Automatic Music Generation by Deep Learning

Juan Carlos García, Emilio Serrano*
juancarlos.garcia.torrecilla@alumnos.upm.es, emilioserra@fi.upm.es

Ontology Engineering Group, Universidad Politécnica de Madrid, Spain

Abstract. This paper presents a model capable of generating and completing musical compositions automatically. The model is based on generative learning paradigms of machine learning and deep learning, such as recurrent neural networks. Related works consider music as a text of a natural language, requiring the network to learn the syntax of the sheet music completely and the dependencies among symbols. This involves a very intense training and may produce overfitting in many cases. This paper contributes with a data preprocessing that eliminates the most complex dependencies allowing the musical content to be abstracted from the syntax. Moreover, a web application based on the trained models is presented. The tool allows inexperienced users to generate automatic music from scratch or from a given fragment of sheet music.

Keywords: Automatic Music, Deep Learning, Recurrent Neural Networks

1 Introduction

Machines carry out a number complex tasks and processes whose automation was unthinkable years ago. Everyday there are more tasks that can be delegated to them. For some of these tasks, such as face recognition, there are not well known steps to undertake them. In these problems, the use of *machine learning* can allow computers to learn a solution from labeled examples. In the last few years, the use of *deep learning*, a subfield of machine learning, has outperformed all previous techniques in a number of problems and fields. More specifically, the use of *Recurrent Neural Networks* (RNN) is the state of the art in speech recognition and other time series problems.

The concept of art is subjective and not clearly defined. Learning creative processes is one of the most challenging problems of artificial intelligence. These processes are typical of human beings, not of living beings in general, which have a creative component that gives them greater complexity. Moreover, it is hard to define correct and incorrect cases that allow machine learning to leverage new knowledge.

This paper presents a study and comparison of several artificial neural networks architectures to model music as a time series prediction. Furthermore,

* ORCID ID: 0000-0001-7587-0703

a responsive web application based on these deep learning models is also presented. The web allows inexperienced users to explore the possibilities of artificial intelligence to enhance their music composition.

The paper revises the related works in section 2. Section 3 describes the methodology followed in the research work and their different phases: data understanding, data preparation, modeling, and evaluation. Section 4 offers the experiments results that are discussed in section 5. The tool implementation is described in section 6. Finally, section 7 concludes and gives future works.

2 Related works

One of the best known related works in automatic music generation is BachBot [17, 14, 15]. This project uses *Long Short-Term Memory* (LSTM) networks to generate and harmonize musical pieces with Batch style. LSTMs are widely used in music generation. This project uses cross entropy as loss function. In a similar vein, DeepBach [12, 11] presents an alternative approach for the same problem that generates musical notes by means of a pseudo-Gibbs sampling procedure.

Magenta [5] google project covers a number of models for music generation such as: Drums RNN [6], Melody RNN [7], Polyphony RNN [10] inspired by BachBot, Performance RNN [8] ane Pianoroll RNN-NADE [9]. This last model combines LSTMs with N*eural Autoregressive Distribution Estimator* (NADE). Most of these models are based on LSTM architectures and all of then employ an encoder/decoder structure to allow a sequence to sequence interface.

The LSTM-NADE combination is also employed by Daniel Johnson [13], whose recent research work employs log-likelihood as cost function. Agarwal et al.[2, 1] also use LSTMs combining them with *Gated Recurrent Unit* (GRU) recurrent neural networks and achieve an accuracy of 65.5%.

Some of these works are open source [13][5][15] and have been studied for the contribution presented in this paper. However, they require a considerable programming and technical expertise to be used for music creation. Therefore, beyond the experimentation with a number of architectures and a new preprocessing method, the web application presented in this paper is an important asset for spreading the artificial intelligence assistance in music composition.

3 Methodology

This research has followed the *Cross Industry Standard Process for Data Mining methodology* (CRISP-DM) [18, 16]. This methodology is composed of 6 phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment. The following sections describe the processes undertaken in these phases over several iterations.

3.1 Data Understanding

The dataset employed in this work is the *Nottingham Music Database*[1], also employed in related works [4][2]. The data is composed of 1037 British and American folk tunes, (hornpipe, jigs, etc.) that was created by Eric Foxley and posted on Eric Foxley's Music Database.

The database was converted to ABC music notation format and was posted on abc.sourceforge.net. The ABC notation is a format of musical notation in ASCII, consisting of a header with metadata and a body with the musical content. The header is composed of several fields, some of the most relevant ones are:

- **T**: Tune title.
- **Q**: Tempo. Default: ♩ = 120.
- **M**: Meter. Default: $\frac{4}{4}$.
- **L**: Unit note length. Default: ♪.
- **K**: Key signature. Default: C.

3.2 Data Preparation

The main goal in this phase has been to eliminate the existing dependencies between the notes and the header making the notes self-contained. The musical content of the body is specially conditioned by the fields K and L, see section 3.1. Therefore, (1) all compositions have been transformed to the default note length; see Figure1a. Subsequently, (2) all the compositions have been transformed to the default key signature, so that the accidents (♯ , ♭) appear explicitly in each note. To avoid alterations between notes of the same bar or measure, the natural tone (♮) has been explicitly indicated in the rest of the notes; see Figure 1b. Finally, since each tone has several representations, (3) the flat notes (♭) have been transformed to their corresponding alternative representation; see Figure 1c. Finally, the symbols that did not provide musical content and accompanying directives were eliminated.

3.3 Modeling

The data was codified using the notes ("=A", "^b", "=c´"...) as temporal symbols instead of the character level approach ("=", "A", "^"...). This prevents neural networks from requiring the syntax of the sheet music to be learned. Padding and masking also were employed to allow RNNs to support training situations including one-to-many, many-to-one, as also support variable length time series in the same mini-batch. Cross entropy is employed as cost function.

The TensorFlow open-source software library was used to model and evaluate different RNN architectures. The first architectures tested count with one

[1] `https://ifdo.ca/~seymour/nottingham/nottingham.html`
[2] `http://www-etud.iro.umontreal.ca/~boulanni/icml2012`

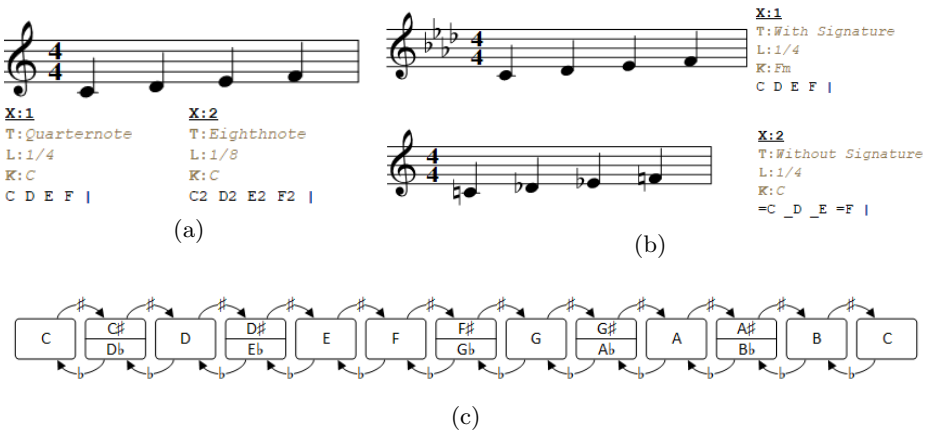

Fig. 1: Data transformations.

recurrent layer of 64 neurons and a fully connected softmax layer as output layer. Different recurrent layers have been assessed: simple RNN, LSTM, and GRU. Several time steps also have been evaluated to be considered in the truncated back propagation: 50 and 100.

- SimpleRNN with 50 timesteps.
- SimpleRNN with 100 timesteps.
- LSTM with 50 timesteps.
- LSTM with 100 timesteps.
- GRU with 50 timesteps.
- GRU with 100 timesteps.

Before these configurations, several experiments were conducted to optimize the hyperparameters. Some of these included different layer widths (16, 32, 64 and 128 neurons) in combination with different block sizes to model the tensors shape (16, 32, 64 and 128 samples).

After evaluating the performance of the different recurrent structures (simple, LSTM, GRU) with 50 and 100 timesteps, tests were performed with two GRU layers and 100 timesteps. An embedding layer was also added at the network input with dense vectors of length 5 and 10. The experiments performed are listed below:

- GRU (64 cells) - GRU (64 cells): 100 timesteps.
- GRU (256 cells) - GRU (128 cells): 100 timesteps.
- Embedding (5 length) - GRU (256 cells) - GRU (128 cells): 100 timesteps.
- Embedding (10 length) - GRU (256 cells) - GRU (128 cells): 100 timesteps.

3.4 Evaluation

Evaluating musical samples is a complex task. There is no standard procedure for evaluating musical quality. However, the standard machine learning eval-

uation splits data into three partitions: training, validation, and test. In this vein, a 60%/20%/20% partition has been undertaken. *Accuracy* is reported in the experiments, i.e. percentage of correct predictions. The test partition is not used at all except for reporting a final accuracy of the chosen model based on experiments over the training and validation data.

4 Results

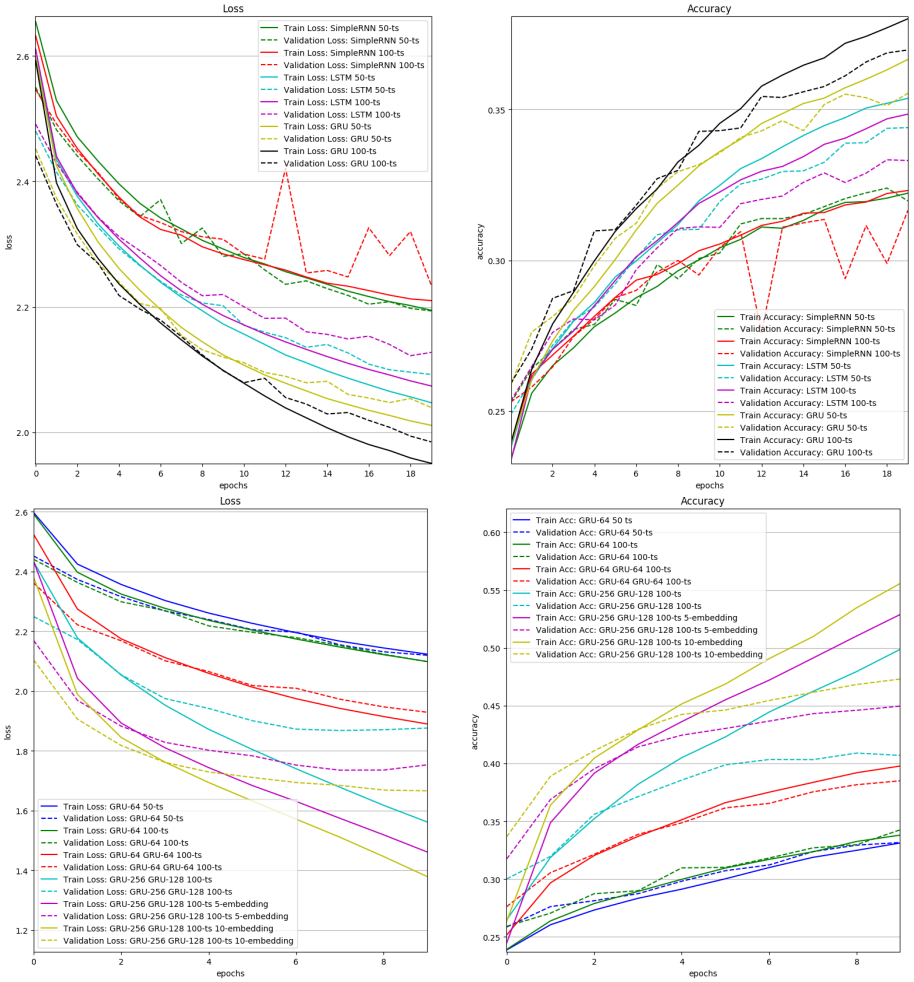

Fig. 2: Experiments with one hidden layer (above) and with two (below).

Figure 2 (above) shows the loss values and accuracy of the experiments performed to compare different recurring layers. All of them employ 64 cells per

layer and have been trained with 64 samples per block. Executions are shown for simple recurring networks, LSTM and GRU with 50 and 100 timesteps, during 20 epochs. Figure 2 (below) details the results for the different GRU RNNs described in section 3.3.

5 Discussion

The simple recurrent models shown in Figure 2 (above) get a similar accuracy regardless of the number of timesteps. This effect may be caused because experimental results show that simple recurrent networks are not able to learn long-term dependencies [3]. In contrast, in LSTM and GRU models there is a slight variation between the test with 50 timesteps and the test with 100 timesteps. In this case, the results show that the GRU models work better than LSTM for the problem addressed, reaching an accuracy around 38% using 100 timesteps. GRU seems to benefit from the increase in timesteps, while LSTM obtains better results in the 50 timesteps test.

Experiments shown in Figure 2 (below) present better validation results than training accuracy with few epochs. However, this tendency is reversed with more epochs presenting the classical overfitting situation. This gap is especially noticeable with the GRU(256)-GRU(128) architecture without embedding layer.

The results with a single GRU layer are quickly improved by the models with two layers. The embedding layer also provides an increase in the performance of the network, preserving similarities between input data and providing a greater number of parameters. The results also show how the models with embedding improve the models without embedding, being slightly higher the one obtained with a dense vector of dimension 10. This model obtained a 44.5% accuracy when evaluated with the test data.

Using the model to generate music and to complete fragments an acceptable musical quality is obtained (see demonstration in section 6). When selecting the next note based on the most likely one, the generated sequences contain repetitions. This specially happens with unusual symbols. As in automatic text generation, the solution is to use the generated probability distribution of the softmax output layer so unlikely notes have a chance of being the next ones. This prevents the model from simply repeating known patterns and somehow providing creativity, as well as different compositions for the same musical input fragments.

6 A deep learning based web application to support musical composition

A web application based on the trained models studied in this paper has been developed. Its interface is shown in Figure 3. The tool allows to introduce notes in the ABC format so the deep learning model can complete the musical composition. Alternatively, the music can be generated from scratch. The tool also

allows to select a musical instrument to play the results in the web browser. Furthermore, the user can choose between completing by predicting the most likely next note or to use the learned probability distribution to "roll a dice" so the least likely musical notes have a chance to appear in the piece of music. Finally, the compositions can be edited, reproduced online, and saved in an MIDI file.

There is a demonstration video available online (`https://youtu.be/MobbCV_SSOA/`) and the source code is also available in GitHub (`https://github.com/Tuxt/AutoScore/`). In the next months, the web service will be deployed in one of the OEG research group servers.

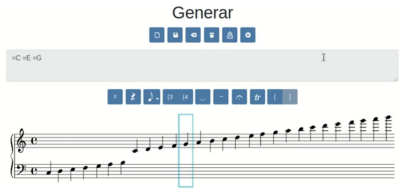

Fig. 3: Web application Interface.

7 Conclusion and future works

This paper describes an approach to automatic music generation through deep learning technologies based on language modeling techniques. An accuracy of 44.5% is achieved by a Recurrent Neural Network (RNN) with Gated Recurrent Units (GRUs) and embedding input layers. The main contribution regarding related works is the preprocessing and transformation of the music compositions used for training. These processes allow RNNs to reduce the necessity of learning the music sheets syntax. The web application, whose demonstration video is given together with its source code in GitHub, allows inexperienced users to use this powerful tool to enhance creativity in music composition.

Some of the future works on this research are: the extension of the training dataset, dealing with inbalanced classification, the experimentation with other neural structures such as Neural Autoregressive Distribution Estimation (NADE), the representation of samples as two-dimensional data (pitch and duration) or multidimensional time vectors, the study and elaboration of a neuronal model that contemplates the ability to produce music with accompaniment, or the implementation of a model that allows music to be created forward and backward.

Acknowledgments

This research work is supported by the Universidad Politécnica de Madrid under the education innovation project "Aprendizaje basado en retos para la Biología

Computacional y la Ciencia de Datos", code IE1718.1003; and by the the Spanish Ministry of Economy, Indystry and Competitiveness under the R&D project Datos 4.0: Retos y soluciones (TIN2016-78011-C4-4-R, AEI/FEDER, UE).

References

1. N. Agarwala, Y. Inoue, and A. Sly. CS224N Final Project. `https://github.com/yinoue93/CS224N_proj`. Accessed: December of 2017.
2. N. Agarwala, Y. Inoue, and A. Sly. Music composition using recurrent neural networks.
3. Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.
4. N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the Twenty-nine International Conference on Machine Learning (ICML'12)*. ACM, 2012.
5. Google Brain Team. Magenta. `https://github.com/tensorflow/magenta`. Accessed: December of 2017.
6. Google Brain Team. Magenta Drums RNN. `https://github.com/tensorflow/magenta/tree/master/magenta/models/drums_rnn`. Accessed: January of 2018.
7. Google Brain Team. Magenta Melody RNN. `https://github.com/tensorflow/magenta/tree/master/magenta/models/melody_rnn`. Accessed: January of 2018.
8. Google Brain Team. Magenta Performance RNN. `https://github.com/tensorflow/magenta/tree/master/magenta/models/performance_rnn`. Accessed: January of 2018.
9. Google Brain Team. Magenta Pianoroll RNN-NADE. `https://github.com/tensorflow/magenta/tree/master/magenta/models/pianoroll_rnn_nade`. Accessed: January of 2018.
10. Google Brain Team. Magenta Polyphony RNN. `https://github.com/tensorflow/magenta/tree/master/magenta/models/polyphony_rnn`. Accessed: January of 2018.
11. G. Hadjeres. DeepBach. `https://github.com/Ghadjeres/DeepBach`. Accessed: December of 2017.
12. G. Hadjeres, F. Pachet, and F. Nielsen. Deepbach: a steerable model for bach chorales generation. *arXiv preprint arXiv:1612.01010*, 2016.
13. D. D. Johnson. Generating polyphonic music using tied parallel networks. In *International Conference on Evolutionary and Biologically Inspired Music and Art*, pages 128–143. Springer, 2017.
14. F. Liang. *BachBot: Automatic composition in the style of Bach chorales*. PhD thesis, Masters thesis, University of Cambridge, 2016.
15. F. Liang, M. Gotham, M. Tomczak, M. Johnson, and J. Shotton. BachBot. `https://github.com/feynmanliang/bachbot`. Accessed: December of 2017.
16. E. Serrano, M. Rovatsos, and J. A. Botía. Data mining agent conversations: A qualitative approach to multiagent systems analysis. *Inf. Sci.*, 230:132–146, 2013.
17. M. Tomczak. *Bachbot*. PhD thesis, Masters thesis, University of Cambridge, 2016.
18. R. Wirth and J. Hipp. CRISP-DM: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39. Citeseer, 2000.