MACHINE LEARNING PROJECT
Multi-step generation of chord progressions for inference in jazz through recurrent neural networks

Valentin Bilot   Paul Lascabettes
Grégoire Blanc   Hugo Manet

January 2019

Abstract

This project aims to generate chord progressions of jazz music, represented as co- herent chord label sequences with the help of probabilistic models. The motivation for this approach comes from a recent article on text-based LSTM networks for automatic music composition Choi et al. [2016]. In this paper, the authors use a recurrent neural network (RNN-LSTM) to generate symbolic chord sequences. They focus on two different approaches named word-RNN and char-RNN. These two variants use the same model architecture but rely on different learning methods. In this project, we will improve the word-RNN approach by doing multi-step prediction and by injecting music theory knowledge through the learning method in order to be able to perform accurate prediction of chord sequence and jazz melody generation. Ultimately, this project could be used to perform automatic accompaniment and improvisation.

Contents

1 Introduction

Computing highly non-linear functions, handling complex datasets, etc. Deep learning techniques have proven to be ways more powerful than HMM (hidden markov models) on these points. This explains why the last research on symbolic music generation is now mostly based on neural networks.

1.1 Context of the project

In this project, we were willing to train a model on the real book dataset, composed of almost 3000 chord progression lists from the well-known real book - which is a strong reference in jazz music. We used Python and PyTorch to elaborate our model.

The different steps were the following :

- Understanding the basis of PyTorch/Python RNN implementation
- Implementing the code as described in the reference article [1], and transferring the algorithm from Keras to Pytorch to prepare further development
- Setting inputs from music theory (eg.: chord distances) to improve the cost function
- Trying out adversarial networks
- Develop, comment, experiment, discover the joyful world of machine learning and AI

1.2 Composition of the database

The database we used in the beginning of the project is basically a text file containing 2847 sequences of chords. For simplification purposes, each repetition of the chord is considered as a quarter note : no bars, no rythm etc. A sequence can look like the following one :

```
_START_ C:maj C:6(9) F:6(9) F:6(9)
G:aug(b7) G:aug(b7) F:maj(b7,9,11,13)
F:maj(b7,9,11,13) E:min9 E:min9 F:min9
E:aug(b7,9) D#:maj9 D#:maj9 D:min(b7,9,11)
G:aug(b7) C:maj G:7/3 A:min F#:hdim
G:aug(b7) G:aug(b7) F:9 F:maj(b7,9,11,13)
E:min7 E:min7 D#:dim D#:dim D:min(b7,9,11)
D:min(b7,9,11) _END_
```

Sequences are separed by `_START_` and `_STOP_` markers.

The database was also available in another form. Each song included in a different `.xlab` file, containing rythm data as well. For instance the beginning of the famous standard *Autumn Leaves* is written down in this way:

```
1:1 0.0 2 0.8275862068965518 G:min7 Gm7 F
1:3 0.8275862068965518 2 1.6551724137931036 C:7 C7 F
2:1 1.6551724137931036 4 3.3103448275862073 F:maj F F
```

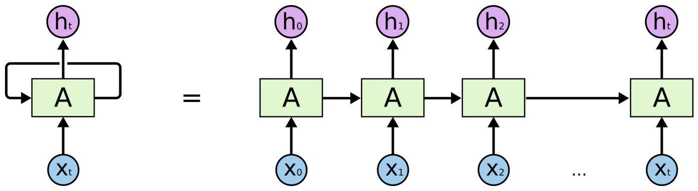


Figure 1: Basic schematic of RNN architecture

```
3:1 3.3103448275862073 2 4.137931034482759 G:min7 Gm7 F
3:3 4.137931034482759 2 4.9655172413793105 A:7 A7 F
...
```

We didn't make use of these complementary information - raw chord sequences was enough for this first approach.

1.3 Chord vocabulary - alphabet

The chords mentioned in the database include a wide range of variations. Beside natural chords (for instance `C`, `D`, `F`, `A`,...) modal variations are considered (`maj` and `min`) as well as seventh, augmented, diminished, suspended chords, etc.

Labels are being reduced and clustered in different dictionaries, built to depict various layers of precision. Harmonic enrichments are indeed not always important to consider.

We named the dictionaries A_i, referencing to A_0 which contained the original labeled chords of the database. For instance A_1 contains only major, minor and diminished chords. The label `N` was used to refer to non-existent chords (for instance, seventh chords in A_1 are referred by `N`).

1.4 Neural networks used in the project

1.4.1 RNN

Recurring Neural Networks is a very common neural network architecture, including some recurring loops in the system. It's a way to include some kind of persistence: previous results are taken into account. Unfolding a recursive loop reveals a chain architecture. Intuitively we an therefore understand why RNN are especially coherent to handle sequences and listed data (cf figure 1).

1.4.2 LSTM

"LSTM" stands for *Long Short-Term Memory*. It's a refined kind of RNN more suitable to handle long-term dependencies. Without getting into the details, LSTM have additional parameters that control when and how the memory is being updated. The module responsible for repetition is changed to a very different structure consisting of four layers connected together.

1.4.3 GRU

Gated Recurrent Units - GRUs are a more recent RNN architecture, introduced about five years ago. It's in fact a variation of LSTM involving less parameters and thus ways faster to train, especially on small datasets (see the accuracy and loss functions we obtained on such networks: fig 9 and 10). However the results are proven to be better on the long run with well-trained LSTM.

2 Elaborating the RNN generator

The very first step consists into transforming the input data into workable elements. All the chord sequences from the dataset are then converted into vectors, on which the network will be able to train.

RNN, LSTM and GRU cells are created as Python classes. the train function and the network in itself are built around these classes as it can be seen on the github repository of the project [7].

The sequences generated are working on a probabilistic basis. An output step consists of a probability vector describing the next chord. There are two possibilities:

- Setting the generated chord according to the highest probability
- Setting the generated chord via a random sampling over the output vector

The sampling technique is the one chosen in the project. Technical details about the implementation of the networks are not commented in this report, and we invite you to read more about this directly on [7].

3 Improving the cost function

3.1 Implementing chord distances in the loss function

In order to generate a chord progression, it is necessary to determine a certain proximity between chords. In fact, chord sequences follow some rules, for example after a `C` we are more used to hear a `a` than a `C#` or a `c`.Consequently, we need to understand the relation between each chords, for this purpose we will determine a distance between two different chords.

Our study is based on the octave-equivalence system, that is to say we identify notes whose interval distance is equal to a multiple of an octave, for instance the A of 440Hz is therefore identify with the A of 880Hz or 220Hz. Moreover, we will represent the twelve musical notes by an elements of $\mathbb{Z}_{12}$. In this way, we will work on the circle of fifths which is described in fig 2.

3.2 Distance between two chords of n notes

A distance or metric is a function with values in $\mathbb{R}^+$ which must respect the three following properties : symmetry, non-negativity and triangle inequality. We will define a metric on chords space with n notes which can be mathematically represented by $\mathbb{Z}_{12}^n$. For example, `major C` which is {`C,E,G`} can be see as the set $\{0, 1, 4\}$.

Let two chords of n notes $A = \{a_1, ..., a_n\}$ and $B = \{b_1, ..., b_n\}$, we define the distance $d : \mathbb{Z}_{12}^n \times \mathbb{Z}_{12}^n \longrightarrow \mathbb{R}^+$ between these chords by :

$$d(A,B) \ = \ \min_{\phi \ \in \ S_n} \left(\sum_{i=1}^{n} \mid a_i - b_{\phi(i)} \mid \right)$$

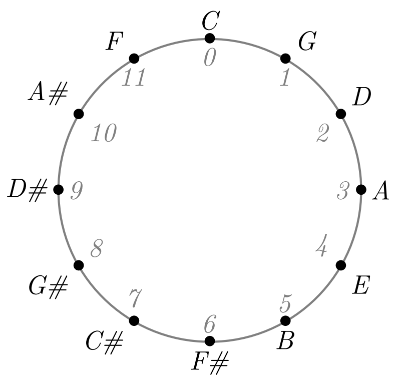


Figure 2: Musical notes represented by elements of $\mathbb{Z}_{12}$ in the circle of fifths

Where S_n is the symmetric group, that is to say ϕ is a bijective function from $\{1, ..., n\}$ to $\{1, ..., n\}$. Moreover, the difference in the absolute value is in $\mathbb{Z}_{12}$ but the sum in $\mathbb{R}^+$, thus, we project the element of $\mathbb{Z}_{12}$ inside of the absolute value by a number of $\{0, ..., 11\} \subset \mathbb{R}^+$. It is clear that this distance satisfies the symmetry and the non-negativity, furthermore we have shown that the triangle inequality is verified. Let try this distance on some chords. We have exposed the relation between `C` and `a` or `C#`, they are respectively represented by $\{0, 1, 4\}$, $\{0, 3, 4\}$ and $\{7, 8, 11\}$. Thus, we have :

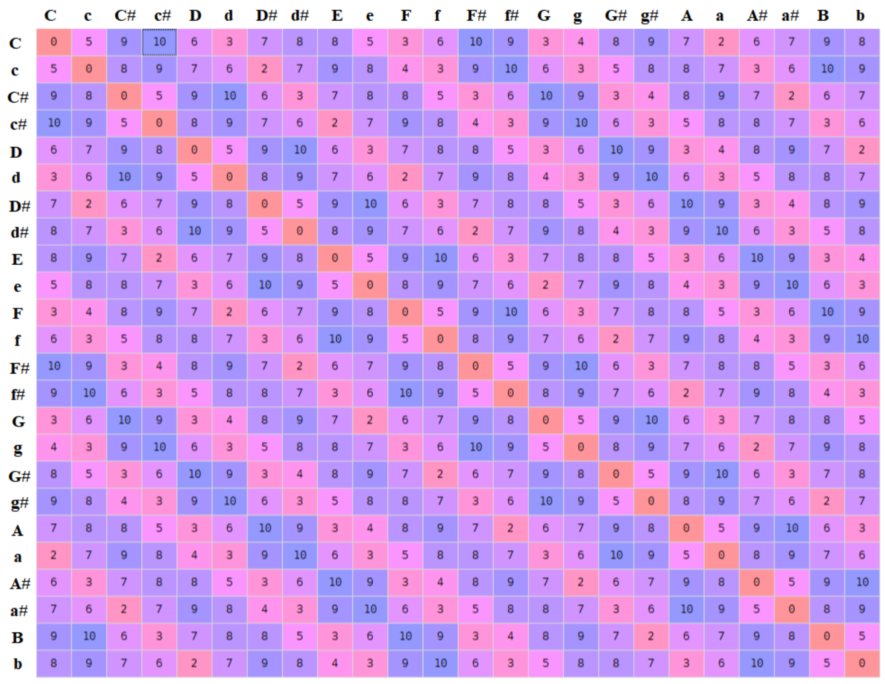
- $d(C, a) = \mid 0 - 0 \mid \; + \; \mid 1 - 3 \mid \; + \; \mid 4 - 4 \mid = 2$
- $d(C, C\#) = \mid 0 - 11 \mid \; + \; \mid 1 - 8 \mid \; + \; \mid 4 - 7 \mid = 1 + 5 + 3 = 9$

We summarize in the table 1 the distance between the chord `C` and the 24 others minor and major chords. The circle of fifths allows to associate coherent musical chords with `C`.Thus, we can determine according to our metric the chords that are closest to `C`, they are : `C`, `d`, `e`, `F`, `G`, `a`.

Table 1: Distance between C and the 24 others minor and major chords

Chords	Distance	Chords	Distance
C	0	c	5
C#	9	c#	10
D	6	d	3
D#	7	d#	8
E	8	e	5
F	3	f	6
F#	10	f#	9
G	3	g	4
G#	8	g#	9
A	7	a	2
A#	6	a#	7
B	9	b	8

To generalize this idea, we have represented in fig 3 the distance between two minor or major chords. The color attributed to each values allows to visualize some patterns into this table.



	C	c	C#	c#	D	d	D#	d#	E	e	F	f	F#	f#	G	g	G#	g#	A	a	A#	a#	B	b
C	0	5	9	10	6	3	7	8	8	5	3	6	10	9	3	4	8	9	7	2	6	7	9	8
c	5	0	8	9	7	6	2	7	9	8	4	3	9	10	6	3	5	8	8	7	3	6	10	9
C#	9	8	0	5	9	10	6	3	7	8	8	5	3	6	10	9	3	4	8	9	7	2	6	7
c#	10	9	5	0	8	9	7	6	2	7	9	8	4	3	9	10	6	3	5	8	8	7	3	6
D	6	7	9	8	0	5	9	10	6	3	7	8	8	5	3	6	10	9	3	4	8	9	7	2
d	3	6	10	9	5	0	8	9	7	6	2	7	9	8	4	3	9	10	6	3	5	8	8	7
D#	7	2	6	7	9	8	0	5	9	10	6	3	7	8	8	5	3	6	10	9	3	4	8	9
d#	8	7	3	6	10	9	5	0	8	9	7	6	2	7	9	8	4	3	9	10	6	3	5	8
E	8	9	7	2	6	7	9	8	0	5	9	10	6	3	7	8	8	5	3	6	10	9	3	4
e	5	8	8	7	3	6	10	9	5	0	8	9	7	6	2	7	9	8	4	3	9	10	6	3
F	3	4	8	9	7	2	6	7	9	8	0	5	9	10	6	3	7	8	8	5	3	6	10	9
f	6	3	5	8	8	7	3	6	10	9	5	0	8	9	7	6	2	7	9	8	4	3	9	10
F#	10	9	3	4	8	9	7	2	6	7	9	8	0	5	9	10	6	3	7	8	8	5	3	6
f#	9	10	6	3	5	8	8	7	3	6	10	9	5	0	8	9	7	6	2	7	9	8	4	3
G	3	6	10	9	3	4	8	9	7	2	6	7	9	8	0	5	9	10	6	3	7	8	8	5
g	4	3	9	10	6	3	5	8	8	7	3	6	10	9	5	0	8	9	7	6	2	7	9	8
G#	8	5	3	6	10	9	3	4	8	9	7	2	6	7	9	8	0	5	9	10	6	3	7	8
g#	9	8	4	3	9	10	6	3	5	8	8	7	3	6	10	9	5	0	8	9	7	6	2	7
A	7	8	8	5	3	6	10	9	3	4	8	9	7	2	6	7	9	8	0	5	9	10	6	3
a	2	7	9	8	4	3	9	10	6	3	5	8	8	7	3	6	10	9	5	0	8	9	7	6
A#	6	3	7	8	8	5	3	6	10	9	3	4	8	9	7	2	6	7	9	8	0	5	9	10
a#	7	6	2	7	9	8	4	3	9	10	6	3	5	8	8	7	3	6	10	9	5	0	8	9
B	9	10	6	3	7	8	8	5	3	6	10	9	3	4	8	9	7	2	6	7	9	8	0	5
b	8	9	7	6	2	7	9	8	4	3	9	10	6	3	5	8	8	7	3	6	10	9	5	0

Figure 3: Distance between two minor or major chords

3.3 Distance between any chords

In this section, we will try to generalize the distance by comparing two chords of n and m notes. Let $A = \{a_1, ..., a_n\}$ and $B = \{b_1, ..., b_m\}$ two chords, and let suppose $n \leq m$. The metric $d : \mathbb{Z}_{12}^n \times \mathbb{Z}_{12}^m \longrightarrow \mathbb{R}^+$ is now define by :

$$d(A, B) \;=\; \min_{\phi \,\in\, S_m \;,\; a_j \in A, \forall j \in \{n,...,m\}} \left(\sum_{i=1}^{n} \mid b_{\phi(i)} - a_i \mid + \sum_{j=n+1}^{m} \mid b_{\phi(j)} - a_j \mid \right)$$

The global idea is to add notes to the chord A it already contains to obtain a chords of m notes. The previous formula can then be applied. For example, the distance between the chords $\{C, E, G\}$ and the chords $\{C, G\}$ represented by $\{0, 1\}$ and $\{0, 1, 4\}$ is 3. We compute this metric in order to generate seventh or ninth chords with minor or major chords.

3.4 Impact on the performances

The impact on the results is quite valuable. If we consider the impact on the accuracy, the results seems to be quite equivalent (fig. 5 and 6).

However, the loss function seems to decrease faster with the customised distance function (fig 7 and 8).

We must admit that it is not obvious, the distance effect might be more visible on more complex dictionaries, and we just implemented the distance function on A_0 dictionary for now.

4 Adversarial networks

Let's now consider Generative Adversarial Networks (GANs). A GAN consists of two entities working in parallel: a *generator* and a *discriminator*. The generator generates, the discriminator discriminates. Simple. The aim of the generator is to create the most valuable samples while the discriminator trends to be as good as possible in determining whether the sample is a real one or a generated one (see figure 4)

The generator takes a sequence from the database and send it to a RNN (which can be as mentioned before a simple RNN, a LSTM or a GRU). Then, the last hidden state of the RNN goes to a fully connected (linear) layer that generates a chord. This chord is added to the sequence, while the first chord of the sequence is removed. We then repeat this process until we generate a new sequence of the desired length.

The discriminator takes a sequence of chords for input, this sequence goes to a RNN, and as before, the last layer of the hidden state of the RNN is used as an input for a linear layer. This linear layer returns a scalar. The discriminator will return:

- 1 if the input data is coming from the dataset
- 0 if it was generated by the generator

Unfortunately, our implementation is not working yet, the back-propagation through the generator seems to have no effect. The generator does not updates weights while training on the discriminator network...

5 Commenting the results obtained

From a musical point of view it is quite satisfying to have obtained some coherent results. If we take a LSTM, 2 layers and 512 hidden blocks we can generate chord sequences with various tastes of coherence and variability depending on the training level of our network.

The maximum accuracy we can obtain is around 70%. For this training level we can get this kind of boring sequence, featuring two chords only:

```
input:
    ['C:maj', 'F:maj', 'C:maj', 'C:maj']
    ['F:maj', 'F:maj', 'C:maj', 'C:maj']
    ['G:maj', 'F:maj', 'C:maj', 'G:maj']
    ['C:maj', 'C:maj', 'F:maj', 'G:maj']
```

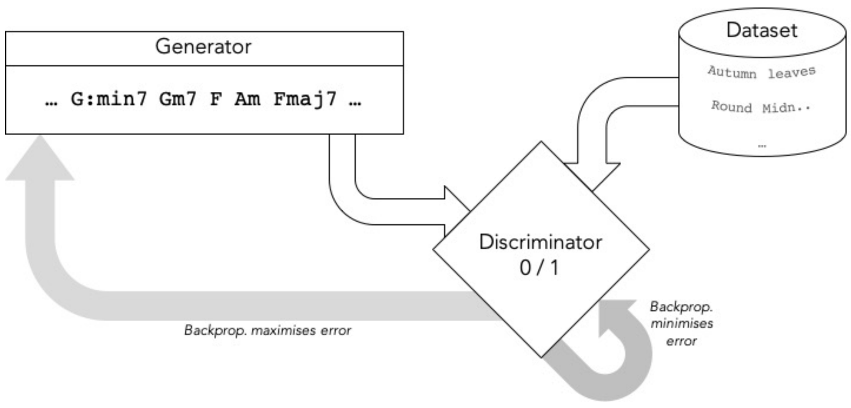

Generator
… G:min7 Gm7 F Am Fmaj7 …
Dataset
Autumn leaves
Round Midn..
…
Discriminator
0 / 1
Backprop. maximises error
Backprop. minimises error

Figure 4: Basic schematic of GAN concept

```
output:
    ['G:maj', 'C:maj', 'G:maj', 'C:maj']
    ['G:maj', 'C:maj', 'G:maj', 'C:maj']
    ['C:maj', 'C:maj', 'C:maj', 'G:maj']
    ['G:maj', 'C:maj', 'C:maj', 'C:maj']
    ['C:maj', 'C:maj', 'C:maj', 'G:maj']
    ['C:maj', 'C:maj', 'C:maj', 'C:maj']
    ['C:maj', 'G:maj', 'C:maj', 'C:maj']
    ['C:maj', 'C:maj', 'C:maj', 'C:maj']
    ['C:maj', 'C:maj', 'C:maj', 'G:maj']
    ['C:maj', 'C:maj', 'C:maj', 'C:maj']
    ['G:maj', 'C:maj', 'G:maj', 'C:maj']
    ['G:maj', 'C:maj', 'C:maj', 'C:maj']
```

Just to compare, generating a sequence with the same network trained at 60% accuracy provides something less coherent but also less steady:

```
input:
    ['C:maj', 'F:maj', 'C:maj', 'C:maj']
    ['F:maj', 'F:maj', 'C:maj', 'C:maj']
    ['G:maj', 'F:maj', 'C:maj', 'G:maj']
    ['C:maj', 'C:maj', 'F:maj', 'G:maj']

output:
    ['C:maj', 'F:maj', 'C:maj', 'F:maj']
    ['N', 'A#:maj', 'A#:maj', 'F:maj']
    ['C:maj', 'F:maj', 'F:maj', 'F:maj']
    ['A#:maj', 'A#:maj', 'N', 'F:maj']
    ['F:maj', 'F:maj', 'F:maj', 'F:maj']
    ['F:maj', 'F:maj', 'F:maj', 'F:maj']
    ['F:maj', 'F:maj', 'A#:maj', 'F:maj']
    ['N', 'A#:maj', 'F:maj', 'F:maj']
    ['C:maj', 'F:maj', 'C:maj', 'A#:maj']
    ['C:maj', 'F:maj', 'F:maj', 'A#:maj']
```

```
        ['F:maj', 'F:maj', 'A#:maj', 'F:maj']
        ['F:maj', 'D#:maj', 'F:maj', 'A#:maj']
```

That was on A_0 dictionary, and the distance function was bypassed. If we now consider our distance function, there is more coherence even at an earlier accuracy rate. An example below for a 56% accuracy rate:

```
input:
    ['C:maj', 'F:maj', 'C:maj', 'C:maj']
    ['F:maj', 'F:maj', 'C:maj', 'C:maj']
    ['G:maj', 'F:maj', 'C:maj', 'G:maj']
    ['C:maj', 'C:maj', 'F:maj', 'G:maj']
output:
    ['G:maj', 'N', 'C:maj', 'C:maj']
    ['C:maj', 'C:maj', 'G:maj', 'G:maj']
    ['C:maj', 'C:maj', 'C:maj', 'C:maj']
    ['C:maj', 'C:maj', 'C:maj', 'G:maj']
    ['D:min', 'G:maj', 'C:maj', 'C:maj']
    ['G:maj', 'C:maj', 'G:maj', 'G:maj']
    ['G:maj', 'N', 'C:maj', 'G:maj']
    ['G:maj', 'C:maj', 'G:maj', 'C:maj']
    ['G:maj', 'C:maj', 'C:maj', 'G:maj']
    ['G:maj', 'C:maj', 'G:maj', 'C:maj']
    ['C:maj', 'G:maj', 'C:maj', 'G:maj']
    ['C:maj', 'G:maj', 'G:maj', 'E:min']
```

But that's a matter of taste of course, there's a lot of subjectivity to be put in the interpretation of the results.

6 Conclusion

To conclude, big efforts have been invested into this project. Generating music through AI is indeed such a fascinating subject. Even on a very restricted problem like this one (restrained chord dictionnary, simple sequences, small dataset, etc.) the quantity of parameters involved can be tremendously huge. Fine-tuning is the key to good results.

Going more in depth on the GAN side with "professor forcing" implementation would be interesting for further work, as well as a better consideration of toy datasets.

References

[1] *Text-based lstm networks for automatic music composition*, Keunwoo Choi, George Fazekas, and Mark Sandler, 2016.

[2] *A probabilistic model for chord progressions*, Jean-François Paiement, Douglas Eck, and Samy Bengio, ISMIR, 2005.

[3] *Generating sequences with recurrent neural networks*, Alex Graves, 2013.

[4] *Towards automatic extraction of harmony information from music signals*, Christopher Harte, PhD Thesis, 2010.

[5] *Professor forcing: A new algorithm for training recurrent networks*, Alex M Lamb, Goyal Anirudh, Ying Zhang, Saizheng Zhang, Aaron C. Courville, and Yoshua Bengio, 2016.

[6] *A graphical model for chord progressions embedded in a psychoacoustic space*, Jean-François Paiement, Douglas Eck, Samy Bengio, and David Barber, 2005.

[7] Github repository of our project :
`https://github.com/ValentinBilot/Jazz_generator_ML_ATIAM`

Appendices

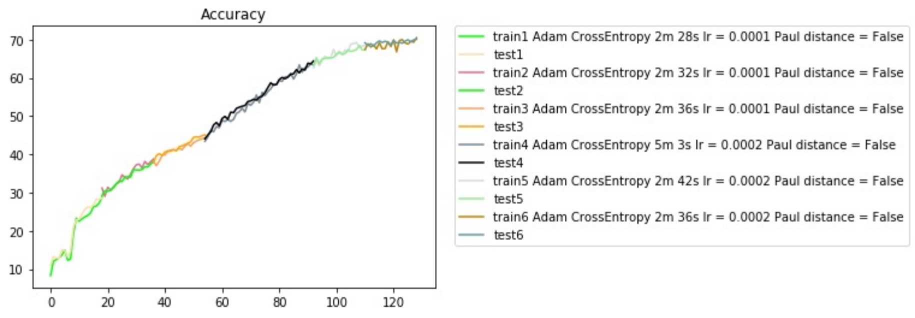


Figure 5: [LSTM] Mapping the accuracy obtained while bypassing the custom-distance function

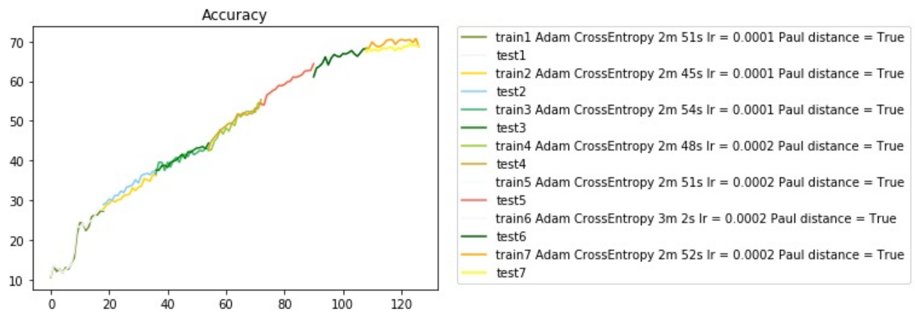


Figure 6: [LSTM] Mapping the accuracy obtained with the custom-distance function

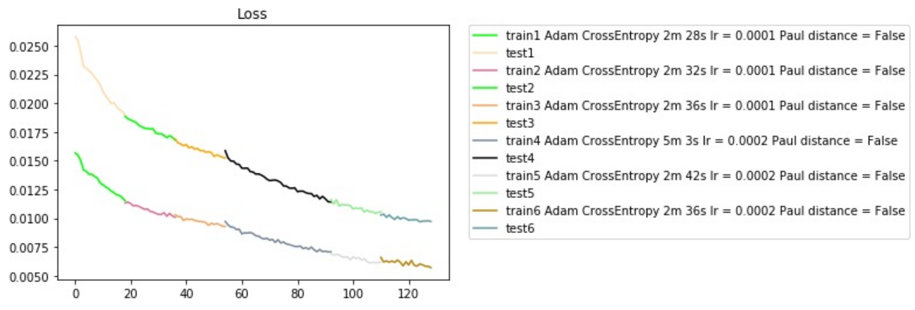


Figure 7: [LSTM] Mapping the accuracy obtained while bypassing the custom-distance function

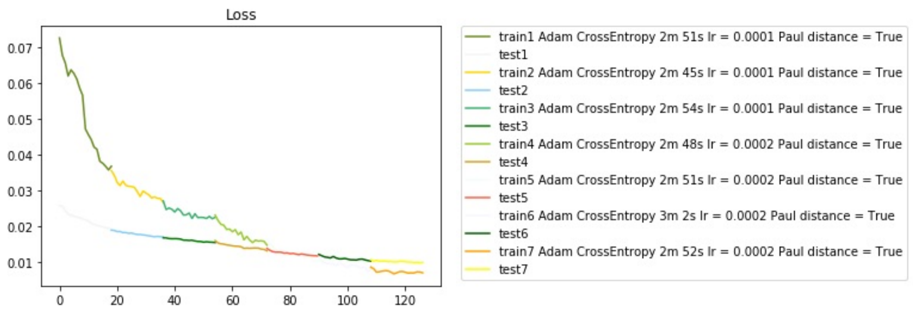


Figure 8: [LSTM] Mapping the loss obtained with the custom-distance function

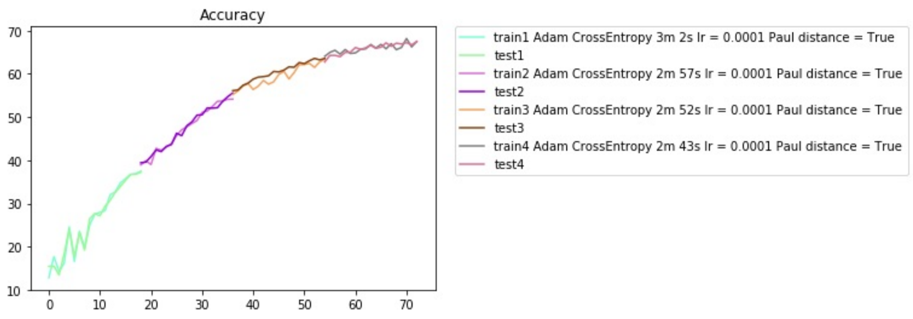


Figure 9: [GRU] Mapping the accuracy obtained with the custom-distance function

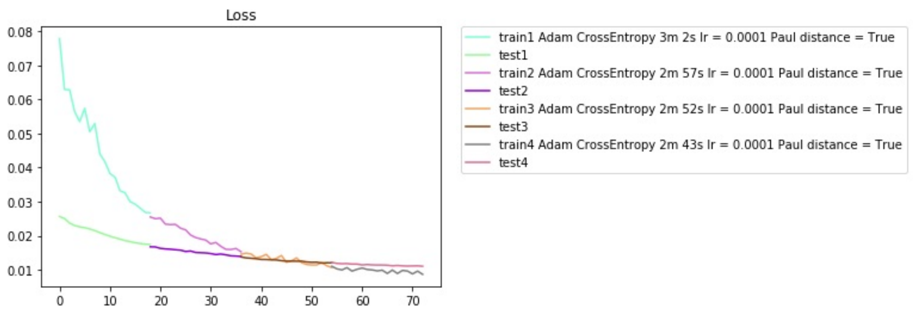


Figure 10: [GRU] Mapping the loss function obtained with the custom-distance function