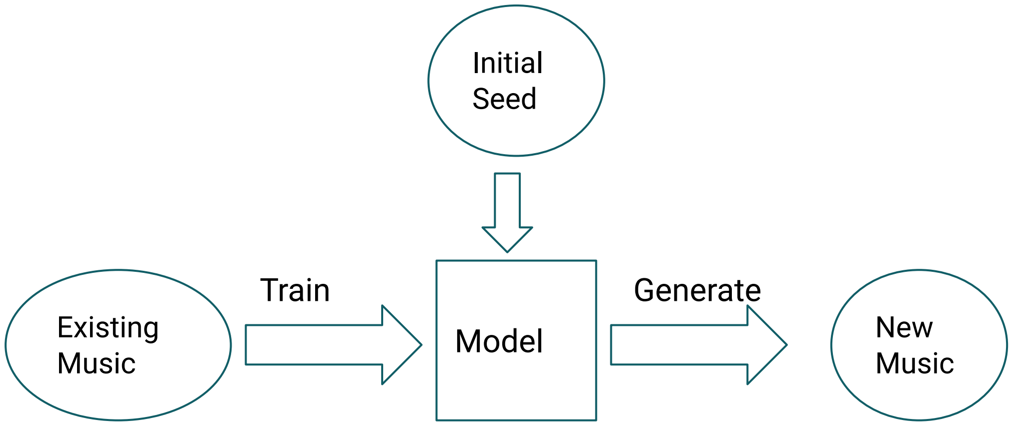
Making Music

May 4, 2019
Porter Perry, Javier Palomares, Spencer Marshall
[Team N.W.A.I]

Overview: Machine Learning for Making Music

- Music is very structured, yet highly dimensional
- We want to train a model to learn aspects of this structure
- Then give the training model an initial seed, and have it generate a song of given length



MIDI (Musical Instrument Digital Interface)

We're using MIDI files to encode music.

- MIDI : list of tracks (plus some messages containing meta information)
  - Track: List of messages
    - Messages: **note_on**, **note_off**, pitch_change,control_change

Used midis with only 1 track

Only trained on the note_on and note_off events

MIDI Messages

Name	Keyword Arguments / Attributes
note_off	channel note velocity
note_on	channel note velocity
polytouch	channel note value
control_change	channel control value
program_change	channel program
aftertouch	channel value
pitchwheel	channel pitch
sysex	data
quarter_frame	frame_type frame_value
songpos	pos
song_select	song
tune_request	
clock	
start	
continue	
stop	
active_sensing	
reset	

MIDI Protocol “note_on” and “note_off” Attributes

- note = [0..127] (Which note [pitch] to play)
- velocity = [0...127] (how hard to strike the note)
- time = [0...N] (in “ticks”)

Midi protocol message with “note_on” indicates the note was “pressed”.

Messages with “note_off” indicates the note was “released”

Encoding MIDIs

- 128 possible notes (values 0-127)
- 128 possible velocities(values 0-127)
- N beats

Midi note variable is categorical, so one-hot encoding applied

Encode as a matrix: N rows, 128 columns (Actually 129 - added one column to indicate no notes [rest beat])

If note ***i*** is played with at beat ***j***, set (***j***, ***i***) = 1. Otherwise elements are 0.

Each row represents a beat in the song.

Matrix is very sparse

Data Collection

Used classical piano midi files as data set since they are readily available for free use.

Scrapped all available midi files from http://www.piano-midi.de/midi_files.htm

Total of 717 midi files, each several minutes long.

Composer	Born	Died	Period
Albéniz, Isaac	1860	1909	Romanticism
Bach, Johann Sebastian	1685	1750	Baroque
Balakirew, Mili Alexejewitsch	1837	1910	Romanticism
Beethoven, Ludwig van	1770	1827	Classicism
Borodin, Alexander	1833	1887	Romanticism
Brahms, Johannes	1833	1897	Romanticism
Burgmueller, Friedrich	1806	1874	Romanticism
Chopin, Frédéric	1810	1849	Romanticism
Clementi, Muzio	1752	1832	Classicism
Debussy, Claude	1862	1918	Romanticism
Godowsky, Leopold	1870	1938	Romanticism
Granados, Enrique	1867	1916	Romanticism
Grieg, Edvard	1843	1907	Romanticism
Haydn, Joseph	1732	1809	Classicism
Liszt, Franz	1811	1886	Romanticism
Mendelssohn, Felix	1809	1847	Romanticism
Moszkowski, Moritz	1854	1925	Romanticism
Mozart, Wolfgang Amadeus	1756	1791	Classicism
Mussorgsky, Modest	1839	1881	Romanticism
Rachmaninov, Sergey	1873	1943	Romanticism
Ravel, Maurice	1875	1937	Romanticism
Schubert, Franz	1797	1828	Romanticism
Schumann, Robert	1810	1856	Romanticism
Sinding, Christian	1856	1941	Romanticism
Tchaikovsky, Peter	1840	1893	Romanticism
Christmas			

Data Preprocessing Step

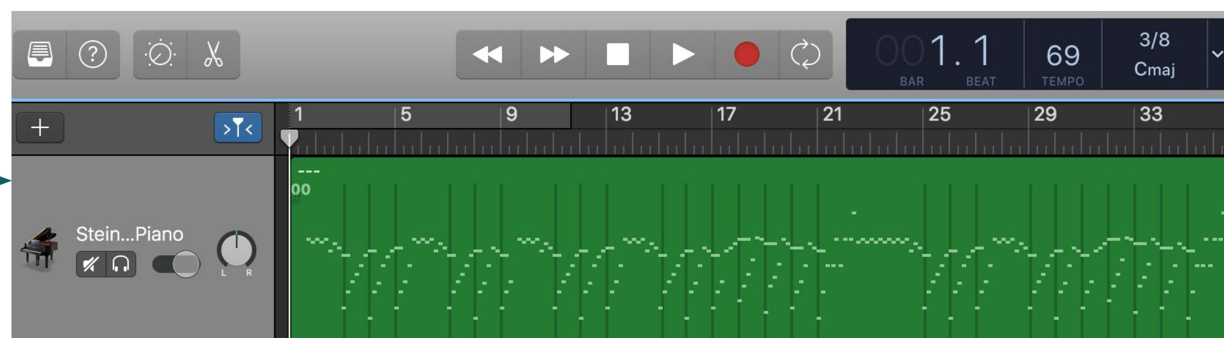
- Reformat original midi files so that piano notes all reside on one midi track/channel (standard midi format 0).

BEFORE

Ex. Beethoven's "Für Elise" midi piano roll representation:

- Some midi files were duplicated in various formats. Remove duplicated midi files as to not skew the model training.

AFTER

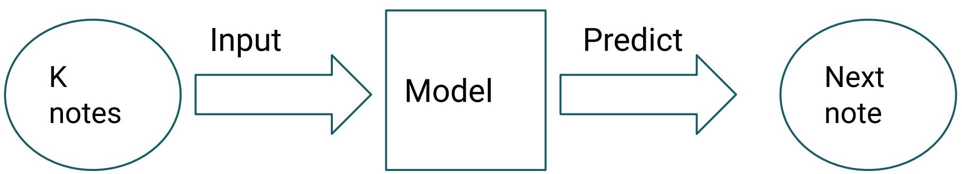


Forming the Training Data

Music generation = “Given the previous k notes, predict the next one”.

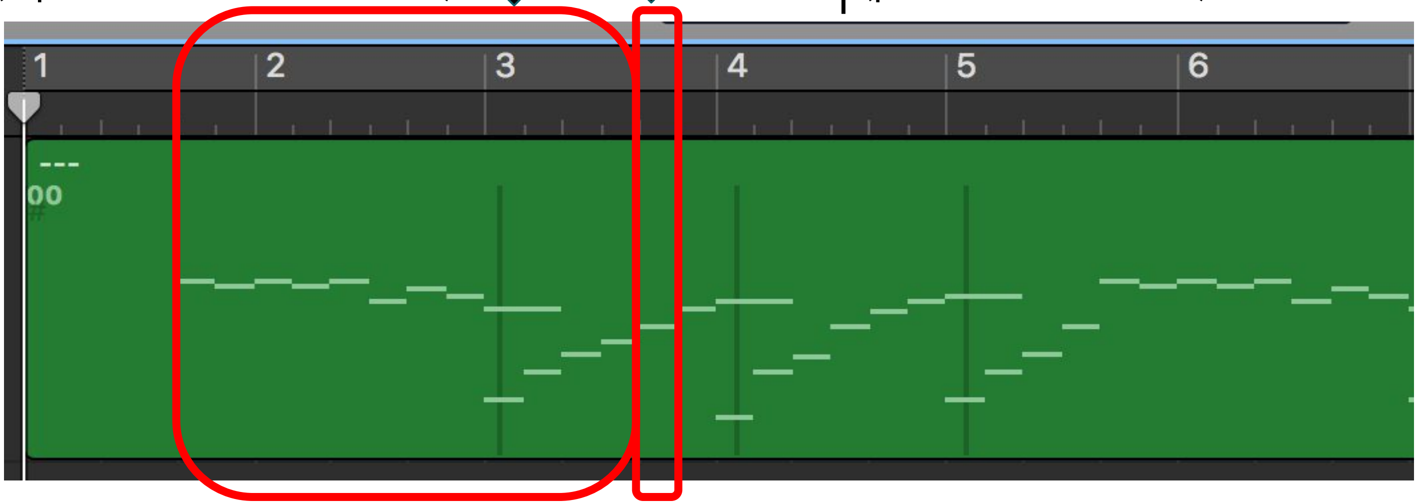
Chose a window size, and formed rolling windows of k consecutive notes (rows from the matrix) , and the note (row) that follows.

The k notes are the input features (X), the note that follows is the predicted value (y).

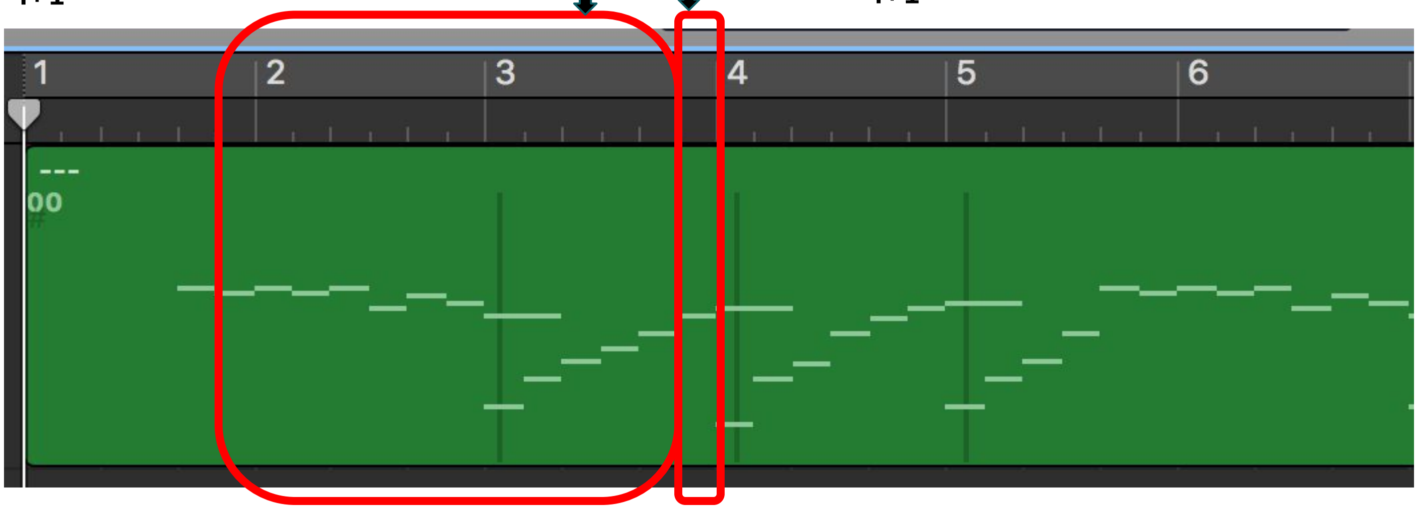


Rolling/Sliding Window Concept

X_i (input feature column notes) Y_i (predicted label note)

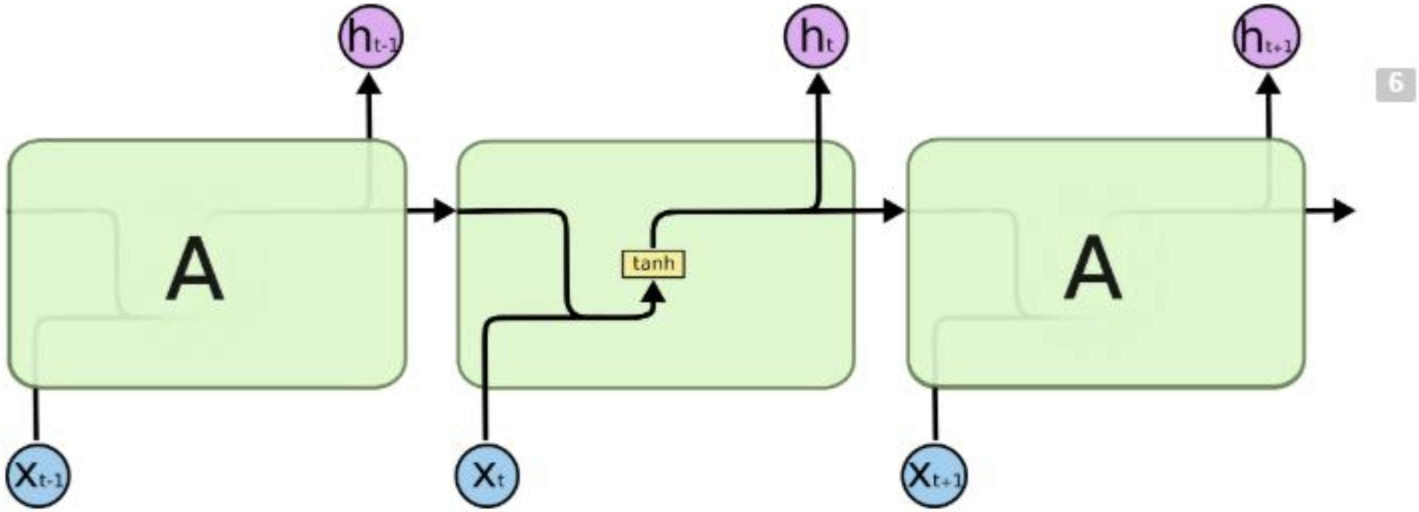


X_{i+1} (input feature column data) Y_{i+1} (predicted label note)



Choosing a Neural Network Model

Used a Long Short Term Memory (LSTM) Recurrent Neural Network (RNN).
They've been successful used in learning time-series forecasting..



The repeating module in a standard RNN contains a single layer.

LSTM Topology

Used 3 LSTM Layers, each with 64 nodes, plus a Dense layer with 129 outputs.

Cross_entropy_classification loss function with dropout rate of .2 to prevent overfitting.

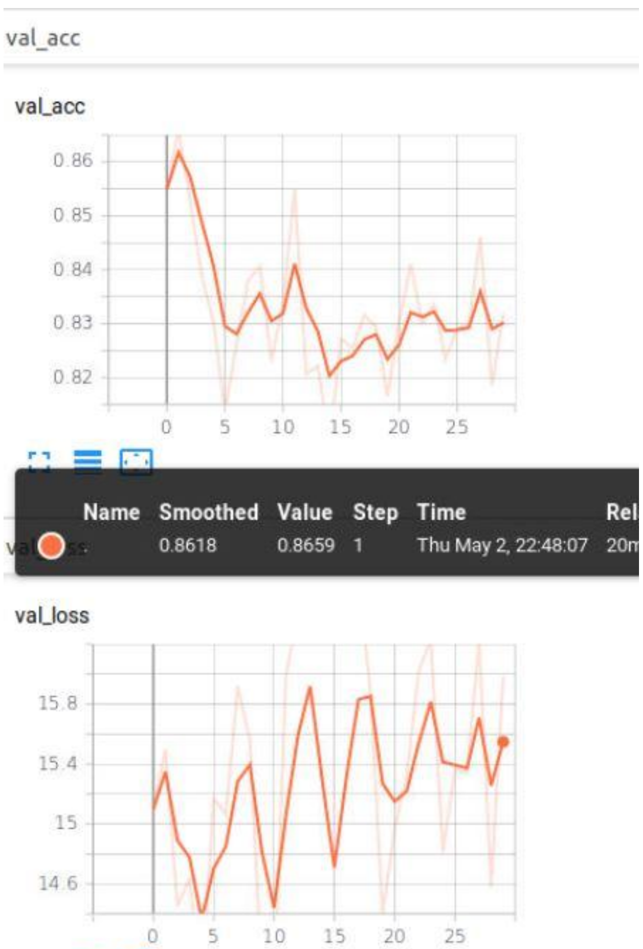
Default Adam optimizer.

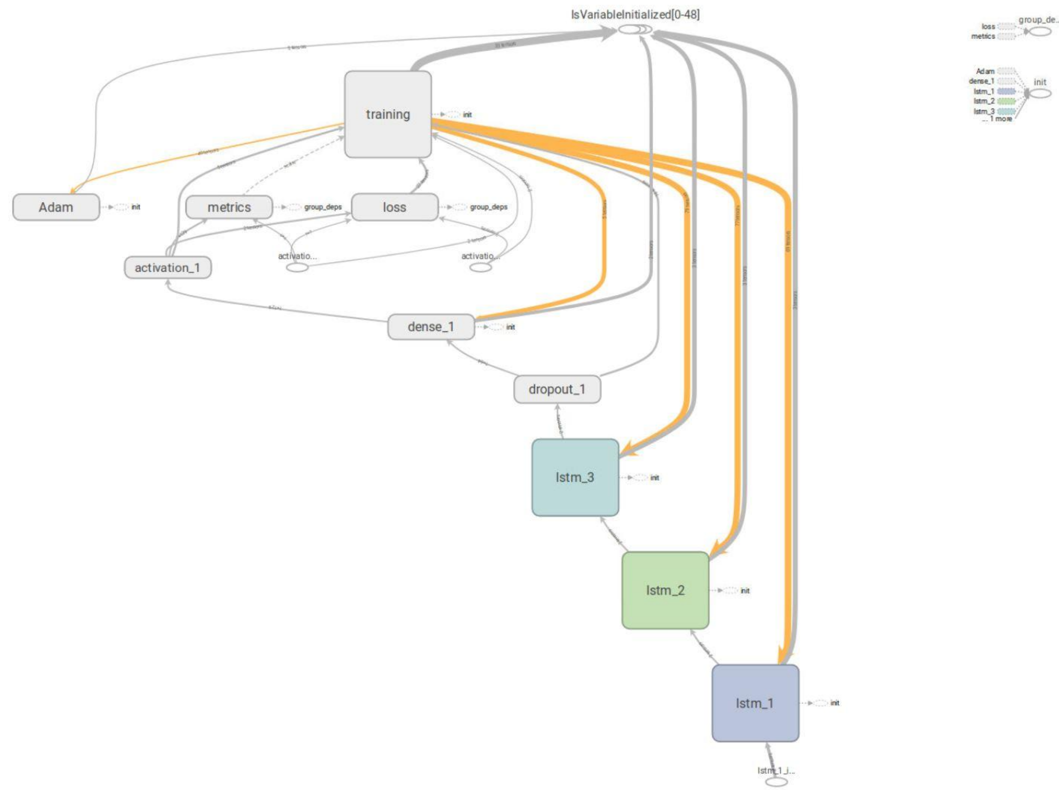
Softmax activation function in last layer for predicted probs.

Will sample from output to predict the next note in the sequence.

Model Training

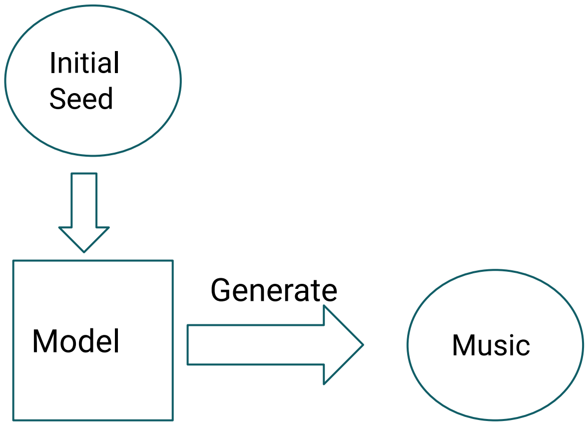
Performed on a Paperspace VM using Keras/Tensorflow-gpu packages on all composer’s music.
Used 30 epochs.





Generating Music

Take a seed from a given song (k rows),
Have the model predict the next note,
On next iteration, pop last note, add in the predicted note.
Repeat until song at desired length.



Parsing the model output

Dense layer with 129 output nodes.
Output = {p0, p1, p2, ..., p127, p128}
Softmax activation so output sums to 1.

p_i is the probability that note ***i*** is played in the note.

Pick a random ***i*** out of {0...128} weighted by the ***p_i*** (discrete distribution).

The ***ith*** i note note is to be played in that frame.

DEMO

What about velocities and time?

We ignored velocities and time in our model training.

In building the midi, we assumed a default velocity of 64 and quarter notes.

These 2 simplifications turned the problem into a classification:

Given X, classify into 1 of 129 classes (0-127 midi note pitches, or rest).

Monophonic

Classifying into 1 class <-> only one note can be played per beat, so no chords.

Getting a better model

- 128 possible notes
- 128 possible velocities
- N beats

Encode as a matrix: N rows, 128 columns. (Actually 129 - added one for rest beat)

If note i is played with at beat j with velocity v_j_i, set (j,i) = v_j_i. Otherwise elements are 0.

Matrix is still very sparse

Getting a better model - LSTM

3 LSTM Layers, each with 64 nodes, plus a Dense layer with 128 outputs.

Lasso loss function with dropout rate of .2 to prevent overfitting.

(Matrix is sparse, and we want sparse predictions).

Default Adam optimizer.

We are now doing regression instead of classification:

Given X (k notes), predict vi (velocity of note i) in the next note for i={0...127}

Didn’t work so good

Output of this model wasn’t very good:

Lasso loss function often gave 0 velocities for all outputs and in a few cases predicted negative velocities.

Also tested with L2 loss function:

Many notes (~50) played at once (didn’t sound good)

Given more time, we would like to continue this approach.

Acknowledgments/ Thank yous

Based our work on what has been done by Branger Briz who created midi-rnn based on Google's Project Magenta
https://brangerbriz.com/blog/using-machine-learning-to-create-new-melodies

Used pretty_midi package to load midi files in a piano roll format.The package is documented at http://craffel.github.io/pretty-midi/

Initially used pymidifile package to load midi files into a Pandas DataFrame. The package is documented at http://github.com/angelfaraldo/pymidifile/

Both pretty_midi and pymidifile are built on top of mido, a package for interfacing directly with midi files to create a python object. The package is documented at https://github.com/mido/mido/

Keras and Tensorflow for Neural Network model implementation

Beautiful Soup and Urllib for data set scraping.