Matthias Buehl Jan 19, 2019

I. Definition

Project Overview

The purpose of this project is to create a machine learning algorithm that produces a melody that are based on existing melodies. More specifically, the algorithm will predict a new note from a sequence of preceding notes. This single note prediction will be used repeatedly to construct a longer sequence of notes or melody.

Problem Statement

Composing music is exclusively a human activity. Composers are highly trained individuals and the craft takes decades to master. Nevertheless, composers tend to be strongly inspired by past works. This fact suggest the possibility that an algorithm could learn from existing works and could create an original piece.

Metrics

I will be predicting a note based on a preceding sequence of notes. Since this is a regression problem, I will be using the mean squared error as a metric.

II. Analysis

Data Exporation

The training data is taken from the music21 (http://web.mit.edu/music21/ (http://web.mit.edu/music21/)) python library. It contains a corpus of thousands of classical pieces.

For the purpose of this project, I used 2 pieces by Bach. I only chose the first part (instrument) of each piece if the piece had multiple parts (instruments). I took out all the timing information of the melody and retained only pitch information. Consequently the source material consists of a sequence of quarter notes.

I made an accidental discovery. By repeating the melody of each piece n times, my training and validation losses would decrease substantially more compared to using each melody once. Therefore, I repeated each of the melodies 30 times.

In [1]:
```
from music21 import *

from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, Activation
from keras.callbacks import ModelCheckpoint, EarlyStopping

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import decimal

from helpers import *
```

```
Using TensorFlow backend.
```

In [2]:

```
# get bach paths
paths = corpus.getComposer('bach')
#paths = corpus.getComposer('mozart')

# get n pieces
#paths = paths[:10]

# get all the notes of the first part of each piece
notes = []
midi_notes = []
# use 2 pieces
paths = paths[:2]
for path in paths:
    # repeat each piece 30 times
    for _ in range(30):
        compStream = converter.parse(path)
        # use only the first part (instrument) of the piece
        notes.extend(list(compStream.parts[0].recurse().notes))

# convert to midi notes
midi_notes = []
for n in notes:
    # handle chords -> get hightest pitch
    if isinstance(n, chord.Chord):
        midi_notes.append(n.pitches[-1].midi)
    else:
        midi_notes.append(n.pitch.midi)

print(f'len(notes): {len(notes)}')
print(f'len(midi_notes): {len(midi_notes)}')
print(midi_notes[:50])

# save a copy  of the original midi notes
midi_notes_org = midi_notes[:]

# write source melody to file
s, fp = to_midi_file(midi_notes_org[:500], 'source.midi')
```

```
len(notes): 5970
len(midi_notes): 5970
[65, 67, 60, 65, 53, 57, 53, 57, 60, 65, 70, 69, 67, 65, 67, 69, 69, 6
5, 67, 69, 70, 72, 70, 69, 67, 67, 72, 72, 71, 69, 71, 72, 65, 65, 70,
69, 67, 65, 60, 65, 60, 60, 57, 53, 60, 60, 60, 57, 65, 67]
```

In [3]:

```
# turn notes in to training sequences of n e.g. 4
# map each sequence to note that follows
# [65, 67, 60, 65] -> 53
# [67, 60, 65, 53] -> 57

# set the length of the sequence that predicts the next note
seq_length = 4

# scale data
midi_notes = np.array(midi_notes_org).reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
midi_notes = scaler.fit_transform(midi_notes)
# print(f'scaled midi_notes:\n{midi_notes}')

data = []
labels = []
for i in range(len(midi_notes)):
    seq_end = i + seq_length

    if seq_end >= len(midi_notes): break

    seq = midi_notes[i:seq_end]
    data.append(seq)
    labels.append(midi_notes[seq_end])

print(f'data:\n{data[0]}')
print(f'labels:\n{labels[0]}')
```

```
data:
[[0.5       ]
 [0.58333333]
 [0.29166667]
 [0.5       ]]
labels:
[0.]
```

```
/Users/mbuehl/.local/share/virtualenvs/ml-music-composition-7FO5GL04/li
b/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversi
onWarning: Data with input dtype int64 was converted to float64 by MinM
axScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [4]:
```
# get the data in shape
data = np.array(data, dtype=float)
data = data.reshape(-1, seq_length, 1)

labels = np.array(labels, dtype=float)
labels = labels.reshape(-1, 1)

# print(f'data.shape: {data.shape}')
# print(f'labels.shape: {labels.shape}')

# print(f'min(data): {min(data.reshape(-1, 1))}')
# print(f'max(data): {max(data.reshape(-1, 1))}')

# print(f'min(labels): {min(labels)}')
# print(f'max(labels): {max(labels)}')

# print('{seq} -> {labels[i]}')
# for i, seq in enumerate(data):
#     print(f'{seq} -> {labels[i]}')

#     if i == 3: break
```

Exploratory Visualization

In [5]:
```
print(f'Shape of training data: {data.shape}')
print(f'Shape of labels: {labels.shape}')
print('Pandas Stats:')
pd.DataFrame(midi_notes_org).describe()
```

```
Shape of training data: (5966, 4, 1)
Shape of labels: (5966, 1)
Pandas Stats:
```

Out[5]:

	0
count	5970.000000
mean	66.482412
std	5.653635
min	53.000000
25%	60.000000
50%	67.000000
75%	70.000000
max	77.000000

Algorithms and Techniques

After some internet research, I concluded that a neural network with LSTM layers would be a good candidate for predicting a singular output from sequential input.
https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/ (https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/)

Another candidate, the RNN, was disqualified since it suffers from the vanishing gradient problem.

Alternatively, I could have used multiclass classification with each possible pitch representing a class.

Benchmark

A clear benchmark for this project is tricky as the quality of a musical composition is highly subjective. Based on several setbacks during this project my benchmarks are as follows:

1. The resulting melody consists of more than 5 different pitches.
2. The resulting melody is not identical to the training melody.
3. The resulting melody does not appear completely random.

III. Methodology

Data Preporcessing

As stated under section exploratory visualization, the training data has shape (5966, 4, 1) and the labels have shape (5966, 1).
The data is normalized between 0 and 1 via a Min Max Scaler.

In [6]:
```
# get test and train data

x_train, x_test, y_train, y_test = train_test_split(data, labels, test_s
ize=0.2, random_state=4)
```

In [7]:
```
# build the model

model = Sequential()
model.add(LSTM((128), input_shape=(seq_length, 1), return_sequences=Fals
e))
model.add(Dense(1))
```

In [8]:
```
# compile the model

model.compile(loss='mse', optimizer='adam')
# print(model.input_shape)
# print(model.output_shape)
# print(model.summary())
```

Implementation

The model consists of a simple LSTM network with one LSTM layer with 128 outputs and one Dense layer with 1 output. The model predicts a note by evaluating the preceding n notes e.g. e.g. [C, D, E, F] -> G.

I use this single note prediction repeatedly to generate a longer melody be convolving forward by one note.

The 'composition' is kicked off by a sequence of notes that I select by hand.

Hand selected sequence: [C, D, E, F]

Predicted note: G

Next sequence (drop first note and append predicted [D, E, F, G]

Predicted note: C (may be)

Next sequence: [E, F, G, C]

and so on

In [9]:
```
print(f'Model Input Shape: {model.input_shape}')
print(f'Model Output Shape: {model.output_shape}')
print(f'Model Summary:')
print(model.summary())
```

```
Model Input Shape: (None, 4, 1)
Model Output Shape: (None, 1)
Model Summary:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 128)               66560
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 66,689
Trainable params: 66,689
Non-trainable params: 0
_________________________________________________________________
None
```

Refinements

I tried quite a few variations of LSTM, Dropout, and Dense layers. I was hoping that adding more LSTM layers would decrease the mse more than just a single LSTM layer. I realized through trial and error that one LSTM layer was just as effective as multiple and took much less time to train.

I also experimented with different activation layers including relu, tanh, and sigmoid. I concluded that the default activation layers were most effective:
LSTM -> activation='tanh', recurrent_activation='hard_sigmoid' Dense -> activation=None

In [10]:
```
# checkpoint
# use early stopping

filepath="weights-improvement-{epoch:02d}-{val_loss:.10f}.hdf5"
checkpoint = EarlyStopping(monitor='val_loss', mode='auto', patience=10,
 verbose=1, restore_best_weights=True )
# checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=0,
 save_best_only=True, save_weights_only=False, mode='auto')

callbacks_list = [checkpoint]
```

In [11]:
```
# fit the model

history = model.fit(x_train, y_train, verbose=0, epochs=1000, batch_size
=64, validation_data=(x_test, y_test), callbacks=callbacks_list)
```

```
Restoring model weights from the end of the best epoch
Epoch 00180: early stopping
```

IV. Results

Model Evaluation and Validation

The scope of evaluation and metric via mse and the loss curves pertains only to the prediction of the note that follows the previous n notes e.g. [C, D, E, F] -> G.

I use the prediction of a single note repeatedly to construct a longer melody. I am currently not evaluating the resulting melody via hard metrics.

Below are charts to evaluate the model.

In [12]:
```
# get the score

score = model.evaluate(x_test, y_test, batch_size=64)
print(f'Score (mse): {score}')
```

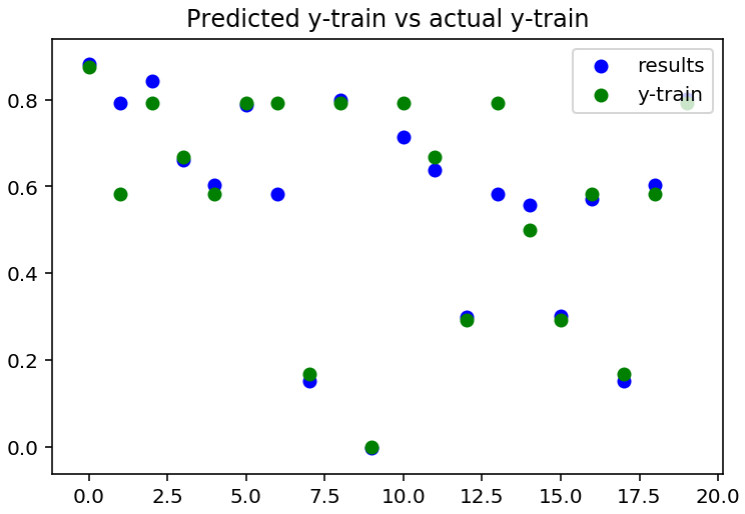
```
1194/1194 [==============================] - 0s 24us/step
Score (mse): 0.006821518428498876
```

In [13]:
```
# save model
#model.save('model.h5')
```

In [14]:
```
# predict y_train
n=20

results= model.predict(x_train)
plt.title('Predicted y-train vs actual y-train')
plt.scatter(range(n), results[:n], c='b')
plt.scatter(range(n), y_train[:n], c='g')
plt.legend(['results', 'y-train'], loc='upper right')
```

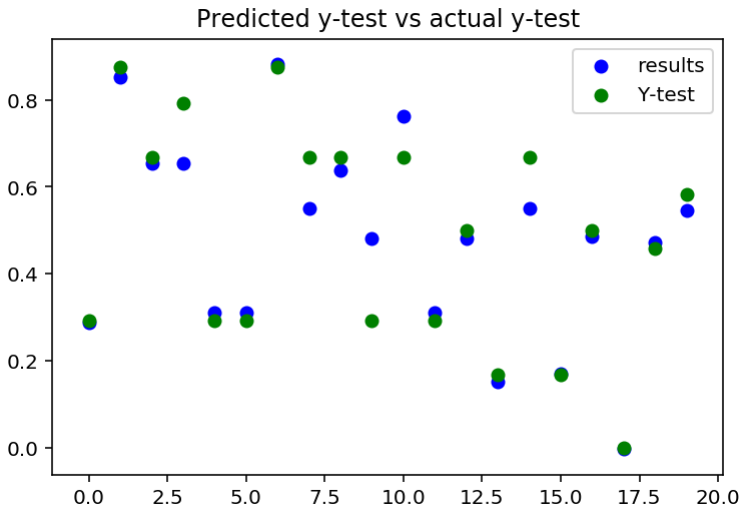
Out[14]:
```
<matplotlib.legend.Legend at 0x10a9ff518>
```



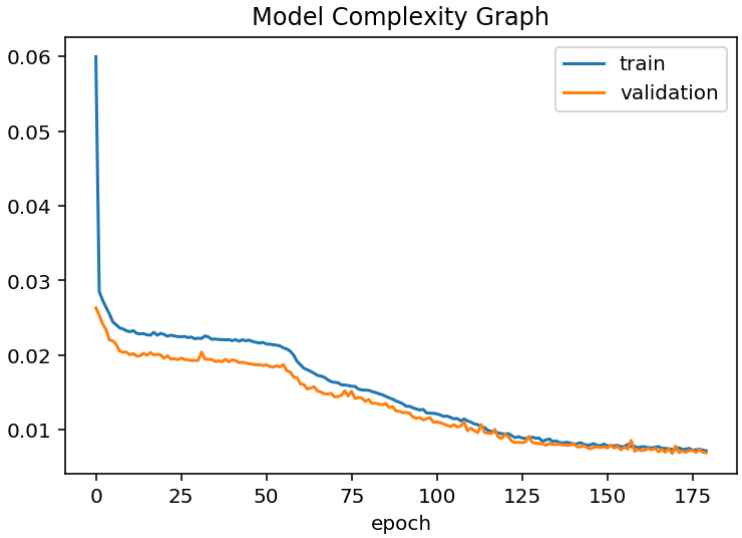
In [15]:
```
# predict y_test

results= model.predict(x_test)
plt.title('Predicted y-test vs actual y-test')
plt.scatter(range(n), results[:n], c='b')
plt.scatter(range(n), y_test[:n], c='g')
plt.legend(['results', 'Y-test'], loc='upper right')
```

Out[15]:
```
<matplotlib.legend.Legend at 0x10aa53320>
```



In [16]:
```
plt.title('Model Complexity Graph')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```



In [17]:
```
# load best weights

# model.load_weights('weights-improvement-57-0.0000455168.hdf5')

# score = model.evaluate(x_test, y_test, batch_size=64)
# print(score)
```

In [33]:
```
# create original melody

max_note = max(midi_notes)
min_note = min(midi_notes)
print(f'min_note: {min_note}')
print(f'max_note: {max_note}')

# get sequence to start things off
start_index = random.randrange(0, midi_notes.shape[0] - seq_length)
print(f'start_index: {start_index}')
seed_seq = midi_notes[start_index:start_index + seq_length]

# set start sequence manually
seed_seq = scaler.transform(np.array([60,67,65,64]).reshape(seq_length,
1))
#seed_seq = scaler.transform(np.array(midi_notes_org[:seq_length]).resha
pe(seq_length, 1))

# holds sequence for next prediction
cur_seq = seed_seq.copy()
cur_seq = cur_seq.reshape(1, seq_length, 1)
print(f'cur_seq.shape: {cur_seq.shape}')

# melody starts with our start sequence seed_seq
melody = cur_seq.copy()
print(f'melody.shape: {melody.shape}')

# melody is 500 notes long
melody_length = 500

# cur_seq moves along with the end of the melody to determine the next n
ote
for _ in range(melody_length):
#     print(f'cur_seq: {cur_seq}')
    next_note = model.predict(cur_seq)
#     print(f'next_note: {next_note}')
    melody = np.append(melody, next_note)
#     print(f'melody: {melody}')
    cur_seq = np.append(cur_seq[0][1:], next_note).reshape(1, seq_length
, 1)


#print(f'melody: {melody}')
```

```
min_note: [0.]
max_note: [1.]
start_index: 1262
cur_seq.shape: (1, 4, 1)
melody.shape: (1, 4, 1)
```

In [37]:
```
# turn melody into a midi file

# denormalize
melody_denorm = scaler.inverse_transform(melody.reshape(-1, 1)).flatten
()

# write new melody to file
s, fp = to_midi_file(melody_denorm, 'machine_bach.midi')
```

Justification

The final melody meets the criteria I established above:

1. The resulting melody consists of more than 5 different pitches.
2. The resulting melody is not identical to the training melody.
3. The resulting melody does not appear completely random.

Below is a visual comparison between the first 50 notes of the composed melody compared to the first 50 notes of the training melody. The corresponding midi files are in the repo:

training melody: source.midi composed melody: machine_bach.midi

V. Conclusion

Free-Form Visualization

Below is a visual comparison between the first 50 notes of the composed melody compared to the first 50 notes of the training melody. The corresponding midi files are in the repo:

training melody: source.midi composed melody: machine_bach.midi

In [42]:
```
new_melody_note_names = [n.pitch.nameWithOctave for n in s.notes]
print(f'new melody note names (first 50): {new_melody_note_names[:50]}')
print('\n')
train_melody_note_names = [n.pitch.nameWithOctave for n in notes]
print(f'train melody note names (first 50): {train_melody_note_names[:5
0]}')
```

```
new melody note names (first 50): ['C4', 'G4', 'F4', 'E4', 'F4', 'F4',
'G3', 'E-4', 'E4', 'F4', 'F4', 'D4', 'C4', 'C4', 'B-2', 'A2', 'G2', 'B-
2', 'C#4', 'C#5', 'C5', 'A4', 'A4', 'C5', 'C5', 'B-4', 'A4', 'B4', 'B
4', 'G4', 'F#4', 'F4', 'F#4', 'G#4', 'D4', 'C4', 'G#3', 'D4', 'A3', 'F
3', 'A3', 'F2', 'E4', 'C#4', 'G3', 'B-3', 'C#4', 'E3', 'D4', 'D4']


train melody note names (first 50): ['F4', 'G4', 'C4', 'F4', 'F3', 'A
3', 'F3', 'A3', 'C4', 'F4', 'B-4', 'A4', 'G4', 'F4', 'G4', 'A4', 'A4',
'F4', 'G4', 'A4', 'B-4', 'C5', 'B-4', 'A4', 'G4', 'G4', 'C5', 'C5', 'B
4', 'A4', 'B4', 'C5', 'F4', 'F4', 'B-4', 'A4', 'G4', 'F4', 'C4', 'F4',
'C4', 'C4', 'A3', 'F3', 'C4', 'C4', 'C4', 'A3', 'F4', 'G4']
```

Reflection

In this project, I created an algorithm that predicts a musical pitch based on 4 preceeding pitches. The pitch is produced by an LSTM network that is trained on 2 Bach pieces. The final melody was produced by using a seed of 4 pitches. The predicted pitch was used as the last pitch in the next input sequence while the first pitch was dropped. The 4 input pitches were convolving forward by one pitch 500 times.

I tried a variety of network architectures and activation functions. A simple network of one LSTM and one Dense layer proved to be most effective.

Initially, I trained the model using hundreds of songs using all of their parts (instruments). That always resulted in a melody that very quickly converged to one note.

After much trial and error I realized that I could create more diverse melodies by training of fewer songs. I could improve the melodies even more by repeating the melody of each training song several times in the training data. I am not sure why at this point.

Improvements

Keeping timing information to the data would make a significant difference. I pan on incorporating rhythm in my next iteration.

Predicting one pitch based on the 4 preceding it may not be the most musical thing to do. It may be more appropriate to use on measure to predict the next.

Currently, the machine learning only applies to the prediction of one pitch based on the 4 pitches that precede it. Consequently that final melody generated in this project is never compared to the source melody. Ideally, the entire melody would be evaluated against the training data and incrementally improved. May be actor/critic would be appropriate.