LEARNING SIMILARITY METRICS FOR MELODY RETRIEVAL

Folgert Karsdorp[1] **Peter van Kranenburg**[1,2] **Enrique Manjavacas**[3]

[1] KNAW Meertens Instituut, The Netherlands, [2] Utrecht University, The Netherlands
[3] University of Antwerp, Belgium

folgert.karsdorp@meertens.knaw.nl

ABSTRACT

Similarity measures are indispensable in music information retrieval. In recent years, various proposals have been made for measuring melodic similarity in symbolically encoded scores. Many of these approaches are ultimately based on a dynamic programming approach such as sequence alignment or edit distance, which has various drawbacks. First, the similarity scores are not necessarily metrics and are not directly comparable. Second, the algorithms are mostly first-order and of quadratic time-complexity, and finally, the features and weights need to be defined precisely. We propose an alternative approach which employs deep neural networks for end-to-end similarity metric learning. We contrast and compare different recurrent neural architectures (LSTM and GRU) for representing symbolic melodies as continuous vectors, and demonstrate how duplet and triplet loss functions can be employed to learn compact distributional representations of symbolic music in an induced melody space. This approach is contrasted with an alignment-based approach. We present results for the Meertens Tune Collections, which consists of a large number of vocal and instrumental monophonic pieces from Dutch musical sources, spanning five centuries, and demonstrate the robustness of the learned similarity metrics.

1. INTRODUCTION

The question of how melodic similarity can be computationally modeled is of crucial importance for various Music Information Retrieval (MIR) tasks [35]. One classic MIR scenario is a user posing a sung or hummed query to a retrieval system in order to retrieve resembling pieces of music from a music collection [6, 24]. This query-by-humming scenario requires melodic matching methods that are robust against different kinds of melodic variation arising from imprecise memory or limited singing skills. Melody matching is also an important aspect in cover-song detection, where the predominant melody contains information for song identification [29]. Musicologists benefit from melodic similarity measures for exploring and mapping folk songs [19, 30], or other collections with monophonic musical material, such as themes from classical compositions [18], or score incipits [34]. Finally, music similarity detection plays an important role in cases of copyright violation [31], where objective similarity measures can support the court in making decisions.

In this paper, we present a Neural Network approach for melodic similarity learning. The neural encoders learn complex mappings from input sequences into distributed melody representations. The primary aim of our system is to be employable for music retrieval. In addition to accurately modeling melodic similarity, desirable properties of a retrieval system are speed and indexability. Neural Networks very well accommodate both. Generally, once trained, a neural network can compute results very fast by making use of GPUs. Moreover, indexability is served by the application of similarity metrics on top of the learned encodings [3].

Various other approaches to melodic similarity have been taken [35], including sequence alignment and other dynamic programming approaches, such as edit distance and dynamic time warping, and, recently, Recurrent Neural network representations [4, 9, 11, 16, 27, 30, 33]. In this study, we contrast our approach with a sequence alignment method that has been successfully applied in the context of folk melodies. Alignment-based methods suffer from at least three drawbacks. First, they typically are first-order, taking into account only adjacent items in sequences. Second, they have quadratic time-complexity. Finally, an alignment score is not a proper metric. Our neural network approach overcomes these disadvantages but does so at the cost of being a supervised learning algorithm.

2. DATA AND FEATURES

2.1 Data sets

The Meertens Tune Collections (MTC) [1] contains a series of data sets with melodic material from Dutch sources (mainly manuscripts, printed sources, and audio recordings), spanning five centuries of music history [20, 22]. These data sets are subsets of the Dutch Song Database, maintained by the KNAW Meertens Institute [21]. The lat-

[1] http://www.liederenbank.nl/mtc/

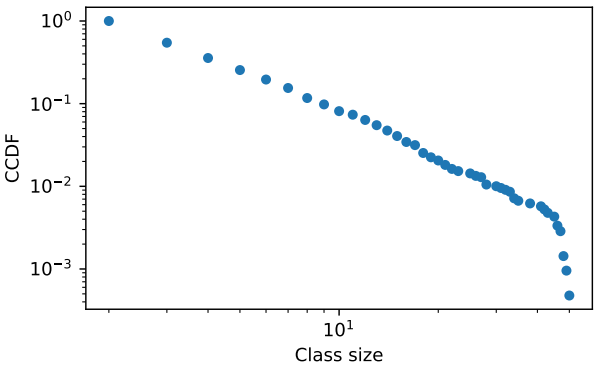

Figure 1. Complementary Cumulative Distribution Function of the tune family sizes in the subset of MTC-FS-INST 2.0 which we use in our experiments.

est release, MTC-FS-INST 2.0, contains 18,109 digitized melodies with rich metadata. Many of these melodies occur in more than one source. Due to oral and semi-oral transmission, these different occurrences typically show melodic variation. As an example, Figure 3 depicts three variant melodies, illustrating the kind and extent of variation. To denote such a group of variant melodies, we adopt the concept of *tune family* from folk song research [1]. In a long-term effort, the collection specialists of the Meertens Institute aim to identify each melody in terms of tune family membership.

In this paper, we use MTC-FS-INST 2.0, which reflects the diversity of the contents of the Dutch Song Database. One main distinction in the data set is between vocal and instrumental music as illustrated in Figure 2. Generally, the instrumental part of the data set dates from the 17th and 18th centuries. It contains melodies that were played in bars and brothels as well as theaters and upper-class private settings. The vocal part of the data set mainly consists of songs from the 19th and 20th centuries. As a whole, the data set provides a rich variety in melodic styles, which renders it a perfect source for training general purpose melodic similarity measures.

To obtain training, development, and test sets, we filter and split the data set. First, we exclude all 5,765 unlabeled melodies and all 3,008 singleton tune families. This leaves a selection of 9,336 melodies in 2,094 tune families. The complementary cumulative distribution function of the class sizes is presented in Figure 1. The distribution of class sizes is heavy-tailed.

An important criterion to measure the level of success achieved by the metric learning approach is its capability to cluster together tune melodies belonging to families unseen during training. In order to make this possible, we perform a controlled test set split, ensuring that all instances from a proportion of tune families do not appear in the training data. The actual proportions of seen and unseen families is shown in Table 1 together with further data set size statistics. [2]

[2] Supplementary material, data sets and code to replicate the experiments are available from `https://github.com/fbkarsdorp/melodic-similarity`.

	# Mel	# TF	# TF in Train	$\mu_{\|TF\|}$	$\sigma_{\|TF\|}$
Train	5,975	1,572		3.80	5.06
Dev	1,492	495	255	3.01	1.64
Test	1,869	611	287	3.06	1.55

Table 1. Composition of the subsets of MTC-FS-INST 2.0 used for training, development and testing. The table provides the number of melodies (Mel) and tune families (TF) in each set, the number of tune families that are shared with the training set, and mean and standard deviation of tune family sizes.

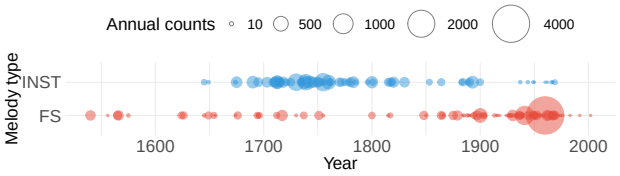

Figure 2. Number of melodies per year in MTC-FS-INST 2.0. The plot displays frequencies for instrumental melodies (INST) and vocal melodies (FS).

2.2 Features

Melodies are represented as sequences of notes, and notes as sets of feature-values. Since the MTC provide a rich melody encoding, including key, meter, and phrase boundaries, we can assemble a diverse feature-set in which various musical parameters are represented: pitch, metric structure, rhythm, tonality, and phrase structure. See the supplementary material for an exact list of features.

3. METHODOLOGY

Our approach is based on two components. First, we deploy distributional melody encoders implemented with Neural Networks. Secondly, we train the encoders with Stochastic Gradient Descent to minimize a Contrastive Loss that we describe below.

3.1 Distributional Encoder

An input melody from the dataset $x_i \in X$ can be represented by a sequence $x_i = [x_{(i,1)}, \ldots, x_{(i,k)}]$ of length $k = |x_i|$, where each $x_{(i,t)}$ is a bundle of m features explained in Section 2.2. For simplicity, we refer to the j^{th} feature of sequence step t as x_t^j, thus dropping data set indices. Our goal is to compute an encoding $h = f(x)$ as a function of the input sequence x, parameterized by a Neural Network $f(x)$.

The encoding process can be described as follows. We first process each time step in the input sequence independently by concatenating all features into a single vector $e_t = [e_t^1; \ldots; e_t^m]$. Categorical features are first encoded into a one-hot vector and projected into their own embedding space with model parameters $W_j \in \mathbf{R}^{JxE}$, where J

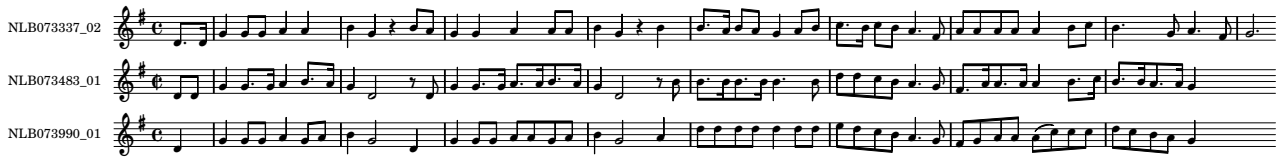

Figure 3. Three members of tune family *Daar was laatstmaal een ruiter 2*, showing various kinds of melodic variation.

is the total number of possible values of the j^{th} categorical feature and E is the dimensionality of the embedding space. Continuous features are normalized to have a mean of 0 and standard deviation of 1. For all feature types, the sequences are padded at the beginning and the end using special symbols in the case of categorical features and the feature mean after re-scaling for continuous features. The resulting sequence of input embeddings is fed to a stack of recurrent layers [3] with the t^{th} hidden activation at layer l given by $h_{(t,l)} = RNN_l(h_{(t,l-1)}, h_{(t-1,l)})$.

We also experiment with bidirectional RNNs, which extend each RNN layer with an additional RNN run backwards. In the case of the bidirectional RNN, the final melody embedding is given by the concatenation of the last activations of the forward and backward RNNs at the last layer: $h = [\overrightarrow{h_{(k,|L|)}}; \overleftarrow{h_{(1,|L|)}}]$. In the case of the unidirectional RNN, the embedding is given by a feature-wise max-pooling operation over the sequence of activations at the last layer, with the p^{th} output feature defined by $h_p = max([h_{(1,|L|)}]_p, \ldots, [h_{(k,|L|)}]_p)$. Instead of traditional RNN cells [7], we use LSTM [14] and GRU [5] cells which have been shown to offer stronger performance and better training behavior.

3.2 Contrastive Loss

The goal of our approach is to learn a distributional encoder such that melodic sequences of the same class are embedded into neighboring regions and far from melodic sequences belonging to different families. To this end, we train the encoder using a contrastive loss [12].

3.2.1 Duplet Loss

Let the encodings of two input sequences with tune family labels y_i and y_j be denoted by x_i and x_j. The goal we want to achieve is that the similarity between x_i and x_j is high when y_i and y_j are equal, and low otherwise. More formally, we seek to achieve the following inequality:

$$D(x_i, x_j) < D(x_i, x_k) + \alpha \tag{1}$$

$\forall (x_i, x_j, x_k) \in X \mid y_i = y_j \wedge y_i \neq y_k$, where D is a distance function and α is a pre-specified margin. In order to achieve this goal, we optimize a contrastive loss function defined over input pairs that is therefore known as the duplet loss. The contrastive loss function is decomposed in a positive term L_+:

$$L_+(x_i, x_j) = (\beta) D(x_i, x_j)^2 \tag{2}$$

[3] During preparatory work, we also experimented with convolutional stacks but found no improvements over the recurrent counter-part, which was, therefore, singled out for the purpose of the present study.

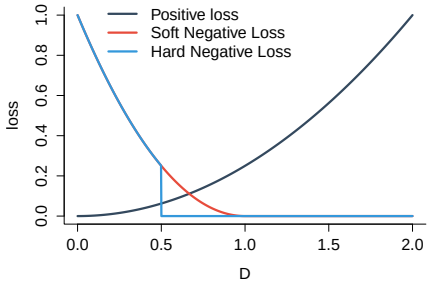

Figure 4. Visualization of the two loss variants with a hard margin and a soft margin.

and a negative term L_-:

$$L_-(x_i, x_j) = \max(0, \alpha - D(x_i, x_j))^2 \tag{3}$$

where β is a parameter used to weight the contribution of the negative term. [4] The two terms can be combined into a single loss function with the help of a variable $Y_{i,j}$ that takes value of 1 for $y_i = y_j$ and 0 when $y_i \neq y_j$:

$$L_D(x_i, x_j) = (Y_{ij}) L_+(x_i, x_j) + (Y_{ij} - 1) L_-(x_i, x_j) \tag{4}$$

For the current study, we restrict ourselves to the cosine distance as defined by Eq. 5:

$$D(x_i, x_j) = 1 - \frac{f(x_i) \cdot f(x_j)}{\|f(x_i)\| \|f(x_j)\|} \tag{5}$$

Naturally, other distance functions are equally applicable, but the two-sided boundedness of the cosine distance (i.e. distances fall between [0, 2]) allows more efficient optimization of the parameters α and β. Moreover, the loss specified above employs a soft margin. By contrast, [28] propose the use of a hard margin, effectively reducing the loss to zero if it falls below some value. With the hard margin, the negative term in Eq. 3 becomes:

$$L_-(x_i, x_j) = \begin{cases} (1 - D(x_i, x_j))^2 & D(x_i, x_j) < \alpha \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Figure 4 visualizes the two loss variants. In the experiments below, we compare both versions of the loss.

3.2.2 Triplet Loss

The triplet loss [13, 32] differs from the duplet loss in that it considers input example triplets x_i, x_j, x_k consisting of a positive example x_i, a negative example x_j such that

[4] The scaling parameter β and margin α are optimized on a development data set, which we will discuss below.

$y_j \neq y_i$ and an anchor x_k such that $y_k = y_i$. The triplet loss shares the goal of the duplet loss from Eq. 1, but employs anchor positive examples to ensure that the distance between any two instances of the same family is less than the distance to an instance of a different family by at least a pre-specified margin α. More formally, the triplet loss is defined by Eq. 7:

$$L_T(x_i, x_j, x_k) = \max(0, D(x_i, x_k) - D(x_j, x_k) + \alpha) \tag{7}$$

The triplet loss, therefore, presents a more relaxed form than the duplet loss, allowing instances of the same family to occupy larger regions. As opposed to the composite form of the duplet loss, the triplet loss is simple. However, the triplet loss is more heavily dependent on the quality of the sampled negative examples and anchors, which might lead to poorer training dynamics.

3.3 Online Duplet and Triplet Mining

We train our encoder using mini-batches of duplets or triplets from the data set. The number of possible duplets and triplets grows, respectively, quadratically and cubically with the number of instances in the data set, which renders exhaustive training costly. Feasibility aside, training with all possible duplets or triplets is not desirable as a large proportion of the resulting duplets and triplets make it either too easy or too difficult to fulfill the objective of Eq. 4 and Eq. 7. Such examples prevent the network from learning, and lead to slower convergence.

As suggested by [32] in the context of face recognition, efficient and fast converging training can be achieved by online selection of 'hard' duplets or triplets, i.e. the most dissimilar *positive* examples, and the least dissimilar *negative* examples. We apply this approach by first sampling a mini-batch of k instances per each of n sampled unique tune families. Subsequently, using the current model we compute the encodings of all $n \times k$ instances and for each instance we sample positive and negative, or anchor and negative examples from the mini-batch. In the case of the duplet loss, for each instance we select all possible positive examples (i.e. all other instances in the mini-batch from the same family) and an equal number of negative examples from the least dissimilar negatives. In the case of the triplet loss, for each instance x_i we select pairs of anchor x_k and negative example x_j such that the distance between positive and anchor is smaller than the distance between negative and anchor, while the difference between the distances lies inside the margin α:

$$0 < D(x_i, x_k) - D(x_j, x_k) < \alpha \tag{8}$$

In case no negative example can be found that satisfies this condition, we select a random negative.

3.4 Baseline: Alignment

We compare our results with the performance of a previously proposed alignment method [23]. In this method, the Needleman-Wunsch-Gotoh algorithm is used [10], which computes a global alignment score for two sequences of symbols. The alignment is constructed by inserting gaps at appropriate locations in the sequences following a dynamic programming approach. The alignment score is based on a similarity function for symbols and a gap scoring scheme. The Gotoh-variant of the algorithm applies an affine gap scoring function in which the continuation of a gap obtains a different score than the opening of a gap, opposed to the basic variant of the algorithm in which all gaps obtain the same score. We use the best scoring configuration in [23], which uses pitch, metric weight and the position of a note in its phrase.

3.5 Evaluation

We formulate the task of tune family identification as a ranking problem: given a query melody q_i and a data set of melodies $X, q_i \notin X$, the models should provide a ranked list of the melodies in X. To evaluate how well our models solve this problem, we measure the performance of the models by means of three evaluation measures: (i) 'Average Precision', (ii) 'Precision at rank 1', and (iii) 'Silhouette Coefficient'. Each of these measures addresses a different aspect of the performance quality of the models. First, Average Precision (AP) addresses the question whether given a query melody, all or most relevant melodies are high up in the ranking:

$$\text{AP} = \frac{\sum_{k=1}^{N} P(k) \times rel(k)}{\text{number of relevant melodies}}, \tag{9}$$

where k is the position in the ranked list of N retrieved melodies. $P(k)$ represents the precision at position k, and $rel(k) = 1$ if the melody at position k is relevant, $rel(k) = 0$ otherwise. By computing the average AP over all query melodies, we obtain the Mean Average Precision (MAP). As a second ranking measure, we focus on the Precision at rank 1 score (P@1), which computes the fraction of queries for which the highest ranked sequence is relevant. Third and finally, we compare these two ranking based evaluation measures with the Silhouette Coefficient, which is a measure of cluster homogeneity and separation. The Silhouette Coefficient contrasts the mean similarity between a sample and all other samples from the same family with the similarity of that sample with members of other families. By taking the average over all silhouette scores, we obtain a measure of cluster homogeneity ranging from -1 (incorrect clustering) to 1 (perfect clustering). [5]

3.6 Training and Hyper-Parameter Optimization

The networks were trained on the training data sets specified in Table 1. We use the Adam optimizer [17] and stop training after no improvement in MAP score was made on the development data for ten consecutive epochs. The neural network consists of a large number of hyper-parameters, making hyper-parameter tuning expensive and time-consuming. Following [2], we perform a randomized hyper-parameter search, in which we train n differ-

[5] See the Supplementary Materials for more information.

	MAP			P@1	Sil.
	all	seen	unseen		
RNN_D	0.72	0.71	0.73	0.78	0.34
RNN_T	0.71	0.70	0.71	0.77	0.29
Alignment	0.69	–	–	0.78	0.23

Table 2. Best evaluation scores for the development data. RNN_D is an RNN with duplet loss, and RNN_T is an RNN with triplet loss.

	MAP			P@1	Sil.
	all	seen	unseen		
RNN_D	0.72	0.70	0.74	0.78	0.33
RNN_T	0.68	0.64	0.71	0.75	0.28
Alignment	0.67	–	–	0.78	0.22

Table 3. Evaluation scores for the test data. RNN_D is an RNN with duplet loss; RNN_T is an RNN with triplet loss.

ent models with random hyper-parameter settings sampled from parameter-specific, relatively flat distributions. [6]

4. RESULTS

Table 2 presents the results for the development data set with MAP scores for the best performing models trained with duplet (RNN_D) and triplet loss (RNN_T). The best RNN_D achieves a MAP of 0.72, which is markedly better than the Alignment method (0.69), and slightly better than the best RNN_T (0.71). However, as will be discussed in more detail below, RNN_T models are significantly harder to optimize than RNN_D models (at least for the current data set). The columns 'seen' and 'unseen' represent MAP scores for queries of which the corresponding tune families were either seen or unseen during training. Crucially, the scores are almost equivalent, indicating that the neural networks are capable of actually learning a similarity metric, and not just a clustering or classification procedure, in which the systems learn to assign sequences to known class labels and data points. The performance differences between the models are further expressed by the Silhouette coefficient, which indicates superior performance of the RNN_D model. However, note that for P@1, all systems perform equally well.

The best performing models were employed to encode the melodies in the test set. The test results in Table 3 show a similar picture. Again, RNN_D outperforms the other systems. Note that the performance of RNN_T slightly dropped in comparison to the development results and that the performance difference with respect to RNN_D has become larger. Overall, the RNNs appear to have adequately learned how to form compact distributional representations of symbolic music in an induced melody space. This capability is further illustrated by the two-dimensional projection of the melody space in Figure 5. The left subplot demonstrates that the learned representations clearly separate vocal (FS) from instrumental melodies (INST). The subplot to the right serves as a validation of the clustering capabilities of the encoder. It highlights the positions of a small number of randomly chosen tune families, the members of which all cluster together.

[6] The full list of hyper-parameters and the predefined priors are listed in Section 2 of the Supplementary Materials.

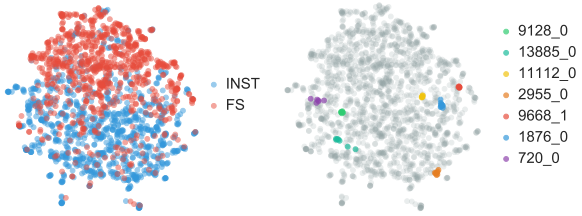

Figure 5. Two-dimensional UMAP [25] projection of the induced melody space obtained with RNN_D. The left subplot visualizes the positions of instrumental melodies (INST) and vocal melodies (FS). The subplot to the right highlights the positions of a small number of randomly chosen tune families.

4.1 Hyper-Parameter Importance

We assess the importance of the different hyper-parameters of the neural networks by modelling their influence on the MAP scores resulting from the randomized parameter search [26]. To this end, we fit the following linear regression model:

$$\text{MAP}_i \sim \mathcal{N}(\mu_i, \sigma) \quad (10)$$

$$\mu_i = \gamma + \beta_l l_i + \beta_m m_i + \beta_h h_i + \beta_d d_i + \beta_b b_i + \beta_c c_i + \beta_{lm} l_i m_i, \quad (11)$$

where Eq. 10 specifies the likelihood function with mean μ and standard deviation σ, and Eq. 11 represents the linear model. Here, γ represents the intercept of the linear model, l_i is the loss type of model i (i.e. triplet or duplet loss), m_i is the margin value α, h_i is the dimension of the hidden layer, d_i refers to the embedding dropout value, b_i dummy encodes whether a model employed bidirectional versus unidirectional RNNs, and c_i is the cell type of the RNN (i.e. LSTM or GRU). Since the margin functions differently in the triplet and duplet loss, we model the interaction between margin and loss type ($\beta_{lm} l_i m_i$). All categorical predictors are dummy encoded, and the continuous predictors are zero centered. β priors are sampled from uninformative Normal distributions, $\mathcal{N}(0, 1)$, and σ is sampled from a weakly regularizing half-Cauchy prior with location 0 and scale 1.

Table 4 presents the posterior distribution estimates of the model along with their estimation errors, their 95% credible intervals (CI_{95}), and the $\hat{R}$ statistic. [7] The mean

[7] Since the linear model is Bayesian, the credible intervals can be interpreted straightforwardly as the 95% probability that the estimates fall in a particular range. The 'No U-Turn Sampler' (NUTS) was used for sampling [15], which is a specific type of Hamiltonian Monte Carlo (HMC).

	Estimate	Error	l-CI$_{95}$	u-CI$_{95}$	$\hat{R}$
γ	0.66	0.00	0.65	0.67	1.0
β_l	-0.08	0.01	-0.09	-0.07	1.0
β_m	0.01	0.01	-0.02	0.04	1.0
β_d	-0.05	0.02	-0.09	0.00	1.0
β_b	0.03	0.01	0.02	0.04	1.0
β_c	-0.05	0.00	-0.06	-0.04	1.0
β_h	0.02	0.00	0.01	0.03	1.0
β_{lm}	-0.18	0.02	-0.22	-0.13	1.0
σ	0.04	0.00	0.04	0.04	1.0

Table 4. Posterior distribution estimates for the hyper-parameters of the Neural Networks. In addition to the mean estimates, the table provides the estimation errors, 95% Credible Intervals, and the $\hat{R}$ statistic.

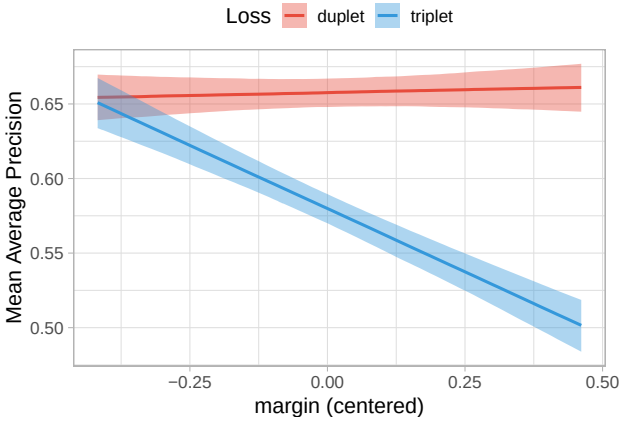

Figure 6. Marginal effects plot showing the interaction between loss type (i.e. Duplet and triplet loss) and the margin α.

intercept $\gamma = 0.66$ represents the mean posterior estimate of MAP values for unidirectional models, fit with GRU cells ($c_i = 0$), duplet loss ($l_i = 0$) and mean (i.e. 0) values for the continuous predictors. Given this base model, several interesting observations can be made. First, as suggested by the negative β_l estimate, triplet loss models markedly underperform duplet loss models with, *ceteris paribus*, a mean drop in performance of 0.08. Second, employing larger hidden dimensions (β_h) and using bidirectional RNNs (β_b) both positively influence the MAP scores. Third, on average adding too much dropout hurts performance ($\beta_d = -0.05$). Fourth, the strong negative posterior distribution estimate for the interaction between the margin α and loss type indicates that careful tuning of α is especially important for the triplet loss. By contrast, different values of α barely impact the performance of duplet loss models. The marginal effects plot in Figure 6 highlights this interaction. Finally, RNNs trained with GRU cells markedly outperform models trained with LSTM cells ($\beta_c = -0.05$). Figure 7 illustrates this performance difference. Additionally, the plot demonstrates the

$\hat{R}$ is a statistic to assess the convergence of the sampler, and should be below 1.1 [8]. For more information about the convergence and parameters, see the Supplementary Information.

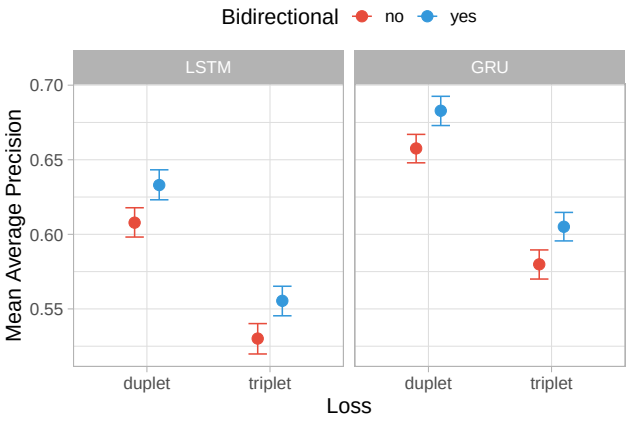

Figure 7. Marginal effects plot of RNN cell type (i.e. LSTM or GRU) and bidirectional versus unidirectional RNNs.

benefits of employing bidirectional RNNs, which consistently outperform unidirectional models.

5. CONCLUSION & FUTURE WORK

This paper proposed a method for end-to-end melody similarity metric learning using deep neural networks. We trained distributional melody encoders to minimize Duplet and Triplet Contrastive loss functions with which we achieve state-of-the-art retrieval performance on a large set of instrumental and vocal melodies. A thorough statistical analysis of the hyper-parameters of the Neural Networks indicates that on average Duplet Loss RNNs are easier to tune and less sensitive to specific hyper-parameter settings. Additionally, RNNs trained with GRU cells consistently outperform LSTM cell implementations. Our system has several major advantages over more traditional, alignment-based methods. First, thanks to its ability to infer complex interactions between input variables, the Neural Network approach is less sensitive to specific feature combinations and feature selection. Second, as shown by our study, the Neural Network approach displays more robustness, achieving similar MAP scores across exclusive sets of tune families (seen vs unseen).

For future work, we have the following three recommendations. First, the applicability of the proposed approach should be carefully examined on more diverse data sets, in order to test for the cross-domain robustness of the learned similarity metrics. Second, a more extensive and thorough comparison (including error analysis) with other existing melodic similarity methods is desired to highlight advantages and possible disadvantages of the neural systems. Finally, we acknowledge that, while successful, the proposed architecture still leaves room for improvement. Inspired by progress in similarity metric learning within the fields of Paraphrase Detection and Semantic Textual Similarity, we would like to experiment with more expressive neural architectures and feature extraction to explore the performance limits of the Neural Network approach on melodic similarity metric learning.

6. REFERENCES

[1] S.M. Bayard. Prolegomena to a study of the principal melodic families of british-american folk song. *Journal of American Folklore*, 63(247):1–44, 1950.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[3] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[4] Tian Cheng, Satoru Fukayama, and Masataka Goto. Comparing rnn parameters for melodic similarity. In *ISMIR*, pages 763–770, 2018.

[5] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. Association for Computational Linguistics, 2014.

[6] Roger B. Dannenberg, William P. Birmingham, Bryan Pardo, Ning Hu, Colin Meek, and George Tzanetakis. A comparative evaluation of search techniques for query-by-humming using the musart testbed. *Journal of the American Society for Information Science and Technology*, 58(5):687–701, 2007.

[7] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[8] Andrew Gelman, Donald B Rubin, et al. Inference from iterative simulation using multiple sequences. *Statistical science*, 7(4):457–472, 1992.

[9] Mathieu Giraud, Ken Déguernel, and Emilios Cambouropoulos. *Fragmentations with Pitch, Rhythm and Parallelism Constraints for Variation Matching*, volume 8905 of *Lecture Notes in Computer Science*, chapter Fragmentations with Pitch, Rhythm and Parallelism Constraints for Variation Matching. Springer, 2014.

[10] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

[11] Maarten Grachten, Josep Lluís Arcos, and Ramon López de Mántaras. Melody retrieval using the implication/realization model. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005)*, 2005.

[12] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.

[13] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[15] Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[16] Berit Janssen, Peter Van Kranenburg, and Anja Volk. Finding occurrences of melodic segments in folk songs employing symbolic similarity measures. *Journal of New Music Research*, 46(2):118–134, 2017.

[17] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, pages 1–15, 2015.

[18] Andreas Kornstädt. Themefinder: A web-based melodic search tool. *Computing in Musicology*, 11:231–236, 1998.

[19] Peter Van Kranenburg. *A Computational Approach to Content-Based Retrieval of Folk Song Melodies*. PhD thesis, Utrecht University, Utrecht, October 2010.

[20] Peter Van Kranenburg and Martine De Bruin. The meertens tune collections: Mtc-fs-inst 2.0. Meertens Online Reports 2019-1, Meertens Institute, Amsterdam, 2019.

[21] Peter Van Kranenburg, Martine De Bruin, and Anja Volk. Documenting a song culture: the dutch song database as a resource for musicological research. *International Journal on Digital Libraries*, 20(1):13–23, 2019.

[22] Peter Van Kranenburg, Martine De Bruin, Louis P. Grijp, and Frans Wiering. The meertens tune collections. Meertens Online Reports 2014-1, Meertens Institute, Amsterdam, 2014.

[23] Peter Van Kranenburg, Anja Volk, and Frans Wiering. A comparison between global and local features for computational classification of folk song melodies. *Journal of New Music Research*, 42(1):1–18, 2013.

[24] Velankar Makarand and Kulkarni Parag. Unified algorithm for melodic music similarity and retrieval in query by humming. In *Intelligent Computing and Information and Communication: Proceedings of 2nd International Conference, ICICC 2017*. Springer Singapore, 2018.

[25] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[26] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*, 2018.

[27] Marcel Mongeau and David Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.

[28] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157, 2016.

[29] Justin Salamon, Joan Serrà, and Emilia Gómez. Tonal representations for music retrieval: from version identification to query-by-humming. *International Journal of Multimedia Information Retrieval*, 2(1):45–58, 2013.

[30] Patrick E. Savage and Quentin D. Atkinson. Automatic tune family identification by musical sequence alignment. In *Proceedings of the 16th International Society for Music Information Retrieval Conference*, pages 162–168, 2015.

[31] Patrick E. Savage, Charles Cronin, Daniel Müllensiefen, and Quentin D. Atkinson. Quantitative evaluation of music copyright infringement. In *Proceedings of the 8th International Workshop on Folk Music Analysis (FMA2018)*, pages 61–66, 2018.

[32] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[33] Julián Urbano, Juan Lloréns, Jorge Morato, and Sonia Sánchez-Cuadrado. Melodic similarity through shape similarity. In Sølvi Ystad, Mitsuko Aramaki, Richard Kronland-Martinet, and Kristoffer Jensen, editors, *Exploring Music Contents*, volume 6684 of *Lecture Notes in Computer Science*, pages 338–355. Springer Berlin Heidelberg, 2011.

[34] Jelmer Van Nus, Geert-Jan Giezeman, and Frans Wiering. Melody retrieval and composer attribution using sequence alignment on rism incipits. In *TENOR 2017 International Conference on Technologies for Music Notation & Representation*, 2017.

[35] Valerio Velardo, Mauro Vallati, and Steven Jan. Symbolic melodic similarity: State of the art and future challenges. *Computer Music Journal*, 40(2):70–83, 2016.