RNN Music Generation

Completed for the Certificate in Scientific Computation Spring 2025

Logan Kronforst
Bachelor of Arts in Music, Classical Guitar Emphasis
Butler School of Music
College of Fine Arts

Shyamal Mitra

Shyamal Mitra, Associate Professor of Instruction, Department of Computer Science

RNN Music Generation

LOGAN KRONFORST**, The University of Texas at Austin, USA
SHYAMAL MITRA†, The University of Texas at Austin, USA

Computer science is a vast field and over-saturated with traditional research in machine learning, computer architecture, and artificial intelligence. An area that has been little explored is the various applications of machine learning and music. Given the hype surrounding generative neural networks, it is a natural motivation to consider music generation using machine learning and deep learning techniques. Recurrent neural networks (RNNs) have proven highly successful in modeling time series and sequential data. In particular long short-term memory (LSTM) neural networks (the child of the RNN) achieve a more efficient gradient-based optimization which minimizes the overestimation of learning parameters and the exploding gradient problem [6]. All forms of music have some temporal component. The music of Johann Sebastian Bach is exceptionally suited for deep learning experimentation, given its structural and temporal patterns and sequences. The goal of this investigation is to produce convincing music using only a vanilla LSTM model, and using only the pitch, step, and time duration of notes processed from all of the available MIDI files of J.S. Bach sourced from www.bachcentral.com

CCS Concepts: • **Computing methodologies** → *Neural networks*; • **Applied computing** → **Sound and music computing**.

Additional Key Words and Phrases: RNN, Music Generation, Machine Learning, Artificial Intelligence, Algorithmic Music Composition, Scientific Computing

ACM Reference Format:
Logan Kronforst and Shyamal Mitra. 2025. RNN Music Generation. 1, 1 (May 2025), 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Computer science is a broad discipline encompassing numerous specialized areas. Over the past few decades, machine learning has gained significant prominence in both academia and popular culture, driving the development of transformative applications across diverse industries and research fields.

Among the various techniques within machine learning, deep learning methods—particularly recurrent neural networks (RNNs)—have demonstrated notable success in modeling temporal dependencies and sequences in musical data [4]. More advanced approaches, such as DeepBach [3], leverage multiple neural networks (one per voice) to generate polyphonic music in the style of J.S. Bach. Before developing deep learning models, classical algorithmic methods for music generation often relied on crafting combinatorial systems of rules, grammars, or automata to enforce harmonic conventions and voice-leading principles.

*Primary Author.
†Research Advisor

Authors' Contact Information: Logan Kronforst, lkronforst@utexas.edu, The University of Texas at Austin, Austin, Texas, USA; Shyamal Mitra, The University of Texas at Austin, Austin, Texas, USA, mitra@cs.utexas.edu.

Manuscript submitted to ACM

In contrast, modern deep learning approaches view music generation not as a purely combinatorial task, but as learning a *discretized approximation* of the underlying musical dynamics. J.S. Bach's music, celebrated for its rigorous adherence to 18th-century counterpoint, is an ideal test bed for such approaches. Its remarkably systematic yet endlessly inventive compositional style provides a rich corpus for training an LSTM to *learn* Bach's idiomatic voice-leading and harmonic rules, rather than hard-coding them as abstractions.

1.1 Computer Science and Music

Algorithmic music composition is not a new subfield. Serious attempts at computer-generated music date back to the 1950s with The Illiac Suite, the first score composed by a computer [3]. The Illiac Suite utilized stochastic models for generation and specific rules to constrain the model and filter generated material based on desired properties. The abstract models mentioned above have significant limitations in their ability to learn natural representations of music. Machine learning techniques enable models to *learn* from an existing corpus of music, without the explicit specification of rules, grammar, or automata.

Recent work utilizing deep learning LSTM neural networks, such as DeepBach [3], creates a dependency network by specifying probability distributions on a finite set of parameters that are maximized using a log-likelihood estimator. DeepBach's model amounts to solving four classification problems of the form:

$$\max \sum \log p_i(\mathcal{V}_i | \mathcal{V}_{\setminus i,t}, \mathcal{M}, \theta_{i,t})$$

Where $\mathcal{M}_i$ denotes the metadata sequence at note i and $\mathcal{V}_{\setminus i,t}$ represents the collection of all other voices at time t, excluding the i-th voice. Each classifier models the conditional probability of one voice given all different voices and metadata, with parameters $\theta_{i,t}$representing the weights of the neural network [3].

DeepBach utilizes four separate neural networks, one for each voice, to model a four-part chorale. In summary, DeepBach maximizes the conditional log-likelihood of predicting each voice $\mathcal{V}_i$, conditioned on:

- the surrounding voices $\mathcal{V}_{\setminus i,t}$,
- the metadata $\mathcal{M}$,
- and the local musical context (neighboring notes),

With all dependencies captured by the parametric neural network $\theta_{i,t}$.

Of course, DeepBach has a far more complex architecture and represents some of the best work in algorithmic music generation in recent years. Using a vanilla LSTM neural network with a single voice, representing a single output, with one hidden layer for this investigation was motivated by model simplicity and to naturally uncover the limitations of modern algorithmic music composition instead of starting with a complex model.

1.2 Data Preprocessing

The most common digital form of music is the MIDI file, which represents pitch, duration, and other details for each note. In this work, we focus solely on J.S. Bach MIDI files, extracting just three features (pitch, step, and duration) per note to form the core of our training data.

Below is a concise outline of the preprocessing pipeline:

(1) **Data Acquisition and Parsing.** We downloaded 228 MIDI files from www.bachcentral.com [2] and used the `pretty_midi` library [11] to read and parse note events (pitch, start time, end time).

(2) **Flattening Polyphony.** In polyphonic passages, simultaneous notes were sorted by pitch and concatenated into a single sequence of note events. While this removes harmonic verticality, it preserves local pitch relationships and timing in a single-stream format.

(3) **Feature Extraction.** For each note:
   (a) *Pitch*: MIDI pitch number ($0 \leq p \leq 127$),
   (b) *Step*: Time gap between note onsets (current note start minus previous note start),
   (c) *Duration*: Length of the note (note end time minus its start time).

   These were stored in a Pandas DataFrame for each file and then concatenated into a single large dataset.

(4) **Optional Transposition.** To reduce key-specific bias, some pieces were (optionally) transposed to a standard key (e.g., C major/A minor).

(5) **Sequence Construction.** The final step was to split the note events into fixed-length subsequences (e.g., 50 notes each) with partial overlap, creating training examples suitable for the LSTM.

Neural networks are known for their sensitivity to hyperparameter tuning, and these minimal features (pitch, step, duration) helped constrain model complexity. Similarly, using a smaller sequence length and maintaining a relatively modest dataset reduced computational overhead while still providing enough variety for the model to learn basic Bach-like patterns.

1.3 History of Bach and 18th Century Counterpoint

J.S. Bach's music is regarded as the gold standard of 18th-century Western music. Bach manages to do seemingly infinite permutations of the same subject material throughout his works. This contrasts with Baroque composers like Domenico Scarlatti, who use multiple subject materials throughout a single work. The phrase "less is more" holds significant weight in the context of Bach's compositional and structural styling. For example, one of Bach's most famous works, the Prelude, Fugue, and Allegro in E-flat major, BWV 998, is a three-movement suite, approximately 12 minutes in length, that sparsely employs a single eight-note subject in various permutations throughout the entire work. Note that the suite BWV 998 is considered a secular work; its subject is similar to Martin Luther's hymn "Vom Himmel hoch da komm ich her" [9].

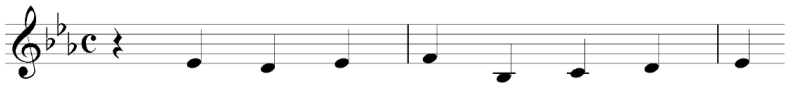

Fig. 1. BWV 998.ii (Fugue Subject)

Given Bach's music's limitations and his expert application of the rules of 18th-century counterpoint voice leading, his musical corpus is seemingly a strong candidate for use as data within a continuous dynamical system that learns a discretized approximation of his compositional style to generate natural and convincing music.

2 Materials and Methods

To assemble the dataset for this project, we downloaded 228 MIDI files from www.bachcentral.com [2], spanning Bach's varied repertoire—violin concertos, four-part chorales, fugues, preludes, cantatas, and solo instrumental works. We used the `pretty_midi` library [11] within a Jupyter Notebook environment to parse these files and verify playback.

Specifically, `pretty_midi` facilitated the extraction of each note's pitch, duration, and step (onset gap), which were then stored in a Pandas DataFrame for further processing.

The deep learning model was implemented with Google Brain's TensorFlow API [1], chosen for its scalability and extensive documentation. While TensorFlow's high-level abstractions streamline model building, leveraging them effectively does require a working knowledge of object-oriented programming, multivariate calculus, and linear algebra. Our final architecture was intentionally minimal, consisting of a single long short-term memory (LSTM) layer with 128 units. This choice aligns with the standard MIDI protocol, which defines 128 distinct pitches, making it a natural fit for representing Bach's musical vocabulary.

2.1 Continuous Dynamical Systems in RNN Music Generation

The primary method used in this investigation was an LSTM (Long Short-Term Memory) neural network, a type of recurrent neural network (RNN). RNNs can be derived from a general nonlinear first-order ordinary differential equation that describes the change of the input state $\vec{s}(t)$ as a function of time, t [12].

$$\frac{d\vec{s}(t)}{dt} = \vec{f}(t) \ + \vec{\phi}$$

Where $\vec{f}(t)$ represents an d-dimensional vector-valued function of time, $t \in \mathbb{R}^+$ describing how the state changes, and $\vec{\phi}$ is a constant d-dimensional vector [12]. Another way to explain $\vec{f}(t)$ is:

$$\vec{f}(t) = \vec{a}(t) + \vec{b}(t) + \vec{c}(t)$$

Whose terms are d-dimensional vector-valued functions of time t. The above equation appears in numerous applications in biology and engineering, and is known as the "Additive Model" in Brain Dynamics research. The vector-valued terms can potentially represent nonlinear rates of change that can model the behavior of neuronal activity $\vec{s}(t)$, in the context of generating music. The input signal state $\vec{s}(t)$can be considered parameters (represented as matrices) that represent sequences of notes, while the additive equation terms $\{\vec{a}(t),\ \vec{b}(t),\ \vec{c}(t)\}$ are learning parameter estimations, and ϕ are a constant vector-valued term.

When training an LSTM on a specific music corpus, such as J.S. Bach chorales, we can view the network as learning a *discrete approximation* of an underlying "musical dynamics" equation. In continuous form, suppose we have a state vector.

$$\vec{s}(t) \in \mathbb{R}^d$$

evolving according to

$$\frac{d\vec{s}(t)}{dt} = \vec{a}(t) + \vec{b}(t) + \vec{c}(t) + \vec{\phi},$$

where $\vec{a}(t)$ $\vec{b}(t)$, and $\vec{c}(t)$ are distinct (possibly nonlinear) contributions to the rate of change in $\vec{s}(t)$, and $\vec{\phi} \in \mathbb{R}^d$ is a constant bias term. Discretizing time (e.g., one step per musical event) yields an update rule akin to

$$\vec{s}_{t+1} = \vec{s}_t + \left[\vec{a}_t + \vec{b}_t + \vec{c}_t\right] + \vec{\phi},$$

where the functions $\vec{a}_t$, $\vec{b}_t$, and $\vec{c}_t$ become learnable parameters in the LSTM. In a music-generation context, the state vector $\vec{s}(t)$ can represent all necessary information for predicting the following note, such as pitch, step, duration, and local harmonic context, thus enabling the model to capture the evolution of musical events over time. As a result, a network trained on Bach chorales naturally learns "Bach-like" motion in pitch and rhythm, mirroring how continuous dynamical systems update their states in an additive manner.

2.2 The Current Approach and Application of the Model to Data

Our approach models each note as a state vector $\vec{s}(t)$ comprising pitch, step, and duration, with $\vec{f}(t)$ serving as the adaptive learning rate parameters optimized by the ADAM algorithm. Because the model is configured for single-note melody generation, we paid particular attention to sparsity, especially crucial given the mixture of polyphonic and monophonic Bach works in the dataset (228 MIDI files from www.bachcentral.com [2]).

Flattening chords into consecutive note events introduced potential overlaps in voice-leading contexts, which can complicate the conditional probability distribution for a monophonic model. Nonetheless, leveraging a sparse architecture offers two key benefits:

(1) **Longer training windows:** With fewer parameters, training can span more epochs without overfitting.
(2) **Faster convergence:** Smaller batches and shorter sequences allow the model to learn local melodic transitions more efficiently.

In practice, reducing the ADAM learning rate consistently yielded greater harmonic and rhythmic coherence output. By emphasizing core pitch–step–duration relationships, this minimalist design converged on Bach-like melodic tendencies while keeping model complexity in check.

3 Results

Our vanilla LSTM model was trained on three core features—pitch, step (interval distance), and duration—extracted from 228 MIDI files of J.S. Bach. Among these, **pitch prediction proved most challenging**, likely because polyphonic source material introduced conflicting harmonic contexts and data leakage, increasing stochasticity in a single-voice setting. Despite this challenge, generated outputs exhibit clear tonal grammar: diatonic stepwise motion, proper leading-tone resolutions to tonic, and concise motifs that imply antecedent–consequent phrasing. Audio samples (e.g., [8], [7]) highlight scalar passages in G major and well-resolved tendency tones that strongly evoke Baroque melodic style.

Rhythmic consistency, however, remains limited: without explicitly encoding barlines or beat positions, the network produces uneven subdivisions and irregular timing that do not reflect Bach's strict metric grids.

Leveraging a **sparse architecture**with zero inputs, accelerated convergence, and simplified optimization. This setup highlights a **bias–variance tradeoff**: architectural simplicity facilitates the rapid learning of local tonal relationships but inherently sacrifices rhythmic precision and polyphonic coherence.

In contrast, **DeepBach** [3] employs four interdependent networks conditioned on neighboring voices and metrical position to generate fully aligned SATB chorales with complex harmonic progressions and rigid rhythmic alignment. Our monophonic LSTM, which captures only immediate melodic context, reveals foundational learning capabilities yet falls short of the multi-voice control required in more sophisticated systems.

Future work could employ quantitative metrics—such as tonal tension curves, step–leap distributions, and onset-variance scores—to systematically assess melodic regularity alongside rhythmic stability.

4 Discussion

18th-century counterpoint is governed by rigorous voice-leading rules that govern 18th-century counterpoint govern 18th-century counterpoint govern 18th-century counterpoint avoiding parallel fifths and octaves, favoring stepwise motion, and balancing phrase lengths—all of which create statistical regularities readily learned by sequence models. By training exclusively on Bach's corpus, these implicit constraints remained intact in the data, allowing our LSTM to internalize elements of his tonal logic without hard-coding any theoretical rules.

Generated lines in melodic excerpts such as [7]demonstrate authentic **tendency-tone resolutions** and **diatonic scalar motion**, reflecting Bach's melodic grammar. Nonetheless, the absence of explicit **metrical encoding** (beats and barlines) produces irregular subdivision patterns that fall short of Bach's precise rhythmic framework.

More advanced systems extend beyond monophonic sequences. **DeepBach** [5] employs four conditional networks—one per voice—alongside Gibbs sampling to enforce polyphonic counterpoint and rhythmic alignment across SATB parts. **BachBot** [10] integrates convolutional layers and attention mechanisms to capture chorales' long-range dependencies and hierarchical structures.

Our vanilla LSTM presents a **minimalist proof of concept**, showing that raw pitch–duration–step sequences can learn core tonal relationships. However, achieving complete stylistic pastiche will require enhancements such as **polyphonic conditioning**, **metrical embedding**, and **constraint-based sampling** techniques (e.g., beam search or Gibbs sampling) to enforce harmonic and rhythmic consistency.

5 Acknowledgments

I want to thank Dr. Shyamal Mitra for his assistance and guidance throughout the project, and Dr. Gustavo Cepparo for helping me understand The mathematical derivations of RNNs.

6 Computer Code

This code snippet represents the RNN model architecture.

```
input_shape = (seq_length, 3)
learning_rate = 0.0025

inputs = tf.keras.Input(input_shape)
x = tf.keras.layers.LSTM(128)(inputs)

outputs = {
  'pitch': tf.keras.layers.Dense(128, name='pitch')(x),
  'step': tf.keras.layers.Dense(1, name='step')(x),
  'duration': tf.keras.layers.Dense(1, name='duration')(x),

}

model = tf.keras.Model(inputs, outputs)

loss = {
      'pitch': tf.keras.losses.SparseCategoricalCrossentropy(
          from_logits=True),
      'step': mse_with_positive_pressure,
      'duration': mse_with_positive_pressure,
}

optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
model.compile(loss=loss, optimizer=optimizer)
model.summary()
\end{verbatim}
```

Listing 1. Python example

6.1 Code Availability

All source code and documentation for this project can be found at https://github.com/logankronforst/RNN-Music-Gen

7 Reflection

This investigation serves as a proof of concept for learning tonal grammar directly from raw monophonic MIDI sequences; however, it reveals several critical limitations and avenues for improvement. The decision to train on both monophonic and polyphonic Bach works introduced conflicting harmonic contexts that the single-voice LSTM could not disentangle, leading to **increased stochasticity** and occasional pitch inaccuracies. Moreover, the absence of explicit **metrical annotations** (beat and bar structure) in the input prevented the network from internalizing consistent rhythmic patterns, as evidenced by irregular note durations and uneven subdivisions in generated outputs.

From a modeling perspective, the use of a **sparse vanilla LSTM** facilitated rapid convergence. It highlighted the core capacity to learn local melodic transitions, but at the cost of sacrificing **hierarchical and polyphonic structure**. Future architectures should incorporate **multi-scale modeling**, where a lower-level network captures fine-grained note-by-note transitions and a higher-level module (e.g., an attention-based or Transformer encoder) learns phrase- and section-level dependencies. Embedding **metrical position vectors** alongside pitch–step–duration inputs would allow the network to align generated notes to a musical grid, thereby restoring rhythmic regularity and bar-level consistency.

To address conflicting harmonic signals, a **voice-separated training regime** could isolate soprano, alto, tenor, and bass lines from SATB chorales, feeding each network only its respective voice while conditioning on others via cross-voice features. Alternatively, a **chordal embedding layer** could supply explicit harmonic context (e.g., root and quality of the underlying chord), enabling the model to ground melodic predictions within a harmonic framework rather than inferring it implicitly from pitch sequences alone.

On the optimization front, introducing an **error-term regularizer** in the loss function—modeling predicted variance through a probabilistic distribution—would help the network manage noisy inputs and avoid overfitting to spurious patterns. **Data augmentation** strategies, such as random transposition, tempo normalization, and dynamic range adjustment, could expand the diversity of training examples and improve model generalization. Furthermore, initializing the model with weights from pre-trained language models in music (e.g., Music Transformer) and fine-tuning it on Bach data offers a promising transfer learning approach.

Finally, rigorous **evaluation frameworks** are essential for assessing progress beyond subjective listening tests. Quantitative metrics—tonal tension curves, pitch class entropy, step–leap ratios, onset-variance scores, and rhythmic histogram similarity—will allow systematic comparison across models. Incorporating human-centered studies with composers and theorists can validate whether generated outputs possess the stylistic nuances of Bach's counterpoint. In the long term, integrating constraint-based sampling methods (Gibbs sampling, beam search with rule-based filters) with learned probability distributions holds the promise of generating **polyphonic, rhythmically coherent** chorales that more fully embody the spirit of 18th-century counterpoint.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Johann Sebastian Bach. n.d.. Prelude in C Major, BWV 846 [MIDI file]. https://www.bachcentral.com/midi.html. Accessed: 2025-04-24.

[3] Jean-Pierre Briot. 2020. From artificial neural networks to deep learning for music generation: history, concepts and trends. *Neural Computing and Applications* 32 (2020), 981–993. doi:10.1007/s00521-019-04123-7

[4] Alex Graves. 2014. Generating Sequences With Recurrent Neural Networks. arXiv:1308.0850 [cs.NE] https://arxiv.org/abs/1308.0850

[5] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. 2017. DeepBach: a Steerable Model for Bach Chorales Generation. arXiv:1612.01010 [cs.AI] https://arxiv.org/abs/1612.01010

[6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[7] Logan Kronforst. 2025. RNN-02: Bach-style Single-Voice Melody. Audio file (RNN-02.mp3). Available at https://github.com/logankronforst/RNN-Music-Gen/RNN-02.mp3 (accessed April 2025).

[8] Logan Kronforst. 2025. RNN-04: Bach-style Single-Voice Melody. Audio file (RNN-04.mp3). Available at https://github.com/logankronforst/RNN-Music-Gen/RNN-04.mp3 (accessed April 2025).

[9] Anne Leahy. 2005. Bach's Prelude, Fugue and Allegro for Lute (BWV 998): A Trinitarian Statement of Faith? *Journal of the Society for Musicology in Ireland* 1 (2005), 33–51. https://jsmi.musicologyireland.ie/index.php/journal/article/view/9 Accessed: 2025-04-27.

[10] Feynman T Liang, Mark Gotham, Matthew Johnson, and Jamie Shotton. 2017. Automatic Stylistic Composition of Bach Chorales with Deep LSTM.. In *ISMIR*. 449–456.

[11] Colin Raffel and Daniel P. W. Ellis. 2014. Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty_midi. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*. https://colinraffel.com/publications/ismir2014_tutorial.pdf

[12] Alex Sherstinsky. 2020. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena* 404 (March 2020), 132306. doi:10.1016/j.physd.2019.132306