```
# pip install music21
# Commented out.. as if run again, doesn't try to install

from music21 import converter, instrument, note, chord, stream
import glob
import pickle
import numpy as np
```

Read a Midi File

```
song1 = converter.parse("midi_songs/8.mid")
print(type(song1))
```

```
<class 'music21.stream.base.Score'>
```

```
song1
```

```
<music21.stream.Score 0x2a00c0d3100>
```

```
# song1 --> object of stream.Score type
#       --> will contain music in form of notes and chords
song1.show('midi')
# This will show the song in playable format
```

```
<IPython.core.display.HTML object>
```

```
# song1.show('text')
# This will show the song in text-format (notes & chords)

# So, the chords and notes are stored in nested forms of containers
# .. to simplify this, store all of them in a single list
# ==> Flatten the elements.
elements_of_song = song1.flat.notes
```

```
print(len(elements_of_song))
print(elements_of_song)
print(type(elements_of_song))
```

```
336
<music21.stream.iterator.StreamIterator for Score:0x2a00cc28a60 @:0>
<class 'music21.stream.iterator.StreamIterator'>
```

```
count = 0
print("Following are some elements of song :-")
for e in elements_of_song:
    print(e, e.offset, type(e))
    count += 1
    if count > 15:
        break
    # e.offset --> will tell the time-duration of element
```

```
Following are some elements of song :-
<music21.note.Note C> 0.0 <class 'music21.note.Note'>
<music21.chord.Chord C5 E4> 0.0 <class 'music21.chord.Chord'>
```

```
<music21.note.Note C> 0.0 <class 'music21.note.Note'>
<music21.note.Note C> 0.0 <class 'music21.note.Note'>
<music21.chord.Chord C5 E4> 0.0 <class 'music21.chord.Chord'>
<music21.note.Note C> 0.0 <class 'music21.note.Note'>
<music21.note.Note G> 1.5 <class 'music21.note.Note'>
<music21.note.Note G> 5/3 <class 'music21.note.Note'>
<music21.note.Note C> 1.75 <class 'music21.note.Note'>
<music21.note.Note C> 1.75 <class 'music21.note.Note'>
<music21.note.Note D> 2.0 <class 'music21.note.Note'>
<music21.chord.Chord D4 B-4> 2.0 <class 'music21.chord.Chord'>
<music21.note.Note B-> 2.0 <class 'music21.note.Note'>
<music21.note.Note D> 2.0 <class 'music21.note.Note'>
<music21.chord.Chord D4 B-4> 2.0 <class 'music21.chord.Chord'>
<music21.note.Note B-> 2.0 <class 'music21.note.Note'>
```

Get the Notes & Chords from the Song

```
elex = elements_of_song[0]
ele2 = elements_of_song[4]
# isinstance(element, classType)
# If the element and its class match with classType --> this returns
True (else False)
flag1a = isinstance(elex, note.Note)
flag1b = isinstance(elex, chord.Chord)
flag2a = isinstance(ele2, note.Note)
flag2b = isinstance(ele2, chord.Chord)
print(flag1a, flag1b, flag2a, flag2b)
```

```
True False False True
```

Processing a Note :-

```
note1 = elements_of_song[3]
print(note1.pitch)
print(type(note1))
# This gives the note in form of a class
print(type(note1.pitch))
# Get the string from the class
currNote = str(note1.pitch)
print(currNote)
# This will recover the note-name from class
```

```
C5
<class 'music21.note.Note'>
<class 'music21.pitch.Pitch'>
C5
```

Processing a Chord :-

```
chord1 = elements_of_song[1]
print(chord1)
print(type(chord1))
# This is a chord, let's figure this out.. how to process this
print(chord1.normalOrder)
```

```
# chord.normalOrder --> Gives the list of nodes in it.
# 2 --> A4
# 6 --> D5
# 9 --> F#4
# (Following some pattern of indexing.. have to figure it out)
print(type(chord1.normalOrder))
# Convert the chord-list into a string, concatenated with "+"
currChord = "+".join(str(x) for x in chord1.normalOrder)
print(currChord)
```

```
<music21.chord.Chord C5 E4>
<class 'music21.chord.Chord'>
[0, 4]
<class 'list'>
0+4
```

Making a list, only of Notes (from Notes) OR (from Chords)

```
notes_of_song = []
# Empty array container for notes & chords

for ele in elements_of_song:
    # If element is a note, store it's pitch
    if(isinstance(ele, note.Note) == True):
        tempNote = str(ele.pitch)
        notes_of_song.append(tempNote)
    elif(isinstance(ele, chord.Chord) == True):
    # Else, element is a chord, split notes, and make string of them
        tempChord = "+".join(str(x) for x in ele.normalOrder)
        notes_of_song.append(tempChord)

print("No. of notes/chords =", len(notes_of_song))
print(notes_of_song)
# for note1 in notes_of_song:
#     print(note1)
```

```
No. of notes/chords = 336
['C5', '0+4', 'C2', 'C5', '0+4', 'C2', 'G4', 'G4', 'C5', 'C5', 'D5',
'10+2', 'B-1', 'D5', '10+2', 'B-1', 'F4', 'F4', 'B-4', 'B-4', 'F5',
'9+0', 'F1', 'F5', '9+0', 'F1', '9+0', 'F1', '9+0', 'F1', '7+11+2',
'G1', '7+11+2', 'G1', 'E5', 'C5', 'C2', 'E5', 'C5', 'C2', 'C2', 'C2',
'7', '7', '0', '0', '2', 'G1', '2', '0', 'G1', '7+10', 'G4', '7+10',
'G4', 'G4', 'B-4', 'G4', 'B-4', 'G1', 'G1', 'G4', 'C5', 'C2', 'G4',
'C5', 'C2', 'C2', 'C2', '7+9', '7+9', '0', '0', '10', 'G1', '10', '0',
'G1', '10+2', '10+2', '0', '0', 'F4', 'D5', 'F4', 'D5', 'G1', 'G1',
'G4', 'E5', 'C2', 'G4', 'E5', 'C2', 'C2', 'C2', '2+7', '2+7', '4+9',
'4+9', '5+9', 'G1', '5+9', '4+9', 'G1', '2+5', '2+5', 'F4', 'D5',
'F4', 'D5', '10', 'G1', '10', 'G1', '5+10', '5+10', 'B-4', 'F5', 'B-
4', 'F5', 'A4', 'E5', 'C2', 'A4', 'E5', 'C2', 'C2', 'C2', '2+7',
'2+7', '4+9', '4+9', '5+10', 'G1', '5+10', '4+9', 'G1', '2+7', '2+7',
'5+10', '5+10', 'D5', 'B-5', 'D5', 'B-5', 'G1', 'G1', 'C5', 'G4',
'C2', 'C5', 'G4', 'C2', 'G5', 'C5', 'G5', 'C5', 'C2', 'C2', '7+0',
```

```
'7+0', '10', 'G1', '10', 'G1', 'A4', 'A4', 'D5', 'D5', 'G4', 'G1',
'G4', 'G1', '5+10', '5+10', 'C5', 'E4', 'C2', 'C5', 'E4', 'C2', 'E4',
'E4', 'F4', 'F4', 'G4', 'G4', 'G5', 'G5', 'C2', 'C2', 'G4', 'G4',
'C5', 'C5', 'C5', 'C5', 'G4', 'G4', '10', 'G1', '10', 'G4', 'G1',
'A4', 'A4', 'D5', 'D5', 'G4', 'G1', 'G4', 'G1', 'B-4', 'E4', 'B-4',
'E4', 'F4', 'F4', 'C5', 'G4', 'C2', 'C5', 'G4', 'F4', 'C2', 'G5',
'C5', 'G5', 'C5', 'C2', 'C2', '7+0', '7+0', '10', 'G1', '10', 'G1',
'A4', 'A4', 'D5', 'D5', 'G4', 'G1', 'G4', 'G1', '5+10', '5+10', 'E4',
'E4', 'C2', 'E4', 'E4', 'C2', 'E4', 'E4', 'F4', 'F4', 'G4', 'G4', '7',
'C2', '7', 'C2', 'G4', 'G4', 'A4', 'A4', 'B-4', 'B-4', 'C5', 'C5',
'G1', 'C5', 'C5', 'G1', 'C5', 'C5', 'D5', 'D5', 'E5', 'E5', 'C5',
'G1', 'C5', 'G1', 'B-4', 'B-4', 'G5', '0+4', 'C2', 'G5', '0+4', 'C2',
'G4', 'G4', 'C5', 'C5', 'F5', '10+2', 'B-1', 'F5', '10+2', 'B-1',
'F4', 'F4', 'B-4', 'B-4', '0+5', 'A4', '9+0', 'F1', '0+5', 'A4',
'9+0', 'F1', '5+9+0', 'A4', 'F1', '5+9+0', 'A4', 'F1', '7+11+2', 'G1',
'7+11+2', 'G1', 'C6', '0+4+7', '0+4+7', '7+0', '0', 'C6', '0+4+7',
'0+4+7', '7+0', '0']
```

Get All the Notes, from all the Midi Files

```
# import glob
# from pathlib import Path

# input_dir = Path.cwd()
# files = list(input_dir.rglob("*.mid"))

# notes = []
# for file in files:
#     song = converter.parse(file)
#     # Convert file into stream.Score object
#     # ..which just contains notes/chords
#     print("parsing", file)
#     file = file.resolve()
#     print(type(file))
#     elements_of_song = song.flat.notes
#     for ele in elements_of_song:
#         # If element is a note, store it's pitch
#         if(isinstance(ele, note.Note) == True):
#             tempNote = str(ele.pitch)
#             notes_of_song.append(tempNote)
#         elif(isinstance(ele, chord.Chord) == True):
#         # Else, element is a chord, split notes, and make string of
them
#             tempChord = "+".join(str(x) for x in ele.normalOrder)
#             notes_of_song.append(tempChord)

notes = []

count = 0
for file in glob.glob("midi_songs/*.mid"):
    midi = converter.parse(file) # Convert file into stream.Score
```

```
Object
    if count < 15:
        print("parsing %s"%file)
    elements_to_parse = midi.flat.notes
    count += 1

    for elex in elements_to_parse:
        # If the element is a Note,  then store it's pitch
        if(isinstance(elex, chord.Chord) == True):
            notes.append("+".join(str(n) for n in elex.normalOrder))
        elif(isinstance(elex, note.Note) == True):
            noteString = str(elex.pitch)
            notes.append(noteString)
            # If the element is a Chord, split each note of chord and
join them with +
```

```
parsing midi_songs\0fithos.mid
parsing midi_songs\8.mid
parsing midi_songs\ahead_on_our_way_piano.mid
parsing midi_songs\AT.mid
parsing midi_songs\balamb.mid
parsing midi_songs\bcm.mid
parsing midi_songs\BlueStone_LastDungeon.mid
parsing midi_songs\braska.mid
parsing midi_songs\caitsith.mid
parsing midi_songs\Cids.mid
parsing midi_songs\cosmo.mid
parsing midi_songs\costadsol.mid
parsing midi_songs\dayafter.mid
parsing midi_songs\decisive.mid
parsing midi_songs\dontbeafraid.mid
```

```
print("Length of notes-array = ", len(notes))
print("Some elements of note array :-")
count = 0
for n in notes:
    print(n)
    count += 1
    if count > 10:
        break
```

```
Length of notes-array =  60764
Some elements of note array :-
4+9
E2
4+9
4+9
4+9
4+9
4+9
4+9
```

```
4+9
11+4
4+9
```

Saving the file, containing all Notes

```
import pickle

with open("notes", 'wb') as filepath:
    pickle.dump(notes, filepath)

# 'wb' --> Write-binary mode (to write data in a file)
# 'rb' --> Read-binary mode (to read data from a file)

with open("notes", 'rb') as f:
    notes = pickle.load(f)
    # This will load whole file-data to variable notes

# print(notes[100:200])
# Viewing a sliced part of dataset
```

Count of Unique Elements in Music :-

```
# In 'wb' and 'rb', same file needs to be referenced.
# Else, Will give error --> "Ran out of data".
print(len(set(notes)))
# This will print unique no. of elements.
# i.e. --> Unique notes/chords in all files.
numElements = len(set(notes))
```

```
398
```

```
# print(notes[:100])
```

Preparing Sequenctial Data for LSTM :-

In Markov chain, we have a window size. So choosing a sequence length. This length also states, how many elements are considered in a LSTM layer.

```
sequenceLength = 100
# Will give 100 elements to a layer, and will predict output for next
layer using them.

uniqueNotes = sorted(set(notes))
print(uniqueNotes)
countNodes = len(uniqueNotes)
print(len(uniqueNotes))
```

```
['0', '0+1', '0+1+3', '0+1+5', '0+1+6', '0+2', '0+2+3+7', '0+2+4+5',
'0+2+4+7', '0+2+5', '0+2+6', '0+2+7', '0+3', '0+3+5', '0+3+5+8',
'0+3+6', '0+3+6+8', '0+3+6+9', '0+3+7', '0+4', '0+4+5', '0+4+6',
'0+4+7', '0+5', '0+6', '1', '1+2', '1+2+4+6', '1+2+4+6+8+10', '1+2+6',
'1+2+6+9', '1+3', '1+3+4+8', '1+3+5', '1+3+5+8', '1+3+6', '1+3+7',
```

'1+3+8', '1+4', '1+4+6', '1+4+6+8', '1+4+6+9', '1+4+7', '1+4+7+10',
'1+4+7+9', '1+4+8', '1+5', '1+5+8', '1+5+9', '1+6', '1+7', '10',
'10+0', '10+0+2', '10+0+2+5', '10+0+3', '10+0+4', '10+0+5', '10+1',
'10+1+3', '10+1+3+4+6', '10+1+3+5+6', '10+1+3+6', '10+1+4',
'10+1+4+6', '10+1+5', '10+11', '10+11+1', '10+11+1+3',
'10+11+1+3+4+6', '10+11+1+4+6', '10+11+3', '10+11+3+4', '10+11+3+5',
'10+2', '10+2+3', '10+2+4', '10+2+5', '10+3', '11', '11+0', '11+0+2',
'11+0+2+4', '11+0+4', '11+0+4+6', '11+0+4+7', '11+0+5', '11+1',
'11+1+2+4', '11+1+3', '11+1+4', '11+1+4+5', '11+1+5', '11+1+6',
'11+2', '11+2+4', '11+2+4+6', '11+2+4+7', '11+2+5', '11+2+5+6',
'11+2+5+7', '11+2+6', '11+3', '11+3+5', '11+3+6', '11+4', '11+4+5',
'2', '2+3', '2+3+7', '2+3+7+10', '2+3+7+9', '2+4', '2+4+5', '2+4+5+9',
'2+4+6+7', '2+4+6+9+11', '2+4+7', '2+4+7+10', '2+4+8', '2+4+9', '2+5',
'2+5+7', '2+5+7+10', '2+5+7+9', '2+5+8', '2+5+8+10', '2+5+8+11',
'2+5+9', '2+6', '2+6+10', '2+6+7', '2+6+7+9', '2+6+8', '2+6+9', '2+7',
'2+7+8', '2+8', '3', '3+4', '3+4+6', '3+4+8', '3+4+8+10', '3+4+8+11',
'3+5', '3+5+10', '3+5+7', '3+5+7+10', '3+5+7+8+11', '3+5+8',
'3+5+8+11', '3+6', '3+6+10', '3+6+7', '3+6+8', '3+6+8+11', '3+6+9+11',
'3+7', '3+7+10', '3+7+11', '3+7+9', '3+7+9+10', '3+8', '3+8+9', '3+9',
'4', '4+10', '4+5', '4+5+7+11', '4+5+7+9', '4+5+9', '4+5+9+0',
'4+5+9+11', '4+6', '4+6+10', '4+6+11', '4+6+7', '4+6+7+9', '4+6+8',
'4+6+8+10+1', '4+6+8+11+1', '4+6+9', '4+6+9+11', '4+7', '4+7+10',
'4+7+11', '4+7+9', '4+7+9+0', '4+7+9+11', '4+8', '4+8+10', '4+8+11',
'4+8+9', '4+8+9+11', '4+9', '4+9+10', '5', '5+10', '5+11', '5+6',
'5+6+10+0', '5+6+8', '5+7', '5+7+0', '5+7+10', '5+7+8+0', '5+7+9+0',
'5+7+9+11', '5+8', '5+8+0', '5+8+10', '5+8+11', '5+9', '5+9+0',
'5+9+11', '5+9+11+0', '6', '6+10', '6+10+0', '6+10+1', '6+10+11',
'6+10+11+1', '6+11', '6+11+0', '6+7', '6+7+0', '6+7+11', '6+7+11+2',
'6+7+9+11', '6+8', '6+8+0', '6+8+1', '6+8+10', '6+8+10+0+3',
'6+8+10+1', '6+8+11', '6+8+11+2', '6+9', '6+9+0', '6+9+0+2', '6+9+1',
'6+9+11', '6+9+11+1', '6+9+11+2', '7', '7+0', '7+10', '7+10+0',
'7+10+0+3', '7+10+1', '7+10+1+3', '7+10+2', '7+11', '7+11+0',
'7+11+2', '7+8', '7+8+0', '7+8+0+3', '7+8+10+2', '7+8+11', '7+9',
'7+9+0', '7+9+1', '7+9+11', '7+9+11+0', '7+9+11+1', '7+9+11+2',
'7+9+2', '8', '8+0', '8+0+1', '8+0+2', '8+0+3', '8+1', '8+10',
'8+10+0', '8+10+1', '8+10+1+3', '8+10+1+4', '8+10+11', '8+10+11+1',
'8+10+11+1+4', '8+10+11+3', '8+10+2', '8+10+3', '8+11', '8+11+1',
'8+11+1+3+4', '8+11+1+4', '8+11+2', '8+11+3', '8+9', '8+9+1',
'8+9+1+3', '8+9+1+4', '8+9+11+1+3', '8+9+2', '9', '9+0', '9+0+2',
'9+0+2+4', '9+0+2+5', '9+0+3', '9+0+3+5', '9+0+4', '9+1', '9+1+2',
'9+1+4', '9+10', '9+10+2', '9+10+2+4', '9+10+3', '9+11', '9+11+0+2',
'9+11+0+4', '9+11+1', '9+11+1+2', '9+11+2', '9+11+2+4', '9+11+2+5',
'9+11+3', '9+11+4', '9+2', '9+2+3', 'A1', 'A2', 'A3', 'A4', 'A5',
'A6', 'B-1', 'B-2', 'B-3', 'B-4', 'B-5', 'B-6', 'B1', 'B2', 'B3',
'B4', 'B5', 'B6', 'C#1', 'C#2', 'C#3', 'C#4', 'C#5', 'C#6', 'C#7',
'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'D1', 'D2', 'D3', 'D4', 'D5',
'D6', 'D7', 'E-1', 'E-2', 'E-3', 'E-4', 'E-5', 'E-6', 'E1', 'E2',
'E3', 'E4', 'E5', 'E6', 'F#1', 'F#2', 'F#3', 'F#4', 'F#5', 'F#6',
'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'G#1', 'G#2', 'G#3', 'G#4', 'G#5',

```
'G#6', 'G1', 'G2', 'G3', 'G4', 'G5', 'G6']
398
```

Mapping Strings (unique-elements) to Integer values :-

```
# As ML models work with numerial data only, will map each string with
a number.
noteMap = dict((ele, num) for num, ele in enumerate(uniqueNotes))

count = 0
for ele in noteMap:
    print(ele, " : ", noteMap[ele])
    count += 1
    if count > 10:
        break
```

```
0  :  0
0+1  :  1
0+1+3  :  2
0+1+5  :  3
0+1+6  :  4
0+2  :  5
0+2+3+7  :  6
0+2+4+5  :  7
0+2+4+7  :  8
0+2+5  :  9
0+2+6  :  10
```

--> As sequenceLength is 100, will take first 100 data to input, and 101st data as output. --> For next iteration, take (2-101) data points as input, and 102nd data as output. --> So on... Sliding window (of size 100) as input, & next 1 data as output.

--> So, total we will get (len(notes) - sequenceLength) datapoints.

```
networkInput = [] # input-data
networkOutput = [] # will try to get output, using input

for i in range(len(notes) - sequenceLength):
    inputSeq = notes[i : i+sequenceLength] # 100 string-values
    outputSeq = notes[i + sequenceLength] # 1 string-value
    # Currently, inputSeq & outputSeq has strings.
    # Use map, to convert it to integer-values.
    # ..as ML-algorithm works only on numerical data.
    networkInput.append([noteMap[ch] for ch in inputSeq])
    networkOutput.append(noteMap[outputSeq])

    # for tempStr in inputSeq:
    #     tempNum = noteMap[tempStr]
    #     # got no. from string, using note-map
    #     networkInput.append(tempNum)
    # for temp in outputSeq:
    #     tempNum = noteMap[tempStr]
```

```
    #     # got no. from string, using note-map
    #     networkOutput.append(tempNum)

print(len(networkInput))
print(len(networkOutput))
```

```
60664
60664
```

Create ready-data for Neural Network :-

```
import numpy as np

# n_patterns = int(len(networkInput)/100)
# No. of rows divided by 100.. as 100 columns, so Distributing data in
3-D format

n_patterns = len(networkInput)

networkInput = np.reshape(networkInput, (n_patterns, sequenceLength,
1))
# LSTM recieves input data in 3-dimensions
print(networkInput.shape)
```

```
(60664, 100, 1)
```

Normalize this data

```
# As the values are from 0 - uniqueNodes
# For better precision, converting data in range [0 - 1]

normNetworkInput = networkInput / float(numElements)

print("Some elements of normNetworkInput[0] array :-")
count = 0
for ele in normNetworkInput[0]:
    print(ele)
    count += 1
    if count > 10:
        break
```

```
Some elements of normNetworkInput[0] array :-
[0.48743719]
[0.92713568]
[0.48743719]
[0.48743719]
[0.48743719]
[0.48743719]
[0.48743719]
[0.48743719]
[0.48743719]
[0.2638191]
[0.48743719]
```

```
# normNetworkInput
# Now, values are in range [0 - 1]

# Network output are the classes, encoded into 1-vector

from keras.utils import np_utils

print("Some elements of networkOutput array :-")
count = 0
for ele in networkOutput:
    print(ele)
    count += 1
    if count > 10:
        break
```

```
Some elements of networkOutput array :-
382
381
381
381
381
381
194
372
194
352
194
```

```
networkOutput = np_utils.to_categorical(networkOutput)
print(networkOutput.shape)

# This will convert output-data to a 2-D format
# In which each key(old-output value) has 229 categorical values
# And, the one which matches has some kind of flag marked to it.
```

```
(60664, 398)
```

Create Model

Download & Import Packages

```
from keras.models import Sequential
from keras.layers import *
from keras.callbacks import ModelCheckpoint, EarlyStopping

# pip install keras
# pip install tensorflow

# import tensorflow as tf
# just to check if tensorflow is working..
```

Creating a Sequential Model :-

```
model = Sequential()
```

Adding Layers to the Model :-

```
# And, this model has first layer as LSTM layer.
model.add(LSTM(units=512, input_shape=(normNetworkInput.shape[1],
normNetworkInput.shape[2]), return_sequences=True))
# As this is the 1st layer, so we need to provide the input-shape (in
argument)
# Here we are passing (100,1) as input_shape, as all data-points have
shape (100,1)
# Also, we have to do return_sequences=True, as this isn't the last
layer, also have further layers.


# After the 1st layer, adding a Dropout
model.add(Dropout(0.3))

# Also adding another LSTM layer.
model.add(LSTM(512, return_sequences=True))
# as this is also not the last layer.. return_sequences=True

# Again adding a Dropout
model.add(Dropout(0.3))

# And, now 1-more LSTM layer.
model.add(LSTM(512))

# And, adding a Dense-layer.
model.add(Dense(256))

# Again adding a Dropout.
model.add(Dropout(0.3))

# Now, the final layer.
#   (Adding dense layer with no. of neurons = countNodes)
#   (Also having an "softmax" activation function)
model.add(Dense(numElements, activation="softmax"))
```

Compiling the model :-

```
model.compile(loss="categorical_crossentropy", optimizer="adam")
# loss="categorical_crossentropy" --> since it has 229 classes.
# Not specifying any metrics (like accuracy), as it would not be a
good metrics to evaluate.
```

This is our Model :-

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

```
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm (LSTM)                  (None, 100, 512)          1052672

 dropout (Dropout)            (None, 100, 512)          0

 lstm_1 (LSTM)                (None, 100, 512)          2099200

 dropout_1 (Dropout)          (None, 100, 512)          0

 lstm_2 (LSTM)                (None, 512)               2099200

 dense (Dense)                (None, 256)               131328

 dropout_2 (Dropout)          (None, 256)               0

 dense_1 (Dense)              (None, 398)               102286

=================================================================
Total params: 5,484,686
Trainable params: 5,484,686
Non-trainable params: 0
_________________________________________________________________
```

Training the Model :-

```
import tensorflow as tf

# (Entire code commented out, to prevent created model, from starting
fit again, and old work getting wasted)

# Creating callbacks for fitting model.

# checkpoint = ModelCheckpoint("model3.hdf5", monitor='loss',
verbose=0, save_best_only=True, mode='min')
# 1st arg  -->  where the model will be saved
# 2nd arg  -->  We have to monitor the loss
# 5th arg  -->  As monitoring loss, so mode = "min", as loss should be
minimum.

# We can also create an earlystopping callback, but lets only keep the
checkpoint.

# Fitting the model :-

# normNetworkInput = np.array(normNetworkInput)
# networkOutput = np.array(networkOutput)

# model_his = model.fit(normNetworkInput, networkOutput, epochs=10,
batch_size=64, callbacks=[checkpoint])
```

```
# No. of epochs = 10 (trying for model3)
# batch size = 64
```

```
Epoch 1/10
948/948 [==============================] - 4003s 4s/step - loss:
4.8030
Epoch 2/10
948/948 [==============================] - 3837s 4s/step - loss:
4.7745
Epoch 3/10
948/948 [==============================] - 3842s 4s/step - loss:
4.7721
Epoch 4/10
948/948 [==============================] - 3839s 4s/step - loss:
4.7687
Epoch 5/10
948/948 [==============================] - 12148s 13s/step - loss:
4.7669
Epoch 6/10
948/948 [==============================] - 3739s 4s/step - loss:
4.7658
Epoch 7/10
948/948 [==============================] - 3994s 4s/step - loss:
4.7650
Epoch 8/10
948/948 [==============================] - 3497s 4s/step - loss:
4.7646
Epoch 9/10
948/948 [==============================] - 3606s 4s/step - loss:
4.7646
Epoch 10/10
948/948 [==============================] - 3747s 4s/step - loss:
4.7644
```

Load Model :-

```
from keras.models import load_model

model = load_model("model4.hdf5")
```

Predictions :-

```
sequenceLength = 100

networkInput = [] # input-data

for i in range(len(notes) - sequenceLength):
    inputSeq = notes[i : i+sequenceLength] # 100 string-values
    # Currently, inputSeq & outputSeq has strings.
    # Use map, to convert it to integer-values.
    # ..as ML-algorithm works only on numerical data.
    networkInput.append([noteMap[ch] for ch in inputSeq])
```

```
print("Some elements of networkInput[0] array :-")
count = 0
for ele in networkInput[0]:
    print(ele)
    count += 1
    if count > 10:
        break
```

```
Some elements of networkInput[0] array :-
194
369
194
194
194
194
194
194
194
105
194
```

```
print(len(networkInput[300]))
```

```
100
```

```
# Each data-point has 100-elements (in networkInput)
# We will give these 100-elements as input, & it will generate 1-
output.
# Will add this 1-output in input, & discard oldest element from
input. (again getting to 100 input-elements)

# This way, we will keep predicting 1-element each time.

startIdx = np.random.randint(len(networkInput)-1)
# This will get any random data-point-index from the input-data
# Data at each random data-point-index means --> 100 elements.

print(startIdx)
```

```
23047
```

```
networkInput[startIdx]
```

```
print("Some elements of networkInput[startIdx] array are :-")
count = 0
for ele in networkInput[startIdx]:
    print(ele)
    count += 1
    if count > 10:
        break
```

```
Some elements of networkInput[startIdx] array are :-
107
0
79
297
97
165
97
97
187
107
0
```

```
# Above 100-element np-array, is the start sequence.

# Right now, we have :-
# element --> integer  mapping

# What is also required is :-
# integer --> element  mapping.

intNoteMap = dict((num,ele) for num,ele in enumerate(uniqueNotes))
# This will have  (integer --> element) mapping.

# uniqueNotes --> has all unique-elements
# noteMap --> has  (element --> integer) mapping.
# countNodes --> count of unique-elements.
print(intNoteMap)
```

```
{0: '0', 1: '0+1', 2: '0+1+3', 3: '0+1+5', 4: '0+1+6', 5: '0+2', 6:
'0+2+3+7', 7: '0+2+4+5', 8: '0+2+4+7', 9: '0+2+5', 10: '0+2+6', 11:
'0+2+7', 12: '0+3', 13: '0+3+5', 14: '0+3+5+8', 15: '0+3+6', 16:
'0+3+6+8', 17: '0+3+6+9', 18: '0+3+7', 19: '0+4', 20: '0+4+5', 21:
'0+4+6', 22: '0+4+7', 23: '0+5', 24: '0+6', 25: '1', 26: '1+2', 27:
'1+2+4+6', 28: '1+2+4+6+8+10', 29: '1+2+6', 30: '1+2+6+9', 31: '1+3',
32: '1+3+4+8', 33: '1+3+5', 34: '1+3+5+8', 35: '1+3+6', 36: '1+3+7',
37: '1+3+8', 38: '1+4', 39: '1+4+6', 40: '1+4+6+8', 41: '1+4+6+9', 42:
'1+4+7', 43: '1+4+7+10', 44: '1+4+7+9', 45: '1+4+8', 46: '1+5', 47:
'1+5+8', 48: '1+5+9', 49: '1+6', 50: '1+7', 51: '10', 52: '10+0', 53:
'10+0+2', 54: '10+0+2+5', 55: '10+0+3', 56: '10+0+4', 57: '10+0+5',
58: '10+1', 59: '10+1+3', 60: '10+1+3+4+6', 61: '10+1+3+5+6', 62:
'10+1+3+6', 63: '10+1+4', 64: '10+1+4+6', 65: '10+1+5', 66: '10+11',
67: '10+11+1', 68: '10+11+1+3', 69: '10+11+1+3+4+6', 70:
'10+11+1+4+6', 71: '10+11+3', 72: '10+11+3+4', 73: '10+11+3+5', 74:
'10+2', 75: '10+2+3', 76: '10+2+4', 77: '10+2+5', 78: '10+3', 79:
'11', 80: '11+0', 81: '11+0+2', 82: '11+0+2+4', 83: '11+0+4', 84:
'11+0+4+6', 85: '11+0+4+7', 86: '11+0+5', 87: '11+1', 88: '11+1+2+4',
89: '11+1+3', 90: '11+1+4', 91: '11+1+4+5', 92: '11+1+5', 93:
'11+1+6', 94: '11+2', 95: '11+2+4', 96: '11+2+4+6', 97: '11+2+4+7',
98: '11+2+5', 99: '11+2+5+6', 100: '11+2+5+7', 101: '11+2+6', 102:
```

'11+3', 103: '11+3+5', 104: '11+3+6', 105: '11+4', 106: '11+4+5', 107:
'2', 108: '2+3', 109: '2+3+7', 110: '2+3+7+10', 111: '2+3+7+9', 112:
'2+4', 113: '2+4+5', 114: '2+4+5+9', 115: '2+4+6+7', 116:
'2+4+6+9+11', 117: '2+4+7', 118: '2+4+7+10', 119: '2+4+8', 120:
'2+4+9', 121: '2+5', 122: '2+5+7', 123: '2+5+7+10', 124: '2+5+7+9',
125: '2+5+8', 126: '2+5+8+10', 127: '2+5+8+11', 128: '2+5+9', 129:
'2+6', 130: '2+6+10', 131: '2+6+7', 132: '2+6+7+9', 133: '2+6+8', 134:
'2+6+9', 135: '2+7', 136: '2+7+8', 137: '2+8', 138: '3', 139: '3+4',
140: '3+4+6', 141: '3+4+8', 142: '3+4+8+10', 143: '3+4+8+11', 144:
'3+5', 145: '3+5+10', 146: '3+5+7', 147: '3+5+7+10', 148:
'3+5+7+8+11', 149: '3+5+8', 150: '3+5+8+11', 151: '3+6', 152:
'3+6+10', 153: '3+6+7', 154: '3+6+8', 155: '3+6+8+11', 156:
'3+6+9+11', 157: '3+7', 158: '3+7+10', 159: '3+7+11', 160: '3+7+9',
161: '3+7+9+10', 162: '3+8', 163: '3+8+9', 164: '3+9', 165: '4', 166:
'4+10', 167: '4+5', 168: '4+5+7+11', 169: '4+5+7+9', 170: '4+5+9',
171: '4+5+9+0', 172: '4+5+9+11', 173: '4+6', 174: '4+6+10', 175:
'4+6+11', 176: '4+6+7', 177: '4+6+7+9', 178: '4+6+8', 179:
'4+6+8+10+1', 180: '4+6+8+11+1', 181: '4+6+9', 182: '4+6+9+11', 183:
'4+7', 184: '4+7+10', 185: '4+7+11', 186: '4+7+9', 187: '4+7+9+0',
188: '4+7+9+11', 189: '4+8', 190: '4+8+10', 191: '4+8+11', 192:
'4+8+9', 193: '4+8+9+11', 194: '4+9', 195: '4+9+10', 196: '5', 197:
'5+10', 198: '5+11', 199: '5+6', 200: '5+6+10+0', 201: '5+6+8', 202:
'5+7', 203: '5+7+0', 204: '5+7+10', 205: '5+7+8+0', 206: '5+7+9+0',
207: '5+7+9+11', 208: '5+8', 209: '5+8+0', 210: '5+8+10', 211:
'5+8+11', 212: '5+9', 213: '5+9+0', 214: '5+9+11', 215: '5+9+11+0',
216: '6', 217: '6+10', 218: '6+10+0', 219: '6+10+1', 220: '6+10+11',
221: '6+10+11+1', 222: '6+11', 223: '6+11+0', 224: '6+7', 225:
'6+7+0', 226: '6+7+11', 227: '6+7+11+2', 228: '6+7+9+11', 229: '6+8',
230: '6+8+0', 231: '6+8+1', 232: '6+8+10', 233: '6+8+10+0+3', 234:
'6+8+10+1', 235: '6+8+11', 236: '6+8+11+2', 237: '6+9', 238: '6+9+0',
239: '6+9+0+2', 240: '6+9+1', 241: '6+9+11', 242: '6+9+11+1', 243:
'6+9+11+2', 244: '7', 245: '7+0', 246: '7+10', 247: '7+10+0', 248:
'7+10+0+3', 249: '7+10+1', 250: '7+10+1+3', 251: '7+10+2', 252:
'7+11', 253: '7+11+0', 254: '7+11+2', 255: '7+8', 256: '7+8+0', 257:
'7+8+0+3', 258: '7+8+10+2', 259: '7+8+11', 260: '7+9', 261: '7+9+0',
262: '7+9+1', 263: '7+9+11', 264: '7+9+11+0', 265: '7+9+11+1', 266:
'7+9+11+2', 267: '7+9+2', 268: '8', 269: '8+0', 270: '8+0+1', 271:
'8+0+2', 272: '8+0+3', 273: '8+1', 274: '8+10', 275: '8+10+0', 276:
'8+10+1', 277: '8+10+1+3', 278: '8+10+1+4', 279: '8+10+11', 280:
'8+10+11+1', 281: '8+10+11+1+4', 282: '8+10+11+3', 283: '8+10+2', 284:
'8+10+3', 285: '8+11', 286: '8+11+1', 287: '8+11+1+3+4', 288:
'8+11+1+4', 289: '8+11+2', 290: '8+11+3', 291: '8+9', 292: '8+9+1',
293: '8+9+1+3', 294: '8+9+1+4', 295: '8+9+11+1+3', 296: '8+9+2', 297:
'9', 298: '9+0', 299: '9+0+2', 300: '9+0+2+4', 301: '9+0+2+5', 302:
'9+0+3', 303: '9+0+3+5', 304: '9+0+4', 305: '9+1', 306: '9+1+2', 307:
'9+1+4', 308: '9+10', 309: '9+10+2', 310: '9+10+2+4', 311: '9+10+3',
312: '9+11', 313: '9+11+0+2', 314: '9+11+0+4', 315: '9+11+1', 316:
'9+11+1+2', 317: '9+11+2', 318: '9+11+2+4', 319: '9+11+2+5', 320:
'9+11+3', 321: '9+11+4', 322: '9+2', 323: '9+2+3', 324: 'A1', 325:
'A2', 326: 'A3', 327: 'A4', 328: 'A5', 329: 'A6', 330: 'B-1', 331: 'B-

```
2', 332: 'B-3', 333: 'B-4', 334: 'B-5', 335: 'B-6', 336: 'B1', 337:
'B2', 338: 'B3', 339: 'B4', 340: 'B5', 341: 'B6', 342: 'C#1', 343:
'C#2', 344: 'C#3', 345: 'C#4', 346: 'C#5', 347: 'C#6', 348: 'C#7',
349: 'C2', 350: 'C3', 351: 'C4', 352: 'C5', 353: 'C6', 354: 'C7', 355:
'D1', 356: 'D2', 357: 'D3', 358: 'D4', 359: 'D5', 360: 'D6', 361:
'D7', 362: 'E-1', 363: 'E-2', 364: 'E-3', 365: 'E-4', 366: 'E-5', 367:
'E-6', 368: 'E1', 369: 'E2', 370: 'E3', 371: 'E4', 372: 'E5', 373:
'E6', 374: 'F#1', 375: 'F#2', 376: 'F#3', 377: 'F#4', 378: 'F#5', 379:
'F#6', 380: 'F1', 381: 'F2', 382: 'F3', 383: 'F4', 384: 'F5', 385:
'F6', 386: 'G#1', 387: 'G#2', 388: 'G#3', 389: 'G#4', 390: 'G#5', 391:
'G#6', 392: 'G1', 393: 'G2', 394: 'G3', 395: 'G4', 396: 'G5', 397:
'G6'}
```

Generate Input-music in playable format :-

```
# Taking the initial input-index pattern
pattern = networkInput[startIdx]
predictionOutput = []

inputMusicElements = []
inputMusic = []
# inputMusic = (uniqueNotes[ele] for ele in pattern)
inputMusic = (intNoteMap[ele] for ele in pattern)

offset = 0
# offset --> instance-time of particular element (note/chord)


# Have to iterate over all elements of predictionOutput
#   --> Checking whether is a note or chord ?

for element in inputMusic:
    # If element is a chord :-
    if('+' in element) or element.isdigit():
        # Possibilites are like '1+3' or '0'.
        notesInChord = element.split('+')
        # This will get all notes in chord
        tempNotes = []
        for currNote in notesInChord:
            # Creating note-object for each note in chord
            newNote = note.Note(int(currNote))
            # Set it's instrument
            newNote.storedInstrument = instrument.Piano()
            tempNotes.append(newNote)
        # This chord can have x-notes
        # Create a chord-object from list of notes
        newChord = chord.Chord(tempNotes)
        # Adding offset to chord
        newChord.offset = offset
        # Add this chord to music-elements
        inputMusicElements.append(newChord)
```

```
    # If element is a note :-
    else:
        # We know that this is a note
        newNote = note.Note(element)
        # Set off-set of note
        newNote.offset = offset
        # Set the instrument of note
        newNote.storedInstrument = instrument.Piano()
        # Add this note to music-elements
        inputMusicElements.append(newNote)
    offset += 0.5
    # Fixing the time-duration of all elements

# For playing them, have to create a stream-object
#    ..from the generated music-elements

midiInputStream = stream.Stream(inputMusicElements)

# Write this midiStream on a midi-file.
midiInputStream.write('midi', fp="testInput5.mid")
# 1st arg  -->  File-type
# 2nd arg  -->  File-name
```

```
'testInput5.mid'
```

```
midiInputStream.show('midi')
```

```
<IPython.core.display.HTML object>
```

Making Prediction :-

```
# Trying to generate (numIteration)-elements of music
numIteration = 1
# Try with different count variations, so named a variable

for noteIdx in range(numIteration):
    predictionInput = np.reshape(pattern, (1,len(pattern), 1))
    # reshaping into (1, 100, 1)
    # 1st argument --> count of data-points (batch-size)
    # As we have, 1-data of 100-length (2nd argument)
    # 3rd argument --> Because LSTM supports data in 3-dimension.

    # Also to predict over it, normalization is required (values
between [0,1])
    predictionInput = predictionInput / float(countNodes)

    # Making prediction
    prediction = model.predict(predictionInput, verbose=0)
```

```
print("Some elements of prediction[0] are :-")
count = 0
# print(prediction[0][0])
for ele in prediction[0]:
    print(ele)
    count += 1
    if count > 15:
        break
```

```
Some elements of prediction[0] are :-
8.852357e-09
6.4902597e-13
5.3691323e-13
8.518334e-15
1.5045504e-08
7.0550076e-13
2.5262173e-11
3.7756975e-13
7.7455734e-13
1.3269509e-08
4.217455e-10
1.0558226e-12
1.6692346e-14
5.0272537e-14
1.0975207e-14
7.41055e-14
```

Analyzing Prediction :-

```
# Let's see, what our model has predicted
print("Measures of Dispersion of data :- \n")
print("Minimum value = ", np.amin(prediction))
print("Maximum value = ", np.amax(prediction))
print("Range of values = ", np.ptp(prediction))
print("Variance = ", np.var(prediction))
print("Standard Deviation = ", np.std(prediction))
print("Length of 1st Prediction-element = ", len(prediction[0]))
print("Count of unique elements = ", countNodes)
```

```
Measures of Dispersion of data :-

Minimum value =  4.5031164e-15
Maximum value =  0.99995935
Range of values =  0.99995935
Variance =  0.0027775299
Standard Deviation =  0.052702274
Length of 1st Prediction-element =  359
Count of unique elements =  398
```

```
# The values are in range [0,1].
# And, no. of values in 1st prediction are equal to the no. of unique
elements we have.
```

```
# So --> it is clear that this has give the probabilities of all
unique-elements.

# So, taking the element with max. probability
```

Again making prediction, with further processing

```
# Trying to generate (numIteration)-elements of music
numIteration = 200
# This time trying a larger no. of iterations.

for noteIdx in range(numIteration):
    predictionInput = np.reshape(pattern, (1,len(pattern), 1))

    predictionInput = predictionInput / float(countNodes)

    # Making prediction
    prediction = model.predict(predictionInput, verbose=0)

    # Taking the element with max. probability
    idx = np.argmax(prediction)
    # No. corresponding to max. probability element

    # The element corresponding to no. (idx) is :
    result = intNoteMap[idx]

    # Appeding this element to prediction-array
    predictionOutput.append(result)

    # Change input-sequence for further predictions
    # Add this into input, & discard the oldes one.
    pattern.append(idx)
    # slicing out the oldest element (0th index)
    pattern = pattern[1:]
    # Size of pattern remained constant at 100.
    #     (as added 1 element, & removed 1)

print(len(predictionOutput))
print(predictionOutput)
```

```
379
['D3', 'C#5', 'B-4', '8+11+1+3+4', 'C#4', 'B-4', '8+11+1+3+4', 'B-4',
'9+11+0+2', '8+11+1', 'B-4', '9+11+0+2', '8+11+1', 'B-4', '9+11+0+2',
'B-4', '9+11+0+2', '8+11+1+3+4', 'B-4', '9+11+0+2', '9+11+4',
'9+11+0+4', 'B-4', '9+11+2+5', '9+11+0+4', 'C3', '9+11+0+2', '9+0',
'9+1+2', '9+0', '8+9+1', '9+0', 'C3', '9+11+0+2', '8+11+1', '10+11+1',
'9', '8+10+11+1', '2+4+6+7', 'B-4', 'D2', '8+10+3', '9+0+2', '8+10+3',
'8+10+3', '9+11', '8+10+3', '8+10+3', '9+0', '8+10+3', '7+9+11+2',
'8+10+2', '8+10+11+1', '8+10+2', '6+11+0', '8+10+3', '5+9+0',
'7+8+11', '2+4+6+9+11', '8+10+3', '6+11+0', '8+11', '6+10+11+1',
'8+10+2', '9+11+2+5', '7+8+11', '7+9+11+2', '9+11', '8+10+3', 'B-2',
```

```
'7+9+11+2', '8+10+3', 'B-1', '7+9+11+2', '8+10+3', '7+9+11+2',
'8+10+3', '7+9+11+2', '8+11+1', '8+10+3', '8+11+1', '7+9+11+2',
'8+10+3', '7+9', '8+10+3', '6+11+0', '8+10+3', '6+11+0', '8+10+3',
'6+11+0', '8+10+3', '6+11+0', '8+10+3', '8', '8+10+3', '7+9+11+2',
'8+10+3', '7+9+11+2', '8+10+3', '9+0+2', '8+10+3', '8+11+1',
'4+5+7+9', '8+10+3', '8+11+1', '7+8+11', 'D2', 'C7', 'B-1', '8+10+3',
'B-3', 'C#2', 'C#7', '8+10+3', '9+10+3', '9+0+2', '7+9+11+2',
'8+10+3', '8+10+3', '8+10+3', '8+10+3', '8+10+3', '9+0+2', '8+10+3',
'8+10+3', '4+5+7+9', '8+10+3', '8+11+1', '8+10+3', '9+0', '8+10+3',
'9+11+3', '8+10+11+1', '8+10+3', '5+7+9+11', '2+4+6+9+11', '8+10+3',
'8+10+3', '5+9+0', '3+4+8+11', 'D2', 'C7', '5+9+0', '9+11+2+4',
'9+0+2', '7+9+11+2', '8+10+3', '8+10+3', '7+9+11+2', '7+9+11+2',
'9+11+2+5', '7+9+11+2', '8+10+3', '9+11+2+5', '7+9+11+2', '8+10+11+1',
'B-1', 'B-1', '9+11+3', '4+5+9+11', '8+10+11+1', '2+4+6+9+11',
'5+9+0', '2+4+6+9+11', '5+9+0', '2+4+6+9+11', '5+9+0', '2+4+6+9+11',
'5+9+0', '2+4+6+9+11', '5+9+0', '2+4+6+9+11', '5+9+0', '2+4+6+9+11',
'5+9+0', '11+4', '5+9+0', '11+4', '5+9+0', '1+4', '5+9+0', '1+4',
'1+4', '5+9+0', '0+3+6+8', '0+3+6+8', '5+9+0', '0+3+6+8',
'2+4+6+9+11', '6+8+11', '11+0+4', '5+9+0', '2+4+6+9+11', '5+9+0',
'2+4+6+9+11', '5+9+0', '2+4+6+9+11', '5+9+0', '2+4+6+9+11', '5+9+0',
'2+4+6+9+11', '5+9+0', '11+4', '5+9+0', '11+4', '5+9+0', '11+4',
'5+9+0', '1+4', '5+9+0', '1+4', '1+4', '5+9+0', '0+3+6+8', '0+3+6+8',
'5+9+0', '0+3+6+8', '2+3+7', '6+8+11', '11+0+4', '2+3+7', '2+3+7',
'11+0+4', '4+5+9+0', '11+1+4+5', '2+3+7', '2+3+7', '2+3+7', '11+0+4',
'2+3+7', '10+11+1+3', '2', '2+3+7', '2+3+7', '2', '6+8+11',
'10+11+1+3', '2+3+7', '2+3+7', '2+3+7', '2', '2+3+7', '6+8+11',
'11+0+4', '2+3+7', '2+3+7', '11+0+4', '4+5+9+0', '11+1+4+5', '2+3+7',
'2+3+7', '2+3+7', '11+0+4', '2+3+7', '10+11+1+3', '2', '2+3+7',
'2+3+7', '2', '6+8+11', '10+11+1+3', '2+3+7', '2+3+7', '2+3+7', '2',
'2+3+7', '6+8+11', '11+0+4', '2+3+7', '2+3+7', '11+0+4', '4+5+9+0',
'11+1+4+5', '2+3+7', '2+3+7', '2+3+7', '11+0+4', '2+3+7', '10+11+1+3',
'2', '2+3+7', '2+3+7', '2', '6+8+11', '10+11+1+3', '2+3+7', '2+3+7',
'2+3+7', '2', '4+9+10', '6+8+11', '4+9+10', '6+8+11', '4+9+10',
'6+8+11', '4+9+10', '6+8+11', '4+9+10', '6+8+11', '4+9+10', '6+8+11',
'4+9+10', '6+8+11', '6+8+11', '6+8+11', '9+0+2+4', '0+6', '2+4+8',
'0+3+6+9', '0+6', '10+11+1+3', '11+1+4+5', '2+4+6+9+11', '11+1+4+5',
'6+8+11', '6+8+11', '1+3+6', '0+3+6+9', '9+11+0+2', '1+3+6', 'C#5',
'9+1+2', '0+4+6', '8+9+1', '0+6', '8+10+11+1', '0+6', '3+4+8+11',
'2+4+8', '2+4+6+7', '2+4+6+7', '7+10+0+3', '4+7+9+11', 'C4', '1+5+8',
'10+11+1', '8+11+1+3+4', '1+4+8', '4+5+7+9', 'C#7', '1+4+8',
'7+10+0+3', 'B-5', '8+10+11+1', '1+4+8', 'C#4', 'D2', '3+4+6', 'A1',
'2+3+7+9', 'A3', '7+10+0+3', '7+10+0+3', 'B-3', '6+8+11+2',
'8+10+11+1', '0+4+6', 'C4', '7+8+11', '5+9+0', '8+11+1+3+4',
'4+5+9+11', '9+1', '8+10+11+1', '7+9+11+2', '9+11', '9+11+0+4',
'9+11+0+2', 'C#4', '8+11', '8+11', '10+11+1+3', '8+10+11+1',
'0+3+6+9', '11', '11+1+4+5', '0', '10+11+1+3', '0']
```

```
# I have trained for only 10 epochs, if more training is done, variety
in notes-chords (music-elements) will be seen.
```

Generate Music out of Predicted-data :

What required is to get a Midi File :-

```
outputMusicElements = []
# Array to store notes & chords.
```

Trying to Create a Note (from string) :-

```
tempStr = 'C4'
# Just copying from the predictionOutput display

# Creating a note-object (using note-package)
note.Note(tempStr)
```

```
<music21.note.Note C>
```

```
# Music-note is generated.
# Similarly we can do for multiple elements.
newNote = note.Note(tempStr)

# Also, the note will have a off-set (timing)
# By default, offset was 0. (setting it manually here)
newNote.offset = 0
# And, the note will have an instrument
#    Can set, using storedInstrument package
newNote.storedInstrument = instrument.Piano()
# outputMusicElements.append(newNote)
# Above element is commented out, as it will get unwanted music like
this random created note.

print(newNote)
```

```
<music21.note.Note C>
```

Creating Music-Elements from String-array :-

```
offset = 0
# offset --> instance-time of particular element (note/chord)


# Have to iterate over all elements of predictionOutput
#   --> Checking whether is a note or chord ?

for element in predictionOutput:
    # If element is a chord :-
    if('+' in element) or element.isdigit():
        # Possibilites are like '1+3' or '0'.
        notesInChord = element.split('+')
        # This will get all notes in chord
        tempNotes = []
        for currNote in notesInChord:
            # Creating note-object for each note in chord
```

```
            newNote = note.Note(int(currNote))
            # Set it's instrument
            newNote.storedInstrument = instrument.Piano()
            tempNotes.append(newNote)
        # This chord can have x-notes
        # Create a chord-object from list of notes
        newChord = chord.Chord(tempNotes)
        # Adding offset to chord
        newChord.offset = offset
        # Add this chord to music-elements
        outputMusicElements.append(newChord)

    # If element is a note :-
    else:
        # We know that this is a note
        newNote = note.Note(element)
        # Set off-set of note
        newNote.offset = offset
        # Set the instrument of note
        newNote.storedInstrument = instrument.Piano()
        # Add this note to music-elements
        outputMusicElements.append(newNote)
    offset += 0.5
    # Fixing the time-duration of all elements

print(len(outputMusicElements))
print(outputMusicElements)
```

```
379
[<music21.note.Note D>, <music21.note.Note C#>, <music21.note.Note B-
>, <music21.chord.Chord G# B C# E- E>, <music21.note.Note C#>,
<music21.note.Note B->, <music21.chord.Chord G# B C# E- E>,
<music21.note.Note B->, <music21.chord.Chord A B C D>,
<music21.chord.Chord G# B C#>, <music21.note.Note B->,
<music21.chord.Chord A B C D>, <music21.chord.Chord G# B C#>,
<music21.note.Note B->, <music21.chord.Chord A B C D>,
<music21.note.Note B->, <music21.chord.Chord A B C D>,
<music21.chord.Chord G# B C# E- E>, <music21.note.Note B->,
<music21.chord.Chord A B C D>, <music21.chord.Chord A B E>,
<music21.chord.Chord A B C E>, <music21.note.Note B->,
<music21.chord.Chord A B D F>, <music21.chord.Chord A B C E>,
<music21.note.Note C>, <music21.chord.Chord A B C D>,
<music21.chord.Chord A C>, <music21.chord.Chord A C# D>,
<music21.chord.Chord A C>, <music21.chord.Chord G# A C#>,
<music21.chord.Chord A C>, <music21.note.Note C>, <music21.chord.Chord
A B C D>, <music21.chord.Chord G# B C#>, <music21.chord.Chord B- B
C#>, <music21.chord.Chord A>, <music21.chord.Chord G# B- B C#>,
<music21.chord.Chord D E F# G>, <music21.note.Note B->,
<music21.note.Note D>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord A C D>, <music21.chord.Chord G# B- E->,
```

<music21.chord.Chord G# B- E->, <music21.chord.Chord A B>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G# B- E->,
<music21.chord.Chord A C>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord G A B D>, <music21.chord.Chord G# B- D>,
<music21.chord.Chord G# B- B C#>, <music21.chord.Chord G# B- D>,
<music21.chord.Chord F# B C>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord F A C>, <music21.chord.Chord G G# B>,
<music21.chord.Chord D E F# A B>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord F# B C>, <music21.chord.Chord G# B>,
<music21.chord.Chord F# B- B C#>, <music21.chord.Chord G# B- D>,
<music21.chord.Chord A B D F>, <music21.chord.Chord G G# B>,
<music21.chord.Chord G A B D>, <music21.chord.Chord A B>,
<music21.chord.Chord G# B- E->, <music21.note.Note B->,
<music21.chord.Chord G A B D>, <music21.chord.Chord G# B- E->,
<music21.note.Note B->, <music21.chord.Chord G A B D>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G A B D>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G A B D>,
<music21.chord.Chord G# B C#>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord G# B C#>, <music21.chord.Chord G A B D>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G A>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord F# B C>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord F# B C>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord F# B C>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord F# B C>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G#>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G A B D>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G A B D>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord A C D>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G# B C#>,
<music21.chord.Chord E F G A>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord G# B C#>, <music21.chord.Chord G G# B>,
<music21.note.Note D>, <music21.note.Note C>, <music21.note.Note B->,
<music21.chord.Chord G# B- E->, <music21.note.Note B->,
<music21.note.Note C#>, <music21.note.Note C#>, <music21.chord.Chord
G# B- E->, <music21.chord.Chord A B- E->, <music21.chord.Chord A C D>,
<music21.chord.Chord G A B D>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G# B- E->,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G# B- E->,
<music21.chord.Chord A C D>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord G# B- E->, <music21.chord.Chord E F G A>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G# B C#>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord A C>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord A B E->,
<music21.chord.Chord G# B- B C#>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord F G A B>, <music21.chord.Chord D E F# A B>,
<music21.chord.Chord G# B- E->, <music21.chord.Chord G# B- E->,
<music21.chord.Chord F A C>, <music21.chord.Chord E- E G# B>,
<music21.note.Note D>, <music21.note.Note C>, <music21.chord.Chord F A
C>, <music21.chord.Chord A B D E>, <music21.chord.Chord A C D>,
<music21.chord.Chord G A B D>, <music21.chord.Chord G# B- E->,

<music21.chord.Chord G# B- E->, <music21.chord.Chord G A B D>,
<music21.chord.Chord G A B D>, <music21.chord.Chord A B D F>,
<music21.chord.Chord G A B D>, <music21.chord.Chord G# B- E->,
<music21.chord.Chord A B D F>, <music21.chord.Chord G A B D>,
<music21.chord.Chord G# B- B C#>, <music21.note.Note B->,
<music21.note.Note B->, <music21.chord.Chord A B E->,
<music21.chord.Chord E F A B>, <music21.chord.Chord G# B- B C#>,
<music21.chord.Chord D E F# A B>, <music21.chord.Chord F A C>,
<music21.chord.Chord D E F# A B>, <music21.chord.Chord F A C>,
<music21.chord.Chord D E F# A B>, <music21.chord.Chord F A C>,
<music21.chord.Chord D E F# A B>, <music21.chord.Chord F A C>,
<music21.chord.Chord D E F# A B>, <music21.chord.Chord F A C>,
<music21.chord.Chord D E F# A B>, <music21.chord.Chord F A C>,
<music21.chord.Chord B E>, <music21.chord.Chord F A C>,
<music21.chord.Chord B E>, <music21.chord.Chord F A C>,
<music21.chord.Chord C# E>, <music21.chord.Chord F A C>,
<music21.chord.Chord C# E>, <music21.chord.Chord C# E>,
<music21.chord.Chord F A C>, <music21.chord.Chord C E- F# G#>,
<music21.chord.Chord C E- F# G#>, <music21.chord.Chord F A C>,
<music21.chord.Chord C E- F# G#>, <music21.chord.Chord D E F# A B>,
<music21.chord.Chord F# G# B>, <music21.chord.Chord B C E>,
<music21.chord.Chord F A C>, <music21.chord.Chord D E F# A B>,
<music21.chord.Chord F A C>, <music21.chord.Chord D E F# A B>,
<music21.chord.Chord F A C>, <music21.chord.Chord D E F# A B>,
<music21.chord.Chord F A C>, <music21.chord.Chord D E F# A B>,
<music21.chord.Chord F A C>, <music21.chord.Chord D E F# A B>,
<music21.chord.Chord F A C>, <music21.chord.Chord B E>,
<music21.chord.Chord F A C>, <music21.chord.Chord B E>,
<music21.chord.Chord F A C>, <music21.chord.Chord B E>,
<music21.chord.Chord F A C>, <music21.chord.Chord C# E>,
<music21.chord.Chord F A C>, <music21.chord.Chord C# E>,
<music21.chord.Chord C# E>, <music21.chord.Chord F A C>,
<music21.chord.Chord C E- F# G#>, <music21.chord.Chord C E- F# G#>,
<music21.chord.Chord F A C>, <music21.chord.Chord C E- F# G#>,
<music21.chord.Chord D E- G>, <music21.chord.Chord F# G# B>,
<music21.chord.Chord B C E>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord B C E>,
<music21.chord.Chord E F A C>, <music21.chord.Chord B C# E F>,
<music21.chord.Chord D E- G>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord B C E>,
<music21.chord.Chord D E- G>, <music21.chord.Chord B- B C# E->,
<music21.chord.Chord D>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord D>,
<music21.chord.Chord F# G# B>, <music21.chord.Chord B- B C# E->,
<music21.chord.Chord D E- G>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord D>,
<music21.chord.Chord D E- G>, <music21.chord.Chord F# G# B>,
<music21.chord.Chord B C E>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord B C E>,

<music21.chord.Chord E F A C>, <music21.chord.Chord B C# E F>,
<music21.chord.Chord D E- G>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord B C E>,
<music21.chord.Chord D E- G>, <music21.chord.Chord B- B C# E->,
<music21.chord.Chord D>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord D>,
<music21.chord.Chord F# G# B>, <music21.chord.Chord B- B C# E->,
<music21.chord.Chord D E- G>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord D>,
<music21.chord.Chord D E- G>, <music21.chord.Chord F# G# B>,
<music21.chord.Chord B C E>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord B C E>,
<music21.chord.Chord E F A C>, <music21.chord.Chord B C# E F>,
<music21.chord.Chord D E- G>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord B C E>,
<music21.chord.Chord D E- G>, <music21.chord.Chord B- B C# E->,
<music21.chord.Chord D>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord D>,
<music21.chord.Chord F# G# B>, <music21.chord.Chord B- B C# E->,
<music21.chord.Chord D E- G>, <music21.chord.Chord D E- G>,
<music21.chord.Chord D E- G>, <music21.chord.Chord D>,
<music21.chord.Chord E A B->, <music21.chord.Chord F# G# B>,
<music21.chord.Chord E A B->, <music21.chord.Chord F# G# B>,
<music21.chord.Chord E A B->, <music21.chord.Chord F# G# B>,
<music21.chord.Chord E A B->, <music21.chord.Chord F# G# B>,
<music21.chord.Chord E A B->, <music21.chord.Chord F# G# B>,
<music21.chord.Chord E A B->, <music21.chord.Chord F# G# B>,
<music21.chord.Chord E A B->, <music21.chord.Chord F# G# B>,
<music21.chord.Chord F# G# B>, <music21.chord.Chord F# G# B>,
<music21.chord.Chord A C D E>, <music21.chord.Chord C F#>,
<music21.chord.Chord D E G#>, <music21.chord.Chord C E- F# A>,
<music21.chord.Chord C F#>, <music21.chord.Chord B- B C# E->,
<music21.chord.Chord B C# E F>, <music21.chord.Chord D E F# A B>,
<music21.chord.Chord B C# E F>, <music21.chord.Chord F# G# B>,
<music21.chord.Chord F# G# B>, <music21.chord.Chord C# E- F#>,
<music21.chord.Chord C E- F# A>, <music21.chord.Chord A B C D>,
<music21.chord.Chord C# E- F#>, <music21.note.Note C#>,
<music21.chord.Chord A C# D>, <music21.chord.Chord C E F#>,
<music21.chord.Chord G# A C#>, <music21.chord.Chord C F#>,
<music21.chord.Chord G# B- B C#>, <music21.chord.Chord C F#>,
<music21.chord.Chord E- E G# B>, <music21.chord.Chord D E G#>,
<music21.chord.Chord D E F# G>, <music21.chord.Chord D E F# G>,
<music21.chord.Chord G B- C E->, <music21.chord.Chord E G A B>,
<music21.note.Note C>, <music21.chord.Chord C# F G#>,
<music21.chord.Chord B- B C#>, <music21.chord.Chord G# B C# E- E>,
<music21.chord.Chord C# E G#>, <music21.chord.Chord E F G A>,
<music21.note.Note C#>, <music21.chord.Chord C# E G#>,
<music21.chord.Chord G B- C E->, <music21.note.Note B->,
<music21.chord.Chord G# B- B C#>, <music21.chord.Chord C# E G#>,
<music21.note.Note C#>, <music21.note.Note D>, <music21.chord.Chord E-

```
E F#>, <music21.note.Note A>, <music21.chord.Chord D E- G A>,
<music21.note.Note A>, <music21.chord.Chord G B- C E->,
<music21.chord.Chord G B- C E->, <music21.note.Note B->,
<music21.chord.Chord F# G# B D>, <music21.chord.Chord G# B- B C#>,
<music21.chord.Chord C E F#>, <music21.note.Note C>,
<music21.chord.Chord G G# B>, <music21.chord.Chord F A C>,
<music21.chord.Chord G# B C# E- E>, <music21.chord.Chord E F A B>,
<music21.chord.Chord A C#>, <music21.chord.Chord G# B- B C#>,
<music21.chord.Chord G A B D>, <music21.chord.Chord A B>,
<music21.chord.Chord A B C E>, <music21.chord.Chord A B C D>,
<music21.note.Note C#>, <music21.chord.Chord G# B>,
<music21.chord.Chord G# B>, <music21.chord.Chord B- B C# E->,
<music21.chord.Chord G# B- B C#>, <music21.chord.Chord C E- F# A>,
<music21.chord.Chord B>, <music21.chord.Chord B C# E F>,
<music21.chord.Chord C>, <music21.chord.Chord B- B C# E->,
<music21.chord.Chord C>]
```

Trying to Play the Output Music :-

```
# For playing them, have to create a stream-object
#    ..from the generated music-elements

midiStream = stream.Stream(outputMusicElements)

# Write this midiStream on a midi-file.
midiStream.write('midi', fp="testOutput5.mid")
# 1st arg  -->  File-type
# 2nd arg  -->  File-name

'testOutput5.mid'
```

Loading the output-midi file :

```
midiStream.show('midi')
# Show the music in playable-format

<IPython.core.display.HTML object>

outputMusic5 = converter.parse("testOutput5.mid")
print(type(outputMusic5))

<class 'music21.stream.base.Score'>

outputMusic5.show('midi')
# This will show the music in playable-format

<IPython.core.display.HTML object>
```

Plotting inputMusicElements VS outputMusicElements :

```
import matplotlib.pyplot as plt

inputMusicNums = networkInput[startIdx]
print("Some elements of inputMusicNums :-")
```

```
# print(inputMusicNums)
count = 0
for ele in inputMusicNums:
    print(ele)
    count += 1
    if count > 10:
        break
```

```
Some elements of inputMusicNums :-
107
0
79
297
97
165
97
97
187
107
0
```

```
outputMusicNums = []
for ele in predictionOutput:
    outputMusicNums.append(noteMap[ele])
print("Some elements of outputMusicNums are :-")
count = 0
for ele in outputMusicNums:
    print(ele)
    count += 1
    if count > 10:
        break
```

```
Some elements of outputMusicNums are :-
357
346
333
287
345
333
287
333
313
286
333
```

Plot inputMusicNums VS outputMusicNums (prediction visualization) :-