Some cool title

Aaron A. Gauthier, Avijeet Kartikay and Pedro Uria Rodriguez

Machine Learning II

GWU

April 24, 2019

Table of Contents

Automatic Music Generation

TODO

- Mmmm

Automatic Music Generation

TODO

- Mmmm
- Woooo

Automatic Music Generation

TODO

- Mmmm
- Woooo

And sooo..

Automatic Music Generation

TODO

- Mmmm
- Woooo

  And sooo..

- Yeahhh

Automatic Music Generation

TODO

- Mmmm
- Woooo

  And sooo..

- Yeahhh
- lolll

Automatic Music Generation

TODO

- Mmmm
- Woooo

  And sooo..

- Yeahhh
- lolll
- Heh

Motivation

TODO

MIDI

- Protocol that stores music data and metadata, and allows different instruments and software to communicate with each other.

MIDI

- Protocol that stores music data and metadata, and allows different instruments and software to communicate with each other.

  It is made up by a series of events, with info regarding:

MIDI

- Protocol that stores music data and metadata, and allows different instruments and software to communicate with each other.

  It is made up by a series of events, with info regarding:
- Location in time
- Duration in time
- Pitch, intensity and tempo
- Other metadata

Data Sources

TODO

Overview

- MIDI files provide us with more info than we need

Overview

- MIDI files provide us with more info than we need
- We are going to use a many-hot encoding approach

Overview

- MIDI files provide us with more info than we need
- We are going to use a many-hot encoding approach

  Thus...

Overview

- MIDI files provide us with more info than we need
- We are going to use a many-hot encoding approach

  Thus...
- Tempo will not be encoded: Too many possible values
- Intensity will not be encoded: Same reason

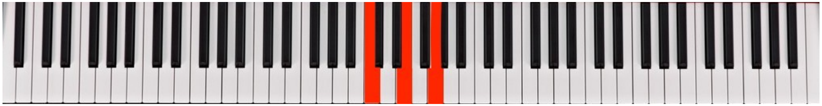
We are essentially losing expressiveness info in order to reduce the complexity of the network.

Many-one-hot Encoding

- Python's `music21` library to read `.mid` files.
- Preprocess the `stream` objects to get the data for the time events.

Many-one-hot Encoding

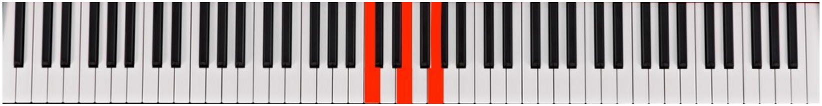
- Python's `music21` library to read `.mid` files.
- Preprocess the `stream` objects to get the data for the time events.



$$\bar{p}_t = [0, 0, \ldots.., 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, \ldots, 0]$$

Many-one-hot Encoding

- Python's `music21` library to read `.mid` files.
- Preprocess the `stream` objects to get the data for the time events.



$$\bar{p}_t = [0, 0, \ldots.., 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, \ldots, 0]$$

- Create a sequence where each input vector $\bar{p}_t$ corresponds to the duration of the shortest note on the piece/s:
  $time\ step \equiv \Delta_t = t_1 - t_0 = t_2 - t_1 = \ldots = cte$

Two extra dimensions

- Rests are essential to music. Add component #88 to encode rests.

Two extra dimensions

- Rests are essential to music. Add component #88 to encode rests.
- If a note is longer than the *time step*, it will be split into more than one vector. Suppose we have:

Two extra dimensions

- Rests are essential to music. Add component #88 to encode rests.
- If a note is longer than the *time step*, it will be split into more than one vector. Suppose we have:

time step = ♩, event at t_i = 𝅗𝅥

Two extra dimensions

- Rests are essential to music. Add component #88 to encode rests.
- If a note is longer than the *time step*, it will be split into more than one vector. Suppose we have:

time step = ♩, event at t_i = 𝅗𝅥 $\Rightarrow \bar{p}_{t_i} = [0, 0, ..., 1, 0, 0, ...]$

Two extra dimensions

- Rests are essential to music. Add component #88 to encode rests.
- If a note is longer than the *time step*, it will be split into more than one vector. Suppose we have:

time step = ♩, event at t_i = 𝅗𝅥 $\Rightarrow \bar{p}_{t_i} = [0, 0, ..., 1, 0, 0, ...]$ and $\bar{p}_{t_{i+1}} = [0, 0, ..., 1, 0, 0, ...] = \bar{p}_{t_i}$

Two extra dimensions

- Rests are essential to music. Add component #88 to encode rests.
- If a note is longer than the *time step*, it will be split into more than one vector. Suppose we have:

time step = ♩, event at $t_i =$ 𝅗𝅥 $\Rightarrow \bar{p}_{t_i} = [0, 0, ..., 1, 0, 0, ...]$ and $\bar{p}_{t_{i+1}} = [0, 0, ..., 1, 0, 0, ...] = \bar{p}_{t_i} \Rightarrow$ ♩ ♩ $\neq$ 𝅗𝅥

Two extra dimensions

- Rests are essential to music. Add component #88 to encode rests.
- If a note is longer than the *time step*, it will be split into more than one vector. Suppose we have:

time step $=$ ♩, event at $t_i =$ 𝅗𝅥 $\Rightarrow \bar{p}_{t_i} = [0, 0, ..., 1, 0, 0, ...]$ and $\bar{p}_{t_{i+1}} = [0, 0, ..., 1, 0, 0, ...] = \bar{p}_{t_i} \Rightarrow$ ♩ ♩ $\neq$ 𝅗𝅥

- Add *hold* component #89, which indicates that the notes played at a time event shall be held. We end up with:

Two extra dimensions

- Rests are essential to music. Add component #88 to encode rests.
- If a note is longer than the *time step*, it will be split into more than one vector. Suppose we have:

time step = ♩, event at t_i = 𝅗𝅥 $\Rightarrow \bar{p}_{t_i} = [0, 0, ..., 1, 0, 0, ...]$ and $\bar{p}_{t_{i+1}} = [0, 0, ..., 1, 0, 0, ...] = \bar{p}_{t_i} \Rightarrow$ ♩ ♩ $\neq$ 𝅗𝅥

- Add *hold* component #89, which indicates that the notes played at a time event shall be held. We end up with:

$\bar{p}_{t_i} = [0, 0, ..., 1, 0, 0, ..., 1], \bar{p}_{t_{i+1}} = [0, 0, ..., 1, 0, 0, ..., 0] \Rightarrow$ 𝅗𝅥.

Combining Melody & Harmony

TODO

hhh

TODO

Whatever

Something

Happy Music Generation!

Some cool pic!