BABEŞ-BOLYAI UNIVERSITY

FACULTY OF MATEMATICS AND COMPUTER SCIENCE

APPLIED COMPUTATIONAL INTELLIGENCE

DISERTATION THESIS

Scientific coordinator

Dr. **CZIBULA Istvan**

Student

TIRIC Radu Ciprian

CLUJ-NAPOCA

2020

BABEŞ-BOLYAI UNIVERSITY

FACULTY OF MATEMATICS AND COMPUTER SCIENCE

APPLIED COMPUTATIONAL INTELLIGENCE

DISERTATION THESIS

Artificial Intelligence in Art: Deep Learning Music

Scientific coordinator

Dr. **CZIBULA Istvan**

Student

TIRIC Radu Ciprian

CLUJ-NAPOCA

2020

Abstract

This paper seeks to present a way in which artificial intelligence can be introduced into art, or rather into musical art. The proposed approach is based on a machine learning model consisting of recurrent LSTM neural networks, trained on a data set with MIDI audio content, that is subsequently capable of generating unique audio content. Some of the original aspects of this paper would be the architecture of the machine learning model, the variety of musical genres in the data set and also the consideration of musical breaks, called rests.

The structure of this paper is divided into three main chapters. The first chapter, as the name suggests, presents some of the existing methods in the literature on this field. The second chapter aims to present the theoretical knowledge needed to understand the approach proposed in chapter three where the development of the application is presented step by step.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Tiric Radu Ciprian

Cluj-Napoca
10.06.2020

Table of contents

Introduction and problem description

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. When we talk about artificial intelligence, people often think of autonomous cars, robots control, medical diagnoses and many other areas where it has reached a high level and makes people's work easier or even replaces it. The idea of „AI artist" has become more and more popular in the last few years. These complex algorithms are creating exclusive works of art. Art is form of communication beyond words, it is a feeling captured and transposed to be felt by other people. They are generating wonderful visuals, tuneful music, profound poetry, and even realistic movie scripts.

The art of music can be defined as a succession of sounds arranged in such a way that harmony, unity and continuity appear. Generating music has developed a lot in the last few years, but the idea has a long history. Somehow, the first automatic music came from nature, a good example would be ancient Greek wind-powered Aeolian harps or the Japanese water instrument suikinkutsu. The first models used in music generation were the Markov models, which appeared in the 1990s. In the late 1990s, Peter M. Todd introduced recurrent neural networks (RNNs) in this area, in an attempt to generate unique music sequences, and unlike Markov models, they achieved much more promising results.

The algorithm presented in this paper is built on LSTM recurrent neural networks and used to automatically produce music and melodies, in the absence of any human interference. The main focus lies on developing a model which is competent of generating a primary set of musical notes, after learning and analyzing an existing set of musical notes, chords and rests. The model must have the capacity to learn the original sequence, to convert it for the learning system and to also reintroduce details and structures of musical notes for future projection of learning sequence.

Chapter 1

Existing approaches in the literature

As previously mentioned, music generation has a long history, starting with inspiration from nature, reaching probability-based Markov models and then introducing the artificial neural networks. As we can speculate, machine learning models quickly became the most promising and also the most accurate. In this section we will provide several examples of different machine learning approaches form the literature which have had a considerable importance in this field.

1.1. DeepHear system

A first example is DeepHear system created by Felix Sun [1] which uses an autoencoding deep belief network (DBN). A deep belief network can be seen as a stack of artificial neurons arranged on layers. Each layer takes the data from the previous layer, changes it because the layer size decreases from layer to layer and passes it on to the next layer, until the output layer. Sun solves the problem of decreasing the size of the layers by autoencoding the network. The final architecture consists of a symmetry between several layers of encoding and decoding that transmit their weights between them with a back-propagation algorithm for training.

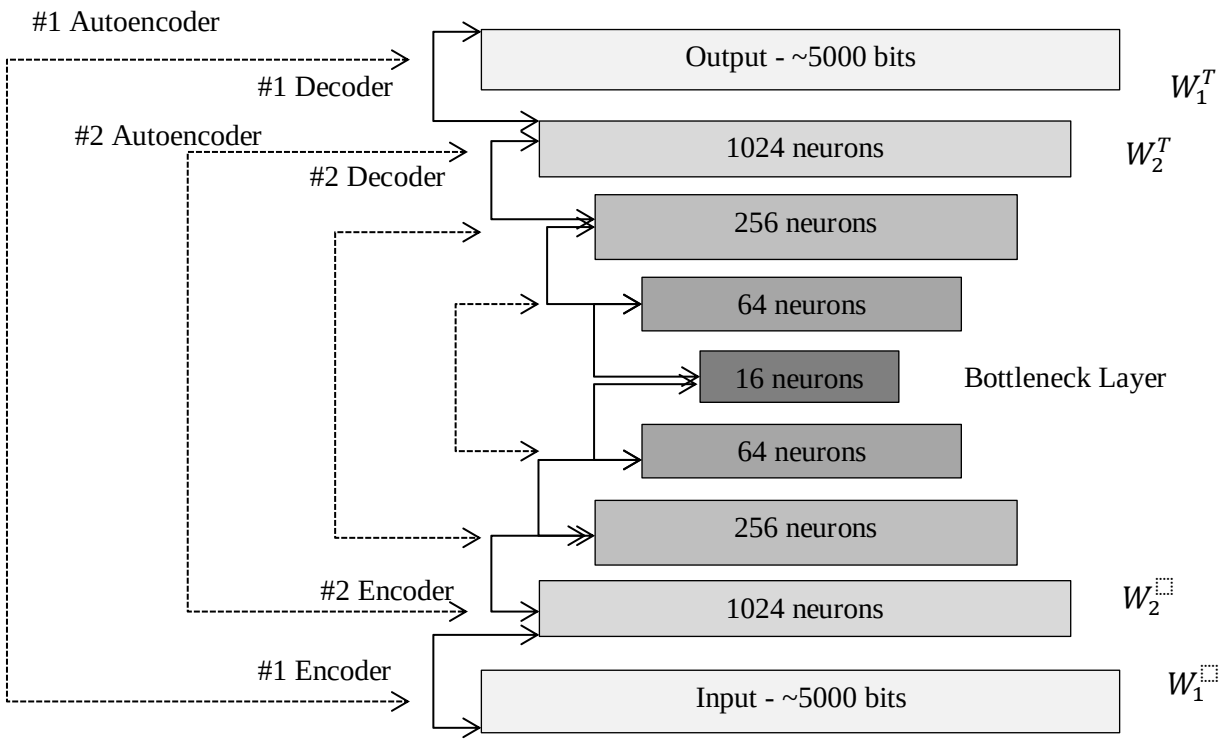


Fig. 1 Autoencoder architecture of DeepHear

For this experiment the author used a corpus of 600 measures of Scott Joplin's ragtime music, split into 4 measures long segments, a piano roll representation with a multi-one-hot encoding and a a 16th note time step. Firstly, a restricted Boltzmann machine (RBM) initialization was used and then the difference between input and its reconstruction was calculated using gradient descent. The result of the averaged error was around 10 wrong notes out of 5000.

The results of this approach claim that there is a fairly large amount of plagiarism because certain musical sequences are almost identical to those in the original dataset. A concrete result regarding the percentage of notes between a generated sequence and a sequence from the training set is 59.6%, which is a fairly high value. The author states that this is due to the small size of the bottleneck hidden layer and he also says that this percentage is not necessarily important because two similar audio sequences as notes may sound completely different because of the melody.

1.2. CONCERT (CONnectionist Composer of ERudite Tunse)

This approach was developed by Mozer [2] in 1994 and it has an architecture based on recurrent neural networks that learns to make predictions influenced by what it has learned in the past. A brief intuition of this model is that it should predict the next note in the current sequence of notes.

As we can see in the Figure 2 the architecture of the CONCERT system contains some special layers:

- the next-note-distributed (or NND) layer contains CONCERT's internal representation of the note-divided into three pools of units, forming distributed representations of pitch, duration and harmony
- the next-note-local (or NNL) layer contains one unit for each alternative pitch, duration and chord
- context sensitive grammar is a collection or rules sufficient for reproducing most or all of the structure inherent in the set (rules are of the form context -> next-note, where context is a string of one or more notes, and next-note is the next note implied by the context)

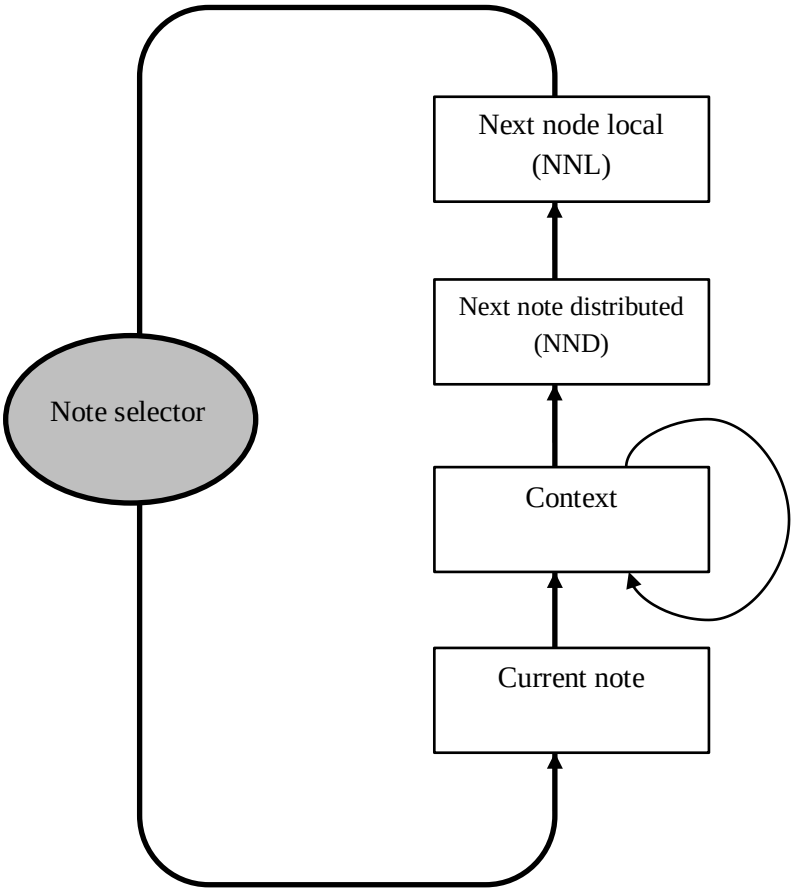


Fig. 2 CONCERT RNN architecture

Pitch, duration and harmonic chord accompaniment are three important aspects referring to the input and output representation:

- Pitch representation (Figure 3) was inspired from Shepard (1982) [10], which is a five-dimensional space where each pitch specifies a point along the pitch height (or PH) dimension, an (x,y) coordinate on the chroma circle (or CC) and an (x,y) coordinate on the circle of fifths (or CF).

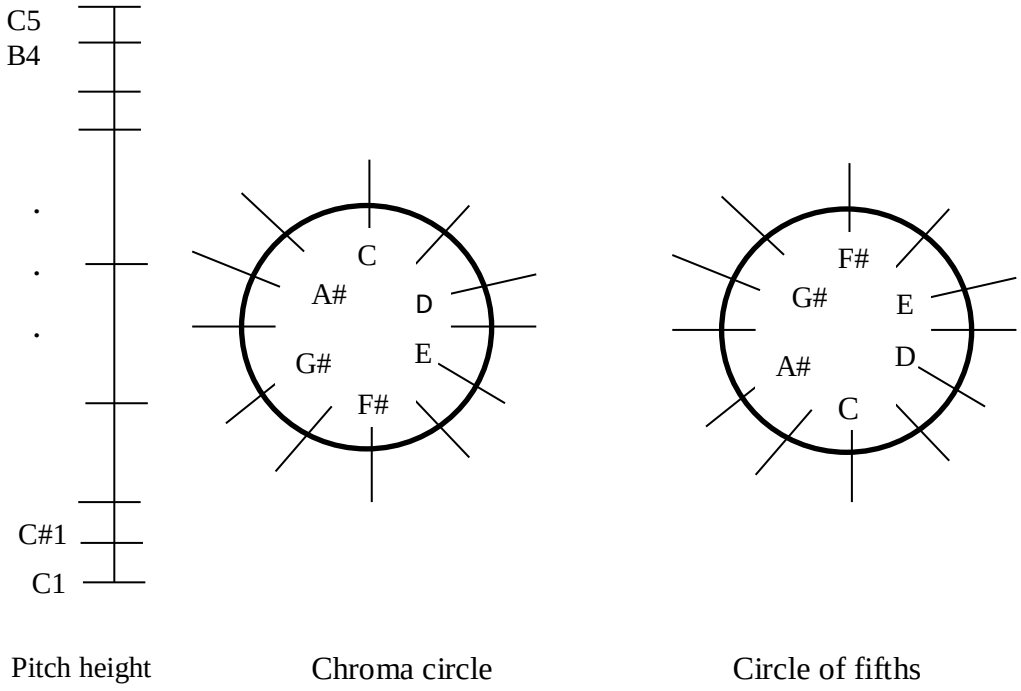


Fig. 3 Shepard's pitch representation

- Duration representation (Figure 4) is also arranged in the form of a five-dimensional space where each beat is divided in twelfths (a quarter-note has the duration of 12/12, an eighth-note 6/12, an eighth-note triplet 4/12, and so on).

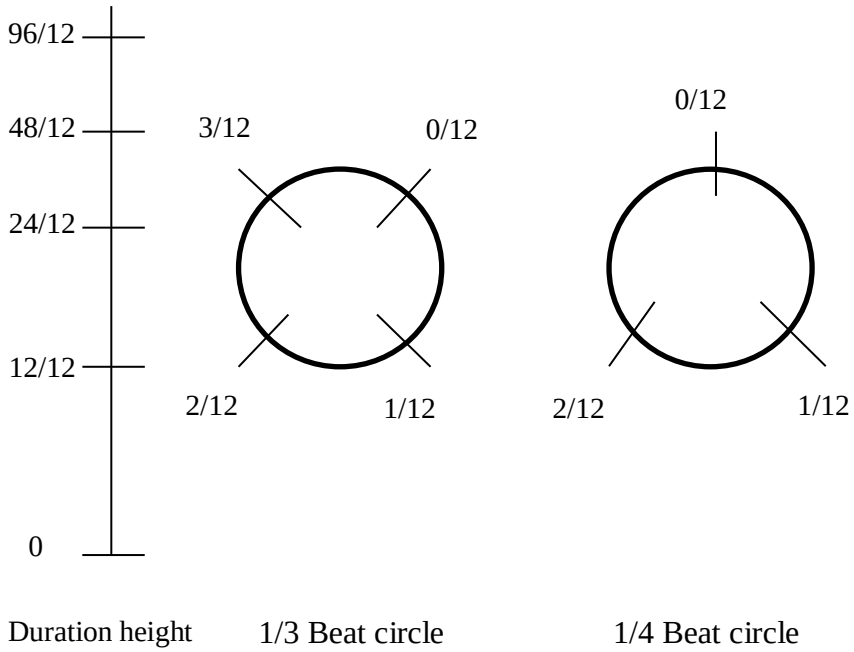


Fig. 4 CONCERT's duration representation

- Chord representation was inspired from Lande and Keef's (1989) where each chord is composed of three of four component pitches (e.g. C-major has C3, E3, G3 component pitches).

For this experiment the author used the sigmoid function rescaled to [-1,1] as an activation function, and the square difference as a cost function. He used three sets of data for training, the first consisting of the melody lines of 10 simple pieces by J. S. Bach, the second of 25 traditional European folk melodies and the third of 25 waltzes by various composers.

The performance of this approach is quite poor on natural music but on simple, structured sequences it increases considerably. One critic claims that few listeners are fooled that the music generated is composed by a human being and not a computer, because these pieces of music do not have the thematic structure, rhythmic and coherent organization like the one created by an artist. The reason for this is because architecture does not cope with the increasing length and complexity of music.

Chapter 2

Theoretical background for generating music with LSTMs

2.1. Data representation (MIDI)

Musical Instrument Digital Interface or MIDI can be defined as a protocol that connects digital instruments to the computer, giving a musician the opportunity to remotely control one or more digital instruments or digital synthesizers. [4]

A midi file contains information about the beginning of a musical note, the end, the intensity and others structured as follows:

- MIDI note number is an integer number between 0-127 which indicates note's pitch
- Velocity is an integer number between 0-127 which leads the intensity (how loud a note is played) of the sound
- MIDI channel is an integer between 0-15 which indicates the instrument
- Note-on indicates that a musical note is played
- Note-off indicates that a musical note is stopped
- Time stamp is an integer number which indicates how many time steps to wait until the note is played either the note is on or off

Time	Message	Channel	Note number	Velocity
60	NOTE_ON	1	67	100
0	NOTE_OFF	2	37	0
20	NOTE_ON	1	41	100
60	NOTE_ON	1	11	90
4	NOTE_OFF	1	120	0
0	NOTE_ON	2	3	40

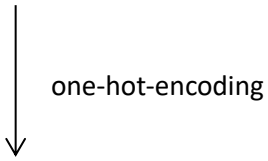
Fig. 5 Midi file representation

An important feature of this type of representation, as well as sheet representation, encoding and storage of absolute synchronization information is done much finer and in a flexible way.

2.2. Data encoding (one-hot-encoding)

One of the most popular data encoding for categorical data in machine learning is one-hot-encoding. It can be defined as a technique which produces a binary vector with the length equal to the number of the categories in the data set. The process can be observed in Figure 6.

Thing	Thing_ID	Price
book	1	10
pen	2	15
book	1	12
t-shirt	3	20



book	pen	t-shirt	price
1	0	0	10
0	1	0	15
1	0	0	12
0	0	1	20

Fig. 6 One-hot-encoding representation

2.3. Architecture (LSTM)

Recurrent neural networks can be defined as a series of memory units (layers) that communicate with each other, trying to solve the upcoming events based on the experience of the previous ones. At the base of these networks states a chain type architecture, see Figure 7, suitable for a multitude of tasks such as machine translation, speech recognition, sentiment analysis and much more. This

type of architecture is illustrated in Figure 7, in which some important variables can be observed, such as:

- **x** which represents the input data (for example this can be a musical note from a song, or any other part of a sequential data),
- **y** which represents the output data (for example, this may be the next musical note in that song, from the network's perspective based on previous musical notes) and
- **h** which represents the hidden layers that contain the weights and activation functions.

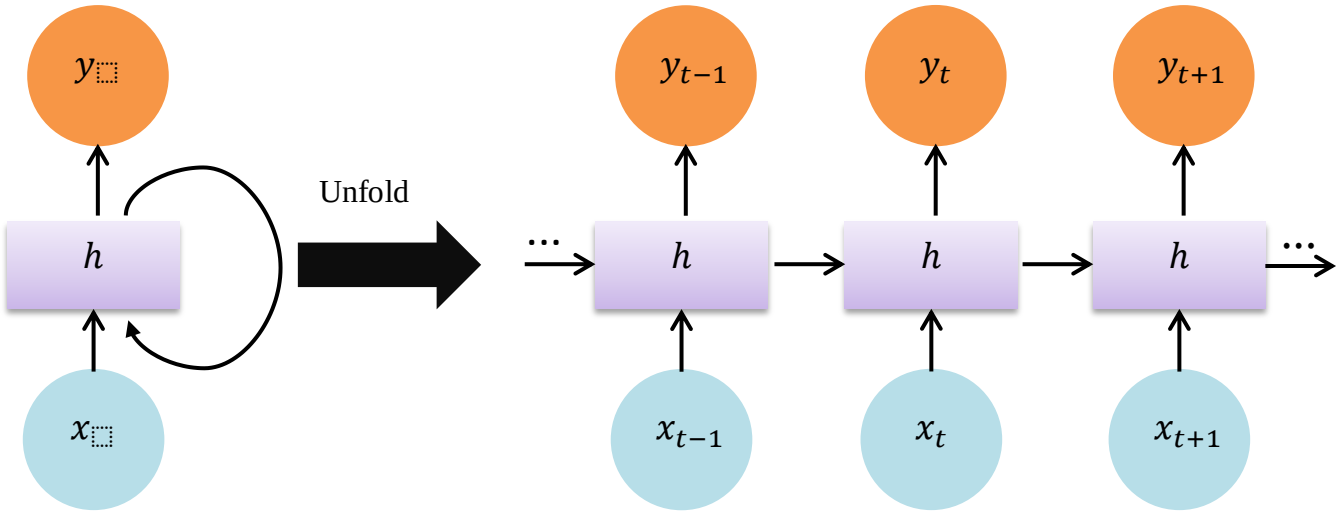


Fig. 7 RNN Architecture

A more detailed view of the hidden layer can be seen in Figure 8, where the most basic form is illustrated, in which the input data from the current step is combined with the output data from the previous step and transmitted to the **tanh** activation function to get the output data for the next step.

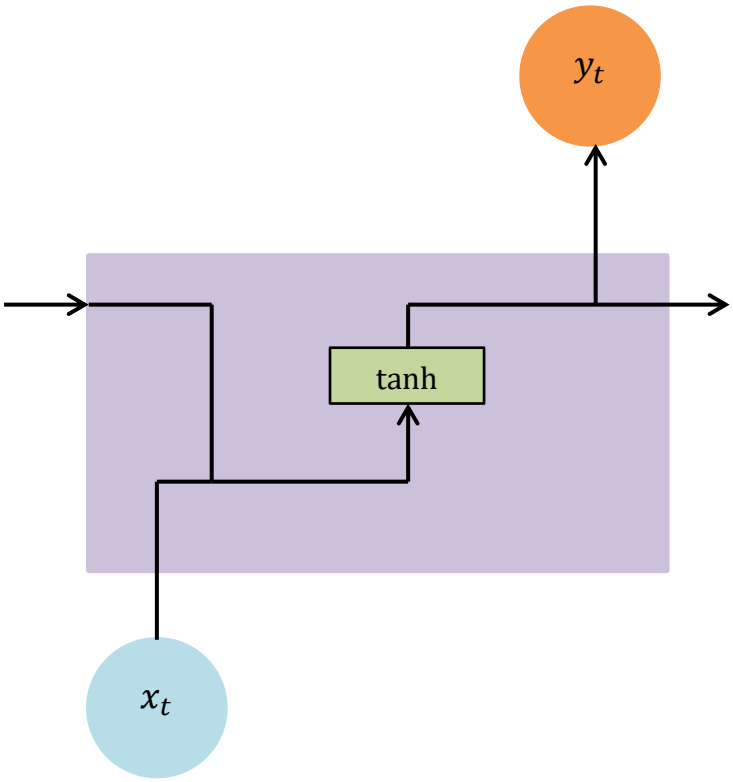


Fig. 8 RNN cell architecture

Due to the fact that this type of architecture does not cope with large sequences, the long-short-term-memory (LSTM) architecture was created. The objective of these networks is to reduce long-term dependencies, keeping in mind the information for a long period of time and solving the problem of vanishing gradient. This problem refers to the fact that a neural network becomes difficult to train due to the fact that as more layers use activation functions are added to the network and the gradients of the loss function approach zero. These networks can be compared to the memory of a computer in terms of the fact that from their cells can read, write or store data. The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. [5]

We can observe in Figure 9 that their interior is more complex than the previous ones, being composed of multiple cells, gates and functions:

- Tanh activation function – gives values between -1 and 1
- Sigmoid activation function – gives values between 0 and 1
- Pointwise multiplication – multiplies the information by a number close to 0 to forget the past information / by a number close to 1 to retain the past information.

- Pointwise addition – merge current information with the past information
- Forget gate – decides what information should be thrown away or kept
- Cell state – transports information
- Input gate – updates cell state
- Output gate – decides what the next hidden state should be

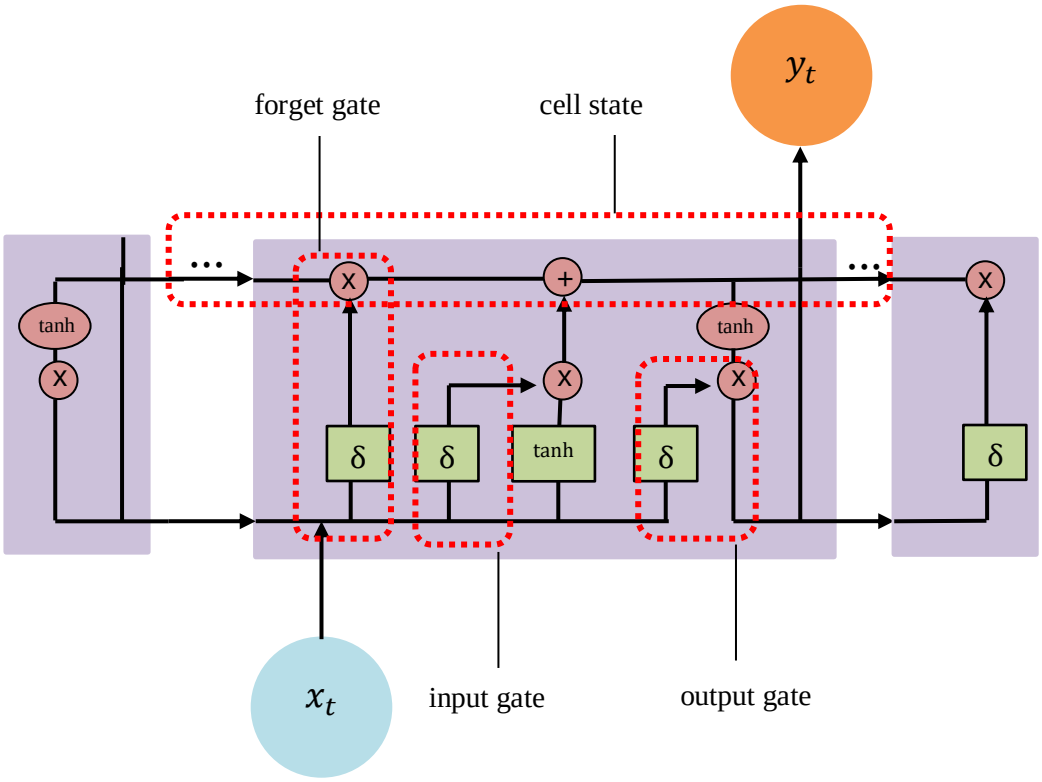


Fig. 9 LSTM cell architecture

Step 1: forget gate

This gate has the job of deciding what information will be retained or forgotten. Based on the current input and the previous hidden state, the information is entered in the sigmoid function which generates a value between 0 and 1 (0- means to forgot, 1- means to keep).

Step 2: Input gate

This step improves the cell state in two ways. The first is the same as in the previous step, namely enter the current input data and the previous hidden data in the sigmoid function, to give them importance. In the second part, the same data from current input and previous hidden state are

introduced in the tanh function, which transforms them in the interval [-1, 1], in order to regularize the network, and then the data modified by both functions is multiplied and passed to the cell state.

Step 3: Cell state

This cell state runs straight down the entire chain, with only some minor linear interactions. This step combines the results received from input gate and forget gate which is done by the pointwise addition function.

Step 4: Output gate

Not all information that runs along the cell state is fit for being output at a certain time. This job of selecting useful information from the current cell state and showing it out as an output is done through the output gate.

Chapter 3

Personal Application

Nowadays music is being used more and more in a lot of areas such as film-making, video-games, commercials and so on. The background sounds have to either be created or used the existing ones but having issues with the copyright and I consider that artificial intelligence can help us in this regard. With this motivation I decided to create a system that can help people who face the problem of the need for audio content.

3. 1. Application overview and technologies used

This application is based on a machine learning model, which uses an audio data set to train a sequential neural network that then manages to learn to produce unique songs, all of which are hidden behind a website.

The main technologies used to implement this application are the following:

- Django Web Application Framework – It is an open-source Python-based framework whose first release appeared in July 2005 [6]. The popularity of this framework has grown due to the ease of creating a web page, clean design, stability and performance. Also another important key of this framework is the architecture, namely Django uses a Model-Controller-View (MCV) architecture also known as Model-Template-View (MTV). The appearance of this type of architecture is due to the evolution of this field, the web pages becoming more and more complex and difficult to implement. MCV is an extremely neat architecture because it has different files for every aspect of the web application. As the name suggests, this architecture contains three important terms, the model which transfers data from and to the database and operates on it, the template which basically controls what should be displayed and how it should be displayed to the user and the view which acts like a controller using a programmed logic to decide what is pulled out from the database through the model and passed to the view or what information comes from the view and entered into the database through the model.
- Keras Deep Learning API – It is a user friendly, modular and Python-based high-level deep learning (API) which means it can act as a front end while other platforms like Tensorflow

or Theano work as back end. This technology has as a strong point the facilitation of the work of a data scientist in the implementation of complex neural networks and it is recommended regardless of the data scientist's experience because it is very well specified and very easy to understand. More information in [7].

- Music21 – It is a Python-based toolkit dedicated to both musicians with minimal programming knowledge and programmers with minimal musical knowledge. It has an object-oriented architecture which makes it more organized and user-friendly. The main occupation of this tool is to study music data sets, to modify them, to create new sounds and most importantly to understand the foundations of music theory.
- MIDI.js – It is a JavaScript midi audio player which uses W3C Web Audio and allows any browser to play midi files without the need to install any plugins or extensions.[9]

A brief presentation of the application is found in Figure 10, where it can be observed that it is divided in two sections, each with its subdivisions.

- User interface (UI):
  - URLs – receive the information from the browser and send it to the controller (view)
  - View – acts like a controller, it connects the user's actions to the application
  - Templates – contains the structure for the comprehensive layout and display features of the website
- Machine learning model:
  - Data processing – extract and encode information from MIDI files (notes, chords, rests)
  - Creating and training model – the architecture of the neural network is defined and the data is introduced in it
  - Generating content – the trained neural network is used to predict a predetermined amount of musical information, which will represent the audio content
  - Evaluation and improvements – calculate the accuracy of the model predictions for several parameters, compare them and try to improve the model

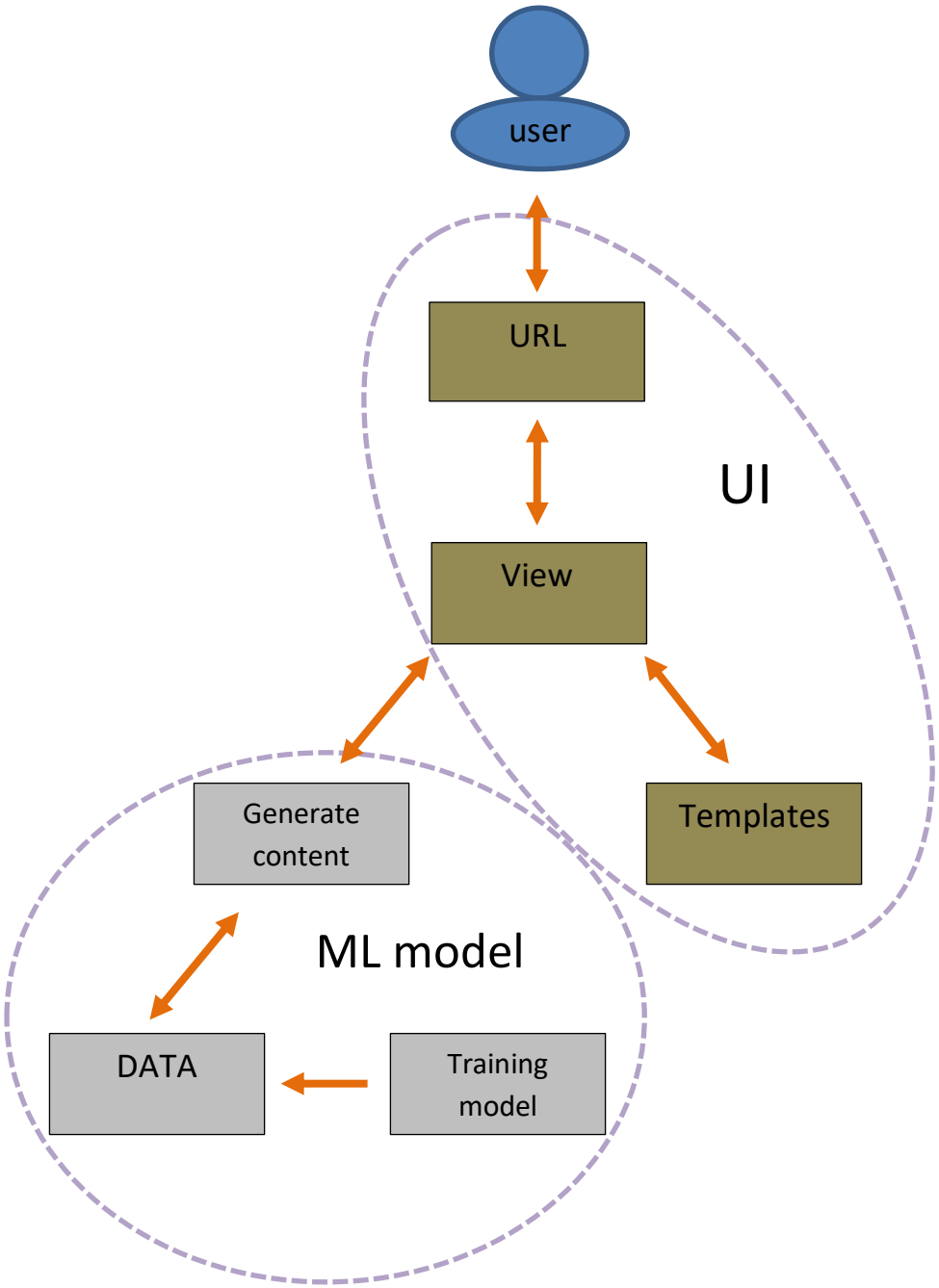


Fig. 10 Application architecture

3. 2. Aspects of implementation

3. 2. 1. User interface

As mentioned in the previous paragraph, this application is hidden behind a website, which represents the user interface. Its implementation has taken place by following some important steps:

Step 1: Template implementation

This step consists in the effective creation of the design and appearance of the web page. This was not a very complicated step because I used a predefined stylized template from the W3 library for a regular site and modified it according to my preferences. Its implementation is in HTML language and has a common structure based on the following sections:

- top bar: it can be described as a narrow white bar containing an "AI – music" named button
- header: this is the most significant part of the site which consists of a background image, a title, a selection bar of the desired instrument, and a button called "Generate" which triggers the appearance of four other buttons "Play song", "Pause", "Resume" and "Download", see Figure 11.
- about section: it can be described as a light-grey paragraph which contains some relevant features of the application
- contact section: it can be described as a grey paragraph which contains some information about how to get in touch with the website's owner
- footer: it can be described as a black paragraph which contains a button "Back to top", some links to social media accounts and the name of the website's designer.

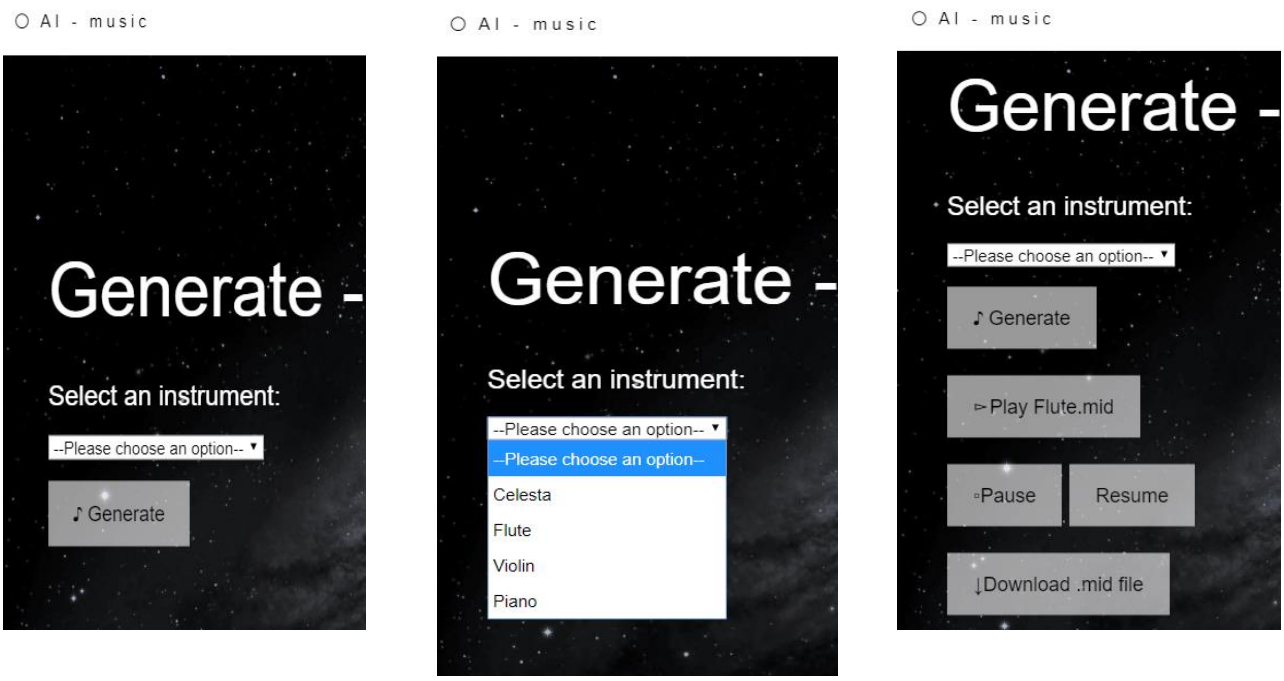


Fig. 11 Main functionalities of the application

Step 2: Implementation of the functionalities

In this step the connection between the template and the user is accomplished, better said the website becomes functional. For this part I chose to use the Django framework, with the motivation that it is very well structured and also it is implemented in Python, the same programming language that I used to implement the machine learning model to generate music. The functionalities of the site are represented by the buttons presented in the previous step, so we will resume them and discuss their implementation:

- "AI - music" button: this button uses the *onClick* function by transmitting through the *href* parameter the new URL to which it refers from the urls.py file which triggers the *home(request)* function from the views.py file which creates an HTTP response and sends it to the browser. This process can be observed in Figure 12.

Fig. 12 "AI – music" button implementation

- "Generate" button: this button is a submit type for the "Select an instrument" form. This top-down form contains two attributes, *action* - which contains the URL where the submitted information will be sent, and *method* - which specifies how the information will be sent, in this case the method is *POST*, which means that the information will be sent as an HTTP post transaction. The URL from urls.py corresponding to the URL from the *action* attribute accesses the *external(request)* function from views.py, which receives the data from the top-down form, runs a Python script which generates a midi file with audio content, and sends the file as a HTTP response to the browser and "Play song", "Pause", "Resume" and "Download" buttons.
- "Play song", "Pause", "Resume" and "Download" buttons: the first three buttons use the *onClick* function, which calls the midi.js library which allow user to listen the generated midi file, respectively pause and resume it, and the last button downloads the midi file.

3. 2. 2. Machine learning model

This section is the most significant part of the application. Next, the necessary steps will be presented to create the machine learning model capable of learning musical information notes and generating unique audio content.

Step 1: Preprocessing data

First of all, the data set consists of 30 downloaded .mp3 songs and then .mdi converted, to be able to extract the information from them. The files are taken one by one, it is divided based on musical instruments using the *instrument.partitionByIntstrument(stream)* function, then it is used an iterator, *stream.iterator.OffsetIterator(stream),* which at each iteration returns a list with all the elements from the same offset, then each of this lists is traversed and each element is checked to which category is part: if it is a note, its pitch is stored, if it is a chord, a string is created that contains each referential musical note delimited by ".", and if it is a rest, the string "rest" is stored. (see Figure 13)

So we have all the music information from all the .mid files, now we are going to create the inputs and the outputs, built in such a way as to be compatible with neural networks. In the first phase, all the unique musical information is taken over, and a dictionary is created in which each musical information is encoded to an integer. In the second phase, *X* - the input, and *y* - the output are created, based on integers from dictionary. For input, all the data are divided into column-vectors of 50 musical information each, taking the whole number from the dictionary corresponding to them. The output must be connected to the input, so for each column vector an output will be retained that represents the integer from the dictionary corresponding to the next musical information after the 50 of the input (the 51st musical information). The last phase is represented by the normalization of the input data, restricting the range [0,591] to [0,1] by dividing each value by float number 593, and by converting the output data into categorical form using *np.utils.tocategorical()* function. (see Figure 13)

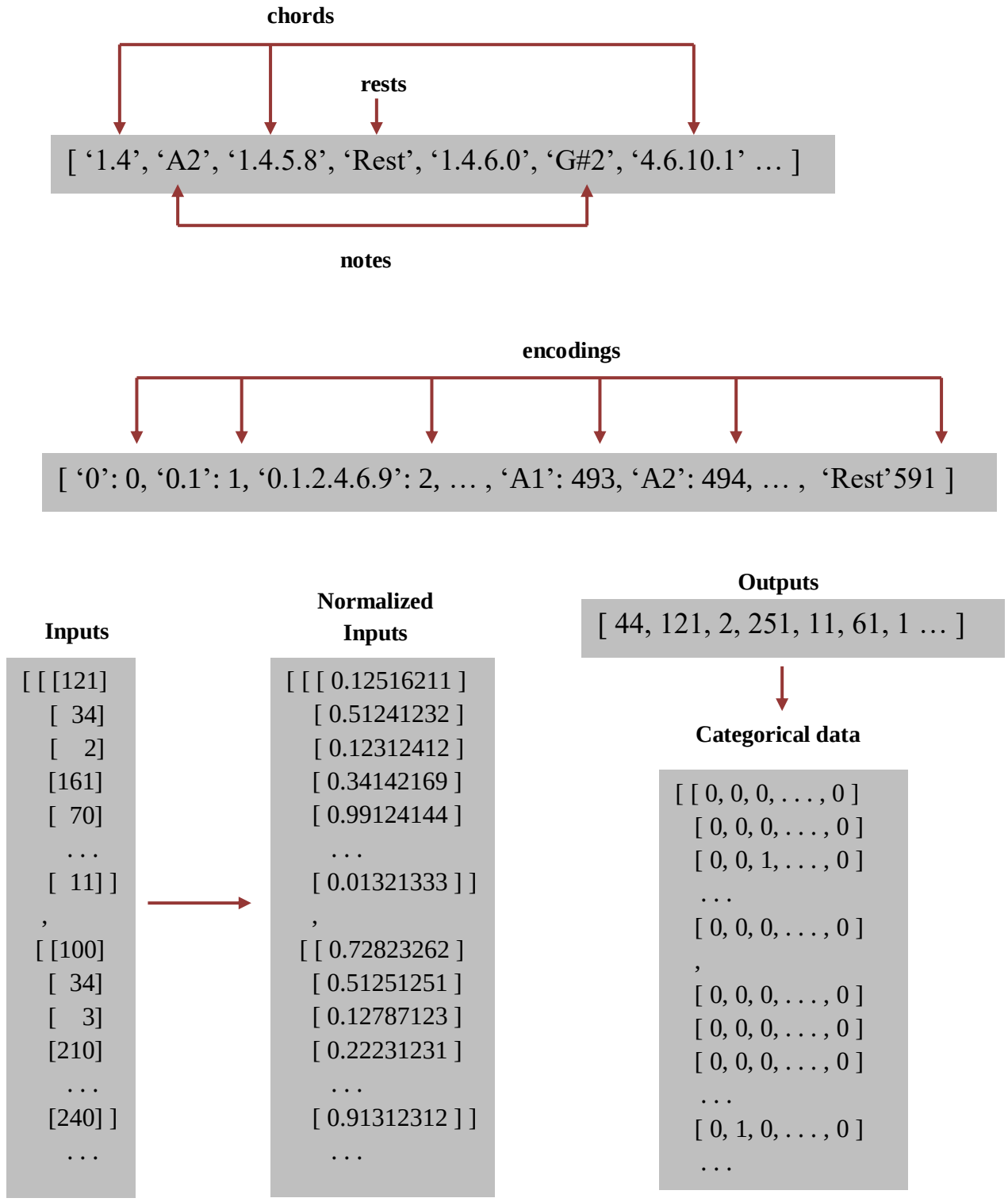


Fig. 13 Data processing (encoding, normalized data, categorical data)

Step 2: Creating and training machine learning model

Neural network has been created in Keras and its architecture (Figure 14) is a sequential one consisting of several layers, two LSTM layers, two dropout layers, one after each LSTM and finally a dense layer and an activation function. Two parameters are used in the LSTM layers, "return_sequnces" set to True, which returns the hidden state output for each input time step, and "stateful" also set to true, which make the model remember what happened in the previous batch by passing states from the previous batch to the next batch. These two parameters are important because our data set is divided into sequences that are related to each other, and certain information must be retained and reused. The dropout layers are used to regularize and prevent overfitting by ignoring, dropping out units/neurons from network with a certain probability set. The dense layer is used to avoid vanishing, to fully connect the inputs to the outputs, having the size equal to the total number of unique music information in the data set, and to apply the activation function, this being the normalized exponential function which will calculate the probability of activation for each neuron. One last thing left at this step is choosing the loss function and the optimizer. For the calculation of the loss, categorical cross entropy is used because our target values is a finite set of values, and the chosen optimizer is Adam because it is a combination of the advantages of the adaptive gradient algorithm and root mean square propagation optimizers, and also it computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. During this step the weights are stored into a HDF5 file which will be used in the next step.

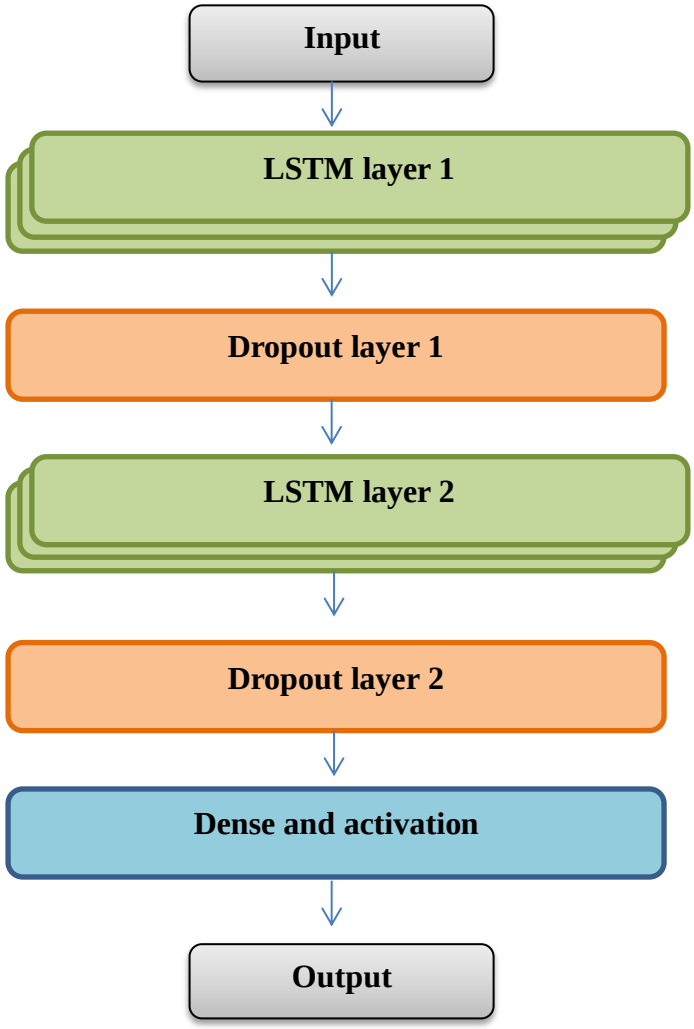


Fig. 14 Neural network architecture

Step 3: Using model to generate audio content

This last step is the most satisfying step, in which we can listen to the first audio content, completely generated by the trained neural network. It can be divided in three parts. First of all, the neural network from the previous step is recreated, but this time it is no longer trained but the best weights are introduced, which were stored in a file during the training process. Second, we load the music information extracted from the midi files in step 1, which is also stored in a file. The generation phase consists in choosing a random sequence of 50 music information and based on which the model will predict 300 (it can be modified) music information. For each prediction, the musical information with the highest probability is chosen, from the probability vector returned by the model, it will be added at the end of the current sequence and retained in a prediction vector. Finally, this prediction vector is reprocessed with the reverse mechanism from step 1, namely a

stream is created with the corresponding musical notes, chords and rests from the encoding dictionary and a midi file is created with this information which represents the audio content.

3. 3. Results and discussion

This subchapter can be seen as a discussion of the initial, intermediate and final results that led to the presentation of the machine learning model from previous subchapter. The following results are based on three important parameters, the number of epochs, the batch size and the length of the input. A first training process of the neural network took place in 40 epochs with the batch size of 128 samples and the input length of 50 musical information with a total duration of about 12 hours and 31 minutes. A good observation was that from the 30th epoch to the 40th the accuracy and loss almost stagnated (Figure 15), so I decided to reduce the number of epochs to 30. Thus the accuracy decreasing from 73.1% to 72.2% but the training time changed drastically, decreasing by almost 5 hours.

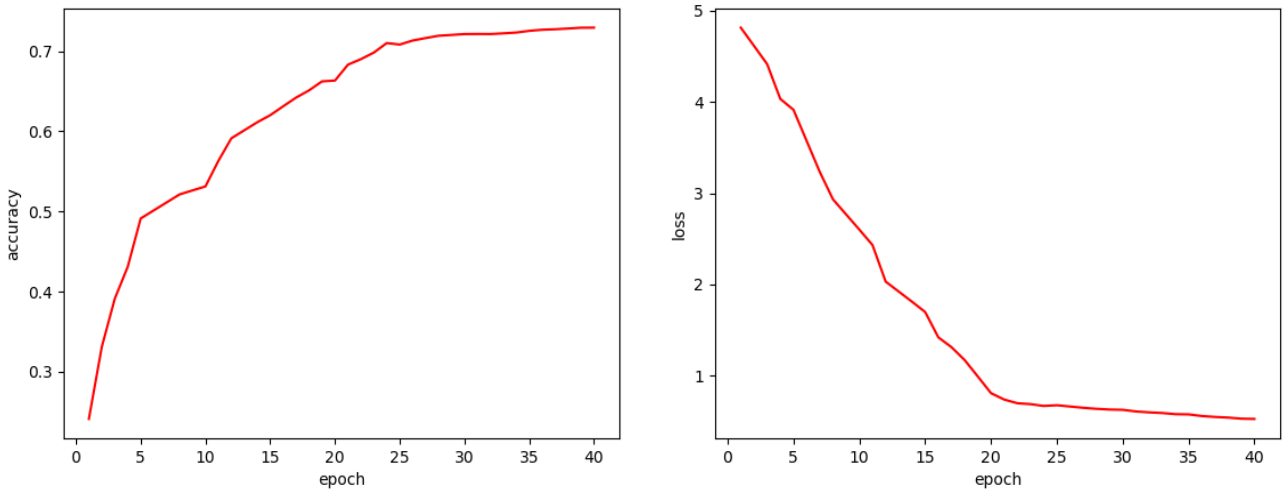


Fig. 15 Accuracy and loss for 40 epochs

Another important aspect is the number of samples introduced into the network at once, namely the size of the batch. Figure 16 shows the difference between three different drive processes each on a certain batch size. Since the best result is the red line (batch size 64), we decided to reduce our current batch size to 64, which slightly increased the duration of the training process.

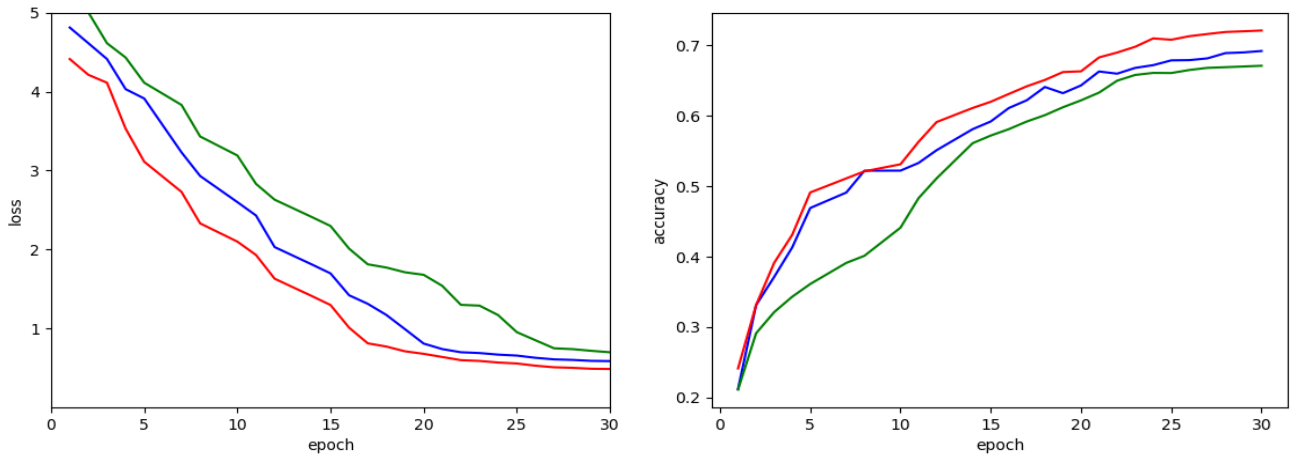


Fig. 16 Accuracy and loss for different batch sizes (red-64, blue-128, green-256)

A last important aspect regarding the quality of the model is represented by the size of the input, namely the number of musical information entered. I initially thought that having a larger input length will definitely increase the accuracy, however the experimentation yielded results that suggests that there is an optimal size and having a larger input length will not contribute to better results, so as we can see in the Figure 18 the most accurate result is given by the model with the input length 50.

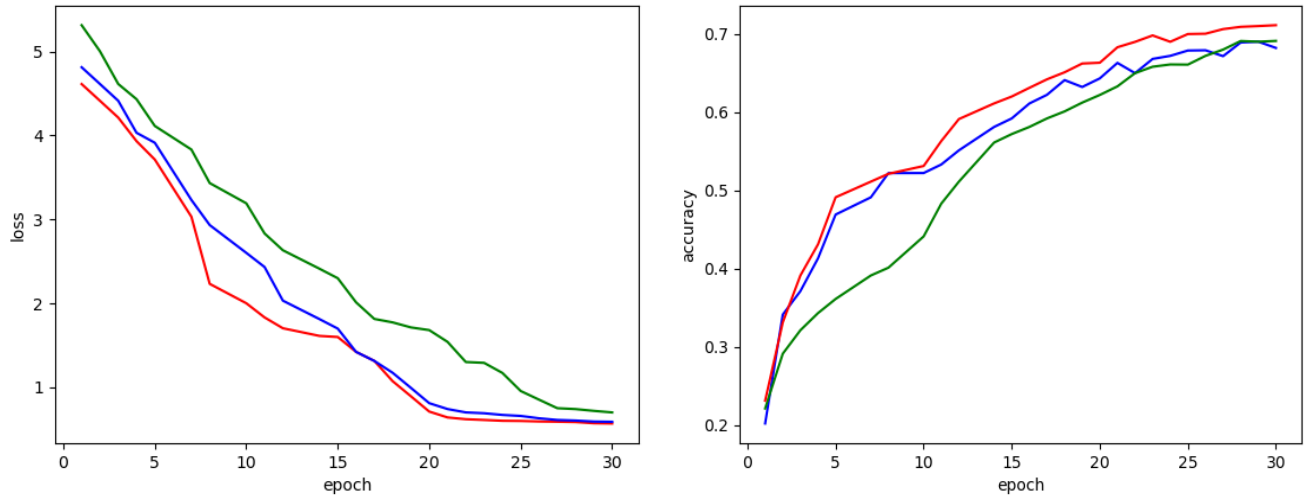


Fig. 17 Accuracy and loss for different input lengths (red-50, blue-80, green-100)

I would like to mention the fact that in addition to these experiments I also did a Touring test. It consisted of collecting the opinions of a few people about which songs sounds better, songs generated by two different models. Most of the opinions, also mine, were directed towards a certain model, a model with accuracy almost 12 percent lower than the other. In view of this fact, I would like to mention that the evaluation of the generated music is an extremely delicate process, and from my point of view its quality is something subjective which cannot be calculated in proportion of 100% by a computer.

Conclusions and future work

This paper aims to present the development of a system capable of generating audio content using long-short-term-memory neural networks. These have proven to be an excellent option in solving problems where the data set is sequential and closely related. As it could be seen in the previous chapter, the results were quite promising, obtaining an accuracy of over 70%. This accuracy reflects in an objective way the quality of predicting a musical information based on a previous sequence of musical information. Nonetheless, the quality of a song also consists of other aspects, which are mostly subjective. Music is a work of art in which the artist adds his personal touch and imposes his own style. As far as I am concerned, a computer does not have the capacity to create something similar, because the feelings which must be conveyed through music are strictly related to humanity.

During this research I noticed quite a lot of challenges, including the representation of audio content, its encoding and finding an architecture, all related to each other. The biggest challenge from my frame of reference, is the evaluation - as each evaluation is designed based on the author's priorities and perspectives. A good example is the CONCERT system in which the author states that the evaluation of his system shows that the audio content has a high percentage of plagiarism, but plagiarism refers not only to repeating musical notes but also to pitch, rhythm and melody.

A continuation of this work would consist primarily in the use of a more complex model trained on a much more powerful computer. Another improvement may be the usage of a better structured data set from which to extract more detailed information related to melodicity, tempo and other aspects that increase the quality of audio content.

To conclude, during this paper I came to the realization that, at this moment, a computer cannot replace an artist, or in other words a computer cannot create artistic audio content, but however, nowadays we need any kind of audio content which can be created with the help of artificial intelligence.

References

[1] Felix Sun. DeepHear – Composing and harmonizing music with neural networks, https://fephsun.github.io/2015/09/01/neural-music.html.

[2] Michael C. Mozer. Neural network composition by prediction: Exploring the benefits of psychophysical constraints and multiscale processing. Connection Science, 1994.

[3] Cedric Seger, An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing, Stockholm, Sweden, 2018

[4] Emilios Cambouropoulos, From MIDI to Traditional Musical Notation, Austrian Research Institute for Artificial Intelligence, Schottengasse 3, A-1010 Vienna, Austria, 2014

[5] Jurgen Schmidhuber, Sepp Hochreiter, LONG SHORT-TERM MEMORY, Fakultat fur Informatik Technische Universitat Munchen, Munchen, Germany, 1997

[6] Adrian Holovaty, Jacob K. Moss, The Definitive Guide to Django: Web Development Done Right, Apress publications, 2007.

[7] Francois Chollet, Deep learning with Python, Manning Publications Co, New York, 2018

[8] Cuthbert, Michael Scott and Christopher Ariza. "music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data." in J. Stephen Downie and Remco C. Veltkamp (Eds.). 11th International Society for Music Information Retrieval Conference (ISMIR 2010), August 9-13, 2010, Utrecht, Netherlands. pp. 637-642.

[9] MIDI.js API, https://www.midijs.net/

[10] Roger N. Shepard, Geometrical Approximations to the Structure of Musical Pitch, Stanford University, 1982

[11] V Kishore Ayyadevara, Neural Networks with Keras Cookbook, Packt Publishing Limited, 2019