Music Creation Using Generative Adversarial Networks (GANs)

Cyprien Lambert[1]

Abstract
This research projects aims at generating music by using Generative Adversarial Networks (GANs) which models include Long Short-Term Memory (LSTM) layers. In order to better analyze the results obtained with such a network, another more classic network also using LSTM network is trained on the same dataset. This allowed to grasp the main issue in this project and similar ones as being the difficulty in creating sequences out of noise by using LSTM layers in a way that is efficient, as is discussed at the end of the paper.

Source code
Simple LSTM network: *https://github.com/sleepingtable/music_generation_gan_lstm/tree/master/LSTM%20network*
GAN: *https://github.com/sleepingtable/music_generation_gan_lstm/tree/master/GAN*

Music Sample: The three generated musics discussed in the article can be found at the root of the git: *https://github.com/sleepingtable/music_generation_gan_lstm*

Author
[1] *Exchange student at Linköping University, 3rd year student at Télécom SudParis, cyprien.lambert@outlook.fr*

Keywords: Generative Adversarial Networks — Long short-term memory — Music Generation

Contents

1. Introduction

One among other common misconceptions about artificial intelligence is that an AI system cannot be ”creative”. Even though this idea is now largely admitted has false, one could argue that AI, for the time being, don’t get a lot of recognition in the arts, which are traditionally considered as creative fields. Though AI systems are now capable convincing photo and music generation, it seems that this domain still has margin for progress and discoveries to be made, which makes this field of study particularly interesting.

The goal of this project is therefore to train an AI system in generating music. In order to achieve this effect, a recent kind of Neural Network, a Generative Adversarial Network (GAN) shall be implemented and trained, each of its two agents containing Long Short-Term Memory (LSTM) layers.

The core idea of this project is to assess the results obtained via the GAN to be implemented, and compare these results with those produced with a classic form of supervised learning using a network identical to the one serving as generator[1] in our GAN.

Even though the GAN should be trainable with other datasets, this project will be reduced to the study of the results yielded after training with a dataset of 92 songs (a large part of them originally from the Final Fantasy series). Furthermore the code will be written with Keras, running on top of TensorFlow.

2. Theory

[1] see 2.3

2.1 Artificial Neural Networks

Artificial neural networks, more commonly referred as neural networks, can be defined as a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs[1].

This type of computing systems is inspired by a simplistic model of the neural networks constituting animal brains[2]. Such systems are "trained" in order to "learn" how to perform a task. This training can however take various forms, which may be broken into three most common forms:

- *Supervised learning:* training a system via a set of inputs paired with their desired outputs, learning to minimize the output prediction error.
- *Unsupervised learning:* training a system via a set of inputs, and learning to minimize a loss function – which is dependent on the task as well as a priori assumptions.
- *Reinforcement learning:* training a system modelling the actions of an agent, trying to devise its policy in its environment, by minimizing the cost of its actions.

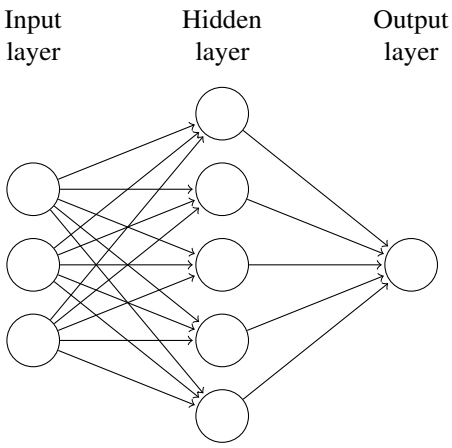


Figure 1. Fully-connected feedforward neural network

Neural networks have a layered structure. Connections between neurons most often do not form a cycle, as such these types of networks are called feedforward neural networks, as in Figure 1, differing from recurrent neural networks[3].

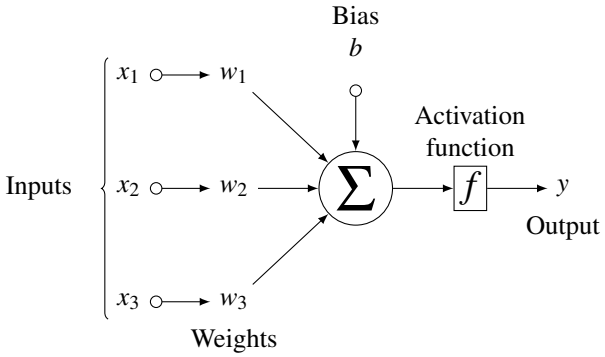


Figure 2. Simple neuron

A neuron receives the weighted values of a certain number of inputs. The output of the neuron is the result of the application of its activation function on the sum of the weighted values (and an optionnal bias). For instance, the output of the neuron in Figure 2 is:

$$y = f(\sum x_i * w_i + b) \tag{1}$$

2.2 Long short-term memory (LSTM)

LSMT are a type of Recurrent Neural Network (RNN). RNNs are a type of network – most often only one layer is considered – designed specifically for working with series of data. They have – in contrast with feedforward network – loops in them that allow information to persist (see Figure 3). However, it has been consistently observed that, while RNNs are theoretically able to connect information separated by a large "gap", such connection are not learned in practice[4].

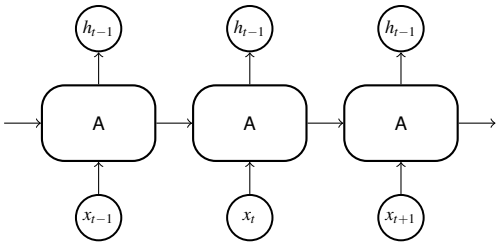


Figure 3. Unfolded basic RNN

Long short-term memory or LSMT are a specific type of RNN which are designed to avoid this long-term dependency issue. They were introduced in 1997 by Sepp Hochreiter and Jürgen Schmidhuber[5] and have proved to be useful in a variety of fields, including composing music.

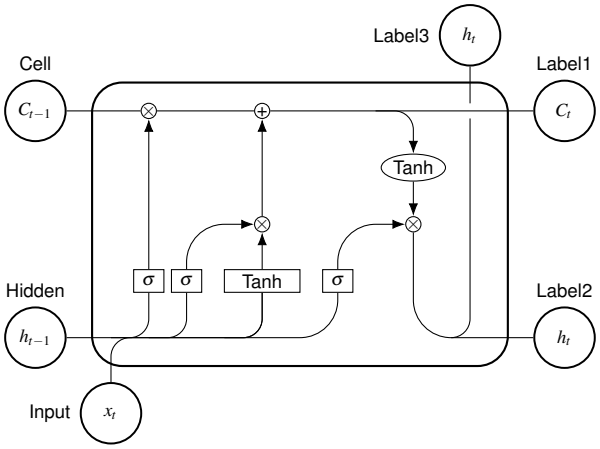


Figure 4. LSTM neuron

The core idea behid LSTM is the cell state C (see Figure 4) which is responsible for keeping track of the dependencies between the elements in the input sequence, intuitively conveying information through the whole chain, with only minor linear interactions[6][7].

2.3 Generative Adversarial Networks

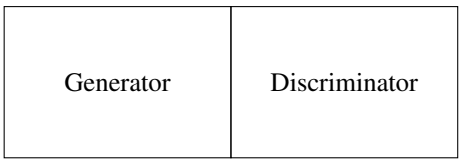
A GAN is a class of machine learning system invented by Ian Goodfellow and his colleagues in 2014[8]. It consists of two neural networks, the *generator* and the *discriminator*, having a contest. Given a training set, this technique learns to generate new data similar to the training set. The generator generates candidates while the discriminator evaluates these candidates. The goal of the generator is to increase the error rate of the discriminator by ”fooling” it into thinking that its candidates are genuine, while the discriminator must minimize its error rate.

GANs have initially been proposed as a form of generative model for unsupervised learning, but proved useful also in semi-supervised learning[9], supervised learning[10] and reinforcement learning[11]. It has proven efficient in generating several types of data – the most impressive arguably being generating portraits of imaginary people[12] – including music[13]. The GAN used here shall be used in the context of supervised learning, as detailed thereafter.

3. Method

3.1 GANs supervised training

While Generative adversarial networks were at the beginning intended as a generative methods for unsupervised learning, it is often used in supervised learning, as is the case here. The method used here in order to train two networks which are dependent upon the results of one another – the generator and the discriminator – is to stack the discriminator over the generator to create a combination of the two.



Training of the discriminator: Selecting a number n_{batch} of sequences from the training set and generating a number n_{batch} of sequences from noise with the generator. Outputs of 1 or 0.9 (in order to prevent the discriminator from converging too fast compared to the generator) is associated with the training inputs, and outputs of 0 are associated with generated inputs.

Training of the generator: Here the combination of the two networks will be used. The discriminator’s weights must not however be changed during this phase, only the generator’s weights (using Keras as I did, this is achieved very easily). A number of n_{batch} noise sequences are generated to serve as input, while the outputs are 1 – so that the backpropagation is positive for the generator if the discriminator is ”fooled”.

3.2 Battery of tests

Generative adversarial networks are notably hard to train; indeed a high loss of the generator does not necessarily yields bad results, which deprives us from an usually easy way to decide of the effectiveness of a change in our network. Moreover there can be cases when the generator can ”collapse” (if the discriminator is too effective and never yields positive rewards).

In order to seek the best results possible, the tests performed must therefore be comparable to one another. This is the reason why all training performed in this project spans upon 60 epochs, each epoch being composed of 228 224 sequences (usually separated in batches of 128). One other approach could have been to use the same training time for each training, as every network will go through one epoch in a different amount of time. The former method has however arbitrarily been chosen.

4. Result

First of all, a network simply using LSTM layers to predict the next note, considering the previous 100 notes, have been trained on a number of 160 epochs (one epoch goes through 57000 sequences). The model’s layers are as follow – translating intuitively Keras’ notations in pseudo-code:

- LSTM(512, return_sequences=True)
- Dropout(0.3)
- LSTM(512, return_sequences=True)
- Dropout(0.3)
- LSTM(512)
- Dense(256)
- Dropout(0.3)
- Dense(output size)

This training yielded the following loss graph, see Figure 5. It is worthy to note that the music quality after 160 epochs was poorer than after 60 epochs. More thoughts on that in the next section.

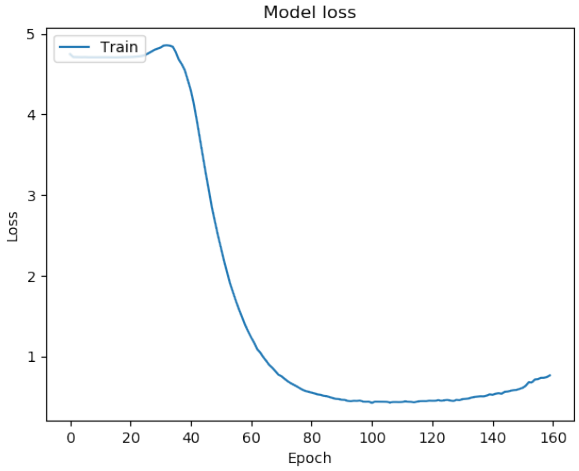


Figure 5. LSTM model – Loss

The second system is a GAN, therefore constituted of two models. It has been trained on 170 epochs. The generator has been trained to produce a sequence of 100 notes from a noise sequence, while the discriminator has been trained to recognize if a 100 notes sequence is authentic or not. The discriminator's layers are as follow:

- LSTM(256, return_sequences=True)
- Bidirectional(LSTM(256))
- Dense(512)
- LeakyReLu(alpha=0.2)
- Dense(256)
- LeakyReLu(alpha=0.2)
- Dense(1)

In a similar way, the generator's layers are as follow:

- Dense(256)
- LeakyReLu(alpha=0.2)
- BatchNormalization(momentum=0.8)
- Dense(512)
- LeakyReLu(alpha=0.2)
- BatchNormalization(momentum=0.8)
- Dense(1000)
- LeakyReLu(alpha=0.2)
- Reshape(sequence shape)
- LSTM(256, return_sequences=True)
- Dropout(0.3)
- LSTM(256, return_sequences=True)
- Dropout(0.3)
- LSTM(256)
- Dropout(0.3)
- BatchNormalization(momentum=0.8)
- Dense(output length)
- Reshape(output shape)

This training yielded the following loss graph, see Figure 6. It is worthy to note that while progress has been very slow, the music quality has been steadily improving throughout the epochs.

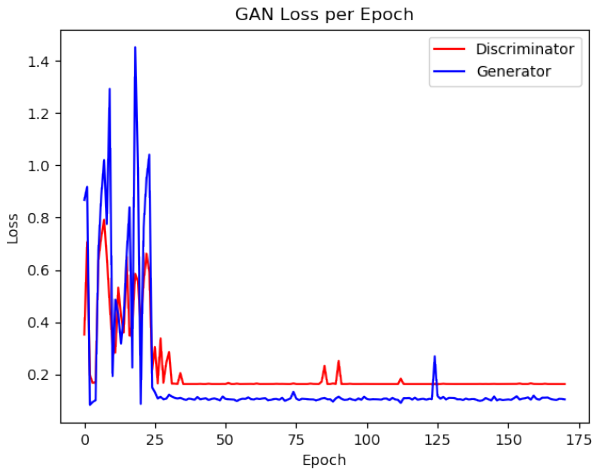


Figure 6. GAN model – Loss

5. Discussion

As mentioned before, the first network – using LSTM layers to predict the note following a previous sequence – actually performed better after 60 epochs than after 160 epochs. The music produced was a lot more complex after 160 epochs, but there was no real structure to the music. On the other hand, while the structure was simpler after 60 epochs, it stayed coherent for the entire 500 notes generated for testing. It was quite simple and yet nice to hear. After 160 epochs, while the beginning of the music was interesting and pleasant to listen to, it actually became quite awful towards the end, probably because the sequence analyzed for producing the next note was composed of notes previously generated, which therefore weren't very well structured and put together.

It seems that such behaviour could occur as a result of overfitting, which is surprising considering that dropout layers were used for the model. It may however be interesting to see if better results could be achieved if using a greater number of sequences for training or if the model were to be simplified. It may also be worth testing the same model while using a smaller learning rate. Still, correct results were achieved after 60 epochs and, looking at Figure 5, it may be that the results at 100 epochs were even better – though no sequence were generated at this point, so the music quality could not be assessed.

As for the results provided by the GAN, we may first be surprised that the genereator and the discriminator seem to

have found a balance in their training ; indeed, the generator may very well have been broken by the discriminator's speed of learning, since the former's model is so much more complex than the latter's. It seems however that these two models interact well and are able to train efficiently, this does not however imply that these models are ideal for the task at hand, only that they don't destroy each other.

As mentioned in the previous section, the music quality provided by the generator throughout the epochs has been slowly but steadily improving. The quality of the audio produced after 160 epochs was getting closer to satisfactory, as a pleasant melody seemed to emerge, though it was still quite subdued. It seemed to be heading toward a similar level as the content produced by the first network after 60 epochs, though the results were far poorer. It would be interesting either to try training this GAN for an even greater amount of time, or to try testing out other combinations of models for the generator and the discriminator.

6. Conclusion

It is as expected easier to get satisfactory results for music generation by predicting the note that follows a note sequence rather than by creating an entire sequence of music out of noise. The second method however has its perks: once the training is finished, it doesn't need any base sequence for generating music. Its music generation is therefore more authentic as it doesn't only extend a preexisting music sequence but actually creates one from the first note to the last one – though the sequence thus created cannot be extended in length.

The way that LSTM layers were used to produce a sequence out of noise – in combination with dense layers – on the generator model seems however unsatisfactory. It seems that there could be a better way to use LSTM layers in order to create sequences out of noise in a more intuitive, natural and efficient manner, but the literature still doesn't have much to offer on this subject. This difficulty therefore seems to be the major challenge to overcome for the specific purpose of this project, but may very well be an interesting field of study for other types of sequence generation as well. Research on this matter could however probably be better achieved by testing out on a different type of network, as GANs are very unstable and hard to train.

Acknowledgments

This project was supported by Linköping University. I would like to thank Pierangelo Dell'Acqua who made possible and supervised this project.

References

[1] Caudill Maureen. Neural networks primer, part i. *AI Expert*, 1987.

[2] Artificial neural network. *Wikipedia*.

[3] Andreas Zell. Simulation neuronaler netze [simulation of neural networks] (in german) (1st ed.). 1994.

[4] Patrice Simard Yoshua Bengio and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE*, 1994.

[5] Hochreiter and Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[6] Long short-term memory. *Wikipedia*.

[7] Christopher Olah. Understanding lstm networks. *colah's blog*, 2015.

[8] Jean; Mirza Mehdi; Xu Bing; Warde-Farley David; Ozair Sherjil; Courville Aaron; Bengio Yoshua Goodfellow, Ian; Pouget-Abadie. Generative adversarial networks. 2014.

[9] Ian; Zaremba Wojciech; Cheung Vicki; Radford Alec; Chen Xi Salimans, Tim; Goodfellow. Improved techniques for training gans. 2016.

[10] Jun-Yan; Zhou Tinghui; Efros Alexei Isola, Phillip; Zhu. Image-to-image translation with conditional adversarial nets. 2017.

[11] Stefano Ho, Jonathon; Ermon. Improved techniques for training gans. 2016.

[12] Michael Doyle. John beasley lives on saddlehorse drive in evansville. or does he? *Courier and Press*, 2019.

[13] Jun Wang Yong Yu Lantao Yu, Weinan Zhang. Seqgan: Sequence generative adversarial nets with policy gradient. page 6, 2016.