Chord2Vec: Learning Chords Embeddings

Sephora Madjiheurem

September 17, 2016

1 Introduction

The word2vec model by Mikolov et al. [4] allows to learn vector representations of words that carry syntactic and semantic meanings and that are useful in various natural language processing (NLP) tasks. In this work, we look at the similar problem on music data. Indeed, arbitrary encodings of polyphonic music provide no useful information regarding the relationships that may exist between the individual musical chords. The main goal of this study is to introduce techniques that can be used for learning high-quality embedding chord vectors from sequences of polyphonic music.

We aim to achieve this by finding chord representations that are useful for predicting the neighboring chords in a musical piece. Formally, given a corpus of chords $\mathcal{T} = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_T\}$ and the neighborhood of size $2j$ of a chord $\boldsymbol{d}_t$ defined by $C(\boldsymbol{d}_t) = \{\boldsymbol{d}_{t+j}, -m \leq j \leq m, j \neq 0\}$, the objective is to maximize the average log probability with respect to some model parameters θ:

$$\max_{\theta} \frac{1}{|\mathcal{T}|} \sum_{\boldsymbol{d}_t \in \mathcal{T}} \sum_{\boldsymbol{c}' \in C(\boldsymbol{d}_t)} \log p(\boldsymbol{c} = \boldsymbol{c}' | \boldsymbol{d}_t, \theta) \,. \tag{1}$$

In the remaining of this report, we will ease the reading by omitting the subscripts of the chords. Throughout this work, will consider a neigborhood window of size 2 (i.e. $j = 1$).

We propose three different models inspired by model architectures used within the field of NLP. We train and evaluate each of these models on five different datasets. Finally we suggest directions for future work.

Linear model

Our first model is a naive adaptation of the Skip-gram model introduced by Mikolov et al. [4]. The main difference with the original Skip-gram model is that a text can be represented as a sequence of words, where each word can be represented as a "one-hot" vector. In the case of music, we need a "many-hot" vector to represent a chord, as more than one note can be heard simultaneously. Hence, we replace the last softmax layer in Skip-gram by a sigmoid layer, to predict each note individually.

The architecture of the first proposed model is shown in Figure 1. It is a feed forward neural network that consists of input, hidden and output layers. At the input layer, a chord is encoded using a fixed length binary vector $\boldsymbol{c} = \{c_1, c_2, \ldots, c_N\} \in \{0,1\}^N$, where N is the number of notes in the vocabulary. In this chord representation, the entries that are set to 1 correspond to the notes that are *on* in the chord, whereas the notes that are *off* are set to 0.

The weights between the input layer and the hidden layer can be represented by a matrix $M \in \mathbb{R}^{D\times N}$. Similarly, the weights between the hidden layer and the output layer can be represented by a matrix $\tilde{M} \in \mathbb{R}^{N\times D}$.

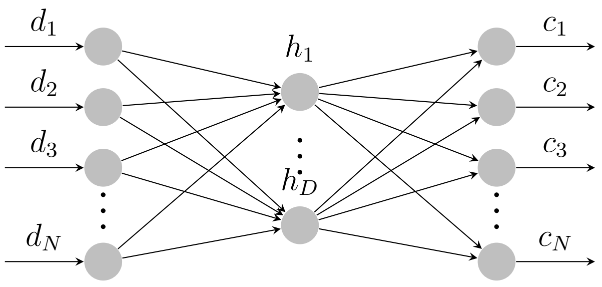

Figure 1: Linear model architecture

The D-dimensional vector representation $\boldsymbol{v}_d$ of the associated chord $\boldsymbol{d}$ is simply the normalized sum of the columns of M that correspond to the notes occurring in chord $\boldsymbol{d}$:

$$\boldsymbol{v}_d = M\frac{\boldsymbol{d}}{\|\boldsymbol{d}\|_1}$$

To compute the probabilities in (1) under this model, a conditional independence assumption between the notes in a context chord $\boldsymbol{c}$ given a chord $\boldsymbol{d}$ is made, i.e

$$p(\boldsymbol{c} = \boldsymbol{c}'|\boldsymbol{d}) = \prod_{i=1}^{N} p(c_i = c'_i|\boldsymbol{d})\,. \tag{2}$$

Using weights matrices M and $\tilde{M}$, we define the scoring function $h : \mathcal{N} \times \mathcal{C} \mapsto \mathbb{R}$:

$$h(i, \boldsymbol{d}) = \tilde{M}_{(:,i)} \boldsymbol{v}_d\,, \tag{3}$$

where $\tilde{M}_{(:,i)}$ denotes the i'th row of $\tilde{M}$.
We then use the sigmoid function to model the conditional probabilities :

$$p(c_i = c'_i|\boldsymbol{d}) = \begin{cases} \sigma\big(h(i,\boldsymbol{d})\big) = \frac{\exp(h(i,\boldsymbol{d}))}{1+\exp(h(i,\boldsymbol{d}))} & c'_i = 1 \\ 1 - \sigma\big(h(i,\boldsymbol{d})\big) = \sigma\big(-h(i,\boldsymbol{d})\big) = \frac{1}{1+\exp(h(i,\boldsymbol{d}))} & c'_i = 0 \end{cases} \tag{4}$$

This model is not expected to perform well, mainly because of the independence assumption between the notes in a chord. In fact, it is very likely that there is a strong dependency between the notes appearing in a chord. In the next section, we introduce a model that does not make such an independence assumption and is therefore expected to lead to improved results.

Autoregressive model

The second proposed model is inspired by the The Neural Autoregressive Distribution Estimator (NADE) which allows to model the distribution of high-dimensional vectors of discrete variables [3].

It decomposes the context chord probability distribution according to the chain rule as follows:

$$p(\boldsymbol{c} = \boldsymbol{c}'|\boldsymbol{d}) = \prod_{i=1}^{N} p(c_i = c'_i|\boldsymbol{d}, c_{<i})\,, \tag{5}$$

where $c_{<i} = \{c_1, \ldots, c_{i-1}\}$.

Using weights matrices $M, L \in \mathbb{R}^{D\times N}$ and $W \in \mathbb{R}^{N\times D}$, we define a new scoring function h:

$$h(i, \boldsymbol{d}, c_{<i}) = W_{:,i} \cdot (\boldsymbol{v}_d + \boldsymbol{v}_{c_{<i}})\,, \tag{6}$$

where $\boldsymbol{v}_{c_{<i}}$ is defined as $\sum_{j<i} L_{:,j} c_j$.

We then use the scoring function to model the individual conditional probabilities:

$$p(c_i = c'_i | \boldsymbol{d}, c_{<i}) = \begin{cases} \sigma\big(W_{:,i} \cdot (\boldsymbol{v}_d + \boldsymbol{v}_{c_{<i}})\big) & c'_i = 1 \\ 1 - \sigma\big(W_{:,i} \cdot (\boldsymbol{v}_d + \boldsymbol{v}_{c_{<i}})\big) & c'_i = 0 \end{cases} \tag{7}$$

where $\boldsymbol{v}_d = M \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|_1}$ is the D-dimensional vector representation of chord and $\sigma : \mathbb{R} \mapsto \mathbb{R}$ is the sigmoid function.

Sequence-to-sequence model

We propose to use another model to learn chord embeddings by learning which notes are compatible in a chord. This model is known as the sequence-to-sequence model [7]. In this setting, a chord is represented as a sequence of notes in some fixed ordering: $c \subseteq \mathcal{N}$, where $\mathcal{N}$ is the ordered set of all possible notes. Chord have arbitrary sizes.

Sequence-to-sequence models allow to learn a mapping of input sequences of varying lengths to output sequences also of varying lengths. It uses a neural network architecture known as RNN Encoder-Decoder. Figure 2 depicts the model architecture. An LSTM encoder is used to map the input sequence to a fixed length vector, and another LSTM decoder is then used to extract the output sequence from this vector. The general goal is to estimate $p(y_1, \ldots, y_{T'} | x_1, \ldots, x_T)$, where $x_1, \ldots, x_T$ and $y_1, \ldots, y_{T'}$ are the input and output sequences respectively, and T' and T need not to be equal.

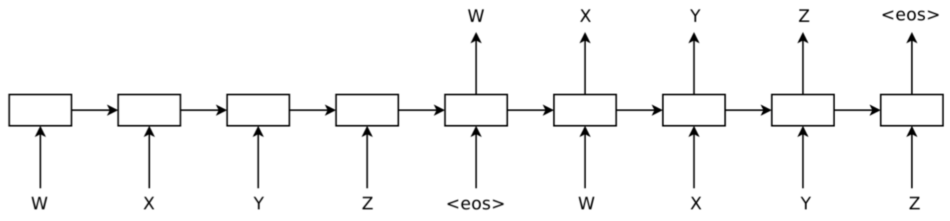

Figure 2: Sequence-to-sequence (Sutskever et al. [7])

The objective is given by:

$$\max_{\theta} \frac{1}{|\mathcal{T}|} \sum_{(X,Y) \in \mathcal{T}} \log p(Y | X, \theta) \,, \tag{8}$$

where Y is a correct output given the input X, $\mathcal{T}$ is the training set and θ is the set of the model parameters. The encoder and decoder are jointly trained to maximize the objective according to θ.

The model estimates the conditional probability $p(y_1, \dots, y_{T'}|x_1, \dots, x_T)$ by first obtaining the fixed-length vector representation v of the input sequence (given by the last state of the LSTM encoder) and then computing the probability of $y_1, \dots, y_{T'}$ with the LSTM decoder:

$$p(y_1, \dots, y_{T'}|x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t|v, y_1, \dots, y_{t-1}) \tag{9}$$

The sequence-to-sequence model can be used to learn embeddings for chords, by training the model to learn the context of a given chord.

At the chord level, the encoder projects all notes in a chord into a joint note representation and a decoder generates all notes in a chord given a chord representation and previously generated notes.

The overall goal is to estimate $p(c_1, \dots, c_T|d_1, \dots, d_{T'})$, where $d_1, \dots, d_{T'}$ and $c_1, \dots, c_T$ are in $\mathcal{N}$, $d = d_1, \dots, d_{T'}$ is an input chord and $c_{t+j} = n_1^{(j)}, \dots, n_T^{(j)}$ is a neighbor of c.

If $C(d)$ denotes the set of chords that are in the neighborhood of the chord d, then the objective in (8) can be written as:

$$\max_{\theta} \frac{1}{|\mathcal{T}|} \sum_{d \in \mathcal{T}} \sum_{c \in C(d)} \sum_{t=1}^{|c|} \log p(c_t|v_d, c_1, \dots, c_{t-1}) \, , \tag{10}$$

where v_d is the vector representation of the input chord d.

Experiments

Data sets

We compare the performance of the different models on five datasets of varying complexity:

JSB Chorales: corpus of 382 four-part harmonized chorales by J.S. Bach with the split of Allan and Williams [1]

Nottingham: collection of 1200 folk tunes with chords instantiated from the ABC format

MuseData: electronic library of orchestral and piano classical music from CCARH

Piano-midi.de: classical piano MIDI archive split according to Poliner and Ellis [6]

Mix: a collection of all above datasets.

In these data sets, the polyphony varies from 0 to 15 and the average number of simultaneous notes is 3.9. The range of notes spans the whole range of piano from A0 to C8, i.e. $N = |\mathcal{N}| = 88$. The the training data was augmented as follows: we transpose each piece by ϕ semi-tones, for $\phi = -6, -5, ..., 4, 5$.

Baseline models

In addition to the previously mentioned models, we compare our results with the two following methods:

- **Random**: the simplest baseline simply models the probability distribution of each note by the uniform probability distribution, i.e.

$$p(c_i = c_i' | d) = 0.5$$

.

- **Marginal distribution**: the second model computes the marginal probability distribution of the notes in the training data:

$$p(c_i = 1|d) = \frac{z_i + \alpha}{|\mathcal{T}| + \alpha},$$

where z_i is the number of the occurrences of note i in the training set $\mathcal{T}$ and $\alpha = 1$ is the constant used for additive smoothing.

Implementation Details

The linear chord2vec model and the autoregressive chord2vec were trained in batches of size 128 with Adam Optimizer using $D = 1024$. Each model is optimized for 200 epochs, but the final model is the one leading to the lowest validation error.

Our sequence-to-sequence model architecture is a multi-layer LSTM cells with 2 layers with 512 units. To allow sequences of varying length, bucketing and padding methods are used. We append to each target chord an end-of-sequence symbol.

We implemented all three models using Google's Tensorflow library.

Results

The negative log likelihoods on the test data for all models and on all data sets are presented in Table 1. First, we observe that our simple linear model, despite the independence assumptions, is able to actually learn something beyond the marginal distribution of the training set. Secondly, as expected, we observe that the autoregressive model achieves better scores than the simple linear model on all data sets, confirming our hypothesis that the notes in a chord are not independent of each other. Lastly, it appears that the Sequence-to-Sequence model is the model that leads to the lowest error overall.

Table 1: Negative log likelihood on test data

Model	JSB Chorales	Nottingham	MuseData	Piano-midi.de	Mix
Random	60.9969	60.9970	60.9970	60.9970	60.9972
Marginal	12.2349	10.4449	23.7489	17.3590	15.2652
Linear c2v	9.7734	5.7609	15.4104	12.6835	12.1712
Autoreg. c2v	6.1853	3.9801	14.4912	10.1847	7.4215
Seq2Seq c2v	1.1090	0.4974	1.5243	1.7795	1.2207

Future work

Our study has revealed that the Sequence-to-Sequence model, which is the model with the most complex architecture, leads to the highest log likelihood on all five datasets. However, in the case of text classification problems, it has been shown that a simpler architecture can lead to results that are comparable to deep learning models in terms of accuracy and are much faster for training and evaluation [2]. With this in mind, we introduce in this section as potential future work yet another model based on the Skip-gram model [4].

We also suggest some directions for measuring the quality of the learned embedding vectors.

One more chord2vec model

Although the architecture of the linear chord2vec model resembles the original Skip-gram's architecture [4], the following model seems to be a more direct adaptation of word2vec to chord2vec:

$$p(\boldsymbol{c} = \boldsymbol{c}'|\boldsymbol{d}) = \frac{\exp(\tilde{\boldsymbol{v}_{c'}} \cdot \boldsymbol{v}_d)}{\sum_{\boldsymbol{e} \in \{0,1\}^N} \exp(\tilde{\boldsymbol{v}}_e \cdot \boldsymbol{v}_d)} \tag{11}$$

where $\boldsymbol{v}_d = M \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|_1}$ and $\tilde{\boldsymbol{v}_{c'}} = \tilde{M} \frac{\boldsymbol{c}'}{\|\boldsymbol{c}'\|_1}$.
With this model, learning the vectors $\tilde{\boldsymbol{v}}_c$ is very expensive since we need to iterate through every possible chord. One solution to this scalability problem is to use *noise-contrastive estimation* [5].

Because this model does not make an assumption of independence between the notes occurring in a chord and given the success of the Skip-gram model for learning words embeddings [4], one can expect such a model to perform well.

Evaluating the Learned Embeddings

After having decided on the model leading to the lowest error on the test data, one could visualize the learned vectors by performing a dimensionality reduction using t-SNE or PCA for instance. An inspection of such low dimension vectors could help understanding whether the embedding vectors capture useful information about chords and their relationship to one anther.

References

[1] Moray Allan and Christopher KI Williams. Harmonising chorales by probabilistic inference. 2005.

[2] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016. URL `http://arxiv.org/abs/1607.01759`.

[3] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. *JMLR: W&CP*, 15:29–37, 2011.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[5] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2265–2273. Curran Associates, Inc., 2013.

[6] Graham E Poliner and Daniel PW Ellis. A discriminative model for polyphonic piano transcription. *EURASIP Journal on Applied Signal Processing*, 2007(1):154–154, 2007.

[7] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL `http://arxiv.org/abs/1409.3215`.