Structured training for large-vocabulary chord recognition

Brian McFee* & Juan Pablo Bello
Small chord vocabularies

- Typically a supervised learning problem
  - Frames → chord labels
- 1-of-K classification models are common
  - 25 classes: N + (12 × min) + (12 × maj)
  - Hidden Markov Models, Deep convolutional networks, etc.
  - Optimize accuracy, log-likelihood, etc.

<table>
<thead>
<tr>
<th></th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>C: maj</td>
<td>C: min</td>
</tr>
<tr>
<td>C#: maj</td>
<td>C#: min</td>
</tr>
<tr>
<td>D: maj</td>
<td>D: min</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>B: maj</td>
<td>B: min</td>
</tr>
</tbody>
</table>
Small chord vocabularies

- Typically a supervised learning problem
  - Frames → chord labels

- 1-of-K classification models are common
  - 25 classes: \( N + (12 \times \text{min}) + (12 \times \text{maj}) \)
  - Hidden Markov Models, Deep convolutional networks, etc.
  - Optimize accuracy, log-likelihood, etc.

- Implicit training assumption: All mistakes are equally bad
Large chord vocabularies

<table>
<thead>
<tr>
<th>Chord quality</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>maj</td>
<td>52.53%</td>
</tr>
<tr>
<td>min</td>
<td>13.63%</td>
</tr>
<tr>
<td>7</td>
<td>10.05%</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>hdim7</td>
<td>0.17%</td>
</tr>
<tr>
<td>dim7</td>
<td>0.07%</td>
</tr>
<tr>
<td>minmaj7</td>
<td>0.04%</td>
</tr>
</tbody>
</table>

Distribution of the 1217 dataset

- Classes are **not well-separated**
  - $C:7 = C:\text{maj} + m7$
  - $C:\text{sus}4$ vs. $F:\text{sus}2$
- Class distribution is **non-uniform**
- **Rare classes are hard to model**
Some mistakes are better than others

Very bad

Not so bad
Some mistakes are better than others

This implies that chord space is structured!
Our contributions

- Deep learning architecture to exploit structure of chord symbols
- Improve accuracy in rare classes
  Preserve accuracy in common classes
- Bonus: package is online for you to use!
Chord simplification

- All classification models need a finite, canonical label set
Chord simplification

- All classification models need a finite, canonical label set
- Vocabulary simplification process:
  a. Ignore inversions
Chord simplification

- All classification models need a finite, canonical label set

- Vocabulary simplification process:
  a. Ignore inversions
  b. Ignore added and suppressed notes
Chord simplification

- All classification models need a finite, canonical label set

- Vocabulary simplification process:
  a. Ignore inversions
  b. Ignore added and suppressed notes
  c. Template-match to nearest quality
Chord simplification

- All classification models need a finite, canonical label set

- Vocabulary simplification process:
  a. Ignore inversions
  b. Ignore added and suppressed notes
  c. Template-match to nearest quality
  d. Resolve enharmonic equivalences

\[ G♭ :9(*5)/3 \quad G♭ :9(*5) \quad G♭ :9 \quad G♭ :7 \quad F♯:7 \]
Chord simplification

- All classification models need a finite, canonical label set

- Vocabulary simplification process:
  a. Ignore inversions
  b. Ignore added and suppressed notes
  c. Template-match to nearest quality
  d. Resolve enharmonic equivalences

Simplification is lossy! (but all chord models do it)
14 \times 12 + 2 = 170 \text{ classes}

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>maj</th>
<th>dim</th>
<th>aug</th>
<th>min6</th>
<th>maj6</th>
<th>min7</th>
<th>minmaj7</th>
<th>maj7</th>
<th>7</th>
<th>dim7</th>
<th>hdim7</th>
<th>sus2</th>
<th>sus4</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>No chord (e.g., silence)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>Out of gamut (e.g., power chords)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Structural encoding

- Represent **chord labels** as **binary encodings**

- Encoding is **lossless** and **structured**:
  - Similar chords with **different labels** will have **similar encodings**
  - Dissimilar chords will have **dissimilar encodings**

- Learning problem:
  - Predict the **encoding** from audio
  - Learn to decode into **chord labels**

* up to octave-folding
The big idea

- Jointly estimate **structured encoding** AND **chord labels**
- Full objective = **root loss** + **pitch loss** + **bass loss** + **decoder loss**
Model architectures

- Input: constant-Q spectral patches

- Per-frame outputs:
  - Root \([\text{multiclass, 13}]\)
  - Pitches \([\text{multilabel, 12}]\)
  - Bass \([\text{multiclass, 13}]\)
  - Chords \([\text{multiclass, 170}]\)

- Convolutional-recurrent architecture (encoder-decoder)

- End-to-end training
Encoder architecture

Hidden state at frame $t$:

$$h(t) \in [-1, +1]^D$$
Decoder architectures

Chords = Logistic regression from encoder state

Frames are independently decoded:

\[ y(t) = \text{softmax}(W h(t) + \beta) \]
Decoder architectures

Chords = Logistic regression from encoder state

Decoding = GRU + LR

Frames are recurrently decoded:

\[ h_2(t) = \text{Bi-GRU}[h](t) \]

\[ y(t) = \text{softmax}(W h_2(t) + \beta) \]
Decoder architectures

Chords = Logistic regression from encoder state

Decoding = GRU + LR

Chords = LR from encoder state + root/pitch/bass

Frames are independently decoded with structure:

\[ y(t) = \text{softmax}(W_r r(t) + W_p p(t) + W_b b(t) + W_h h(t) + \beta) \]
**Decoder architectures**

1. **CR1**
   - Input
   - Encoder [512]
   - Chords

2. **CR2**
   - Input
   - Encoder [256]
   - Bi-GRU [256]
   - Chords

3. **CR1+S**
   - Input
   - Encoder [512]
   - Root
   - Pitches
   - Bass
   - Chords

4. **CR2+S**
   - Input
   - Encoder [256]
   - Bi-GRU [256]
   - Root
   - Pitches
   - Bass
   - Chords

---

- Chords = Logistic regression from encoder state
- Decoding = GRU + LR
- Chords = LR from encoder state + root/pitch/bass
- All of the above
What about root bias?

- Quality and root should be independent
- But the data is **inherently biased**
- Solution: **data augmentation!**
  - *muda* [McFee, Humphrey, Bello 2015]
  - Pitch-shift the audio and annotations simultaneously
- Each training track $\rightarrow \pm 6$ semitone shifts
  - All qualities are observed in all root positions
  - All roots, pitches, and bass values are observed

http://photos.jdhancock.com/photo/2012-09-28-001422-big-data.html
Evaluation

- 8 configurations
  - ± data augmentation
  - ± structured training
  - 1 vs. 2 recurrent layers
- 1217 recordings
  (Billboard + Isophonics + MARL corpus)
  - 5-fold cross-validation
- Baseline models:
  - DNN [Humphrey & Bello, 2015]
  - KHMM [Cho, 2014]
Data augmentation (+A) is necessary to match baselines.
Structured training (+S) and deeper models improve over baselines.
Results

Improvements are bigger on the harder metrics (7th's and tetrads)

CR1: 1 recurrent layer
CR2: 2 recurrent layers
+A: data augmentation
+S: structure encoding
Results

Substantial gains in maj/min and MIREX metrics

CR2+S+A wins on all metrics
Error analysis: quality confusions

Errors tend toward simplification

Reflects maj/min bias in training data

Simplified vocab. accuracy: 63.6%
Summary

- Structured training helps
- Deeper is better
- Data augmentation is critical
  - pip install muda
- Rare classes are still hard
  - We probably need new data
Thanks!

- Questions?

- Implementation is online
  - https://github.com/bmcfee/ismir2017_chords
  - pip install crema

brian.mcfee@nyu.edu
https://bmcfee.github.io/
Extra goodies
Error analysis: CR2+S+A vs CR2+A

Reduction of confusions to major

Improvements in rare classes: aug, maj6, dim7, hdim7, sus4
Learned model weights

- Layer 1: Harmonic saliency
- Layer 2: Pitch filters (sorted by dominant frequency)
Training details

- Keras / TensorFlow + pescador
- ADAM optimizer
- Early stopping @20, learning rate reduction @10
  - Determined by decoder loss
- 8 seconds per patch
- 32 patches per batch
- 1024 batches per epoch
Inter-root confusions

Confusions primarily toward P4/P5
Inversion estimation

- For each detected chord segment
  - Find the most likely bass note
  - If that note is within the detected quality, predict it as the inversion

- Implemented in the crema package

- Inversion-sensitive metrics ~1% lower than inversion-agnostic
Pitches as chroma