Symbolic Music Generation with Deep Learning Techniques

github.com/ffsgfy

August 15, 2023

Abstract

Deep learning-based language modeling has received much attention in recent years from both the general public and the academic research community, catalyzed by the invention of the Transformer architecture. Symbolic music generation can be formulated as a highly expressive language modeling task with a small vocabulary and relatively simple structure, which should allow usage of language models with a low parameter count. This work evaluates the performance of two similar sized ($\sim$4 million parameters) models — an LSTM and a Transformer — on the task of symbolic music generation in a low-data, low-compute setting. Results show that, contrary to popular belief, even at small sizes the Transformer can outperform the LSTM, both in terms of testing loss and the quality of generated data.

1 Introduction

Instrumental music, at its core, consists of various musical pitches, or notes, sounding at specific points in time. The number of available notes can be rather small (88 on a standard piano, for instance), so the bulk of musical complexity comes from the way they are arranged into chords, melodies, rhythms and motifs. This makes music generation an attractive target for deep learning research, which typically excels at recognizing implicit patterns in raw data.

For this purpose, several different music representations can be used, such as: raw audio signal [23]; piano roll images, with axes for time and note pitch [1, 17]; sequence of textual [22] or domain-specific tokens [20, 13, 11]. For any specific representation, certain model architectures can be expected to perform better than others — for instance, it is well-known that convolutional networks work well for signal-based data (audio or images), while recurrent networks are better suited for modeling sequences of discrete tokens. Since this work aims to

compare the performance of an LSTM and a Transformer in the same setting, a custom token-based encoding scheme, which should be suitable for both model types, is utilized.

LSTMs are recurrent networks, which means they process input tokens one at a time, updating their internal state along the way [10]. More specifically, at time step t they receive as input token x_t, hidden state h_{t-1} and cell memory c_{t-1}, and produce as outputs new cell memory c_t and hidden state h_t, the latter of which can be passed to downstream network layers. The use of a separate cell memory enables the network to combat vanishing gradients and capture longer-term dependencies compared to plain RNNs.

Transformer networks were introduced in the seminal paper *Attention Is All You Need* [24]. They rely solely on their attention mechanism, which essentially computes a weighted average of all input tokens based on a pairwise token similarity metric. Specifically, input token embeddings are first linearly projected[1] to form three different sets of vectors, commonly referred to as queries, keys and values. Then, for each query, its similarity to all keys is computed using a dot product, these similarities are converted to weights using the softmax function, and those weights are used to compute a weighted average of all values. The output is then passed through a small (typically 2-layer) feed-forward network; it contains the same number of tokens as the input. This procedure is usually modified by splitting input token projections into several independent parts, computing attention inside each part separately, then concatenating the outputs and passing them through an extra linear projection, forming multi-headed attention. Furthermore, since all input time steps are processed in parallel, the architecture, unlike recurrent networks, has no intrinsic knowledge of the order in which its inputs are arranged, so this information has to be injected into the model in the form of absolute or relative positional embeddings [24, 21]. In order to apply Transformers to autoregressive sequence generation tasks, a causal inductive bias is introduced by masking attention weights (before softmax) such that any given token only attends to tokens preceding it in time. A full Transformer model, then, consists of an input embedding, a series of multi-headed masked attention modules with positional embeddings, and a final feed-forward decoder network.

Much of the success of Transformers in language modeling can be attributed to their exceptional ability to learn from large corpora of data, which also requires a very large number of trainable parameters and gives rise to so-called large language models (LLMs). However, as this work demonstrates, even small transformers can achieve remarkable performance when applied to music generation, with overall output quality superior to that of an LSTM with a similar number of parameters.

[1] This describes self-attention; in general, queries and keys/values can be obtained from different inputs.

2 Related Work

LSTMs were first used for music generation over 20 years ago in [4], where a piano roll representation was utilized. A more recent and complex approach involving two LSTMs — acting separately along the time and pitch axes — was presented in [14] and further refined in [17]. A token-based LSTM was shown in [18] to be able to create expressive performances with little long-term structure. The effects of injecting various conditioning tokens into LSTM inputs were studied in [6], with some combinations displaying significant performance gains.

The most influential early paper on Transformers in the context of music generation was [12], which described an efficient way of computing relative positional embeddings introduced in [21] and demonstrated their model's ability to maintain long-term structure and coherence. In [13] an alternative music encoding scheme, named REMI, was introduced, which, among other things, embeds a metrical grid into the input data. In an effort to separate processing of different token types, such as pitch and timing tokens, [11] proposed using compound words in a Transformer model where different heads were used to process different token types. In [7] a long-context autoregressive Transformer architecture was introduced, which was able to directly attend to tens of thousands of tokens by mapping them into a small latent space using cross-attention.

3 Data Preparation

The dataset used in this work consists of soundtracks from Final Fantasy game series, arranged for piano and encoded in MIDI format. They were chosen for their abundance and relative compositional simplicity (especially in comparison with classical music), which makes them easier to learn for smaller networks. The files were collected from `https://www.oocities.org/groveman64/ffpc.htm` and split into training and testing sets, totaling 214 and 24 files respectively.

3.1 Encoding Scheme

Data is encoded in a fashion similar to REMI [13], albeit with significant simplifications. Specifically, only two main musical objects are encoded: bars and notes. A bar is encoded as a pair of tokens `(offset, bar)`, where `offset` represents the time offset from the previous bar token, and `bar` is a special marker token; this allows generated musical bars to vary in length. A note is encoded as a triplet of tokens `(offset, pitch, duration)`, where `offset` represents the time offset from the previous bar token, `pitch` represents this note's pitch, and `duration` — the duration of time for which this note is to be held.

A unit of time is equal to a single 16th note. Any notes shorter than a 16th are discarded, and offsets and durations are truncated if necessary. Offset values range from 0 to 31, and duration values — from 1 to 32. If a longer time period needs to be encoded, the corresponding musical object is split; for example, a bar of length 40 becomes two bars of lengths 31 and 9, and a note with duration 36 becomes two notes with durations 32 and 4. Note pitches range from 0 to 87 — one for each key on a standard piano. In total this yields 153 different tokens: 1 bar token, 32 offset tokens, 88 pitch tokens, and 32 duration tokens.

4 Model Specifications

Both models, for each input time step, produce a logarithmic probability distribution for the next token given the current and all previous input tokens. These outputs can then be converted to proper discrete probability distributions by applying a softmax function to them.

4.1 LSTM

The LSTM-based model consists of the following layers, in order:

1. input embedding of dimension 64;
2. 2 LSTM layers with hidden sizes of 512;
3. linear layer with output size of 1024;
4. linear layer with output size of 512;
5. linear layer with output size of 153.

Layer normalization [2] and ReLU activation [5] is applied to the outputs of layers #3 and #4. Additionally, to the outputs of those layers, as well as the outputs of both LSTM layers, dropout regularization [9] with probability of 0.5 is applied. This model has 4 426 457 trainable parameters in total.

4.2 Transformer

The Transformer-based model consists of the following layers, in order:

1. input embedding of dimension 256;
2. 8 causal self-attention layers of dimension 256 split into 16 heads;

3. linear layer with output size of 153.

Each self-attention layer, in turn, contains the following sublayers:

1. linear projection into sets of queries, keys and values, each of dimension 256;
2. causal scaled dot product self-attention with relative positional embeddings;
3. linear projection of the result of head concatenation;
4. linear layer with output size of 512 and ReLU activation [5];
5. linear layer with output size of 256.

These layers include dropout regularization [9] with probability 0.5, layer normalization [2] and residual connections as described in the original paper [24]. Relative positional embeddings are implemented according to [12], with maximum relative distance between tokens limited to 239 (i.e. for a total of 240 positional embedding vectors to be learned); they are shared between all 16 heads.

Additionally, residual attention links between self-attention layers are added, as introduced in [8]. This model has 4 326 041 trainable parameters in total.

5 Experimental Setup and Results

5.1 Inputs and Outputs

Data is loaded from MIDI files using `muspy` [3] and converted into its token-based representation, from which overlapping windows of length 257 are extracted. Each such window is a single data point; the first 256 of its tokens are used as model inputs, and the last 256 tokens — as output targets (i.e. the models are trained to predict not only the "next" token, but the entire input token sequence shifted one time step into the future), with average negative log-likelihood being the optimization target. In total, the training and testing datasets contain 639 470 and 70 248 data points respectively.

5.2 Training and Inference

Both models are implemented in PyTorch [19]. They were trained on a single Nvidia GTX 960 GPU with a batch size of 8. In early experiments, using the entire training dataset for each epoch led to very quick overfitting of both models, possibly due to the fact that many data points significantly overlap with each other, so epoch size was limited to 2% of the training dataset (sampled at random). After

each epoch, models were saved and evaluated on the testing dataset. Training was halted when testing loss stopped improving, after which the model from the epoch with the lowest testing loss was selected.

Adamax [16] with default settings was used as the optimizer. For the LSTM model, learning rate was set to exponentially decay by a factor of 0.8 per epoch from an initial value of $5 \cdot 10^{-4}$ down to a lower limit of $5 \cdot 10^{-5}$. The Transformer model was trained with a fixed learning rate of 10^{-3}.

Since the data encoding scheme has to maintain a certain structure to remain valid (e.g. a `pitch` token must be followed by a `duration` token), that structure is artificially enforced at inference time by masking out log probabilities of invalid tokens with $-\infty$. These probabilities are then passed through a softmax, and the resulting discrete probability distribution is sampled to produce the next token in autoregressive fashion.

5.3 Results

The best testing loss achieved by the LSTM was 1.425 after 9 training epochs, and by the Transformer — 1.237 after 6 training epochs. It should be noted that the Transformer model was much more sensitive, compared to the LSTM, to variations in certain hyperparameters such as batch size and learning rate, which added to its training difficulty (given that a full hyperparameter sweep was infeasible due to limited compute resources).

Figures 1 and 2 show samples of sequences of 1024 tokens generated unconditionally[2] by both models. The Transformer model is able to generate bass lines with clear chord-based structure, which can also evolve over time without losing consistency; its melodies harmonize fairly well with the accompaniment and largely stay in one place, utilizing repeated motifs with various — at times even complex — rhythms. The LSTM is able to generate rather interesting melodic patterns, but it lacks any sort of long or mid-term consistency, both in terms of note pitches and durations; its bass lines typically consist of single-key repetitions with occasional movement, and the overall harmony is often disturbed by unpleasant, dissonant sounds.

[2]To be precise, they were conditioned on a single `bar` token.

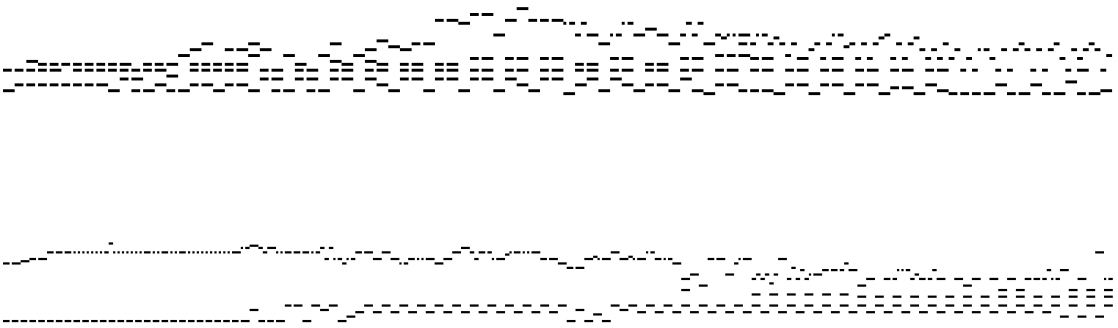

Figure 1: Transformer unconditional generation sample piano rolls

Figure 2: LSTM unconditional generation sample piano rolls

Figures 3 and 4 show samples of sequences of 768 tokens generated by both models conditioned on the same sequence of 256 tokens taken from the test dataset (shown in grey). The Transformer fails to capture the main motif of its conditioning sequence, but nonetheless maintains melody locality and basic structure — repeated single-note movements interleaved with wide chords. The generated melody, while distinctly different in style, connects smoothly to the seeded part and then develops further in a fairly graceful manner. The LSTM model generates music almost completely disconnected from the seed, not only in terms of style, but also pitch range and rhythm. However, these generated sequences in themselves are more consistent, complex and expressive than those created unconditionally.

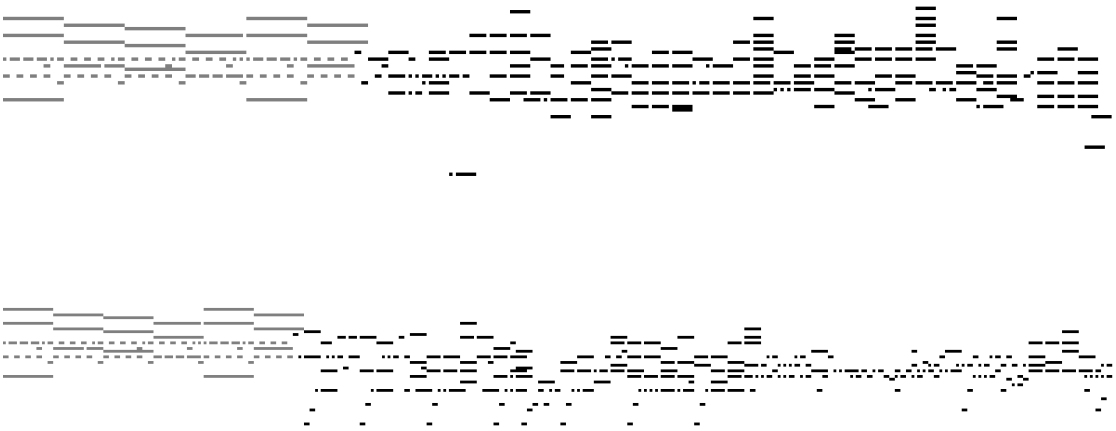

Figure 3: Transformer conditional generation sample piano rolls

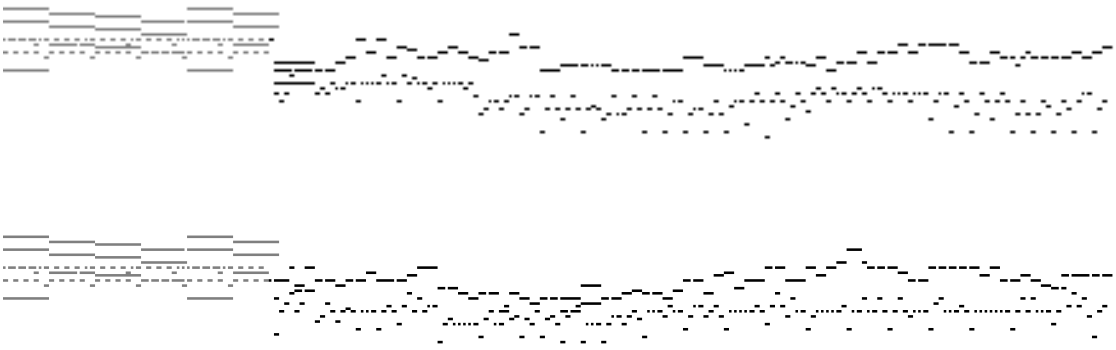

Figure 4: LSTM conditional generation sample piano rolls

6 Conclusion

Evidently, even small Transformer models are capable of generating music of surprisingly good and consistent quality, given proper configuration and a suitable dataset. Furthermore, as Transformers are known for their ability to learn from large amounts of data, the quality could be further improved via simple dataset augmentation, which was omitted here due to resource limitations. In comparison, a similar sized LSTM model seems to quickly lose its sense of direction and starts to generate sequences of notes that, while being relatively harmonious, lack any long-

term structure; it should be noted, however, that conditional generation results seem to surpass unconditional ones in terms of overall quality.

One major drawback of vanilla Transformers is their $\mathcal{O}(L^2)$ time complexity compared to LSTM's $\mathcal{O}(L)$, where L is the length of the generated sequence. It is possible to overcome this limitation by reformulating autoregressive transformers as RNNs, as has been demonstrated in [15], yielding linear time complexity at inference time. As such, it appears that Transformers are superior to LSTMs in practically every aspect, at least when evaluated on the particular task examined in this work.

Model implementation and training code, conditional and unconditional inference examples, and generated samples, both in `.mid` and `.mp3` formats, can be found at `https://github.com/ffsgfy/music-generation`. Model training checkpoints can be found at `https://drive.google.com/file/d/1MQie3Nu8M9y3KjH8ZtTUUh2ydvgyyhRd/view`.

References

[1] Lilac Atassi, *Generating symbolic music using diffusion models*, 2023.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton, *Layer normalization*, 2016.

[3] Hao-Wen Dong, Ke Chen, Julian McAuley, and Taylor Berg-Kirkpatrick, *Muspy: A toolkit for symbolic music generation*, 2020.

[4] Douglas Eck and Juergen Schmidhuber, *A first look at music composition using lstm recurrent neural networks*, 2002.

[5] Kunihiko Fukushima, *Visual feature extraction by a multilayered network of analog threshold elements*, IEEE Transactions on Systems Science and Cybernetics **5** (1969), no. 4, 322–333.

[6] Benjamin Genchel, Ashis Pati, and Alexander Lerch, *Explicitly conditioned melody generation: A case study with interdependent rnns*, 2019.

[7] Curtis Hawthorne, Andrew Jaegle, Cătălina Cangea, Sebastian Borgeaud, Charlie Nash, Mateusz Malinowski, Sander Dieleman, Oriol Vinyals, Matthew Botvinick, Ian Simon, Hannah Sheahan, Neil Zeghidour, Jean-Baptiste Alayrac, João Carreira, and Jesse Engel, *General-purpose, long-context autoregressive modeling with perceiver ar*, 2022.

[8] Ruining He, Anirudh Ravula, Bhargav Kanagal, and Joshua Ainslie, *Realformer: Transformer likes residual attention*, 2021.

[9] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, 2012.

[10] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural Computation **9** (1997), no. 8, 1735–1780.

[11] Wen-Yi Hsiao, Jen-Yu Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang, *Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs*, 2021.

[12] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck, *Music transformer*, 2018.

[13] Yu-Siang Huang and Yi-Hsuan Yang, *Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions*, 2020.

[14] Daniel D. Johnson, *Generating polyphonic music using tied parallel networks*, Computational Intelligence in Music, Sound, Art and Design (Cham) (João Correia, Vic Ciesielski, and Antonios Liapis, eds.), Springer International Publishing, 2017, pp. 128–143.

[15] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret, *Transformers are rnns: Fast autoregressive transformers with linear attention*, 2020.

[16] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 2017.

[17] Huanru Henry Mao, Taylor Shin, and Garrison Cottrell, *DeepJ: Style-specific music generation*, 2018 IEEE 12th International Conference on Semantic Computing (ICSC), IEEE, Jan 2018.

[18] Sageev Oore, Ian Simon, Sander Dieleman, and Doug Eck, *Learning to create piano performances*, NIPS 2017 Workshop on Machine Learning and Creativity, 2017.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, *Pytorch: An imperative style, high-performance deep learning library*, Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[20] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck, *A hierarchical latent vector model for learning long-term structure in music*, Proceedings of the 35th International Conference on Machine Learning (Jennifer Dy and Andreas Krause, eds.), Proceedings of Machine Learning Research, vol. 80, PMLR, 10–15 Jul 2018, pp. 4364–4373.

[21] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani, *Self-attention with relative position representations*, 2018.

[22] Bob L. Sturm, João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova, *Music transcription modelling and composition using deep learning*, 2016.

[23] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu, *Wavenet: A generative model for raw audio*, 2016.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, 2017.