LEARNING SEMANTIC SIMILARITY IN MUSIC VIA SELF-SUPERVISION

Mason Bretan
Samsung Research America
mason.bretan@samsung.com

Larry Heck
Samsung Research America
larry.h@samsung.com

ABSTRACT

Neural networks have been used to learn a latent "musical space" or "embedding" to encode meaningful features and provide a method of measuring semantic similarity between two musical passages. An ideal embedding is one that both captures features useful for downstream tasks and conforms to a distribution suitable for sampling and meaningful interpolation. We present two new methods for learning musical embeddings that leverage context while simultaneously imposing a shape on the feature space distribution via backpropagation using an adversarial component. We focus on the symbolic domain and target short polyphonic musical units consisting of 40 note sequences. The goal is to project these units into a continuous low dimensional space that has semantic relevance. We evaluate relevance based on the learned features' abilities to complete various musical tasks and show improvement over baseline models including variational autoencoders, adversarial autoencoders, and deep structured semantic models. We use a dataset consisting of classical piano and demonstrate the robustness of our methods across multiple input representations.

1. INTRODUCTION

Music is inherently complex. A single motif can be described along a multitude of dimensions. Some of these dimensions may describe the motif in broad terms and capture properties that offer a more aggregate representation including tonality, note density, complexity, and instrumentation. Others may consider the sequential nature and temporal facets of music such as syncopation, harmonic progression, pitch contour, and repetition. While these features may describe specific attributes about the music, they are intrinsically related and when combined can be used to predict or classify higher level musical descriptors such as genre, style, or even mood and emotion.

In recent years, neural networks have been used to learn a low dimensional latent "musical space" or "embedding" to encapsulate such features and provide a method of measuring semantic similarity between two musical passages [26]. Ideally, the embedding $f(x) \in \mathbb{R}^d$ for a passage x is learned such that the Euclidean distance $D_{i,j} = ||f(x_i) - f(x_j)||^2$ between two passages describes their semantic relationship. For the task of autonomous music generation learning this space effectively is important in order to influence the generator such that its outputs conform to human expectations. This is particularly true in interactive applications where a machine's response is typically conditioned on a human performer. Thus, an effective embedding is one that is capable of interpreting music in a manner which correlates with human perception.

Previously, musical embeddings have been learned using restricted boltzman machines (RBMs) [9, 22, 27], autoencoders (with various denoising techniques) [2, 3], siamese network models [4,13,23], word2vec models [11], and sequence prediction models [1, 6, 18]. Variational autoencoders (VAEs) have also demonstrated some success with monophonic inputs [25]. VAEs learn a normally distributed latent space which has shown to make sampling and embedding manipulation more effective [15]. Though VAEs are useful for constraining the statistical properties of the learned space, it has also been shown that, like word embeddings in language, improved features can be learned when networks are trained to reconstruct the context. The resulting features perform well on prediction and composer classification tasks [2].

An ideal embedding is one that both captures useful features and conforms to a distribution suitable for sampling and meaningful interpolation. In this work, we present two methods for learning musical embeddings. The methods leverage context and simultaneously impose a shape on the feature space distribution via backpropagation using an adversarial component. We focus on the symbolic domain and target short musical units such as a three second clip or a sequence consisting of a small of number notes. The goal is to project these units into a continuous low dimensional space that has semantic relevance. We evaluate relevance based on the learned features' abilities to complete several music-related tasks. We use a dataset consisting of classical piano and demonstrate that it is possible impose a prior distribution on the embeddings while maintaining the quality of the features.

2. RELATED WORK

One possible scenario for learning a space which encodes semantic similarity is to explicitly label pairs of musical units as being similar and train a model in a supervised fashion, thus, pulling units labeled as similar closer together and pushing non-related units further apart in the latent space. This type of metric learning can be effective, however, the requirement for constructing such a dataset makes it challenging. In this work we focus on self-supervised methods that don't require explicitly labeled data.

RBMs, autoencoders, and prediction are all examples of self-supervised learning paradigms. They rely on what is readily available in the data to serve as a proxy to human labeled data, thus, autonomously constructing the supervising signal. Many of the methods have been inspired from the natural language processing community including skip-gram and language models [20]. For example, both [11] and [18] used skip-gram inspired techniques to embed chords. The learned tonal space had similarities to the circle of fifths. Context is also leveraged in [4] where a Deep Structured Semantic Model (DSSM) is used learn one, two, and four bar embeddings [12]. Such siamese network techniques have proven useful for learning semantic similarity in language. In [21] labeled pairs were used to train siamese networks to effectively learn sentence similarity. A pairwise ranking loss was similarly used for the task of hit song prediction [30].

RBM and autoencoder methods don't leverage context, but have still demonstrated the ability to learn good features compared to manually designed features. This is particularly true in the audio domain [9, 16, 29]. In [7] autoencoders were used to learn a latent space encoding timbre. Autoencoders are particularly useful because they inherently support contain a generative component and if learned effectively the embeddings can be manually manipulated for interactive applications [3]. VAEs, in particular, have shown promise and utility for music because the latent space is regularized in a manner that makes sampling and manipulation more convenient and meaningful [15, 24, 25].

The methods in this work are inspired by adversarial autoencoders [19]. Like VAEs, the goal of this type of autoencoder is to constrain the latent codes to some arbitrary prior distribution. However, instead of using a KL-divergence penalty, the autoencoder incorporates an adversarial method to train the distribution of the latent codes to match that of the prior distribution.

While both VAEs and adversarial autoencoders are useful for generation, it has been shown that autoencoder features are not as effective for downstream tasks compared to methods that include context or prediction [2]. In this work we propose a solution that computes a pairwise loss based on context and includes an adversarial component to regularize the latent space.

3. METHODS

We present two methods. At a high level the embeddings of semantically similar units should be geometrically closer in the latent space than units that are dissimilar. For each method the objective is to learn a space that achieves this by leveraging context while adhering to a predefined distribution. In lieu of explicitly labeled data we train the networks using the assumption that two adjacent units (i.e. two adjacent measures in a composition) are related. In other words the distance between two adjacent units should be smaller than two random units in the database.

3.1 Adversarial DSSM

Our first method is a modified implementation of the DSSM. If $q(z)$ represents the aggregated posterior distribution of all the embeddings of length d generated by the DSSM $f(x)$ for $x \in X$ then the goal here to is match $q(z)$ to a prior distribution $p(z)$ we define as $z_i \sim \mathcal{N}_d(\mu, \sigma^2)$ where $\mu = 0$ and $\sigma^2 = 1$. This is achieved by connecting a discriminator to the last layer of the DSSM as shown in Figure 3. This discriminator is trained adversarially in coalescence with the generator which also happens to be the DSSM itself.

The standard DSSM training procedure is well-suited for metric learning as it explicitly trains parameters to produce embeddings that are closer together (according to a distance metric) for related items while pushing non-related items away. However, the number of negative examples and the ratio of easy to hard examples is usually greatly biased towards the easy-end. This often produces poor performance since many examples can satisfy the constraint with a very small loss that provides no real meaningful update during backpropagation [5]. This typically leads to high inter-class and low intra-class variance making fine grained categorization or meaningful similarity measures (important for music) challenging or impossible.

To address this problem a bootstrapping method was used in [5] in which particularly difficult examples were manually mined from the dataset and used during different stages of training. In this work, the adversarial component naturally helps to mitigate this problem by enforcing the prior distribution. The network parameters must find a way to achieve the desired similarity metric, but adhere to a distribution that does not allow for a learned space in which most examples can easily satisfy the similarity constraint.

The adversarial DSSM is trained in two stages: 1) Using the standard DSSM technique compute a softmax loss with negative examples and 2) Using the adversarial network train the generator and discriminator so that the generator is trained to produce embeddings that look as if they have been sampled from the predefined distribution $p(z)$. Thus, the parameters are being optimized according to two different losses with one learning the similarity metric and the other learning to describe the data such that the aggregated posterior distribution of the embeddings are Gaussian and continuous.

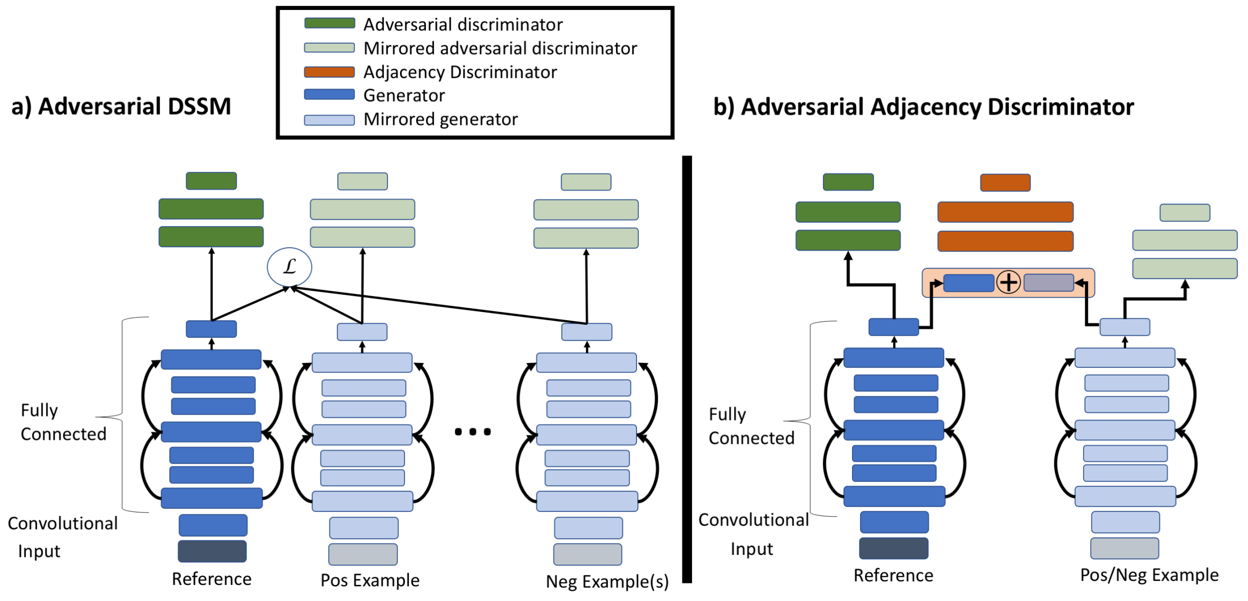

Figure 1. Architectures for a) Adversarial DSSM and b) Adversarial Adjacency Model

For the first part the network is trained using Euclidean similarity.

$$sim(\tilde{X},\tilde{Y}) = \frac{1}{1 + D(\tilde{X},\tilde{Y})} \tag{1}$$

Negative examples are included in a softmax function to compute $P(\tilde{R}|\tilde{Q})$ where $\vec{R}$ is the reconstructed vector and $\vec{Q}$ is the input vector.

$$P(\tilde{R}|\tilde{Q}) = \frac{\exp(sim(\tilde{Q},\tilde{R}))}{\sum_{\tilde{d}\epsilon D}\exp(sim(\tilde{Q},\tilde{d}))} \tag{2}$$

The network learns the parameters by minimizing the following loss function using gradient descent:

$$\mathcal{L} = -log \prod_{(Q,R)} P(\tilde{R}|\tilde{Q}). \tag{3}$$

For the second part generative adversarial network (GAN) training procedures are used [8]. First, the adversarial discriminator is trained to distinguish between the generated embeddings and vectors sampled from $q(z)$. Second, the generator (also the DSSM or $f(x)$) is trained to fool the discriminator. We use a deterministic version of the GAN where stochasticity comes solely from the data distribution. In other words no additional randomness is incorporated.

Training alternates between the DSSM and GAN procedures until Eqn. 3 converges. We found that a higher learning rate for the GAN procedures (particularly for updating the generator) relative to the DSSM loss was necessary in order to get the desired results. Otherwise, the GAN based updates had very little to no effect resulting in a model with a very similar behavior to the vanilla DSSM without the adversarial component.

3.2 Adversarial Adjacency Model

The second method we propose is also inspired by siamese network paradigms. However, unlike the DSSM, the embeddings are not directly optimized for the desired metric. Instead, a classifier is trained to determine whether two units are related or not. Though, because we use adjacency as the self-supervising surrogate signal in lieu of manually designed similarity labels the classifier is really being trained to determine whether two units would be contiguous or not in a composition.

A simple version of this classifier would concatenate *both* units into a single input and be trained to produce a binary classification from this concatenated vector. Our goal, however, is to be able to embed a *single* unit and having a network which requires two units as input would prevent this. Therefore, we use tied weights in which the lower layers of the network are identical and the embeddings are not concatenated until several layers deep into the network (see Figure 3). In other words the two inputs are embedded independently, but use the same parameters to do so.

A much smaller classifier is attached to the top of the concatenated embeddings to discriminate between related and non-related inputs. By using only a couple layers to perform the actual discrimination most of the good features for classification will need to be learned by the embedding portion. This enforces the network to embed an input in a manner that not only efficiently encodes itself, but also can effectively distinguish itself from unrelated inputs. Our thinking was that hopefully the network would achieve this by embedding related units closer together. (Note, for ease of comparison the architecture of the embedding network here is identical to the DSSM network in the previous section).

Like the previous method we attach an adversarial component to the end of the embedding portion of the network. The goal is the same in that the aggregated posterior distribution of the embeddings should conform to a predefined distribution. The model is trained in two stages: 1) Train the classifier to discriminate between related and non-related inputs and 2) Train the embeddings to fit a prior distribution using the GAN scheme. Therefore, the embedding portion of the model is updated during both stages.

For the first part the classifier is trained using cross entropy with two classes (related and non-related).

$$-\sum_{c=1}^{M} y'_c \log(y_c) \tag{4}$$

where $M = 2$, y' is the predicted probability and y is the ground truth. The GAN portion is trained similarly to the previous method. We found that this method was inherently more stable than the previous method and much less tuning of the learning rates between the two losses were necessary.

4. EXPERIMENTS

In order to test the various networks and training procedures we used a collection of piano compositions from 27 artists. The distribution of compositions among artists is depicted in Table 4. At least two songs from each artist were held out for testing. We augmented the data by transposing each piece into all keys. This also prevented the networks from simply learning the bias any composers might have had for specific key signatures. We also augmented the data by altering the tempo randomly within a range of .95 to 1.05 of the original.

Composer	Num. Train Songs	Num. Test Songs
Albeniz	15	2
J.S. Bach	8	2
Bartok	21	4
Beethoven	30	4
Borodin	8	2
Brahms	31	5
Burgmueller	10	2
Byrd	34	4
Chopin	49	6
Clementi	17	2
Couperin	10	2
Debussy	10	2
Galuppi	6	2
Grieg	17	2
Handel	20	3
Haydn	20	3
Scott Joplin	57	4
Liszt	17	2
Mendelssohn	16	2
Mozart	22	3
Mussorgsky	9	2
Rachmaninov	10	2
Ravel	5	2
Scarlatti	6	2
Schubert	30	4
Schumann	25	3
Tschaikovsky	13	2

Table 1. Piano Music Dataset.

4.1 Baseline Models

We compare our methods against four baseline models for a total of six methods:

1. Variational autoencoder (VAE)
2. Adversarial autoencoder (AAE)
3. Deep structured semantic model (DSSM)
4. Adjacency Discriminator (AdjD)
5. Adversarial deep structured semantic model (A-DSSM)
6. Adversarial Adjacency Discriminator (A-AdjD)

4.2 Input Representation

Often performance of a music-based model is ultimately determined by the input representation of the data. Therefore, we test our system using two different input representations.

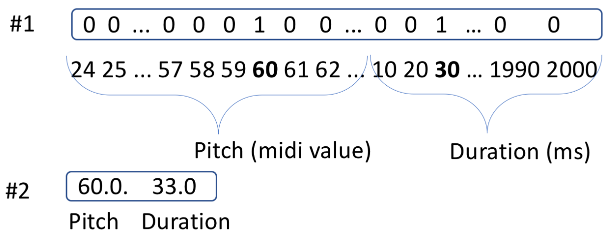

Figure 2. Two representations for encoding middle 'C' (midi note 60) and a 33ms interval to the next pitch. Input representation #1 uses a two-hot encoding that discretizes time in 10ms chunks. Input representation #2 uses a single floating point value for both pitch and time.

The first representation is inspired by Google Magenta's event based method [28] in which each event is one-hot encoded. Because much of the data does not include relevant volume information we do not include it in our representation. We also use only the onset time and use a fixed duration for each note, therefore, it is not necessary to include note off events. This was primarily to simplify the input space and attempt to better interpret the results. Pitch is represented in a one-hot manner on a vector representing midi values 24 to 96. We encode intervals between pitch events discretely in 10ms intervals from 0 to 2000ms, thus, the time portion of the vector consists of 200 values. The single vector for all possible events including pitch and interval has a length of 272. A two-hot encoding method is used so both the pitch and interval before the next pitch is encoded using the 272 values (see Figure 4.2).

The second representation is much less conventional. We represent both pitch and time intervals continuously. The pitch of a note is represented by a single floating point value determined by its midi value and the interval between notes is also represented by a single floating point value determined by the number of milliseconds. Therefore a vector for a single event only has a length of two (4.2).

On the surface, this continuous pitch representation does not make a lot of sense as the Euclidean distance in this space does not really translate to pitch distances that are particularly meaningful in Western music or piano. An autoencoder trained to reconstruct this input using a Euclidean-based loss is unlikely to learn many relevant musical features. However, the two methods proposed in this work do not compute the loss in the original space. The adversarial DSSM optimizes the latent space by computing the loss directly on the embeddings and the adversarial adjacency model optimizes the latent space through a classification task using a cross entropy loss. Therefore, we hypothesize that our methods should be more robust against varying input representations compared to models that compute a loss in the original space (e.g. autoencoders). Additionally, a method that is robust when using this representation can be useful for styles of music or particular instruments in which continuous pitch representations are more appropriate.

In this continuous representation the pitches and intervals are standardized to a mean of zero and standard deviation of one. In our experiments we did not find a difference in our final results when compared to using non-standardized input vectors, however, the learning was faster using standardized vectors.

We focus on short musical units consisting of exactly 40 notes. This means that the input vector to the network using the first representation has a length of 10880 (40 * 272). The input vector using the second representation has a length of only 80 (40 pitches and 40 intervals). Forty notes was chosen because it provides enough content to capture local structure, yet, the vector length using the first representation is not overwhelmingly large.

4.3 Architectures

The first layer of the network is convolutional using a filter with an input length equivalent to a vector containing one note and one interval (i.e. 272 for the first representation and 2 for the second). The filter is convolved over the entire vector using an equivalent stride length (272 or 2), thus, the filter learns to encode a single pitch and interval. After this initial convolutional layer all remaining layers are fully connected.

For each method the portion of the network which performs the embedding is the same aside from marginal differences in the number of parameters for the first convolutional layer between the two representations. The network encodes the input into a 32-dimensional vector. Each network has eight layers between the final embedding and input vector and uses residual connections [10] as depicted in Figure 3.

It is plausible that higher capacity networks (both wider and deeper) may improve results further, however, the primary objective in this work is to compare various training methods and not architectures. We designed this architecture because it allowed us to leverage the utility of deep learning and specific techniques (e.g. convolution, residual connections, etc.), yet, it is not too large that training time would become problematic during experimentation. Each layer uses LeakyReLU and the parameters are updated using Adam optimization [14, 17].

4.4 Experiments

We perform five different experiments based on music-related tasks. Performance on these tasks will be used to determine the efficacy of the learned latent space and features for the various models.

Ranking The primary measure for evaluation is based on a ranking task. Given a reference unit and a group of 100 units consisting of 99 random units from the database and one unit that is adjacent to the reference (either before or after) the task is to rank all 100 units according to their Euclidean distance to the reference in the latent space. This is repeated for each unit in the test set and a mean rank is reported where the a lower rank indicates a higher similarity. The assumption is that adjacent units should be geometrically closer in the latent space relative to non-adjacent units.

Composer Classification We evaluate how useful the learned features are for classifying the units according to their composers. Using the embeddings as inputs we train a simple two layer network to perform classification. In the test set there are 27 possible composers. Though works by these composers are seen during the training phase, the test set consists of unique compositions that were not available during training.

Pitch Chroma Prediction We evaluate whether the embedding retains enough information about the input units to reconstruct a unit's chromagram representation. We train a two layer network to minimize a softmax function (Eqn. 3) over cosine similarity. Thus, in Eqn. 2 we replace the Euclidean similarity (Eqn. 1) with $sim(\tilde{X}, \tilde{Y}) = \frac{\tilde{X}^T \cdot \tilde{Y}}{|\tilde{X}||\tilde{Y}|}$. The positive example is the true chroma extracted from the unit computed directly from the original input vector. Negative examples come from chroma extracted from random units in the data.

Note Density Regression We evaluate whether the embedding retains enough information about the input units to describe their note density. Because we use a fixed number of notes per unit the network is trained to predict the unit's duration in seconds. We use root mean square error (RMSE) to measure performance.

Forward Prediction We evaluate whether a sequential model can be trained to predict the embedding of the next unit in a composition given a sequence of the previous seven units. This task is related to the first Ranking task, but focused on generation and prediction rather than general distances in the learned manifold. We train two stacked LSTM cells, each with 200 units, to predict the next step of a sequence. Specifically, at each time step, the input to the network is a the 32-dimensional embedding vector of a 40 note unit x_i. We train this network to predict the 8^{th} unit, x_{i+7} given the previous 7 units $x_i, z_{i+1}, \dots x_{i+6}$. This means that 280 notes of context are provided before the prediction is made.

For each given target x_{i+7} as described above, we create a set of 32 embedding vectors: one is $f(x_{i+7})$, the true embedding for the target. The other 99 vectors are embeddings of randomly selected units in the data set. The Euclidean distance is measured between the output of the

LSTM and the encodings of each unit. The distances are then sorted and ranked similarly to the first Ranking experiment.

5. RESULTS

The results for each experiment are reported in Tables 2-6.

Method	Representation #1	Representation #2
VAE	10.8	12.3
AAE	9.9	12.3
DSSM	11.2	14.8
AdjD	5.8	6.3
A-DSSM	8.8	9.2
A-AdjD	5.1	5.6

Table 2. **Ranking Results.** Geometric means are reported for the adjacency ranking task. A lower score indicates a better result.

Method	Representation #1	Representation #2
VAE	.11	.087
AAE	.11	.091
DSSM	.18	.10
AdjD	.26	.26
A-DSSM	.23	.19
A-AdjD	.27	.28

Table 3. **Composer Classification** Macro-F1 scores are reported for the composer classification task. A higher score indicates a better result.

Method	Representation #1	Representation #2
VAE	.56	.43
AAE	.57	.44
DSSM	.34	.27
AdjD	.71	.68
A-DSSM	.72	.67
A-AdjD	.83	.78

Table 4. **Chroma Predication** Geometric means of cosine similarities between predicted and ground truth chroma. A higher score indicates a better result.

Method	Representation #1	Representation #2
VAE	.33	.28
AAE	.33	.29
DSSM	.20	.18
AdjD	.26	.20
A-DSSM	.21	.19
A-AdjD	.21	.19

Table 5. **Note Density Regression** RMSE values are reported for note density regression. The deviations were measured in seconds. A lower score indicates a better result.

Method	Representation #1	Representation #2
VAE	5.9	8.7
AA	5.9	8.6
DSSM	10.7	11.5
Adj	2.7	3.0
A-DSSM	4.8	5.2
A-Adj	2.6	2.7

Table 6. **Forward Prediction** Geometric means are reported for the adjacency ranking task. A lower score indicates a better result.

5.1 Discussion

The vanilla DSSM performed relatively poorly on all tasks except for note density regression. Without the adversarial component it learns the most significant feature (in this case note density in time), yet, fails to learn much more beyond this. By fitting the latent space to a prior distribution, the adversarial component seems to do what it was designed for – preventing the model from satisfying the similarity constraint without actually learning too many meaningful features.

Our adjacency discriminator model worked reasonably well even without the adversarial training. We found that without the adversarial component the latent embeddings were naturally fairly close to a Gaussian distribution, thus, adding an adversarial discriminator had much less of an effect than when used with the DSSM. Though the additional fine tuning did improve performance across the tasks albeit marginally.

Finally, the differences in performance for the two input representations were much less pronounced for our methods. This suggests our proposed methods (the Adversarial Adjacency Discriminator in particular) may be useful for non-Western music and instruments capable of continuous pitch spaces.

6. CONCLUSION

In this work we described new approaches to self-supervised metric learning. The learned features showed improved results on downstream tasks over various baseline methods. Most importantly, the quality of the features were either improved or maintained when imposing a prior distribution on the embeddings. The next steps for this work are to: 1) develop decoders from the latent spaces learned from our methods and 2) measure the perceptual significance of the learned space.

7. REFERENCES

[1] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[2] Mason Bretan, Sageev Oore, Doug Eck, and Larry Heck. Learning and evaluating musical features with

deep autoencoders. In *KDD Workshop on Machine Learning for Creativity*, 2017.

[3] Mason Bretan, Sageev Oore, Jesse Engel, Douglas Eck, and Larry Heck. Deep music: towards musical dialogue. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[4] Mason Bretan, Gil Weinberg, and Larry Heck. A unit selection methodology for music generation using deep neural networks. In *Proceedings of the 8th International Conference on Computational Creativity, Atlanta*, 2017.

[5] Yin Cui, Feng Zhou, Yuanqing Lin, and Serge Belongie. Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1153–1162, 2016.

[6] Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE workshop on neural networks for signal processing*, pages 747–756. IEEE, 2002.

[7] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1068–1077. JMLR. org, 2017.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[9] Philippe Hamel and Douglas Eck. Learning features from music audio with deep belief networks. In *ISMIR*, volume 10, pages 339–344. Utrecht, The Netherlands, 2010.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Cheng-Zhi Anna Huang, David Duvenaud, and Krzysztof Z Gajos. Chordripple: Recommending chords to help novice composers go beyond the ordinary. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pages 241–250. ACM, 2016.

[12] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338. ACM, 2013.

[13] Yu-Siang Huang, Szu-Yu Chou, and Yi-Hsuan Yang. Similarity embedding network for unsupervised sequential pattern learning by playing music puzzle games. *arXiv preprint arXiv:1709.04384*, 2017.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[16] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

[17] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[18] Sephora Madjiheurem, Lizhen Qu, and Christian Walder. Chord2vec: Learning musical chord embeddings. In *Proceedings of the constructive machine learning workshop at 30th conference on neural information processing systems (NIPS'2016), Barcelona, Spain*, pages 1–5, 2016.

[19] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[21] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[22] Juhan Nam, Jorge Herrera, Malcolm Slaney, Julius O Smith, et al. Learning sparse feature representations for music annotation and retrieval. In *ISMIR*, pages 565–570, 2012.

[23] Xiaoyu Qi, Deshun Yang, and Xiaoou Chen. Triplet convolutional network for music version identification. In *International Conference on Multimedia Modeling*, pages 544–555. Springer, 2018.

[24] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[25] Adam Roberts, Jesse Engel, and Douglas Eck. Hierarchical variational autoencoders for music. In *NIPS Workshop on Machine Learning for Creativity and Design*, 2017.

[26] Fanny Roche, Thomas Hueber, Samuel Limier, and Laurent Girin. Autoencoders for music sound synthesis: a comparison of linear, shallow, deep and variational models. *arXiv preprint arXiv:1806.04096*, 2018.

[27] Jan Schluter and Christian Osendorfer. Music similarity estimation with the mean-covariance restricted boltzmann machine. In *2011 10th International Conference on Machine Learning and Applications and Workshops*, volume 2, pages 118–123. IEEE, 2011.

[28] Ian Simon and Sageev Oore. Performance rnn: Generating music with expressive timing and dynamics. *Magenta Blog: https://magenta. tensorflow. org/performancernn*, 2017.

[29] Jan Wülfing and Martin A Riedmiller. Unsupervised learning of local features for music classification. In *ISMIR*, pages 139–144, 2012.

[30] Lang-Chi Yu, Yi-Hsuan Yang, Yun-Ning Hung, and Yi-An Chen. Hit song prediction for pop music by siamese cnn with ranking loss. *arXiv preprint arXiv:1710.10814*, 2017.