

Finally, although music is mostly translation-invariant for small shifts, there is a difference between high and low notes in practice, so we also give as input to each network instance the MIDI pitch number it is associated with. These inputs are concatenated and then fed to a set of LSTM layers, which are implemented as described above. Note that we use LSTM blocks instead of regular RNNs to encourage learning long-term dependencies.

As in the RNN-NADE model, the output of this parallel network instance should be an expression for $p(v_n|\mathbf{v}_{<n})$. We can adapt the equations of NADE to enable them to work with the parallel network as follows:

$$\begin{aligned}\mathbf{h_n} &= \sigma(\mathbf{b}_h^{(n,t)} + W\mathbf{x_n}) \\ p^{(t)}(v_n = 1|\mathbf{v}_{<n}) &= \sigma(b_v^{(n,t)} + V\mathbf{h_n}) \\ p^{(t)}(v_n = 0|\mathbf{v}_{<n}) &= 1 - p^{(t)}(v_n = 1|\mathbf{v}_{<n})\end{aligned}$$

where $\mathbf{x_n}$ is formed by taking a 2-octave window of the most recently chosen notes, concatenated with a set of bins. This is performed in the same manner as for the input to the LSTM layers, but instead of spanning the entirety of the notes from the previous timestep, it only uses the notes in $\mathbf{v}_{<n}$. The parameters $\mathbf{b}_h^{(n,t)}$ and $b_v^{(n,t)}$ are computed from the final hidden activations $\mathbf{y}^{(n,t)}$ of the LSTM layers as

$$\begin{aligned}b_v^{(n,t)} &= b_v + W_{yb_v}\mathbf{y}^{(n,t)} \\ \mathbf{b}_h^{(n,t)} &= \mathbf{b}_h + W_{yb_h}\mathbf{y}^{(n,t)}\end{aligned}$$

One key difference is that b_v is now a scalar, not a vector, since each instance of the network only produces a single output probability, not a vector of them. The full output vector is formed by concatenating the scalar outputs for each network instance. The left side of Fig. 3 is a diagram of a single instance of this parallel network architecture. This version of the architecture will henceforth be referred to as Tied-Parallel LSTM-NADE (TP-LSTM-NADE).

2.2 Bi-Axial LSTMs

A downside to the architecture described in Sect. 2.1 is that, in order to apply the modified NADE, we must use windowed and binned summaries of note output. This captures some of the most important relationships between notes, but also prevents the network from learning any precise dependencies that extend past the size of the window. As an alternative, we can replace the NADE portion of the network with LSTMs that have recurrent connections along the note axis. This combination of LSTMs along two different axes (first along the time axis, and then along the note axis) will be referred to as a "bi-axial" configuration, to differentiate it from bidirectional configurations, which run recurrent networks both forward and backward along the same axis.

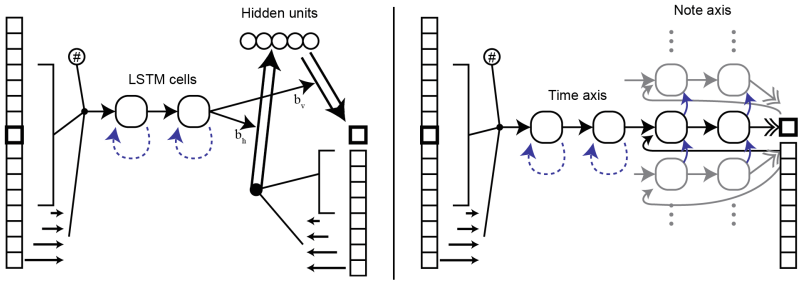

Fig. 3. On the left, schematic of a network instance for the tied parallel LSTM-NADE network. This instance is responsible for producing the output probability for the note indicated with the thick-outlined box. On the right, a schematic of an instance in the bi-axial LSTM network, showing a single instance of the time-axis network and three note-steps of the note-axis network. For each network, we concatenate a window of the note's vicinity, bins, and MIDI note number of the current note. Concatenations are indicated by lines connected by a solid black circle. Dashed arrows represent time-delayed connections, blue arrows represent recurrent connections, thick double-line-arrows represent the modified NADE estimation, and double-headed arrows indicate sampling a binary value from its corresponding probability.

In each "note-step", these note-axis LSTM layers receive as input a concatenation of two sources: the activations of the final time-axis LSTM layer for this note, and also the final output of the network for the previous note. The final activations of the note-axis LSTM will be transformed into a probability $p^{(n,t)}(v_n = 1|\mathbf{v}_{<n})$ using softmax activation. Note that, just as each note has a corresponding tied-weight time-axis LSTM network responsible for modeling temporal relationships for that single note, each timestep has a corresponding tied-weight note-axis LSTM network responsible for modeling the joint distribution of notes in that single timestep. Sequentially running the network for each note in a timestep allows us to determine the full conditional distribution for that timestep. This modification to the architecture is shown on the right side of Fig. 3, and will be referred to as Bi-Axial LSTM (BALSTM).

2.3 Training and Generation

We demonstrate our architecture by applying it to a polyphonic music prediction task, as described in Boulanger-Lewandowski et al. [3]. We train our network to model the conditional probability distribution of the notes played in a given timestep, conditioned on the notes played in previous timesteps. Specifically, we interpret the output of the n^{th} tied-weight network instance at timestep t as the probability for playing note n at t, conditioned on previous note choices. Training our model thus amounts to maximizing the log-likelihood of each training sequence under this conditional distribution.

To calculate the log-likelihood of a given sequence, since we already know the notes that are chosen at all timesteps, we can use those notes as the inputs

into the model, and then sum the log-likelihoods of the sequence being generated across all notes and all timesteps. Letting $q^{(n,t)}$ represent our network's estimate of $p^{(t)}(v_n = 1|\mathbf{v}_{<n})$, our cost is given by

$$C = -\frac{1}{TN}\sum_{t=1}^{T}\sum_{n=1}^{N}\ln\left[v_n^{(t)}q^{(n,t)} + (1 - v_n^{(t)})(1 - q^{(n,t)})\right],$$

where T is the number of timesteps and N is the number of possible notes.

Importantly, in each of the variants of our architecture described above, interaction between layers flows in a single direction; i.e. the LSTM time-axis layers depend only on the chosen notes, not on the specific output of the note-axis layers. During training, we already know all of the notes at all timesteps, so we can accelerate our training process by processing each layer independently: first preprocessing the input, then running it through the LSTM time-axis layers in parallel across all notes, and finally either using the modified NADE or the LSTM note-axis layers to compute probabilities in parallel across all timesteps. This massively parallel training process is ideal for training on a GPU or on a multi-core system.

Once trained, we can sample from this trained distribution to "compose" novel sequences. In this case, we do not know the entire sequence in advance. Instead, we must run the full network one timestep at a time. At each timestep, we process the input for that timestep, advance the LSTM time-axis layers by one timestep, and then generate the next timestep's notes. To do this, we sample from the conditional distribution as it is being generated: for each note, we choose $v_n^{(t)}$ from a Bernoulli distribution with probability $q^{(n,t)}$. Then, this choice is used to construct the input for the computation of $q^{(n+1,t)}$. Once all of the notes have been processed, we can advance to the next timestep.

The bottleneck of sampling from the distribution before processing the next timestep makes generation slower on GPUs or multi-core systems, since we can no longer parallelize the activation of note-axis computation. However, this can be mitigated somewhat by generating multiple samples simultaneously.

3 Experiments

3.1 Quantitative Analysis

We evaluated two variants of the tied-weight parallel model, along with a non-parallel model for comparison:

- *LSTM-NADE:* Non-parallel model consisting of an LSTM block connected to NADE as in the RNN-NADE architecture. We used two LSTM layers with 300 nodes each, and 150 hidden units in the NADE layer.
- *TP-LSTM-NADE:* Tied-parallel LSTM-NADE model described in Sect. 2.1. We used two LSTM layers with 200 nodes each, and 100 hidden units in the modified NADE layer.

Table 1. Log-likelihood performance for the non-transposed prediction task. For LSTM-NADE, TP-LSTM-NADE, and BALSTM, the two values represent the best and median performance across 5 trials. Data for other models is reproduced from Boulanger-Lewandowski et al. [3] and Vohra et al. [25].

Model	JSB Chorales	MuseData	Nottingham	Piano-Midi.de
Random	−61.00	−61.00	−61.00	−61.00
RBM	−7.43	−9.56	−5.25	−10.17
NADE	−7.19	−10.06	−5.48	−10.28
RNN-RBM	−7.27	−9.31	−4.72	−9.89
RNN (HF)	−8.58	−7.19	−3.89	−7.66
RNN-RBM (HF)	−6.27	−6.01	−2.39	−7.09
RNN-DBN	−5.68	−6.28	−2.54	−7.15
RNN-NADE (HF)	−5.56	−5.60	−2.31	−7.05
DBN-LSTM	−3.47	−3.91	−1.32	−4.63
LSTM-NADE	−6.00, −6.10	−5.02, −5.03	−2.02, −2.06	−7.36, −7.39
TP-LSTM-NADE	−5.88, −5.92	−4.32, −4.34	−1.61, −1.64	−5.44, −5.49
BALSTM	−5.05, −5.86	−3.90, −4.41	−1.55, −1.62	−4.90, −5.00

Table 2. Log-likelihood performance for the transposed prediction task. The two values represent the best and median performance across 5 trials.

Model	JSB Chorales	MuseData	Nottingham	Piano-Midi.de
LSTM-NADE	−9.04, −9.16	−5.72, −5.76	−3.65, −3.70	−8.11, −8.13
TP-LSTM-NADE	−5.89, −5.92	−4.32, −4.33	−1.61, −1.64	−5.44, −5.49
BALSTM	−5.08, −5.87	−3.91, −4.45	−1.56, −1.71	−4.92, −5.01

– *BALSTM:* Bi-axial LSTM with windowed+binned input, described in Sect. 2.2. We used two LSTM layers in the time-axis direction with 200 nodes each, and two LSTM layers in the note-axis direction with 100 nodes each.

We tested the ability of each model to predict/generate note sequences based on four datasets: *JSB Chorales*, a corpus of 382 four-part chorales by J.S. Bach; *MuseData*[1], an electronic classical music library, from CCARH at Stanford; *Nottingham*[2], a collection of 1200 folk tunes in ABC notation, consisting of a simple melody on top of chords; and *Piano-Midi.de*, a classical piano MIDI database. Each dataset was transposed into C major or C minor and segmented into training, validation, and test sets as in Boulanger-Lewandowski et al. [3]. Input was provided to our network in a piano-roll format, with a vector of length 88 representing the note range from A0 to C8.

[1] www.musedata.org.
[2] ifdo.ca/~seymour/nottingham/nottingham.html.

Dropout of 0.5 was applied to each LSTM layer, as in Moon et al. [15], and trained using RMSprop [21] with a learning rate of 0.001 and Nesterov momentum [19] of 0.9. We then evaluated our models using three criteria. Quantitatively, we evaluated the log-likelihood of the test set, which characterizes the accuracy of the model's predictions. Qualitatively, we generated sample sequences as described in Sect. 2.3. Finally, to study the translation-invariance of the models, we evaluated the log-likelihood of a version of the test set transposed into D major or D minor. Since such a transposition should not affect the musicality of the pieces in the dataset, we would expect a good model of polyphonic music to predict the original and transposed pieces with similar levels of accuracy. However, a model that was dependent on its input being in a particular key would not be able to generalize well to transpositions of the input.

Table 1 shows the performance on the non-transposed task. LSTM-NADE, TP-LSTM-NADE, and BALSTM correspond to the architectures described here, where the two values represent the best and median performance of each architecture, respectively, across 5 trials. Data for baseline models is reproduced from Vohra et al. [25] (for RNN-DBN and DBN-LSTM) and Boulanger-Lewandowski et al. [3] (for all other models). In particular, "Random" shows the performance of choosing to play each note with 50% probability, and the other architectures are variations of the original RNN-RBM architecture, which we do not describe thoroughly here.

Our tied-parallel architectures (BALSTM and TP-LSTM-NADE) perform noticeably better on the test set prediction task than did the original RNN-NADE model and many architectures closely related to it. Of the variations we tested, the BALSTM network appeared to perform the best. The TP-LSTM-NADE network, however, appears to be more stable, and converges reliably to a relatively consistent cost. Both tied-parallel network architectures perform comparably to or better than the non-parallel LSTM-NADE architecture.

The DBN-LSTM model, introduced by Vohra et al. [25], has superior performance when compared to our tied-parallel architectures. This is likely due to the deep belief network used in the DBN-LSTM, which allows the DBN-LSTM to capture a richer joint distribution at each timestep. A direct comparison between the DBN-LSTM model and the BALSTM or TP-LSTM-NADE models may be somewhat uninformative, since the models differ both in the presence or absence of parallel tied-weight structure as well as in the complexity of the joint distribution model at each timestep. However, the success of both models relative to the original RNN-RBM and RNN-NADE models suggests that a model that combined parallel structure with a rich joint distribution might attain even better results.

Note that when comparing the results from the LSTM-NADE architecture and the TP-LSTM-NADE/BALSTM architectures, the greatest improvements are on the MuseData and Piano-Midi.de datasets. This is likely due to the fact that those datasets contain many more complex musical structures in different keys, which are an ideal case for a translation-invariant architecture. On the other

hand, the performance on the datasets with less variation in key is somewhat less impressive.

In addition, as shown in Table 2, the TP-LSTM-NADE/BALSTM architectures demonstrate the desired translation invariance: both parallel models perform comparably on the original and transposed datasets, whereas the non-parallel LSTM-NADE architecture performs worse at modeling the transposed dataset. This indicates that the parallel models are able to learn musical patterns that generalize to music in multiple keys, and are not sensitive to transpositions of the input, whereas the non-parallel model can only learn patterns with respect to a fixed key. Although we were unable to evaluate other existing architectures on the transposed dataset, it is reasonable to suspect that they would also show reduced performance on the transposed dataset for reasons described in Sect. 2.

3.2 Qualitative Analysis

In addition to the above experiments, we trained the BALSTM model on a larger collection of MIDI pieces with the goal of producing novel musical compositions. To this end, we made a few modifications to the BALSTM model.

Firstly, we used a larger subset of the Piano-Midi.de dataset for training, including additional pieces not used with prior models and pieces originally used as part of the validation set. To allow the network to learn rhythmic patterns, we restricted the dataset to pieces with the 4/4 time signature. We did not transpose the pieces into a common key, as the model is naturally translation invariant and does not benefit from this modification. Using the larger dataset maximizes the variety of pieces used during training and was intended to allow the network to learn as much about the musical structures as possible.

Secondly, the input to the network was augmented with a "temporal position" vector, giving the position of the timestep relative to a 4/4 measure in binary format. This gives the network the ability to learn specific temporal patterns relative to a measure.

Thirdly, we added a dimension to the note vector $\mathbf{v}$ to distinguish rearticulating a note from sustaining it. Instead of having a single 1 represent a note being played and a 0 represent that note not being played, we appended an additional 1 or 0 depending on whether that note is being articulated at that timestep. Thus the first timestep for playing a note is represented as 11, whereas sustaining a previous note is represented as 10, and resting is represented as 00. This adjustment enables the network to play the same note multiple times in succession. On the input side, the second bit is preprocessed in parallel with the first bit, and is passed into the time-axis LSTM layers as an additional input. On the output side, the note-axis LSTM layers output two probabilities instead of just one: both the probability of playing a note, and the probability of rearticulating the note if the network chooses to play it. When computing the log-likelihood of the sequence, we penalize the network for articulating a played note incorrectly, but ignore the articulation output for notes that should not be played. Similarly, when generating a piece, we only allow the network to articulate notes that have been chosen to be played.

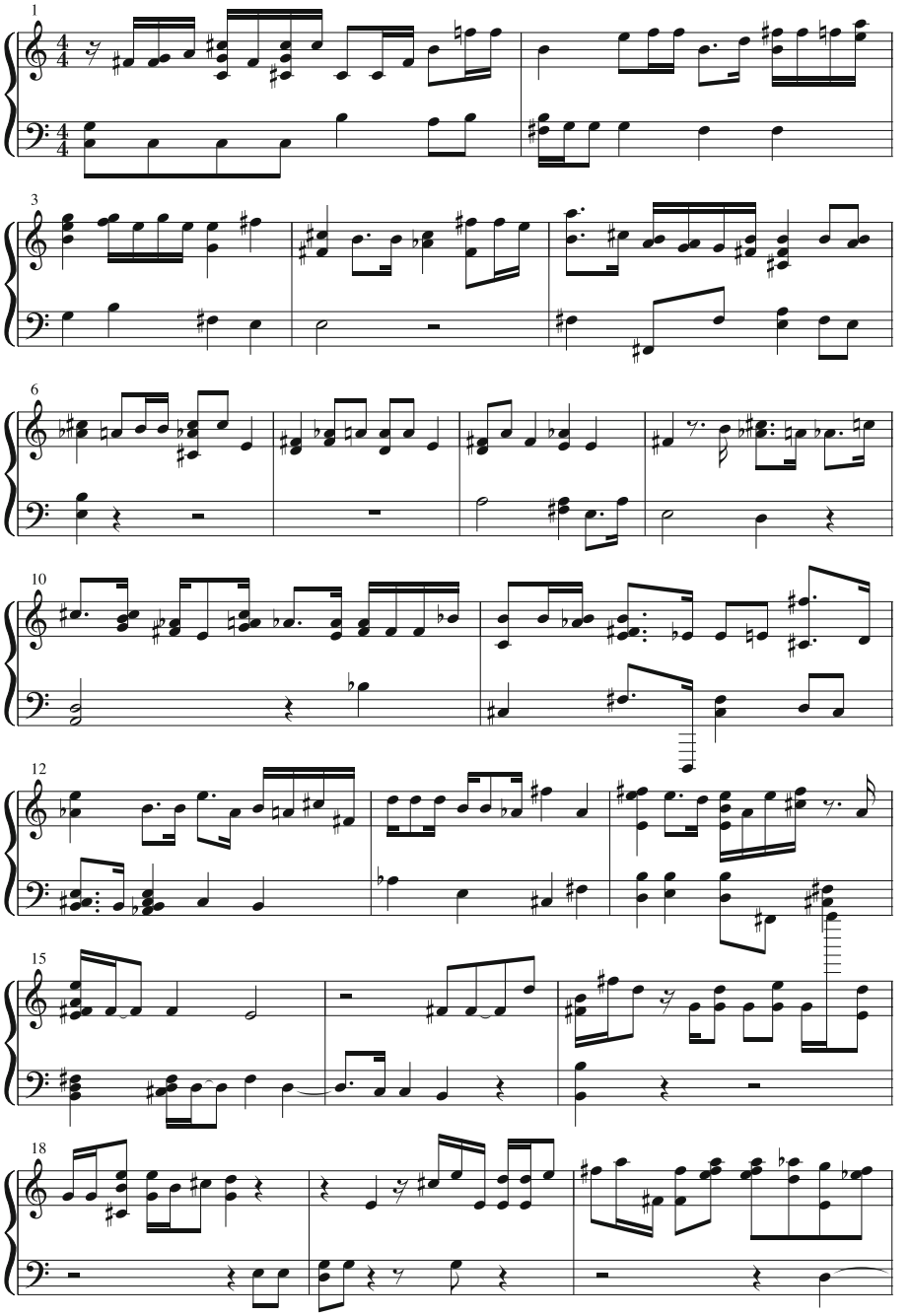

Fig. 4. A section of a generated sample from the BALSTM model after being trained on the Piano-Midi.de dataset, converted into musical notation.

Qualitatively, the samples generated by this version of the model appear to possess complexity and intricacy. Samples from the extended BALSTM model demonstrate rhythmic consistency, chords, melody, and counterpoint not found in the samples from the RNN-RBM and RNN-NADE models (as provided by Boulanger-Lewandowski et al. [3]). The samples also seem to possess consistency across multiple measures, and although they frequently change styles, they exhibit smooth transitions from one style to another.

A portion of a generated music sample is shown in Fig. 4. Samples of the generated music for each architecture can be found on the author's website.[3]

4 Future Work

One application of the music prediction task is improving automatic transcription by giving an estimate for the likelihood of various configurations of notes [18]. As the music to be transcribed may be in any key, applying a tied parallel architecture to this task might improve the results.

A noticeable drawback of our model is that long-term phrase structure is absent from the output. This is likely because the model is trained to predict and compose music one timestep at a time. As a consequence, the model is encouraged to pay more attention to recent notes, which are often the best predictors of the next timestep, instead of on planning long-term structures. Modeling this structure will likely require incorporating a long-term planning component into the architecture. One approach to combining long-term planning with recurrent networks is the Strategic Attentive Writer (STRAW) model, described by Vezhnevets et al. [24], which extends a recurrent network with an action plan, which it can modify incrementally over time. Combining such an approach with a RNN-based music model might allow the model to generate pieces with long-term structure.

We also noticed that the network occasionally appears to become "confused" after playing a discordant note. This is likely because the dataset represents only a small portion of the overall note-sequence state space, so it is difficult to recover from mistakes due to the lack of relevant training data. Bengio et al. [2] proposed a scheduled sampling method to alleviate this problem, and a similar modification could be made here.

Another potential avenue for further research is modeling a latent musical style space using variational inference [11], which would allow the network to model distinct styles without alternating between them, and might allow the network to generate music that follows a predetermined musical form.

5 Conclusions

In this paper, we discussed the property of translation invariance of music, and proposed a set of modifications to the RNN-NADE architecture to allow it to

[3] https://www.cs.hmc.edu/~ddjohnson/tied-parallel/.

capture relative dependencies of notes. The modified architectures, which we call Tied-Parallel LSTM-NADE and Bi-Axial LSTM, divide the music generation and prediction task such that each network instance is responsible for a single note and receives input relative to that note, a structure inspired by convolutional networks.

Experimental results demonstrate that this modification yields a higher accuracy on a prediction task when compared to similar non-parallel models, and approaches state of the art performance. As desired, our models also possess translation invariance, as demonstrated by performance on a transposed prediction task. Qualitatively, the output of our model has measure-level structure, and in some cases successfully reproduces complex rhythms, melodies, and counterpoint.

Although the network successfully models measure-level structure, it unfortunately does not appear to produce consistent phrases or maintain style over a long period of time. Future work will explore modifications to the architecture that could enable a neural network model to incorporate specific styles and long-term planning into its output.

Acknowledgments. We would like to thank Dr. Robert Keller for helpful discussions and advice. We would also like to thank the developers of the Theano framework [20], which we used to run our experiments, as well as Harvey Mudd College for providing computing resources. This work used the Extreme Science and Engineering Discovery Environment (XSEDE) [23], which is supported by National Science Foundation grant number ACI-1053575.

References

1. Bellgard, M.I., Tsang, C.P.: Harmonizing music the boltzmann way. Connect. Sci. **6**(2–3), 281–297 (1994)
2. Bengio, S., Vinyals, O., Jaitly, N., Shazeer, N.: Scheduled sampling for sequence prediction with recurrent neural networks. In: Advances in Neural Information Processing Systems, pp. 1171–1179 (2015)
3. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription. In: Proceedings of the 29th International Conference on Machine Learning (ICML-2012), pp. 1159–1166 (2012)
4. Eck, D., Schmidhuber, J.: A first look at music composition using LSTM recurrent neural networks. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale (2002)
5. Fernández, J.D., Vico, F.: AI methods in algorithmic composition: a comprehensive survey. J. Artif. Intell. Res. **48**, 513–582 (2013)
6. Greff, K., Srivastava, R.K., Koutnk, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: A search space odyssey. arXiv preprint arXiv:1503.04069 (2015)
7. Hild, H., Feulner, J., Menzel, W.: HARMONET: a neural net for harmonizing chorales in the style of JS Bach. In: NIPS, pp. 267–274 (1991)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural comput. **9**(8), 1735–1780 (1997)
9. Kaiser, Ł., Sutskever, I.: Neural GPUs learn algorithms. arXiv preprint arXiv:1511.08228 (2015)

10. Kalchbrenner, N., Danihelka, I., Graves, A.: Grid long short-term memory. arXiv preprint arXiv:1507.01526 (2015)
11. Kingma, D.P., Welling, M.: Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114 (2013)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
13. Larochelle, H., Murray, I.: The neural autoregressive distribution estimator. In: International Conference on Artificial Intelligence and Statistics, pp. 29–37 (2011)
14. Lewis, J.P.: Creation by refinement and the problem of algorithmic music composition. In: Music and Connectionism, p. 212 (1991)
15. Moon, T., Choi, H., Lee, H., Song, I.: RnnDrop: a novel dropout for RNNs in ASR. In: Automatic Speech Recognition and Understanding (ASRU) (2015)
16. Mozer, M.C.: Induction of multiscale temporal structure. In: Advances in Neural Information Processing Systems, pp. 275–275 (1993)
17. Nierhaus, G.: Algorithmic Composition: Paradigms of Automated Music Generation. Springer Science & Business Media, Verlag (2009)
18. Sigtia, S., Benetos, E., Cherla, S., Weyde, T., Garcez, A.S.d., Dixon, S.: An RNN-based music language model for improving automatic music transcription. In: International Society for Music Information Retrieval Conference (ISMIR) (2014)
19. Sutskever, I.: Training recurrent neural networks. Ph.D. thesis, University of Toronto (2013)
20. Theano Development Team: Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints abs/1605.02688, May 2016. http://arxiv.org/abs/1605.02688
21. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. In: COURSERA: Neural Networks for Machine Learning 4 (2012)
22. Todd, P.M.: A connectionist approach to algorithmic composition. Comput. Music J. **13**(4), 27–43 (1989)
23. Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., Hazlewood, V., Lathrop, S., Lifka, D., Peterson, G.D., et al.: XSEDE: accelerating scientific discovery. Comput. Sci. Eng. **16**(5), 62–74 (2014)
24. Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., Agapiou, J., et al.: Strategic attentive writer for learning macro-actions. In: Advances in Neural Information Processing Systems, pp. 3486–3494 (2016)
25. Vohra, R., Goel, K., Sahoo, J.: Modeling temporal dependencies in data using a DBN-LSTM. In: IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2015. 36678 2015, pp. 1–4 (2015)