Generate Polyphonic Music with Multivariate Masked Autoregressive Flow

Massimiliano Sirgiovanni [1] and Daniele Castellana[1]

Università degli Studi di Firenze - Department of Statistics, Informatics and Applications, Viale Morgagni, 59, Firenze - Italy

Abstract. This paper explores the use of the Masked Autoregressive Flow (MAF) model for music generation, specifically addressing its limitation to univariate time series. To extend MAF for polyphonic melodies, three approaches are proposed and tested on the Lakh Pianoroll Dataset. The results show promising accuracy and the model's ability to generate original, pleasing melodies, demonstrating the potential of this innovative interdisciplinary approach.

1 Introduction

Music generation theory aims to decode the mathematical principles behind melody creation, enabling the reproduction of pleasing tunes. to this end, key challenges such as data representation, adherence to melodic rules, quality evaluation, and modifying generated compositions should be faced.

In the literature, melodies are often treated as monophonic, meaning only one note is played at a time. For example, in [1], the authors employ an adversarial autoencoder to generate new music samples. However, the melodies are treated as temporal sequences in which each time step corresponds to a single played note or a silent state. Similarly, [2] proposes a recurrent VAE with a hierarchical decoder to improve long-term dependencies, but still within a monophonic framework.

While there are other approaches which generate polyphonic melodies (e.g. [3]), the application of Masked Autoregressive Flow (MAF) [4] in this context is unexplored. MAF is an autoregressive flow-based model that is used to perform density estimation. In [4], this architecture has been applied only to univariate time series and images.

The goal of this paper is to extend the MAF architecture to multivariate time series to generate polyphonic melodies. A polyphonic melody can be interpreted as a temporal sequence of vectors of size D, where D is the number of possible notes. We propose three MAF architectures to handle multivariate sequences with different inductive biases: the first treats each note separately with shared parameters; the second uses separate weights for each note while assuming independence; and the third accounts for dependencies between notes across time steps.

2 MAF Models for Music Generation

The MAF [4] is a flow-based model leveraging Masked Autoregressive for Density Estimation (MADE) [5] as a fundamental component to effectively manage sequential data and compute exact data densities. Let $[x_1, \ldots, x_T]$ be an input sequence. MAF assumes that it is generated as follows:

$$x_t = u_t \cdot \exp \alpha_t + \mu_t \quad \text{with} \quad u_t \sim \mathcal{N}(0, 1), \tag{1}$$

where $\mu_t = f_{\mu_t}(x_{1:t-1})$ and $\alpha_t = f_{\alpha_t}(x_{1:t-1})$ are estimated using the previous elements in the sequence $x_{1:t-1}$. Here, u_t represents a latent random variable sampled from a simple known distribution, such as a standard normal distribution. Thanks to this formulation, MAF enables the generation of new sequences by simply sampling new values of u_t. Notably, Eq. (1) is invertible, allowing transformation between data x and latent variables u.

Going into details, MAF assumes that each observable variables x is generated from an univariate Gaussians with parameters μ_t and α_t, representing the mean and log-variance respectively. These parameters are computed via two invertible autoregressive functions f_{μ_t} and f_{α_t}, enabling transformations between complex data distributions and simpler ones. These two functions are implemented using a MADE network [5], which consist of a stack of masked linear layers followed by activation functions. At each layer, MADE produces a new sequence $[h_1, \ldots, h_T]$, where $\boldsymbol{h}_t \in R$ represents the hidden representation at time-step t. Thanks to the masked linear layer, MADE allows density evaluations without the typical sequential loop of autoregressive models, thus making MAF computationally efficient and well-suited for parallel execution on modern architectures such as graphics processing units (GPUs).

In the above presentation (and in [1]), the input sequence is assumed to be univariate (i.e., each element x_t is a real number). In the following, we propose three different MAF architectures to handle *multivariate* sequence (each element $\boldsymbol{x}_t$ is a vector of size D, i.e. $\boldsymbol{x}_t \in \mathbb{R}^D$; thus, we represent a multivariate time-series as a matrix of size $T_{in} \times D$). To this end, we modify the masked linear operator in order to handle multiple features at each time-step. This step is crucial to employ MAF in the context of polyphonic music generation where each element in the sequence is a vector of size D, where D is the number of notes that could be played.

Shared Univariate MAF (SU-MAF) treats each melody as D distinct univariate sequences processed with the same parameters:

$$h_{ui}^{l+1} = \sum_{t=1}^{T_{in}} W_{ut}^l M_{ut}^l h_{ti}^l + b_u^l, \tag{2}$$

where $W^l \in \mathbb{R}^{T_{out} \times T_{in}}$ and $b^l \in \mathbb{R}^{T_{out}}$ are the parameters of the l-th layer, and M^l is a mask which ensures that the autoregressive property is ensured. The output of the layer is again a multivariate sequence with T_{out} elements of size D (thus, it is represented by a matrix of size $T_{out} \times D$).

Different Univariate MAF (DU-MAF) again processes the melody as D univariate sequence; however, it employs a different set of parameters for each note:

$$h_{ui}^{l+1} = \sum_{t=1}^{T_{in}} W_{uti}^{l} M_{ut}^{l} h_{ti}^{l} + b_{ui}^{l}, \tag{3}$$

where $W^l \in \mathbb{R}^{T_{out} \times T_{in} \times D}$ and $b_i^l \in \mathbb{R}^{T_{out} \times D}$ are the parameters. Again, the mask M ensures the autoregressive property and therefore it does not depend on the note.

Tensor-Multivariate MAF (TM-MAF) aims to capture the relationships between different notes at different time steps. To achieve this, we cannot process each note separately as a univariate sequence (as we have done in the previous architectures):

$$h_{uk}^{l+1} = \sum_{t=1}^{T_{in}} \sum_{i=1}^{D} W_{utik}^{l} M_{ut}^{l} h_{ti}^{l} + b_{uk}^{l}, \tag{4}$$

where $W^l \in \mathbb{R}^{T_{out} \times T_{in} \times D \times K}$ and $b_i^l \in \mathbb{R}^{T_{out} \times K}$ are the parameters of the l-th layer. The mask M ensures the autoregressive property and therefore it does not depend on the note.

The output of the layer is again a multivariate sequence with T_{out} elements of size K (thus, it is represented by a matrix of size $T_{out} \times K$). The dimension K is a hyper-parameter and it allows the generation of vectors with a size that is different from the input one.

The mask M remains unchanged from the previous case because the notes do not have any autoregressive dependence, meaning a note at the i-th time step depends on all the states assumed by all the notes in the preceding $i-1$ time steps. Therefore, the mask is applied only to the time steps, adjusting its dimensions to be able to multiply it with the weights.

3 Experimental Analysis

To assess the differences among the proposed architectures, we conduct an experimental analysis on the Lakh Pianoroll Dataset Cleansed[1], containing 21,425 multi-track piano rolls. We focus only on melodies that use $\frac{4}{4}$ (the most commonly used); therefore, each measure consists of four beats, each of which is valued at $\frac{1}{4}$. In the case of this study, tracks were divided into sub-tracks of two measures each. The granularity chosen was the sixteenth note, where one time step represents $\frac{1}{16}$ of a whole note. As done in [1], a sliding window approach was employed on the data. The unit considered is a measure, so all combinations of two consecutive measures were considered. Sequences were removed from the obtained tracks if they (1) repeat in the dataset, (2) contain only silent states

[1] `https://salu133445.github.io/lakh-pianoroll-dataset/`

Model	N. Layers	Hidden Sizes	N. Parameters per layer
SU-MAF	2	$[500, 1000]$	7'540'000
DU-MAF	4	$[50, 100]$	21'801'000
TM-MAF	3	$[(10, 10), (14, 10)]$	20'574'320

Table 1: Best configuration for each variant of MAF. The number of parameters of SU-MAF is lower due to the parameter sharing among notes.

(32 bars containing only zero values), (3) contain a hold state as the first time step of the first measure.

In a MIDI file, the playable notes range from 0 to 127. We also add a new state called "hold state" to represent the situation where a note is played for a period longer than a time step continuously. This distinguishes this case from when the same note is played multiple times in succession but not continuously. For simplicity, only note activation information was considered, disregarding the velocity attribute in MIDI. Also, only MIDI programs with a value of 0, which usually corresponds to the "Acoustic Piano" instrument, were considered. After all the aforementioned transformations were applied, around 300,000 tracks were obtained. For each MAF architecture proposed, we select the best hyper-parameters through a grid search. We validate the number of layers and the value of hidden sizes in MADE in such a way that all the architectures have a comparable number of parameters and we select the best model and we select the best configurations (Table 1) via model selection.

To evaluate the quality of the generations obtained from the models, it was not possible to apply traditional generative model evaluation metrics, such as reconstruction error. In fact, the intrinsic properties of the MAF model always ensure a perfect reconstruction of the input sequence. Therefore, some metrics have been developed to compare them with the melodies from the training set. Since the training set consists of human-composed melodies, this approach allows an indirect comparison with human compositions. However, many music-based metrics require a deep understanding of musical theory, making them complex for those outside the field. To circumvent this issue, we rely on more accessible metrics, such as: number of held notes (hold state), number of silent notes, number of played notes, highest pitch note, lowest pitch note, average pitch of the notes, and simultaneously played notes. Each model generated samples of 1000 melodies to apply the aforementioned metrics.

Results Discussion. By observing Table 2, there is a clear discrepancy between the training set values and those generated by SU-MAF: due to the parameter sharing among notes, the model is not able to generate good samples. Instead, DU-MAF produced results that were much closer to those of the training set. In particular, the number of silent notes decreased significantly compared to the SU-MAF case, while the number of played notes increased. Furthermore, the songs exhibited more polyphony, averaging 16 time steps with simultaneous

notes. Finally, TM-MAF obtains mixed results: while silent time steps were still common, they were fewer than in the SU-MAF case. Consequently, the number of played notes and the polyphonic time steps differed substantially from the training set values. On the other hand, certain features, such as the number of held notes and the highest and lowest pitches, closely aligned with the training set. Although TM-MAF is the most expressive architecture, it struggles to capture all the data characteristics. We believe that this is due to small hidden size choosing to limit the number of parameters.

	Tr set	SU-MAF	DU-MAF	TM-MAF
Held Notes	7.89	0.04	9.19	6.57
Silent Timesteps	11.55	27.59	10.12	20.29
Played Notes	26.16	5.00	26.57	7.51
Highest Pitch	60.97	78.27	71.25	55.89
Lowest Pitch	43.02	47.15	36.65	38.13
Mean Pitch	51.81	63.07	49.85	45.84
Polyphonic	14.25	4.32	17.76	5.99

Table 2: Comparison of Generation Metrics between the training set and 1000 samples generated by: SU-MAF, DU-MAF, and TM-MAF

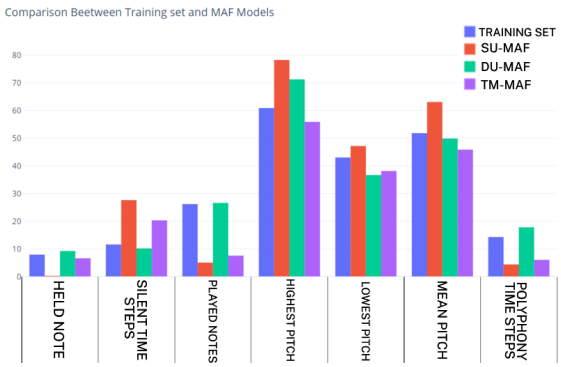

Fig. 1: Histogram representation comparing the Training Set with the generations obtained through the three models. The metrics depicted are those associated with the evaluation of generations and reported in Table 2.

4 Conclusions

The main goal of this work is to explore the application of the MAF architecture in the context of music generation. In this context, the time series are multivariate and thus cannot be handled directly by MAF. We overcome this limitation by proposing three approaches to extend the MAF architecture to multivariate time series. The results obtained on the generation of MIDI songs show that

processing all the notes with the same parameter is a sub-optimal choice. Nevertheless, the most complex approach that considers also the interactions between the nodes struggles to learn the intrinsic characteristics of the data due to the limited hidden size.

Our results encourage us to deepen the application of MAF architectures in the context of multivariate time series. Music is just one possible application of this model. A multivariate model of this kind could be valuable for various types of complex data, particularly sequential data, as explored in this study. More broadly, it could be applied to high-dimensional datasets where capturing intrinsic features and their interactions is essential. The approach used can certainly be improved. In particular, it would be interesting to employ tensor decomposition to overcome the limitation of TM-MAF; thanks to tensors decompositions, we would be able to model interaction among notes without limiting the hidden size to reduce the number of parameters. This approach has been already applied with success on various domains, from structured data [6, 7, 8] to images [9]. On the other hand, to further develop the music use case, the work could be extended by incorporating additional musical properties, such as velocity, and adopting more advanced music-specific metrics, that would enable a comparison with commonly used approaches in the literature.

References

[1] Andrea Valenti, Antonio Carta, and Davide Bacciu. Learning style-aware symbolic music representations by adversarial autoencoders, 2020.

[2] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In *International conference on machine learning*, pages 4364–4373. PMLR, 2018.

[3] Gino Brunner, Andres Konrad, Yuyi Wang, and Roger Wattenhofer. Midi-vae: Modeling dynamics and instrumentation of music with applications to style transfer. In *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR 2018)*, pages 747–754. dblp, 2018.

[4] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

[5] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pages 881–889. PMLR, 2015.

[6] Daniele Castellana and Davide Bacciu. A tensor framework for learning in structured domains. *Neurocomputing*, 2021.

[7] Daniele Castellana and Davide Bacciu. Learning from non-binary constituency trees via tensor decomposition. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3899–3910. International Committee on Computational Linguistics, dec 2020.

[8] Chenqing Hua, Guillaume Rabusseau, and Jian Tang. High-order pooling for graph neural networks with tensor decomposition. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.

[9] Weihong "Grace" Guo Yinan Wang and Xiaowei Yue. Tensor decomposition to compress convolutional layers in deep learning. *IISE Transactions*, 54(5):481–495, 2022.