Accelerating Multimodal Sequence Retrieval with Convolutional Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Given a large database of sequential data, a natural problem is to find the entry in the database which is most similar to a query sequence. Warping-based similarity metrics such as the dynamic time warping (DTW) distance can be prohibitively expensive when the sequences are long and/or high-dimensional. To mitigate these issues, [1] utilizes a convolutional network to map sequences of feature vectors to downsampled sequences of binary vectors. On the task of matching synthetic renditions of pieces of music to a large database of audio recordings of songs, this approach was able to efficiently discard 99% of the database with high confidence. We extend this approach to the multimodal setting where rather than synthetic renditions, a matrix representation of the piece's score is used, demonstrating that this approach is adaptable to the changes in the underlying representation.

1 Introduction

The ability to compute a similarity metric for sequences of feature vectors is necessary for the task of retrieving the most similar entry (nearest-neighbor search) in a database of sequences. A natural way to compare sequences is to first find their optimal alignment and then compute the total distance between aligned feature vectors. Aligning the sequences before computing the total distance makes metrics of this type robust to timing distortions (e.g. offset, skew, or cropping), and can be achieved in quadratic time using dynamic programming [2]. For feature vectors which are effectively compared with Euclidean distance (e.g. those with continuously-valued feature vectors), the most commonly used method of this type is dynamic time warping (DTW) [3], which will be the focus of this work.

The quadratic cost of the dynamic programming-based alignment operation can make nearest-neighbor search infeasible for for databases with many entries and/or long sequences. Furthermore, traditional DTW involves computing the pairwise distance between all feature vectors in the two sequences being compared, which can outweigh the cost of the alignment for high-dimensional data. A common way to avoid these costs is to use "pruning" techniques, which use heuristics to skip a large portion of the database. A wide variety of pruning methods have been proposed; in [2] it is shown that their successful application can enable nearest-neighbor search in databases with trillions of sequences. Despite their benefits, pruning methods typically rely on various constraints on the comparisons being made, such as the query sequence always being a subsequence of its correct match in the database or that the total number of aligned frames is fixed. Furthermore, these methods suffer losses in efficiency when sequences are oversampled and/or high dimensional.

To avoid these issues, in [1] we proposed a learning-based method which utilizes a convolutional network to map sequences of feature vectors to downsampled sequences of binary vectors. The resulting "hash sequences" can be much more efficiently compared using dynamic time warping, which enables flexible, problem-adaptive pruning. In this paper, we will show that this framework is

additionally flexible to multimodal settings, where the sequences being compared represent different types of data.

2 Learning a More Efficient Representation for DTW

Because the constraints and requirements of a sequence retrieval problem can vary based on the data, a natural question is whether sequence comparison can be made more efficient in a learning-based manner. In particular, if sequences are very long and high-dimensional, mapping sequences to shorter, lower-dimensional sequences such that similarity is preserved would provide quadratic speed gains when comparing sequences with DTW. Motivated by this possibility, in [1] we developed a system with the following capabilities:

Maps to a Hamming space: By replacing continuous-valued feature vectors with bitvectors in an embedded Hamming space, computing the distance between a pair of feature vectors simplifies to a single exclusive-or and a table lookup: The exclusive-or of two bitvectors a and b will yield a bitvector consisting of 1s where a and b differ and 0s elsewhere, and the number of 1s in all bitvectors of length D can be precomputed and stored in a table of size 2^D.

Downsamples sequences: Rather than creating a one-to-one correspondence between the original featured vectors and mapped bitvectors, groups of subsequent feature vectors are mapped to a single bitvector, giving a quadratic increase in efficiency.

Preserves similarity: The system is trained with an objective which seeks to produce a mapping where aligned feature vectors from matching sequences have a small Hamming distance in the embedded space, and non-matched feature vectors have a large distance.

Learns its representation: Our approach is entirely data-driven, which allows it to adapt to different problem settings including multimodal data, as we show in Section 3.

In this section, we will given an overview of our system; for a more thorough description, see [1] §3.

2.1 Similarity-Preserving Hashing

To begin with, our model requires a training set of sequences which are both matched and aligned. This training set can be constructed by obtaining a collection of sequences for which matching pairs are known, and then using DTW to find the optimal alignment of feature vectors in matching sequences. We call this collection of matching feature vectors $\mathcal{P}$, such that $(x, y) \in \mathcal{P}$ indicates that x is the feature vector in some sequence from one modality which has been aligned to y in a matching sequence from another modality. We then construct $\mathcal{N}$, a set of "dissimilar" pairs, by repeatedly randomly choosing two pairs $(x_1, y_1), (x_2, y_2) \in \mathcal{P}$ and swapping their entries to construct $(x_1, y_2), (x_2, y_1) \in \mathcal{N}$. Given this training data, our goal is to map feature vectors to a Hamming space where pairs in $\mathcal{P}$ have a small distance and pairs in $\mathcal{N}$ have a large distance. Motivated by the multimodal hashing technique of [4], we use the following objective function to measure the quality of the mapping:

$$\mathcal{L} = \frac{1}{|\mathcal{P}|} \sum_{(x,y)\in\mathcal{P}} \|f(x) - g(y)\|_2^2 - \frac{\alpha}{|\mathcal{N}|} \sum_{(a,b)\in\mathcal{N}} \max(0, m - \|f(a) - g(b)\|_2)^2$$

where f and g are learned nonlinear functions, α is a parameter to control the importance of separating dissimilar items, and m is a target separation of dissimilar pairs. As long as f and g are differentiable functions, this system can be optimized by using gradient descent to adjust the parameters of f and g such that $\mathcal{L}$ is minimized.

After minimizing $\mathcal{L}$ on a training set, pairs of sequences of feature vectors in either modality can then be mapped to pairs of sequences of hash vectors using f and g. Once mapped, we can perform DTW much more efficiently to compute a distance between the hash sequences. Ideally, the DTW distance between the original sequences will be well-approximated by the DTW distance of the hash vectors, which can be computed more efficiently. Overviews of DTW and its calculation can be found in [5, 2] and [1] §2.2.

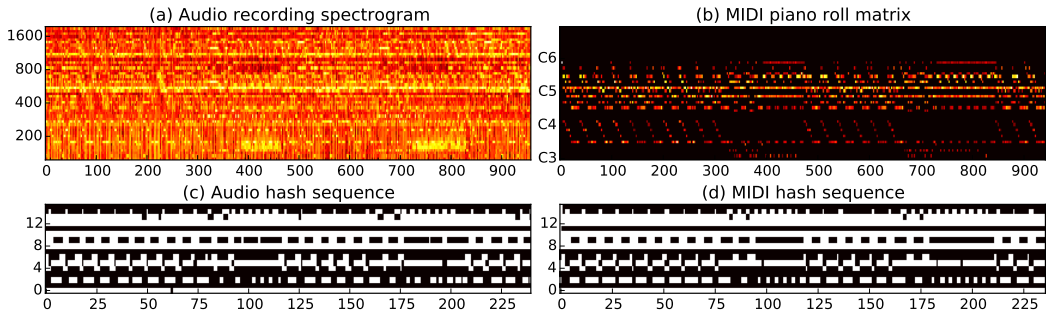

Figure 1: Example audio spectrogram and MIDI piano roll and their corresponding hash sequences (cf. [1] Figure 2). (a) Normalized beat-synchronous constant-Q spectrogram of an audio recording, with frequency in Hz on the vertical axis and beat index on the horizontal axis. (b) Piano roll matrix for a transcription of the same song, with notes on the vertical axis and beat indices on the horizontal axis. (c) Resulting hash sequence for the audio spectrogram, with pooled beat indices and Hamming space dimension on the horizontal and vertical axes respectively. (d) Hash sequence for the MIDI piano roll.

2.2 MIDI to Audio Matching Experiment

We tested the effectiveness of this approach on the task of matching synthetically generated renditions of pieces of music to recordings of the same pieces in a large database of audio recordings. For the first modality, synthetic recordings of music were obtained by using the `fluidsynth` program to synthesize musical scores in the form of MIDI files, a widely used format for transcribing music. For the second, we used audio recordings from the Million Song Dataset (MSD) [6, 7]. The goal was therefore to efficiently match a synthetic rendition to its original recording among the million recordings in the MSD. For both modalities, we used feature vectors of log-magnitude, constant-Q (i.e. log-frequency) spectra with 48 frequency bins from 130 to 1047 Hz. To normalize differences in tempo, we estimated beat locations and computed spectrograms (sequences of spectra) on a timescale where each spectrum corresponded to the span of an estimated beat. An example of this representation can be seen in Figure 1(a). MIDI and audio analyses were performed with `pretty_midi` [8] and `librosa` [9, 10] respectively.

In [4], dense feed-forward neural networks are used for the learnable functions f and g; we instead opted for convolutional networks due to the fact that sequential data tends to exhibit invariances over time which convolutional networks can learn to model with fewer parameters. In addition, by utilizing max-pooling layers to downsample over time, the resulting hash sequences can be made shorter, which makes computing the DTW distance quadratically faster. In our experiments, we used the same network structure for f and g, which consisted of a convolutional network followed by a dense, fully-connected network. Each layer in the network used a rectified linear unit (ReLU) nonlinearity except the last, which as in [4] used a $\tanh$ nonlinearity. This allows us to obtain binary hash vectors by testing whether each output dimension is greater or less than zero. We used an output dimensionality of 16 so that the resulting hash vectors could be represented as 16-bit unsigned integers. All weight matrices and filters were initialized using He's method [11], and all biases were initialized to zero. Our model was implemented using `theano` [12, 13] and `lasagne` [14].

For training data, we assembled a collection of MIDI files which were determined to be high-quality transcriptions of corresponding audio recordings. We held out 10% of this training set as a validation set, which was used to estimate the performance of our networks for early stopping and hyperparameter tuning, described below. We normalized all input feature vectors by their L^2-norm, then z-scored (standardized) feature vectors according to the statistics of the training set. Training data was presented to the network as minibatches of 50 randomly-cropped length-100 subsequences. To train the networks, we backpropagated the gradient of $\mathcal{L}$ through the networks' parameters and used RMSProp for optimization [15]. After each 100 minibatches, we tested whether the loss $\mathcal{L}$ computed on the validation set was was less than 99% of the previous lowest; if it was, we trained for 1000 more iterations (minibatches).

	(a) Synthesized MIDI					(b) Piano roll				
Rank	1	10	100	1000	10000	1	10	100	1000	10000
Percent $\leq$	15.2	41.6	62.8	82.7	95.9	10.9	34.6	55.6	74.9	92.8

Table 1: Percentage of MIDI/MSD pairs where the DTW distance ranked below each threshold, using (a) the synthesized MIDI spectrogram or (b) the MIDI piano roll as a representation.

To maximize performance, we performed Gaussian Process-based hyperparameter optimization using Whetlab, which was a web API implementing the techniques described in [16]. The scale of the validation loss will vary with the α and m regularization hyperparameters, so as an objective we instead used the Bhattacharyya coefficient between the distributions of distances between hash vectors produced from matching and non-matching sequences from the validation set. A small Bhattacharyya coefficient therefore indicates that a given system produces hash vectors which preserve similarity. We used Whetlab to optimize the number of convolutional/pooling layers, the number and size of the fully-connected layers, the RMSProp learning rate and decay parameters, and the α and m regularization parameters of $\mathcal{L}$. The best performing hyperparameter configuration produced a Bhattacharyya coefficient of 0.488, indicating a high degree of separation between the distributions.

After training a model which effectively maps sequences of feature vectors to shorter sequences of binary vectors, we evaluated its performance on a held-out test set of MIDI files for which we knew a priori the correct match in the MSD. For each MIDI file in our test set, we computed its hash sequence representation and the resulting DTW distance to every hash sequence of every audio recording in the MSD. To measure performance, we then ranked MSD entries according to their hash sequence DTW distance and determined the rank of the correct match. In our test set of 1,537 pairs, our system achieved a mean reciprocal rank of 0.241, indicating that the correct match tended to be ranked highly. To further measure performance, we report the percentage of MIDI files in the test set where the correct match in the MSD ranked below a certain threshold for various thresholds in Table 1(a).

As can be seen from this table, the correct entry in the MSD only had the smallest distance about 15.2% of the time, which suggests that this approach cannot reliably used in isolation. However, because it was able to rank the correct entry in the top 1% of the MSD (corresponding to 10,000 entries) 95.9% of the time, it can instead be used as an effective pruning method. Our best-performing model included two 2×2 max-pooling layers, which resulted in the number of hash vectors in a given sequence being 1/4 the number in the corresponding feature vector sequence. This, combined with the speed-up obtained by using binary vectors instead of continuously-valued ones, resulted in our approach being 100 times faster than standard DTW on the raw feature vectors. As a result, a concise way to state the success of our system is that it was able to discard 99% of entries of the MSD in 1% of the time.

3 Shared Representations for Multimodal Sequences

Our proposed system is structured to learn mappings from two different modalities thanks to the fact that two separate networks f and g are trained. Despite this capability, in [1] we utilized a representation of MIDI files which is similar to the one used for audio recordings. One drawback of using the constant-Q spectra of the synthesized MIDI files as feature vectors is that the resulting feature extraction pipeline is inefficient; for a complex MIDI file it can take tens of seconds. In addition, utilizing this representation does not test the multimodal capabilities of our model.

We therefore repeated the experiment from [1] using a representation of MIDI data which is much more efficient to calculate and substantially different than the one used for audio files. Instead of synthesizing the MIDI file and computing a constant-Q spectrogram, we instead computed a beat-synchronous "piano roll" matrix directly from the MIDI data itself. A piano roll matrix P is constructed such that $P[i, j] > 0$ denotes that note i is playing on some instrument during the jth beat, and $P[i, j]$ is 0 otherwise. An example piano roll matrix can be seen in Figure 1(b). Raw MIDI data can be readily converted to this format; for complex MIDI files, constructing such a matrix takes tens of milliseconds using `pretty_midi`'s `get_piano_roll` and `get_beats` functions

[8]. While the resulting representation is still a time-frequency matrix, it contains substantially less information (e.g. notes played on different instruments have the same representation) and is not directly comparable to the spectrograms we used for audio files.

For ease of comparison, we utilized the system outlined in Section 2.2 exactly except that we replaced the synthesized spectrograms with beat-synchronous piano roll matrices. After training on this representation, the hashing network achieved a Bhattacharyya coefficient of 0.538 which indicates that the networks succesfully preserved similarity, although not as effectively as when spectrograms of synthesized MIDI files were used as a feature representation. The resulting hash sequences for the audio spectrogram shown in Figure 1(a) and the MIDI piano roll shown in 1(b) can be seen in Figure 1(c) and 1(d) respectively, which at a high-level appear much more similar than the original feature vector sequences.

When evaluated on the task of matching our test set of MIDI files to the MSD, the model trained on piano roll matrices performed slightly worse than the system outlined in Section 2.2, as can be seen in Table 1(b). In particular, only 92.8% (vs. 95.9%) of the correct MSD matches for the MIDI files in the test set ranked below 10,000 (1% of the MSD). While this difference is significant, it nevertheless demonstrates that our system was able to learn an effective mapping despite the fact that the feature representations in each modality were substantially different.

4 Future Work

In this paper, we have summarized the system proposed in [1] for mapping sequences of feature vectors to shorter sequences of hash vectors, and have shown that it is robust to multimodal problem settings. Given the success of this approach on the MIDI-to-audio recording matching experiment described in Section 2.2, we are interested in applying it to other large-scale sequence retrieval tasks. One source of difficulty with the use of our system is that it requires a training set of matched and aligned sequences. We are currently investigating methods to relax this constraint by embedding sequences of feature vectors as fixed-length vectors in a common space where similarity is preserved. This approach will also replace the DTW-based comparison of hash sequences with a simple distance calculation, which could afford further efficiency gains. For researchers interested in verifying and extending our results, all of the code used in these experiments is available online.[1]

References

[1] Colin Raffel and Daniel P. W. Ellis. Large-scale content-based matching of MIDI and audio files. In *Proceedings of the 16th International Society for Music Information Retrieval Conference*, 2015.

[2] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, et al. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 262–270, 2012.

[3] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics Speech and Signal Processing*, 26(1):43–49, 1978.

[4] Jonathan Masci, Michael M. Bronstein, Alexander M. Bronstein, and Jürgen Schmidhuber. Multimodal similarity-preserving hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):824–830, 2014.

[5] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[6] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 591–596, 2011.

[7] Alexander Schindler, Rudolf Mayer, and Andreas Rauber. Facilitating comprehensive benchmarking experiments on the million song dataset. In *Proceedings of the 13th International Society for Music Information Retrieval Conference*, pages 469–474, 2012.

[8] Colin Raffel and Daniel P. W. Ellis. Intuitive analysis, creation and manipulation of MIDI data with `pretty_midi`. In *Proceedings of the 15th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, 2014.

[9] Brian McFee, Matt McVicar, Colin Raffel, et al. `librosa`: v0.4.0. `https://github.com/bmcfee/librosa`, 2015.

[1] `https://github.com/craffel/midi-dataset`

[10] Brian McFee, Colin Raffel, Dawen Liang, et al. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th Python in Science Conference*, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[12] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, et al. Theano: new features and speed improvements. In *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop*, 2012.

[13] James Bergstra, Olivier Breuleux, Frédéric Bastien, et al. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3, 2010.

[14] Sander Dieleman, Jan Schlüter, Colin Raffel, et al. Lasagne: First release. `https://github.com/Lasagne/Lasagne`, 2015.

[15] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

[16] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.