Audio Track Accompaniment

Marion Lepert and Jérôme Nowak

***Abstract*—This paper investigates three methods of generating a musical accompaniment track from three input tracks. Our approach is limited to discrete notes using the MIDI (Musical Instrument Digital Interface) standard. We formulate generating an accompaniment track as a classification problem, where the algorithm predicts a note from one of 128 pitch classes at each time interval. We compare the performance of a classical method, Support Vector Machines, with a modern Convolutional Neural Network, and find that both have low performance. We attribute one source of our low success to a class imbalance problem in our dataset and propose several methods of addressing this challenge. The code for our project can be found at the following git repository: https://github.com/MarionLepert/cs229**

I. Introduction

Music composition, improvisation and accompaniment are creative acts which are typically challenging to capture algorithmically. However human creativity may be emulated to some extent given enough examples and a function fitting or a generative algorithm. Following this principle, we wish to apply the methods studied in class and compare them with modern deep learning methods to train a neural network on generating an accompanying audio track for a given set of input tracks from three different instruments. The scope of our study is limited to MIDI files containing at least piano, guitar, bass, and strings. Our input is a MIDI file with piano, guitar, and strings tracks, and our output is the MIDI file for a bass track.

II. Related Work

Artificial music accompaniment has been studied for several decades. Starting in the 20th century, online algorithms such as [5], [15], were developed to do real-time accompaniment. A computer was given an existing piece of music and was tasked with synchronizing it to a live performer. However, these methods relied on existing audio tracks, and we are interested in generating notes from scratch. Simon and Morris present a stepping stone towards this in [16] using a Hidden Markov Model (HMM) to automatically choose chords to accompany a vocal melody.

More recently, algorithms have been developed that do not rely on an existing audio track, but rather generate new music themselves. Dong et al. [6] use generative adversarial networks (GANs) to both generate music from scratch and generate accompanying tracks for an existing track. However, as noted by the authors, their results are "musically and aesthetically behind the level of human musicians." Zhu et al. [17] present an end-to-end multi-track song generation model that relies on Attention Cells to share information between tracks and ensure a harmonious arrangement. They show improvements compared to existing models but still do not achieve comparable performance to human composed music.

A broader class of research has investigated music generation without focusing on accompaniment. Commonly used algorithms for this type of work include Recurrent Neural Networks (RNNs), such as in [3], [8] and Long Short Term Memory Networks (LSTMs), such as in [2]. RNNs rely on inputs at a current time step and inputs in the hidden layer of the previous time step to make predictions, allowing the temporal relation between samples to be learned by the model. LSTMs are a modification of the RNN architecture that use "gates" to allow gradients to flow through nodes unchanged, allowing information to flow across many time steps without vanishing or exploding [12]. As a result, methods that use LSTMs tend to have better global music structure. Additional approaches include interactive evolutionary computation (IEC), a less data intensive method which involves a human collaborating with the computer to explore a space of candidate accompaniments, such as Hoover and Szerlip in [7]. This method identifies functional relations between patterns of notes and is reported to produce melodies indistinguishable from fully-human composition for the untrained ear, though it is not applicable to melodies whose relationships are more complex.

III. Dataset

Our training dataset is taken from the Lakh Dataset [13], which has over 170,000 MIDI files. We select a subset of those that have guitar, piano, strings, and bass tracks. We model our problem as a classification problem. We define 128 different possible classes, where each class corresponds to a unique music note pitch. We choose the number 128 because MIDI files have 128 different possible music note pitches. We split each song into consecutive segments of duration T = 10ms. A training example consists of one segment and has 128 notes $\times$ 3 instruments $= 384$ features. Each feature corresponds to a music note pitch associated with an instrument. The feature is either a 0 or a 1 depending on whether the instrument is playing that note during a segment. More specifically, if a note starts playing in the middle of a segment, we model it as playing during the entire segment. Conversely, if a note ends during a segment, we model it as not playing at all during that segment. A visualization of a sample from our data with this time discretization is shown in Figure 1.

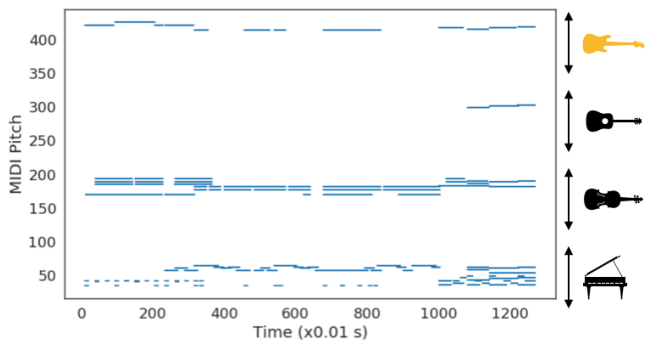

Fig. 1. Visualization of a MIDI file with four instruments where, for plotting purposes, pitch levels 0-127 correspond to the first instrument, piano, while pitch levels 128-256 correspond to strings, pitch levels 257-384 correspond to acoustic guitar, and pitch levels 385-512 correspond to bass.

We split our data using an 80% train, 10% val, and 10% test split and use 50,000 training examples.

IV. Methods

A. Baselines

For our first baseline, we used the Support Vector Machines (SVM) algorithm to predict what note the bass instrument should play in each time interval. We chose SVM because according to Ng [9], it is a classifier that is known to be one of the best "off-the-shelf" self-supervised learning algorithms. This classifier finds the hyperplanes that separate the data into different classes. More specifically, the classifier solves an optimization problem, shown in equation (1), to find the hyperplanes with the maximum geometric margins. We add a regularization term to this optimization problem to allow for non-linearly separable datasets and reduce the algorithm's sensitivity to outliers.

$$\begin{aligned} &\min_{\gamma,w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\zeta_i \\ &\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \zeta_i, \quad i = 1, ..., n \\ &\qquad \zeta_i \geq 0, \ i = 1, ..., n \end{aligned} \tag{1}$$

To solve this problem more efficiently, we construct the dual of the optimization problem and use the kernel trick to calculate only the inner products between input features, as shown in equation (2). We experiment with a linear and a Gaussian kernel.

$$\begin{aligned} &\max_{\alpha} W(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle \\ &\text{s.t. } 0 \leq \alpha_i \leq C, \ i = 1, ..., n \\ &\qquad \sum_{i=1}^{n}\alpha_i y^{(i)} = 0 \end{aligned} \tag{2}$$

To apply the SVM algorithm to our data, we assigned a unique label to each training example, consisting of the note the bass instrument is playing during a given time segment. To simplify our approach, we assumed that the bass instrument can only play a single note during each segment. Therefore, if the bass instrument is playing several notes during a segment, we randomly select a single note as the label. We tested both a linear and a Gaussian kernel.

B. Convolutional Neural Networks

We compare the performance of our baseline algorithms to convolutional neural networks (CNNs). We chose this method because CNNs have shown impressive performance in the field of computer vision because of their ability to find translation-invariant patterns, and we hypothesized that by treating our MIDI files as "images" we could achieve better classification performance compared to classical methods. Neural networks learn a model by taking the gradient of a loss function from the output of fully-connected layers and updating the weights and biases of these layers with the appropriate gradients. CNNs build off of neural networks by adding convolutional layers between the inputs of the model and the fully-connected layers. These layers act as filters that are learned by the network and allows the network to find patterns in the input data [4].

To apply CNNs to our problem, we consider our MIDI data to be an image where a pixel corresponds to a note played by an instrument which can take on the value of 0 or 1. We perform a 1D convolution over our "image" along the time dimension as shown in Figure 2.

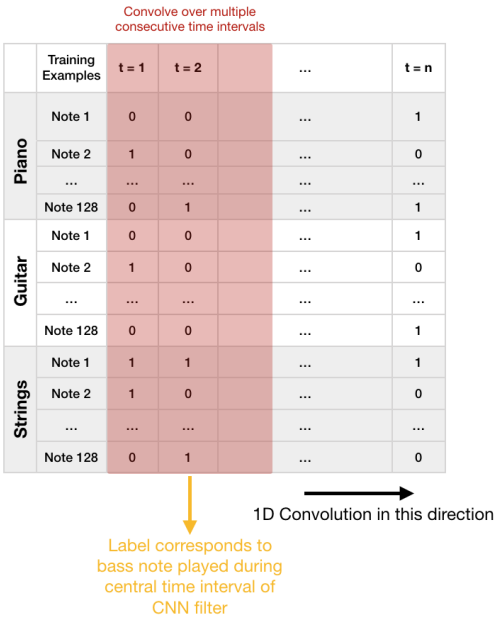

Fig. 2. A visual representation of how 1D convolution is applied to our dataset along the temporal axis.

Our CNN architecture is shown in Figure 3. We used two 1D Convolutional layers followed by three fully connected layers. The size of our convolution kernel along the temporal direction

was 5. We added the ReLU activation function between these layers. Our first convolutional layer had 384 input features, while our second one had 64 input features. Our first fully connected layer had 672 features, our second one had 400, and our final layer had 250 input features. The final output is a vector with 128 components, corresponding to the 128 different notes the fourth instrument can play.

We experimented with two different loss functions. The first was a Binary Cross Entropy Loss Function. This function applies the Cross Entropy Loss function individually to each component of the output vector. In this case, each element of the output vector of the network corresponds to the probability that the note is being played (1) or not played (0). Therefore, with this loss function, our network is solving 128 binary classification problems in parallel.

We also tried to formulate our problem as a multi-class classification problem by using a Cross Entropy Loss Function over the entire output vector with 128 components. In this case, each element of the output vector represents a probability of how likely a note should be playing with respect to all the other notes.

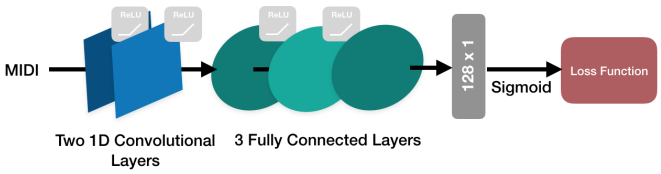

Fig. 3. The architecture of our CNN

C. Evaluation Metrics

We evaluate the performance of our algorithms with a variety of metrics. The first is accuracy. We aim for high accuracy but recognize that perfect accuracy is not desirable, as we want our algorithm to generate new music, not copy music.

$$\text{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{All examples}}$$

In addition, we look at precision and recall, defined below. Intuitively, precision describes how likely an element that is labeled as positive is actually positive and recall describes how many of the actual positives were correctly identified. [1].

$$\text{Precision} = \frac{\text{True Positives}}{\text{True positives + False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True positives + False Negatives}}$$

Lastly, we consider the False Positive Rate (FPR) which describes how many false positives are in the actual negatives set.

$$\text{False Positive Rate (FPR)} = \frac{\text{False positives}}{\text{False Positives + True Negatives}}$$

V. Results

For our first experiment, we ran SVM on a subset of the Lakh Dataset. We tested a linear kernel and a Gaussian Kernel. A visualization of the SVM classifier's predictions on the test set with a Gaussian kernel are shown in Figure 4.

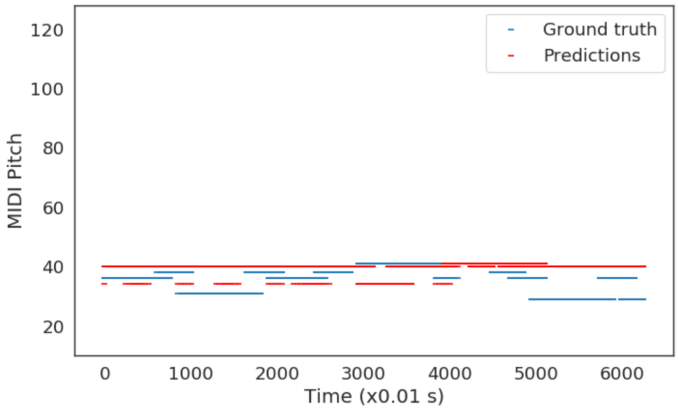

Fig. 4. Visualization of SVM predictions with a Gaussian kernel vs the ground truth on the test set. Note that the algorithm only predicts a single note for a given time interval. The apparent overlap in notes in this plot is due to the size of the markers and the resolution of the time axis.

The accuracy, precision, and recall metrics for our SVM algorithm are shown in Table V. Our accuracy is quite low at 11% for the linear kernel and 10% for the Gaussian kernel. The average precision and recall metrics for these kernels is also quite low, although we note that it is higher for some notes, such as note 41 with a precision of 0.32 and recall of 0.54. This indicates that our algorithm is rarely able to predict a note that matches ground truth. We found that the linear and Gaussian kernels had similar accuracy and recall and the Gaussian kernel had slightly higher precision than the linear kernel. This was surprising as we expected substantially better performance from the Gaussian kernel given that our data is not linearly separable.

Table 1. Evaluation metrics for our SVM algorithm using a linear and Gaussian kernel

	Accuracy	Weighted Avg Precision	Weighted Avg Recall
Linear Kernel	0.11	0.07	0.11
Gaussian Kernel	0.10	0.11	0.10

Next, we analyze the results from our CNN. We used stochastic gradient descent and selected a mini-batch size of 256 because it was the highest power of two that did not result in a memory allocation error. We wanted to maximize our mini-batch size to reduce the variance in the gradient updates, since the gradient is averaged over the entire mini-batch. Given this mini-batch size, we used a learning rate of 0.1 because it allowed us to decrease our training loss reasonably fast while still being able to converge, which was not always the case with a higher learning rate like 1.0.

Figures 5, 6, and 7 present results for a CNN with a Binary Cross Entropy Loss. Although our loss on the training and test set converges, we obtain very low performance. Our precision recall curve shown in Figure 6 demonstrates that we have very low precision, meaning that we almost never predict that a note is playing when it is actually playing.

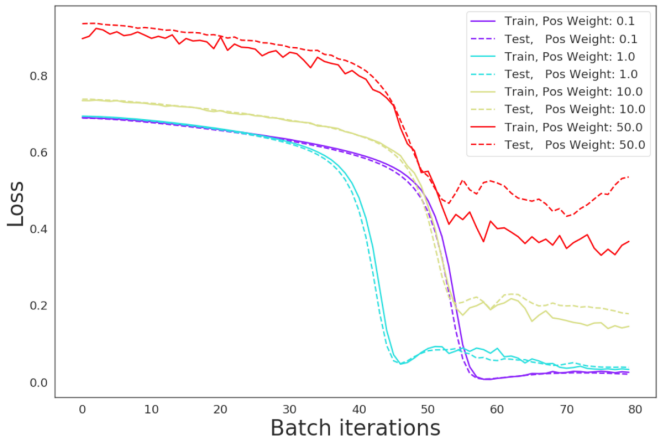

Fig. 5. Loss vs iterations for the training and test sets for different up-weighting values when using a CNN with Binary Cross Entropy Loss

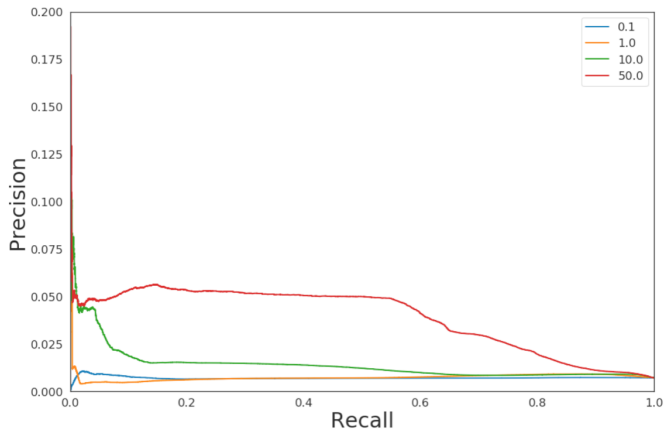

Fig. 6. Precision recall curves for different up-weighting values when using a CNN with a Binary Cross Entropy Loss

Our ROC Curve, shown in Figure 7, indicates that if we want to predict every note that should be playing, we will also falsely play about 40% of notes that should not be playing.

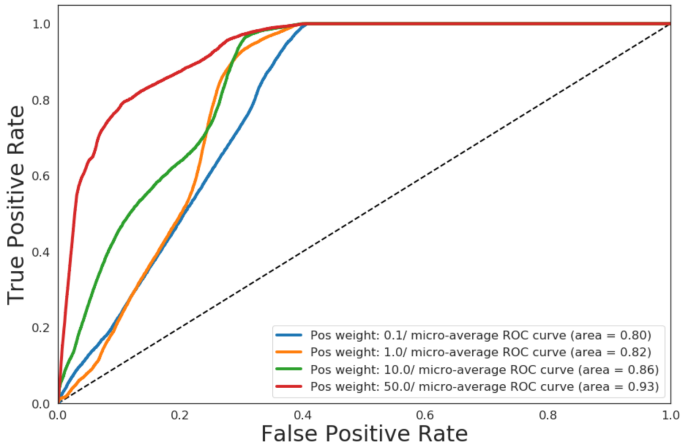

Fig. 7. ROC curve for different up-weighting values when using a CNN with Binary Cross Entropy Loss

We believe our low performance is because we have a large class imbalance problem. This imbalance stems in part from the fact that we are doing binary classification on each note, and each note spends a much longer period of time not being played than being played. Another way of looking at this problem is recognizing that the model achieves 99.2% accuracy when the model always outputs zeros (predicting that no note is ever being played). We tried to address this imbalance by up-weighing positive samples in our loss function. However, this resulted in minimal improvements, as shown in Figures 6 and 7.

To reduce the class imbalance problem, we reformulate our problem as a multi-class classification problem where our loss function is the Cross Entropy Loss function. Because the bass instrument is often playing a note, as shown in Figure 1, the model will not be as bias towards predicting that no notes should be played. Indeed, as shown in Table V, if the model always predicts that no note is being played, then the model only achieves 20% accuracy, a significant reduction from the model with Binary Cross Entropy Loss. Conversely, when the algorithm predicts a note according to the trained model, it achieves 26% accuracy. We also notice that our trained model has five times higher precision than a model that always predicts zeros. However, the performance of our model with CE loss is still too low to generate a good accompaniment track, but it achieves more than twice the accuracy achieved with SVM.

	Model always predicts no note is being played	Trained model with CE loss
Accuracy	0.20	0.26
Precision	0.04	0.20
Recall	0.26	0.26

VI. Conclusion

Our project was a first attempt at generating an accompanying musical track for a bass instrument given the music score of three other input instruments, and we formulated

this problem as a classification problem. While a CNN with a Cross Entropy Loss had slightly better performance than the classic SVM method, both methods had overall limited success.

One of the limitations of our SVM implementation is that it does not consider how notes played by the three input instruments before and after a given time interval influence what note the fourth instrument should play during that time interval. We hypothesize that including a larger time window of notes played by the first three instruments to predict the fourth instrument's note would increase the algorithm's performance.

For our convolutional neural networks, there are several parameters that we did not tune that could have increased performance. These include changing the length of our time discretization interval T, the size of the kernel, the stride applied by the convolution, the number of layers, and the number of input features for each layer. Although we could improve our existing model by tuning parameters, implementing a different neural network architecture used in the literature, such as an RNN or LSTM, will most likely lead to better performance.

In addition to our model, the class imbalance problem in our data also severely limited the performance of our algorithms. Because some of the instruments we looked at only played a few notes per song, our data had a disproportionate number of time intervals in which an instrument was not playing music. Other researchers using the Lakh dataset have reported needing to merge tracks of similar instruments to deal with this class imbalance problem [6].

Finally, we need to improve our evaluation metrics. Because we are interested in generating new music, not copying music, accuracy on an unseen piece of music should not be our sole evaluation metric. One way to better evaluate our results would be to synthesize the bass MIDI into bass sounds, and use a Turing test to evaluate the quality of the generated music. For this test, we would ask human subjects to label original vs machine generated bass tracks to determine whether or not our algorithms' music can be distinguished from original tracks. Listening to the generated music could also help us understand what quantitative metrics are important for the music to sound good.

Although we did not achieve the results we hoped for, this project was an opportunity to better understand the shortcomings of some classification algorithms on music accompaniment, importance of data engineering, and the difficulties in manipulating large datasets to train neural networks.

VII. Contribution

Team members contributed equally to this project.

References

[1] Precision-recall. URL https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html.

[2] Gino Brunner, Yuyi Wang, Roger Wattenhofer, and Jonas Wiesendanger. Jambot: Music theory aware chord based generation of polyphonic music with lstms. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 519–526. IEEE, 2017.

[3] Hang Chu, Raquel Urtasun, and Sanja Fidler. Song from pi: A musically plausible network for pop music generation. *arXiv preprint arXiv:1611.03477*, 2016.

[4] CS231N. Cs231n lecture notes on convolutional neural networks. URL http://cs231n.github.io/convolutional-networks/.

[5] Roger B Dannenberg. An on-line algorithm for real-time accompaniment. In *ICMC*, volume 84, pages 193–198, 1984.

[6] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[7] Amy K Hoover, Paul A Szerlip, and Kenneth O Stanley. Generating musical accompaniment through functional scaffolding. In *Proceedings of the Eighth Sound and Music Computing Conference (SMC 2011)*, 2011.

[8] MICHAEL C. MOZER. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3):247–280, 1994. doi: 10.1080/09540099408915726. URL https://doi.org/10.1080/09540099408915726.

[9] Ma Tengyu Ng, Andrew. Cs 229 lecture notes part v, October 2019.

[10] 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. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-li pdf.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S. Chang, and T. Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, May 2019. ISSN 1941-0484. doi: 10.1109/JSTSP.2019.2908700.

[13] Colin Raffel. *Learning-based methods for comparing*

sequences, with applications to audio-to-midi alignment and matching. PhD thesis, Columbia University, 2016.

[14] Colin Raffel and Daniel PW Ellis. Intuitive analysis, creation and manipulation of midi data with pretty_midi. In *15th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, pages 84–93, 2014.

[15] Christopher Raphael. A probabilistic expert system for automatic musical accompaniment. *Journal of Computational and Graphical Statistics*, 10(3):487–512, 2001. doi: 10.1198/106186001317115081. URL https://doi.org/10.1198/106186001317115081.

[16] Ian Simon, Dan Morris, and Sumit Basu. Mysong: automatic accompaniment generation for vocal melodies. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 725–734. ACM, 2008.

[17] Hongyuan Zhu, Qi Liu, Nicholas Jing Yuan, Chuan Qin, Jiawei Li, Kun Zhang, Guang Zhou, Furu Wei, Yuanchun Xu, and Enhong Chen. Xiaoice band: A melody and arrangement generation framework for pop music. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 2837–2846, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3220105. URL http://doi.acm.org.stanford.idm.oclc.org/10.1145/3219819.3220105.