A tutorial of the LSTM-Music-Genre-Classification-master demo program

KH Wong

## Introduction

The Music genre classification (from <https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification> ) is a LSTM (Long Short Term Memory) neural network system that is able to learn from a set of training music recordings with known class labels. After learning it can classify successfully a new music recording to a suitable class with an accuracy of more than 70 %. So it is a supervised learning method. The system uses a library called “LIBROSA to handle music signal feature extraction. And the neural network model is written under the framework of TENSORFLOW. This tutorial will discuss the structure of the dataset and how they are being used. We also explain the LIBROSA feature extraction method and how parameters are used. Finally we will see how to modify the TENSORFLOW LSTM model to improve the result. We believe it is a useful tool to serve as a framework for many similar audio signal processing applications such as music sentiment analysis and music information retrieval (MIR) applications.

## Background and theory

Music genre classification has been a popular research topic recently. It has enormous potential in the entertainment and music industry.

## Music signal features: please read <http://www.cse.cuhk.edu.hk/~khwong/www2/cmsc5707/5707_03_features.pptx>

LSTM Long Short term Memory Neural network

Please read <http://www.cse.cuhk.edu.hk/~khwong/www2/cmsc5707/5707_11_rnn_lstm.pptx>

## Installation and overview of the structure

Assume you are using win10 with anaconda2, python3 and TENSORFLOW tf2, installation of these tools can be found at [hong'slinks - 1.Tensor\_windows (installation through Anaconda3) (google.com)](https://sites.google.com/site/hongslinks/1-tensor_windows-installation-through-anaconda3?authuser=0) . See the appendix for details. Method of recording, editing and converting sound files to be used in this system can also be found in the appendix.

If you are using TENSORFLOW2 (tf2), you need to change the source code because keras is already in TENSORFLOW you have to tell the system about it. See below or Appendix for details.

#from line 46 of lstm\_genre\_classifier\_keras.py

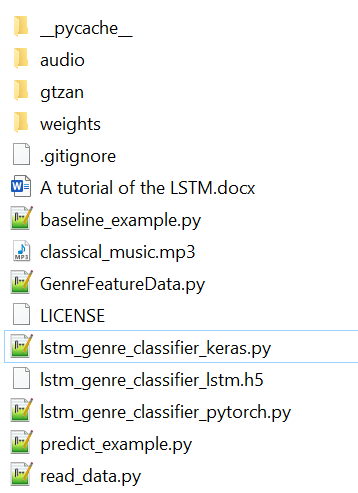
from tensorflow.python.keras.models import Sequential

from tensorflow.python.keras.layers.recurrent import LSTM

from tensorflow.python.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

After installation of the above tools, you may download the zip files from <https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification>  and unzip to the current directory (..\), where you should find the main python code, such as GenreFeatureData.py and lstm\_genre\_classifier\_keras.py. A picture of the system (current) directory view is shown below.



### Overview of the programming structure.

The main program is The main program is lstm\_genre\_classifier\_keras.py. After it is executed

By the command tf-cpu (or tf-gpu) under anaconda>> python is lstm\_genre\_classifier\_keras.py

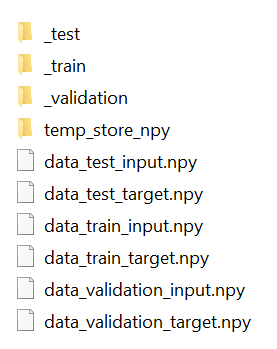
It will first check if the \*.npy under \gztan exist or not. If yes, those \*.npy file will be used to train the :LSTM system. Otherwise, GenreFeatureData.py will be execiuted to generate those \gztan\\*.npy files . After that these .npy files will be used for the training of the LSTM system. Therefore if you wantto start from some fresh training data, you should delete the existing \gztan\\*.npy feels . So new .npy files will be generated from \*.au files..\gztan\(\_test, \_train, \_validation) to produce the new training data.

### Overview of the dataset

The directory with data set should is called gztan. The three directories (\_test, \_train, \_validation) are for testing , training and validation respectively. Some preloaded data are already there when you download the zip file from <https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification>

If you want to use more music data sets, you can download more directly from <https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification>

As discussed, those ..\gztan\\*.npy files are numpy data arrays to be used by LSTM. In the main directory, GenreFeatureData.py reads\*.au audio files (Sun format) from \_test, \_train, \_validation and save them to the corresponding .npy files.



As discussed the sound data files are inside directories \\_train, \\_test and \\_validate. The directory \\_train is for training the system. After the LSTM network is training, we use the files in \test to test the system to find out the accuracy rate. The directory \\_validate contain files to validate your system, you may use it tune up your approach. Raw new data can be found in gztan <https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification> . Usually people use 60 % for training, 20% for testing and 20% for validation. Using the existing \*.au data preload in the 3 directories are enough for your testing.

\\_train

\\_test

\\_validate

the files have the type such as 'classical', 'disco' etc prefixed.

class GenreFeatureData:

### How to use the dataset

We will use the training phase to illustrate how to use the sound data. Assume you delete all existing .npy file under \gztan, GenreFeatureData.py will be called to generate new \*.npy from the audio sound files from the directory \\_train.

In GenreFeatureData.py , genre\_list from line 13

    "Music audio features for genre classification"

    hop\_length = None

    genre\_list = [

        "classical",

        "country",

        "disco",

        "hiphop",

        "jazz",

        "metal",

        "pop",

        "reggae",

    ]

8 (or 10 in some versions) different classes are shown in the program. They correspdon o te file preficed file name fo the training sound files \*.au.

#### Naming of the training files define their classes

As you can see the .au files in \\_train: they are classical.00030.au, classical.00031.au, .. etc.

Each file is a training sample in .au format the file name is prefixed by the class name , for example, classical., hiphop. etc.

You may create a new class, but you have to change 1) GenreFeatureData.py , genre\_list from line 13, 2) the file name of the audio file

#### How long is the default data length (samples and time) it uses for training the LSTM

As explained below, since you sampling (sr) is 22050 samples/second, self.timeseries\_length represents how much data you feed to the LSTM training. In summary:

Total recording time is 30.012s for an .au file, so each file 22050\*30.012≈661764 samples

Hop\_length (hop\_size, or stride, or non-overlapping sample)= 661764/512≈1293 samples

#### What can you change?

Now we are reading the first 3.065 seconds of each music file, if you change self.timeseries\_length = 256, you read 6 seconds. The highest you can use is 1293 but the system seems not be able to tolerate such large data for subsequent processing. You may try yourself, I tried self.timeseries\_length = 256 , itis ok, but 1293 fails.

Line 53 of GenreFeatureData.py ,

self.timeseries\_length = (

128 #original is 128, #max is 1293, tested result

) # sequence length == 128, default fftsize == 2048 & hop == 512 @ SR of 22050

# equals 128 overlapped windows that cover approx ~3.065 seconds of audio,

# which is a bit small!

# hop=512samples=512/22050 ms,sample len(30s for an .au file),

# 30000/23.22=1292 frames

#total length of a. au file:30s =30/(512/22050) =1291.99,using 128(30s)is too small

#### How to change the features obtained from the mfcc extraction library LIBROISA?

See <https://librosa.org/doc/latest/generated/librosa.feature.mfcc.html>, **librosa.feature.mfcc(*y=None*, *sr=22050*, *S=None*, *n\_mfcc=20*, *dct\_type=2*, *norm='ortho'*, *lifter=0*, *\*\*kwargs*)**

There are many different ways you can use the MFCC/LIBROSA function, these are they things that I tried.

* Lifter (to highlight ow or high time parameters in cepstral domain), see <http://www.cse.cuhk.edu.hk/~khwong/www2/cmsc5707/5707_03_features.pptx>
* You may skip MFCC 0, if you are processing speech, by modifying line 130 onward of GenreFeatureData.py :

#use all 13 mfcc features including mfcc0, read the above pptx for the reason

data[i, :, 0:13] = mfcc.T[0:self.timeseries\_length, :] #orginal

data[i, :, 13:14] = spectral\_center.T[0:self.timeseries\_length, :]

data[i, :, 14:26] = chroma.T[0:self.timeseries\_length, :]

data[i, :, 26:33] = spectral\_contrast.T[0:self.timeseries\_length, :]

### if you skip mfcc0, the data will be 1 column shorter

# data[i, :, 0:12] = mfcc.T[0:self.timeseries\_length, 1:13] #khw, skip mfcc0, .T is transpose

# data[i, :, 13-1:14-1] = spectral\_center.T[0:self.timeseries\_length, :]

# data[i, :, 14-1:26-1] = chroma.T[0:self.timeseries\_length, :]

# data[i, :, 26-1:33-1] = spectral\_contrast.T[0:self.timeseries\_length, :]

Conclusion:

The open source code from <https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification> ) is good demo program for audio signal processing and neural networks. You can learn how to extract audio features such as MFCC (Mel scale Cepstral coefficients) from audio files and how to use them for a LSTM (Long Short Term memory) neural network model to build useful machine learning applications.

Appendix

**Music Genre Classification tool**

|  |
| --- |
| Handle sound recording  1) <https://www.goldwave.com/,> download and install the free version.  2) Record/edit you sound and save it using .au (Sun)  file format, it is for <https://librosa.org/> , software to convert .au sound to mfcc code.  3) Install tensorflow 2 if you have not installed it before. See  [hong'slinks - 1.Tensor\_windows (installation through Anaconda3) (google.com)](https://sites.google.com/site/hongslinks/1-tensor_windows-installation-through-anaconda3?authuser=0)  /////////////////////////////////////////////////////////////////////////////////////// , the main demo program is "lstm\_genre\_classifier\_keras.py" //////////////////////////////  readme\_genre\_classification\_lstm\_201012.txt  Perquisite: Installation Tensorflow guide, see below  <https://www.tensorflow.org/tutorials> or [hong'slinks - 1.Tensor\_windows (installation through Anaconda3) (google.com)](https://sites.google.com/site/hongslinks/1-tensor_windows-installation-through-anaconda3?authuser=0)  error fix (if you use tensorflow2) from https://github.com/keras-team/keras/tree/master/      fix: keras --> tensorflow.keras (for <https://github.com/keras-team/keras>)      fix:from keras.utils.data\_utils import get\_file --> from tensorflow.python.keras.utils.data\_utils import get\_file  You need these parts:  (1) <https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification> ,  (2) gtzan:  <https://www.kaggle.com/carlthome/gtzan-genre-collection>  (3) In tensorflow under anaconda (admin) : conda>> pip install librosa #( [http://librosa.org](http://librosa.org/), required for turning  music files to mfcc codes)  ======  download and install LSTM-Music-Genre-Classification ==2020 Oct 12; 2020 Sept 2, khwong  1) download the zip file from  <https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification>  2) unzip to a directory , assume "LSTM-Music-Genre-Classification-master\_ok\_test2"  3) Assume you are working on  C:\\_projects\\_5d3\_tensorflow\_ok\tensorflow\_tested\_ok\_200804\LSTM-Music-Genre-Classification-master\_ok\_test2  You may delete all .npy files in  ...\LSTM-Music-Genre-Classification-master\_ok\_test2\gtzan .  Reason: if you don't use new sound files, you may use the original .npy data from sound files preinstalled  by the author of this githib software for testing you LSTM software.  4) Download gtzan data set from <https://www.kaggle.com/carlthome/gtzan-genre-collection>  and unzip to ...\LSTM-Music-Genre-Classification-master\_ok\_test2\gtzan  5) Select and arrange you test sound files, save them in the following directories:     test (20%), train(60%),validation(20) data, using this percentage arrangement or some other choices.  ...\LSTM-Music-Genre-Classification-master\_ok\_test2\gtzan\\_test  ...\LSTM-Music-Genre-Classification-master\_ok\_test2\gtzan\\_train  ...\LSTM-Music-Genre-Classification-master\_ok\_test2\gtzan\\_validation  ============How this software works===================  tf-cpu (or tf-gpu) >> python python lstm\_genre\_classifier\_keras.py  If you run  >python lstm\_genre\_classifier\_keras.py  it will read all audio files .au from  C:\\_projects\\_5d3\_tensorflow\_ok\tensorflow\_tested\_ok\_200804\LSTM-Music-Genre-Classification-master\_ok\_test2\gtzan\\_test  C:\\_projects\\_5d3\_tensorflow\_ok\tensorflow\_tested\_ok\_200804\LSTM-Music-Genre-Classification-master\_ok\_test2\gtzan\\_train  C:\\_projects\\_5d3\_tensorflow\_ok\tensorflow\_tested\_ok\_200804\LSTM-Music-Genre-Classification-master\_ok\_test2\gtzan\\_validation  The sound files will be turned into .npy (python data) in  C:\\_projects\\_5d3\_tensorflow\_ok\tensorflow\_tested\_ok\_200804\LSTM-Music-Genre-Classification-master\_ok\_test2\gtzan  That's why before you use this software , delete the original .npy files. But if you don't use new sound files,  you may use the original .npy files for testing you LSTM software.  ==== if you want to add more lstm layers, line 92 of  lstm\_genre\_classifier\_keras.py========  model.add(LSTM(units=128, dropout=0.05, recurrent\_dropout=0.35, return\_sequences=True, input\_shape=input\_shape))  #model.add(LSTM(units=32,  dropout=0.05, recurrent\_dropout=0.35, return\_sequences=True))#return\_sequences=True if you have mor elayers  #model.add(LSTM(units=32,  dropout=0.05, recurrent\_dropout=0.35, return\_sequences=True))#added  model.add(LSTM(units=32,  dropout=0.05, recurrent\_dropout=0.35, return\_sequences=False))#added  model.add(Dense(units=genre\_features.train\_Y.shape[1], activation="softmax"))  ==========  data structure ==============================  in GenreFeatureData.py  it defines the types: and see the files in  \\_train  \\_test  \\_validate  the files have the type such as 'classical', 'disco' etc prefixed.  class GenreFeatureData:      "Music audio features for genre classification"      hop\_length = None      genre\_list = [          "classical",          "country",          "disco",          "hiphop",          "jazz",          "metal",          "pop",          "reggae",      ]  (See above, you may edit this file  (GenreFeatureData.py ) to change the directory structure and names used,  Recommendation: not to change it  #################################################################################################  General description of the Music genre classification demo program in <https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification>, See         Sound file handling: It uses Librosa (librosa.org)  to convert sound files (.au format) to MFCC (Mel-frequency cepstral coefficients ). Data is save in ..\gztan\\*.npy format (python numpy format). When you run lstm\_genre\_classifier\_keras.py, it will earch for .npy under ..\gztan\, if not found , it will convert sound file s(.au) from ..\gtzan\\_train, \gtzan\\_train  to become the suitbale .npy files suhc as data\_train\_input.npy, data\_test\_input.npy, they will be used for training and testing.         How to encode the class for samples: the class for each input is placed at the file name. Such as classical.00000.au, classical.00000.au, the class is Classical, the program uses “.” To separate the class and the testing sample serial number of the sound file.  f you want to change the class name:  o   Change Line 10 of GenreFeatureData.py, i.e., ‘Classical  ‘AnotherClassName’  etc. Do it for the other class name, such as ‘Hihop’, ‘Jazz’ etc.  o   Also change files name in ..\gztan\\_tarin, ..\gztan\test  accordingly etx.         How to change the network architecture: The orginal system has 2 LSTM layers, you may add more to it to improve result: See line92 of  lstm\_genre\_classifier\_keras.py  model.add(LSTM(units=128, dropout=0.05, recurrent\_dropout=0.35, return\_sequences=True, input\_shape=input\_shape))  model.add(LSTM(units=32,  dropout=0.05, recurrent\_dropout=0.35, return\_sequences=False))  model.add(Dense(units=genre\_features.train\_Y.shape[1], activation="softmax"))    Change “return\_sequences=False” is for the final hidden layer. Change to “return\_sequences=False”  to “return\_sequences=True” if it is not the last layer. Try and figure out how to add layer yourself.   ================= analysis of  GenreFeatureData.py=============  from line 53          # compute minimum timeseries length, slow to compute, caching pre-computed value of 1290          # self.precompute\_min\_timeseries\_len()          # print("min(self.timeseries\_length\_list) ==" + str(min(self.timeseries\_length\_list)))          # self.timeseries\_length = min(self.timeseries\_length\_list)          self.timeseries\_length = (              256 #orginal is 128, #max is 1293, tested result          )   # sequence length == 128, default fftsize == 2048 & hop == 512 @ SR of 22050          #  equals 128 overlapped windows that cover approx ~3.065 seconds of audio, which is a bit small!          # hop=512samples=512/22050 ms,sample len(30s for an .au file),          # 30000/23.22=1292 frames          #total length  of a. au file of 30s  =30/(512/22050) =1291.9921875, using 128 is too small      def load\_preprocess\_data(self):          print("[DEBUG] total number of files: " + str(len(self.timeseries\_length\_list)))          # Training set          self.train\_X, self.train\_Y = self.extract\_audio\_features(self.trainfiles\_list)          with open(self.train\_X\_preprocessed\_data, "wb") as f:              np.save(f, self.train\_X)          with open(self.train\_Y\_preprocessed\_data, "wb") as f:              self.train\_Y = self.one\_hot(self.train\_Y)              np.save(f, self.train\_Y)          # Validation set          self.dev\_X, self.dev\_Y = self.extract\_audio\_features(self.devfiles\_list)          with open(self.dev\_X\_preprocessed\_data, "wb") as f:              np.save(f, self.dev\_X)          with open(self.dev\_Y\_preprocessed\_data, "wb") as f:              self.dev\_Y = self.one\_hot(self.dev\_Y)              np.save(f, self.dev\_Y)          # Test set          self.test\_X, self.test\_Y = self.extract\_audio\_features(self.testfiles\_list)          with open(self.test\_X\_preprocessed\_data, "wb") as f:              np.save(f, self.test\_X)          with open(self.test\_Y\_preprocessed\_data, "wb") as f:              self.test\_Y = self.one\_hot(self.test\_Y)              np.save(f, self.test\_Y)      def load\_deserialize\_data(self):          self.train\_X = np.load(self.train\_X\_preprocessed\_data)          self.train\_Y = np.load(self.train\_Y\_preprocessed\_data)          self.dev\_X = np.load(self.dev\_X\_preprocessed\_data)          self.dev\_Y = np.load(self.dev\_Y\_preprocessed\_data)          self.test\_X = np.load(self.test\_X\_preprocessed\_data)          self.test\_Y = np.load(self.test\_Y\_preprocessed\_data)      def precompute\_min\_timeseries\_len(self):          for file in self.all\_files\_list:              print("Loading " + str(file))              y, sr = librosa.load(file)              self.timeseries\_length\_list.append(math.ceil(len(y) / self.hop\_length))      def extract\_audio\_features(self, list\_of\_audiofiles):          data = np.zeros(              (len(list\_of\_audiofiles), self.timeseries\_length, 33), dtype=np.float64          )          target = []          for i, file in enumerate(list\_of\_audiofiles):              y, sr = librosa.load(file) #y= sound data , sr = sampling rate              #print('661794 by expeirment , 661794/22050=30.133 sec, ok, len(y)=')              #print(len(y))                mfcc = librosa.feature.mfcc(                  y=y, sr=sr, hop\_length=self.hop\_length, n\_mfcc=13              )              spectral\_center = librosa.feature.spectral\_centroid(                  y=y, sr=sr, hop\_length=self.hop\_length              )              chroma = librosa.feature.chroma\_stft(y=y, sr=sr, hop\_length=self.hop\_length)              spectral\_contrast = librosa.feature.spectral\_contrast(                  y=y, sr=sr, hop\_length=self.hop\_length              )              splits = re.split("[ .]", file)              genre = re.split("[ /]", splits[1])[3]              target.append(genre)              #use all 13 mfcc features including mfcc0              data[i, :, 0:13] = mfcc.T[0:self.timeseries\_length, :] #orginal              data[i, :, 13:14] = spectral\_center.T[0:self.timeseries\_length, :]              data[i, :, 14:26] = chroma.T[0:self.timeseries\_length, :]              data[i, :, 26:33] = spectral\_contrast.T[0:self.timeseries\_length, :]                ### if you skip mfcc0              # data[i, :, 0:12] = mfcc.T[0:self.timeseries\_length, 1:13] #khw, skip mfcc0, .T is transpose              # data[i, :, 13-1:14-1] = spectral\_center.T[0:self.timeseries\_length, :]              # data[i, :, 14-1:26-1] = chroma.T[0:self.timeseries\_length, :]              # data[i, :, 26-1:33-1] = spectral\_contrast.T[0:self.timeseries\_length, :]                    # dd1,dd2= mfcc.shape              # print('it is mfcc features = 13, dd1=')              # print(dd1)              # print('661794 total\_samples/512\_hop ~=1292.56, 1293 by expeirmentdd2=')              # print(dd2)                # print('data.size=')              # print(data.size)              # d1,d2,d3 = data.shape              # print('d1=')              # print(d1)              # print('d2=')              # print(d2)              # print('d3=')              # print(d3)                  # print('len(data[i, :,0]=')              # print(len(data[i, :, 0]))                # print('i=')              # print(i)                # print('Total num of files in ..\gztan\\_train\=len(list\_of\_audiofiles),=')              # print(len(list\_of\_audiofiles))              # input('pause') |