Midi Music Generation with Keras

Installation

Das Machine Learning Model, Training und Prediction läuft in Phyton mithilfe von Keras auf dem Tensorflowbackend. Die Api zum umwandeln der Midifiles in Arrays läuft auf einem NodeJS WebServer.

Python

Requirements

```
1. Python amd64
2. Tensorflow
3. Keras
4. Matplotlib
6. Numpy
```

Node JS

Requirements

```
1. NodeJS
2. Express (npm install im aktuellen Projekt sollte reichen)
```

How to Run:

NodeJS:

```
0.  Installiere alle prerequirements
1.  Fuege den Ordner mit Midi Files der Analysiert werden soll, im Webstormproject
zu unter ./Music ein.
2.  Aendere in Webstorm unter ./routes/index.js den Foldernamen so das er dem
Folder entspricht
    let folderName = "bachOneChannel";
3.  Starte den Server mithilfe von Webstorm oder direkt via NodeJS durch ausfuehren
der ./bin/www.js
```

Python:

```
1.  Stelle die Parameter (siehe untent) in der training.py so ein wie sie gefallen.
Oder lasse Sie so wie sie sind.
2.  Starte training.py mit Python / Pycharm er zieht sich automatisch die Midi
Files vom Webstorm.
3.  Wenn das Training fertig ist sollte eine Plot angezeigt werden und das Model
sollte unter ./ Models gespeichert werden.
4.  Waehle das erstellte Model fuer den gewuenschten Channel in runPrediction aus
```

```
und starte Training.
```

Python Files and Functions:

training.py

Diese Datei muss ausgeführt werden um die Models zu trainieren.

Initialisierung wichtiger Variablen für das Training

```
epochen = 10
sequence_length = 15
batch_size = 200
hl_firstLSTM = 131
hl_secondLSTM = 256
hl_Neuronen_Noten = 256
hl_Neuronen_Duration = 155
```

Erstellen der Ordner in denen die Models gespeichert werden und iteration über jeden Channel der Midi Files. Erstellt für jedes File einen eigenen Projekt Ordner

def printHisory()

Plottet den Verlauf des Trainings 'history': Training des Models gibt ein history-object zurück, das hier als Eingabe verwendet wird

runPrediciton.py

Diese Datei wird ausgeführt um eine Prediction zu starten. Channel Name und Folder Name müssen vorher entsprechend angepasst werden.

apiCalls.py

gets Notes as oneHotEncoded plus duration and time as float32 bitte ändere hier die Serveradresse wenn die Serveradress sich ändert.

```
def GetNotesPlusFloatDuration():
    r = requests.post("http://localhost/getJSOnOfMidiFolder")
    input = json.loads(r.text)
    return input
```

Convert Output Array to Midi File, the Midifile will be stowed on the server.

```
def covertArrayToJSON(resultVektorArray,midiName,channel):
    r = requests.post("http://localhost/convertArrayToMidi", {"midAsJson":
json.dumps(resultVektorArray), "name": midiName, "channel":channel})
    return r.text
```

To grap one big JSON with channels inside for newest not correct working model

```
def GetAllTogether():
    r = requests.post("http://localhost/withAnyThingToInputArray")
    input = json.loads(r.text)
    return input
```

modelDefinitions.py

def specialModel(...)

Dies ist die Definition des aktuell genutzen Keras Models sie ist kurz aber die umwandlung aller Daten ist kompliziert.

```
def
specialModel(notes,channelName,folderName,sequence_length,epochen,batch_size,hl_fir
stLSTM,hl_secondLSTM):
    [....]
    [....]
    #Erstellung des Models
    mainInput = Input(shape=(sequence_length, uploadResult.shape[1]),
dtype='float32', name='mainInput')
    print(mainInput)
    mainLSTM = LSTM(hl_firstLSTM, return_sequences=True)(mainInput)
    dropOut = Dropout(0.2)(mainLSTM)
    secondLSTM = LSTM(hl_secondLSTM)(dropOut)
    dropOut2 = Dropout(0.2)(secondLSTM)
    noteDenseLayer = Dense(128, activation='softmax',
name='noteDenseLayer')(dropOut2)
    durationDenseLayer = Dense(durationAmount,
name='durationDenseLayer',activation='softmax')(dropOut2)
    timeDenseLayer = Dense(timeAmount,
name='timeDenseLayer',activation='softmax')(dropOut2)

    model = Model(inputs= mainInput, outputs=[noteDenseLayer,
durationDenseLayer,timeDenseLayer])
    model.compile(optimizer='Adadelta',
                  loss={'noteDenseLayer': 'categorical_crossentropy',
'durationDenseLayer': 'categorical_crossentropy', 'timeDenseLayer':
'categorical_crossentropy'},metrics=['accuracy'])
    [....]
    [....]
```

def categorieze(duration)

Diese Funktion categorisiert Float Arrays und liefert für jeden Wert eine ID

def shapeData(uploadResult,sequence_length):

Function ändert die Art wie die Daten formartiert sind.

predictionFunctions.py

def makeSpecialPrediction(model,actualTrain,midiName,channel,durations,times):

Hier findet die Tatsächliche Prediktion statt. Es wird das aktuell ausgewählte model geladen und x Schritte in die Zukunft predicted. Der Output des jeweiligen Prediction Schrittes wird an den vorherigen Output angehängt. So das es einen neuen Input Vektor erstellt.

def runPrediction(folderName,channel,midiName):

Diese Funktion wird von run Prediction ausgeführt Sie initializiert den ersten Input Vektor und führt dann make Special Prediction aus.

Interressante NodeJS Funktionen:

router.post('/convertArrayToMidi', function(req, res) {[...]});

Diese Methode konvertiert ein ankommendes Array zurück zu Midi

let folderName = "bachOneChannel";

Kopiere den Music Ordner zu ./public music und stelle hier den Namen des Ordners vor dem Training ein.

router.post('/getJSOnOfMidiFolder', function(req, res) {[...]});

Diese Methode wandelt Midi zu JSON.

1. function tracks(folder){}Filtert alle Tracks aus allen Files in den Ordnern
2. function makeLSTMInputVektorOutOfTracks(notes){}macht input Vektoren aus den TrackArray
3. function getChannels(TrackArray){}Sortiert die Tracks des Midi Files nach Channels
4. function getOnlyChannelsWithOverXAmount(channelArray,minAmount){Filtered alle Channels mit mehr als minAmount Notenevents
5. function getChannelNotes(TrackArray, channel){}Sucht sich die Noten in den Channels