CMPS 287 Project

Mirabelle Dib, Yousif Mansour

May 08, 2019

Abstract

In music, counterpoint is the relationship between voices that are harmonically interdependent (polyphony) yet independent in rhythm and contour. This project aims to produce a voice that is in counterpoint with an input voice. Using counterpuntal compositions of J.S. Bach, voices in a simillar style are produced.

An encoder-decoder model utilizing the LSTM cell is used to encode the input sequence of notes and produce an output sequence.

Introduction

The function that we want to learn looks like this:
$f(x_0, x_1, ..., x_{63}) = (y_0, y_1, ..., y_{63})$.
Each x_i and y_i is an interger between 0 and 11 (representing notes in an one octave).
This is a funciton that maps a sequence of a fixed length to another sequence of the same length, which is 64 here.
The 64 numbers represent a 1 bar phrase of musical notes. The function takes this representation and outputs another 1 bars that would sound fitting if played with the input.

This results in 2 voices being in counterpoint with each other.
Notes in the output sequence can reference notes that are before it in the sequence, and also all the notes from the input.

An encoder-decoder model is used to learn to solve this sequence to sequence problem. The encoder layer goes through the input sequence, updating its state for every note. This state is then fed into the decoder layer, which uses it as it's initial state and tries to predict the next note in the output sequence.

Using 1000 units and a batch size of 64 in training, the model performs well and produces intersting output sequences.

Dataset

The dataset used to train this model is built from the two collections of fugues (The Well Tempered Clavier I and II) by J.S. Bach.
The data for these fugues are provide in MIDI format online. Using python-midi, the midi files are parsed to produce an array of note values and positions.
From the midi files, we are able to read the melody lines of each voice. Then the voices are used to create the dataset (x, y).

A fugue on averge has 3 or 4 voices. They can be represetned as follows:

voice1: $[a_0] - [a_1] - [a_2] - ... - [a_{n-1}]$
voice2: $[b_0] - [b_1] - [b_2] - ... - [b_{n-1}]$
voice3: $[c_0] - [c_1] - [c_2] - ... - [c_{n-1}]$
voice4: $[d_0] - [d_1] - [d_2] - ... - [d_{n-1}]$

Each $[x]$ represents 1 bars of notes here, where each is an instance of 64 numbers in the range (0, 11) corresponding to the notes (C, C$^{\#}$, D, ... B) .
With this representation a fugue would like this:
voice1: $[a_0, a_1, ..., a_{63}] - [a_0, a_1, ..., a_{63}] - ...$
voice2: $[b_0, b_1, ..., b_{63}] - [b_0, b_1, ..., b_{63}] - ...$
voice3: $[c_0, c_1, ..., c_{63}] - [c_0, c_1, ..., c_{63}] - ...$
voice4: $[d_0, d_1, ..., d_{63}] - [d_0, d_1, ..., d_{63}] - ...$

Representing the notes this way, the $i'th$ value of the 64 values for each 1 bar phrase represents the note at that position.
For each instance, a START and END special symbol is added, making the length 64+2=66.

An (x, y) data instance therefore is the 64 values at the same position in two voices (which are in counterpoint), surrounded by START and END symbols.

After reaching this point, the size of the data set was 4522. To increase it, further sampling was done. For every pair of data points (x_i, y_i) and (x_{i+1}, y_{i+1}), new instances were created by taking the last 36 notes of (x_i, y_i) and the first 8 notes of (x_{i+1}, y_{i+1}), and then repeatedly doing the same 8 note shift to generate new data.

From $(a_0, a_1, ..., a_{63})$ and $(b_0, b_1, ..., b_{63})$ the following points are created:

$(a_8, a_9, ..., a_{63}, b_0, ..., b_7)$
$(a_{16}, a_{17}, ..., a_{63}, b_8, b_9..., b_{15})$
$(a_{32}, a_{17}, ..., a_{63}, b_{16}, b_{17}, ..., b_{31})$
And so on.

This increased the size of the dataset to 35724. The dataset is also then converted into one-hot vectors.

Model

In training, this is how things look like:

$[\text{START}, x_0, x_1, x_2, ..., x_{63}, \text{END}] \rightarrow \text{Encoder} \rightarrow [h, c, \text{output}]$
$[\text{START}, y_0, y_1, y_2, ..., y_{63}] \rightarrow \text{Decoder}(state = [h, c]) \rightarrow [h^{'}, c^{'}, y_0, y_1, y_2, ..., y_{63}, \text{END}]$

Here h and c are the state vectors of the LSTM, and the encoder output is not used.
The model is trained to learn to predict $(y_0, ..., y_{63})$ from the input $(x_0, ..., x_{63})$ and

the START symbol.

A set of weights is learned for the encoder and decoder layers.
This is simillar to the problem of translating from English to French, but here the sequence lengths are equal.

In prediction, we repeatedly feed the output note of the decoder as input to it to get the next note:

$[\text{START}, input_0, input_1, input_2, ..., input_{63}, \text{END}] \rightarrow \text{Encoder} \rightarrow [h, c, \text{output}]$
$[\text{START}] \rightarrow \text{Decoder}(state = [h, c]) \rightarrow [h^{'}, c^{'}, y_0]$
$[y_0] \rightarrow \text{Decoder}(state = [h^{'}, c^{'}]) \rightarrow [h^{''}, c^{''}, y_1]$
$[y_1] \rightarrow \text{Decoder}(state = [h^{''}, c^{''}]) \rightarrow [h^{'''}, c^{'''}, y_2]$
...
This repeats untill we get all 64 notes $[y_0, ..., y_{63}]$ or the END symbol.

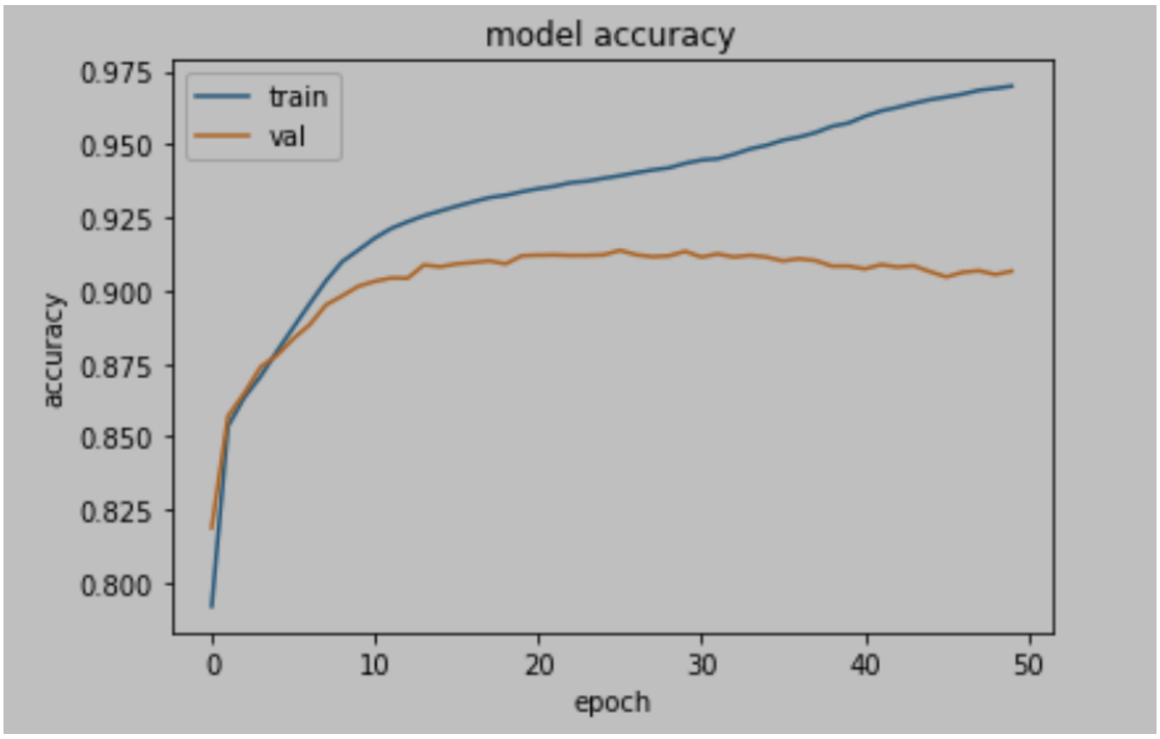
Categorical-crossentropy was used as a loss function and RMSprop was the optimizer used in the compilation of the model.

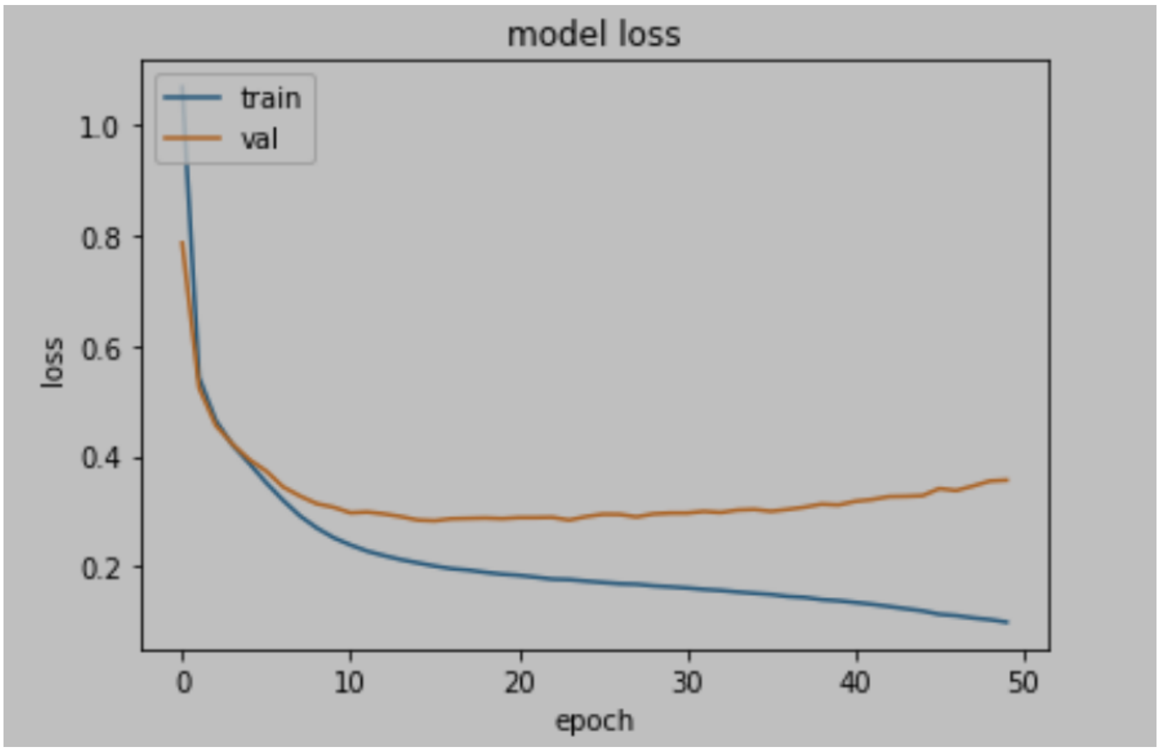
Results

Because the size of the dataset was only 35724 and the sequence length is 64 (which is relatively long), all notes were converted into one octave to reduce the size of the vocabulary. Also, it was not possible to have a very high degree of generalization. Training the encoder-decoder model with 250 and 500 units for the LSTM did not provide interesting results. Using 1000 units (around 8 million parameters), an accuracy of 90% was acheived on the test set.

The model would also perform poorly when trained with a batch size that is large (larger than 64), probably because of the samll dataset size. Further sampling of the dataset and using other compositions to increase the dataset size in training can possibly further improve the performence and quality of the output sequences, and make it possible to work with longer sequences.

The trained model outputs interesting sequences of notes. When given an input





that is very simple, having only 3 or 4 note values, it is more likely to output sequences with more than 4 notes in a pattern (although not always). The notes in the output sequences also tend to change with the notes in the input sequence at the same positions, showing that patterns in the input are being picked up by the network.

For an input that is very complex, the output is likely to have fewer than 4 notes. If the input size has a reasonable number of notes and complexity, the output also has a simillar number of notes. This reflects how the voices are related in Bach's fugues. It is also possible to produce very intersting passages by repeatedly feeding the output voice as an input to get another voice.

References:

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation
A ten-minute introduction to sequence-to-sequence learning in Keras
The 'Well- tempered Clavier', Book I, BWV 525-771
The 'Well- tempered Clavier', Book II, BWV 870-893
python-midi