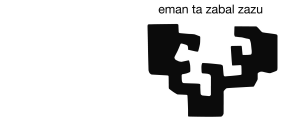

University of the Basque Country

Machine Learning & Neural Networks

P66: Deep learning models for music generation

Xabier de Domingo Yesa
Jon Pérez Etxebarria

2020/2021

1 Description of the problem

The task we have to solve is to solve a classification problem concerning music. Given the sampling of some segments of a song, we have to classify it in oine of the ten genres of our dataset. This dataset is the GTZAN dataset, which contains 10 genres and each song for each genre, making the data perfectly balanced.

As the instances are songs, they do not have any attribute, so each song just has itself and its genre. The genres are:

- Blues
- Classical
- Country
- Disco
- Hip hop
- Jazz
- Metal
- Pop
- Reggae
- Rock

In order to tackle this problem, we have defined two DNNs, a CNN and a RNN, which will be explained in detail below.

2 Description of our approach

We divided the description of our project's approach in the following parts:

1. Preprocessing of the data.
2. Definition of different DNNs for getting the best result.
3. Failed experiments.

2.1 Preprocessing of the data

The data was loaded using *librosa* package. As we can see in Figure 1, sound signals are not continuous when they are digitalized, samples are taken through the time for representing the signal the best possible. Thus, we took each song's wave amplitude at the first 660000 samples for the representation of the data. Thus, each song is represented by an numpy array of 660000 floats.

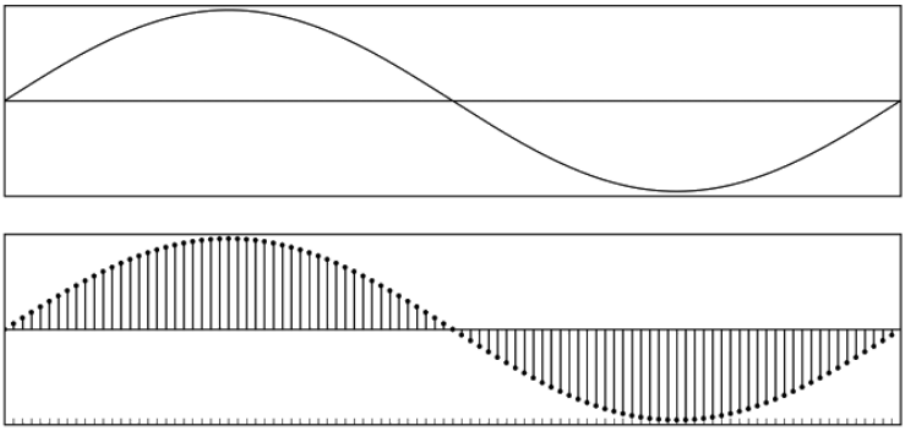

Figure 1: Above, common representation of a signal. Below, how a digitalised signals sample the data

As another approach, we tried to convert each song into an image: the audio's spectrogram. This is explained in the *2.3 Failed experiments* section.

The songs are sorted by genre, so we decided to mix them up for the DNNs to learn better. Also, we made the class to be given as one-hot array instead of by the genre's name, so the genres would have the following representation:

- [1 0 0 0 0 0 0 0 0 0] for blues.
- [0 1 0 0 0 0 0 0 0 0] for classical.
- [0 0 1 0 0 0 0 0 0 0] for country.
- [0 0 0 1 0 0 0 0 0 0] for disco.
- [0 0 0 0 1 0 0 0 0 0] for hip hop.
- [0 0 0 0 0 1 0 0 0 0] for jazz.
- [0 0 0 0 0 0 1 0 0 0] for metal.
- [0 0 0 0 0 0 0 1 0 0] for pop.
- [0 0 0 0 0 0 0 0 1 0] for reggae.
- [0 0 0 0 0 0 0 0 0 1] for rock.

The songs are too long, so we splitted each song in 100 parts, and then we splitted those 100 parts from each song in another 60 parts, so we had each song divided in 6000.

We took out 5 songs of each genre for the test, so the model learnt 95 songs for each genre. The validation was included in the train data, but that is not a problem since we already excluded the test data before.

2.2 Creating the neural networks

We used the package keras for creating our neural networks. We created different neural networks for taking the one with the best results and for comparing each one's accuracy. We decided to name all of the neural networks after a famous classic

musician.

CNN: Johann Sebastian Bach

We created a CNN called Johann Sebastian Bach, whose summary is the one we can see in Figure 2.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 56, 100)           55100
_________________________________________________________________
dropout (Dropout)            (None, 56, 100)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 54, 300)           90300
_________________________________________________________________
dropout_1 (Dropout)          (None, 54, 300)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 53, 180)           108180
_________________________________________________________________
dropout_2 (Dropout)          (None, 53, 180)           0
_________________________________________________________________
flatten (Flatten)            (None, 9540)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 9540)              0
_________________________________________________________________
dense (Dense)                (None, 10)                95410
=================================================================
Total params: 348,990
Trainable params: 348,990
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Summary of Johann Sebastian Bach

As can be seen, we added dropout after each convolutional layer. The dropout was completely necessary in our neural network, without it the neural network overfitted with the train data and the validation error grew a lot. After implementing the dropout, the validation error did not stop decreasing.

Also, we tried adding max pooling to the neural network, but the results were as good as without it, so we decided not to use it as it only made the program to take more time.

The number of neurons for each layer was decided by trial and error, after running the code a few times we saw that these values worked well for this task.

The Johann Sebastian Bach CNN is explained in-depth in the code.

RNN: Ludwig van Beethoven

We also created a RNN called Ludwig van Beethoven for resolving this task, but it was not as good as the CNN. As in the CNN, we can observe in the Figure 3 that after each layer, we used dropout for not overfitting the model.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 60, 100)           84400
_________________________________________________________________
dropout (Dropout)            (None, 60, 100)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 60, 100)           80400
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 100)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 10)                1010
=================================================================
Total params: 246,210
Trainable params: 246,210
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Summary of Ludwig van Beethoven

We used 3 layers of fully connected LSTM cells, 100 in each layer. The activation function we used in the last layer is softmax, not using any in the previous layers. We decided to use a look back of 60. As with the CNN, we decided to use Categorical Cross-Entropy as the loss function.

The Ludwig van Beethoven RNN is explained in-depth in the code.

2.3 Failed experiments

We tried to approach the project's task with different solutions from the ones we mentioned previously. The main failed experiments we tried to apply are the following ones:

1. Spectrograms as input
2. GAN

Spectrogram images as input

As we mentioned before, we thought about trying to process the data in a different way than we did previously, converting each song into an array of floats. This new approach consisted in processing the spectrogram of each song as an image, as we can see in Figure 4.

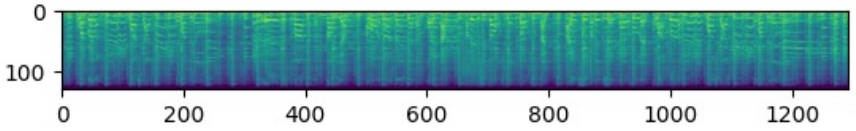

Figure 4: Example of the spectrogram of a song

It did not work as expected: the processing of the data had an expected time of arrival of 40 hours, as it had to create a spectrogram for each of the 1000 songs. Because of this, we decided not to use this method.

GAN

We also tried to generate music from the songs we have in the database, so we implemented a GAN. Nonetheless, this neural network did not work properly as the database is too short and it is not aimed at the generation of new data.

Despite of that, we managed to have an output as a result, but it was not what we wanted: the generated song was almost a unique static song, so we decided not to present this neural network as part of our solution for the project.

3 Implementation

All the project's code was implemented in Python, in two jupyter notebooks. We used the *numpy* and *librosa* packages for loading the data, *tensorflow* and *keras* for creating the neural networks, *mathplotlib* for visualising the errors of the NNs, *sklearn* for using the *train_test_split* function and *os* for using local paths.

The code is available in the notebooks *Bach* (code of the CNN) and *Beethoven* (code of the RNN).

4 Results

4.1 CNN results

The CNN's loss function representation can be observed in the figures 5 and 6. We can deduce that, more the epochs, less the loss. Anyways, using more epochs was not very useful as the CCE does not decrease much more and it may take too long to finish the training.

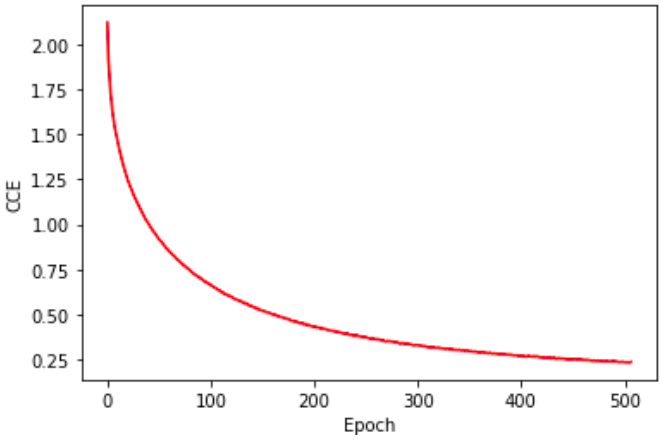

Figure 5: CCE for the train data through the epochs.

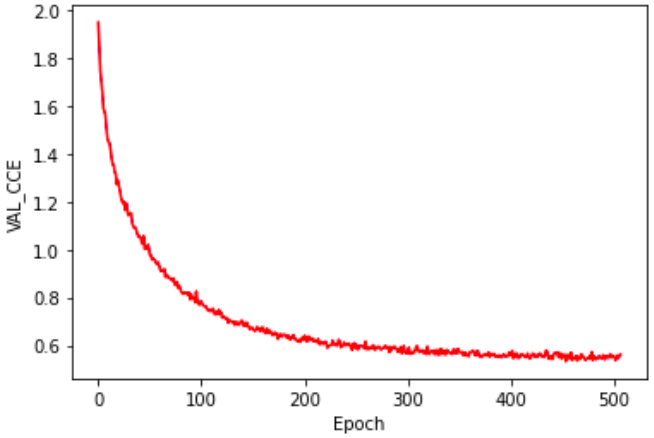

Figure 6: CCE for the validation data through the epochs.

The accuracy we got is 0.8258. This accuracy represents the probability for each one of the 100 parts the song was splitted into to be correctly classified. Thus, the higher probability of having a song incorrectly classified is, assuming that every incorrect class has the same probability:

$$\left(\frac{1-0.8258}{9}\right)^{11} = 1.4284402 \times 10^{-19}$$

Due to this low probability of classifying incorrectly a song, we can say we have almost an accuracy of 1. This is also thanks to the dropout, used for avoiding overfitting in our model.

We took out 5 songs of each genre before the training for the test. Some of the results of this test can be observed in Figure 7.

```
[  0.   0.   2.  92.   1.   0.   0.   1.   0.   4.]
[  0.   0.   1.  96.   0.   0.   1.   0.   0.   2.]
[  0.   0.   1.  90.   0.   0.   0.   9.   0.   0.]
[  0.   0.   0.   4.  71.   0.  16.   8.   1.   0.]
[  8.   2.   0.  33.   0.   0.  45.   0.   0.  12.]
```

Figure 7: Results of the classification

These results belong to the 5 disco songs used for the test. The row represents each song, and the columns represent the genre (the order of the genres is the same we used for mentioning them in the first section). Thus, the numbers represent how many parts of the song were classified in that genre. We can observe that the first three songs were correctly classified, having each one 92, 96 and 90 parts assigned in their genre. On the other hand, the fourth and fifth songs are not correctly classified. The fourth was classified as a hip hop song and only had 4 parts of it in the correct class, while the fifth song was classified as a metal song but having at least a third part of its fragments correctly classified.

From the 50 songs we used for the test, we classified correctly 45 out of them, the CNN has an accuracy of 0.9.

4.2 RNN

As with the CNN, we used the Categorical Cross-Entropy as loss function. We can see the representations of this loss in the figures 8 and 9.

As the CNN, the RNN has a high accuracy for predicting songs's parts: 0.8344. Even if it is higher than the CNN's one, the final accuracy for classifying the entire song is lower, because of the variance in the RNN.

5 Conclusions

As a conclusion, we can observe that the CNN learns enough for a good classification with 100 epochs, but if we want the best results, it is better to get close to the 600 epochs. Thus, the training can take too long. Running the codes in the jupyter notebooks may take a lot of time. We used a computer with an i9 9900k as CPU

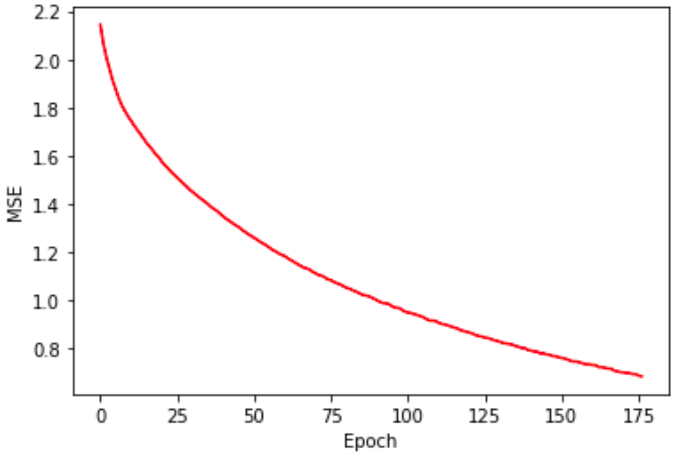

Figure 8: CCE for the train data through the epochs.

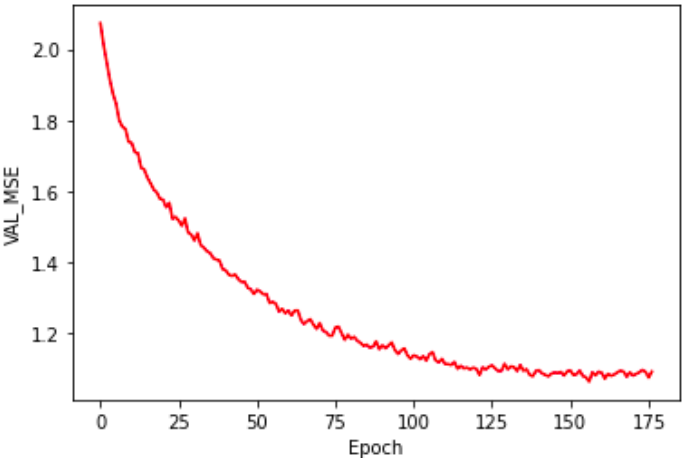

Figure 9: CCE for the validation data through the epochs.

and a GTX 1080Ti as GPU.

We also observed that many other solutions both the one from our classmates and other found on some papers used the spectrogram approach. We looked at their results and concluded that those were too costly and in many times far worse than our 90% accuracy. It is thus than we conclude the spectrogram is harder to classify that the sample array most likely due the higher complexity of the information thus demanding more power and more complex DNNs.

Finally, we noticed that the CNN is better than the RNN and trains faster, that is why we chose it as our first solution. Furthermore, it is remarkable how important is to preprocess the data correctly, being this even more important than how the neural networks are created.

6 Bibliography

GTZAN Genre Collection

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition,

Leland Roberts. Understanding the Mel Spectrogram.

Martin F. McKinney, Jeroen Breebaart. Features for Audio and Music Classification.

Keunwoo Choi, Gyorgy Fazekas, Mark Sandler. Explaining deep convolutional neural networks on music.