Production Systems and Information Engineering
Volume 11 (1), pp. 1–15.
https://doi.org/10.32968/psaie.2023.1.1

DEVELOPMENT OF A MUSIC GENERATOR APPLICATION BASED ON ARTIFICIAL INTELLIGENCE

JANKA TAMÁS
University of Miskolc
Hungary Institute of Information Technology
tamas.janka95@gmail.com

LÁSZLÓ ÁRVAI
University of Miskolc
Hungary Institute of Information Technology
arvai.laszlo@iit.uni-miskolc.hu

Abstract. Automatic music generation is a process [1] that composes a short piece of music with minimal human intervention. It has a longer history than we might think, having been researched and developed for more than 60 years. With the emergence of deep learning, the automatic generation of different content (e.g. images, texts) has become a topical goal [2], so it is no wonder that many works have been done on music, such as melody generation, chord generation, performance generation, vocal tone generation, etc. Deep music generation refers to the automatic generation of music by computers using deep learning network architectures.

Keywords: music generation, deep learning, long short-term memory

1. Introduction

From the very beginning, the world of music has been greatly influenced by constantly evolving technologies. Since the advent of the first sound recordings, new technologies and tools have played an increasingly important role [3] not only in the processing and editing of sounds, but also in the processes of music creation and composition. The invention of electronic musical instruments has almost revolutionized and accelerated [4] the development of new forms of music, further stimulated by decades of research, discoveries and technological breakthroughs in the field of artificial intelligence, which are pushing the boundaries of the world of music and music creation. Artificial intelligence can create new sounds, write melodies, develop entire compositions and even recreate human singing. With technological advances of this magnitude and scope, questions arise as to how and to what extent AI will influence the music industry's creations and, even more excitingly, whether in the future will it take over the job of musicians?

There is no precise or clear definition to the seemingly simple question "What is music?", as it is a very complex and abstract concept that can be examined and defined from many different angles. By examining the definitions of thinkers from different eras [5], we have tried to summarize the characteristics on which there is overwhelming agreement.

In general, music is a series of temporally ordered alternations of sounds and 'silences' [6], which are temporally limited and auditory [7]. The perception of music is, however, highly subjective, being a hair's breadth away from the concept of noise, as the listener's perspective [8] determines how he or she identifies what is heard. The more orderly the observer perceives the noise he hears, the more he interprets the sounds (and silences) as music. As a consequence, it can be concluded that music is in fact interpreted noise.

2. The past of generated music

The idea of automatic music composition is not a new idea. In fact, long before the era of artificial neural networks, there were already many attempts [9] to automate the process of composing music.

As early as 1787, Mozart invented a so-called "dice game" for this purpose, in which he randomly selected melodies from a pre-composed set of 272 melodies based on the sum of 2 dice. This method enabled beginners to compose polonaises, minuets, waltzes, etc.

In the early 1950s, Iannis Xenakis used statistics and probability to compose music. The resulting music became known as stochastic music. He defined music as a series of randomly occurring elements (or sounds). His method of randomly selecting elements was based solely on mathematical calculations. In 1964, Koenig, another composer, implemented the PROJECT1 algorithm [10], which used serial composition and other techniques (such as Markov chains) to automate the generation of music.

Another interesting approach to music generation is the use of musical grammar (L. M. Zbikowski: *Foundations of Musical Grammar*). Musical grammar covers the knowledge needed to arrange and combine musical sounds accurately and to perform musical compositions correctly. The disadvantage of this method is that different rules have to be created to suit different types of music. The compositional process is often accompanied by constant intervention and fine-tuning in the post-production phase.
Recently, so-called deep learning architectures have become the method of choice for automatic music generation, with researchers achieving many outstanding results and unstoppable progress.

3. Overview of the elements of music

Since the subject of this paper is music generation, it is worth clarifying the basic concepts of music and what music consists of. Like anything else, music can be broken down into elements and basic units. These are defined [11] [12] below:

Sound: the smallest unit of music. The four characteristics of musical sound are pitch, volume, duration and timbre.

Beat: on the one hand, the smallest unit of rhythm, the unit of time between the relatively most weighty parts of a rhythm, which are regular and usually follow each other at equal intervals. On the other hand, a sequence of notes within a given unit.

Phrase: a set of beats (and notes within them). A relatively independent part of a song, sometimes considered as a whole in itself.

Song: a repetition of phrases. The musical work itself or a part of it.

Within the musical sound, it is important to focus specifically on the pitch attribute and the information that determines it. Pitch gives the height or depth of a sound. Pitch is determined by two pieces of information: the note and the octave.

Note: in music, there are seven types of notes, which are expressed by A, B, C, D, E, F and G, or modifications of these. Three modifiers are used to modify the notes: flats (♭), which lower the pitch a half-step, sharps (♯), which raise the pitch a half-step, and the natural symbol (♮) is used to cancel the effect of a flat or sharp.

Octave: defines the pitch range of a note. On a classical piano there are 88 keys (notes), which means that there are 7 octaves to determine the range of notes and to place notes within these octave ranges.

4. AI technologies for music generation

Deep learning is an advanced type of machine learning. While machine learning, as a subset of artificial intelligence, uses algorithms to identify patterns in the data and thus make increasingly accurate predictions over time, deep learning uses networks of algorithms inspired by the structure of the human brain, called neural networks, whereby each question that it answers leads to a set of related questions [13].

With the rise in popularity of deep learning, DL algorithms or models [14] have become the main method for researches about generating. The first such neural network architecture for music generation was the RNN (Recurrent neural network) model. However, the problem of gradient descent makes it difficult for RNNs to store long historical information about sequences. To solve this problem, a special RNN architecture was designed to help the network to memorize and retrieve the information in the sequence. The model is called LSTM, Long-Short-Term Memory Network. Although several successful results were achieved with the architecture, there were still difficulties in capturing the musical structure with the long term dependency, so researchers made some further attempts to innovate the RNN model (Melody RNN, Anticipation-RNN, BALSTM, etc.).

Later on, powerful deep generative models such as VAE (Variational autoencoder), GAN (Generative adversarial network) and Transformer appeared. MusicVAE is a hierarchical VAE model that is able to capture the long-term structure of polyphonic music, while also providing outstanding interpolation and reconstruction performance. GAN is a latent variable model coupled with a binary regularizer, which has performed effectively in improvised accompaniment generation. However, it has the disadvantage of being difficult to train and is typically used for non-sequential data.

Audio synthesis (the technique of generating sounds using electronic hardware or software) has made a major breakthrough in recent years with the advent of the WaveNet program, which can be used to generate speech and music waves. This was followed by GANSynth, which uses a GAN model to produce coherent sound by modeling logarithmic magnitudes and instantaneous frequencies with the appropriate frequency resolution.

5. Choosing genre, format and method

5.1. Music genre

Since there are already many composer and generator softwares on the market, it was very important for me to generate music in a genre that had not been encountered before and was unique. For this reason, the genre of Hungarian folk music was chosen. Folk songs are the embodiment and expression of human thoughts and emotions [15], the most ancient form of folk music regardless of culture. Folk songs have been passed down through oral tradition and handed down from generation to generation, so their authors are unknown. Hungarian folk music is extremely complex and consists of many different styles, due to their different origins and different levels of development. However, it is possible to identify some stylistic features that are characteristic of the majority of folk songs:

- Monophony.
- The melodic style is psalmody, psalm-like.
- It has a characteristic rhythm.
- Its performance style is parlando, i.e. narrative.
- Its tempo is rubato, i.e. free.
- The melodic range is narrow.
- The songs are characterized by pentatonicism, i.e. five degrees.

5.2. Music formats

As the AI will need a database of music files as input for music generation, it is worth getting to know the most commonly used music formats before embarking on any design or implementation process.

Image format

The largest amount of music resources available in written form, in the form of sheet music. Sheet music is easy for humans to read, but requires special processing for computers. This can be done using compiler programs based on artificial intelligence, but unfortunately these programs often produce inaccurate and erroneous results.

Audio format
When talking about music, most people think of sounds, recordings. It seems obvious to create the input data set from sound material. The biggest pitfall of this method is that the sound recordings are not clean and noisy, and they contain too much unnecessary information that misleads the AI. Furthermore, the most common audio format is .mp3, which is a lossy compression format.

standard MIDI
MIDI is an acronym for Musical Instrument Digital Interface [16]. Standard MIDI is a standardized protocol designed for communicating between synthesizers. MIDI files take up only five to six kilobytes of space for a one-minute piece of audio, containing only the pitch, rhythm and the instrument to be played.

ABC notation
The most common textual representation in music is the ABC notation. The .abc format is designed [17] to be easy to read and require little storage space, so that this “sheet format” is written in the language of the musical ABC notation.

5.3. Neural networks

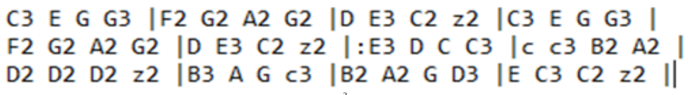
In a task as complex as composing music, the sequence of information determines the result itself [18]. Because if you mix up the sequence, you get a confused result. To get the right output, you need a neural network that has access to some prior knowledge about the information, the data. This is where the concept of recurrent neural networks (RNNs), mentioned earlier, comes in, which deals with sequence prediction problems. Since RNNs have an excellent ability to handle input and output of different types and lengths, they can be used in a variety of domains such as emotion classification, image captioning, language translation, etc.

However, recent breakthroughs in data science have shown that for almost all sequence prediction problems, a type of RNN, the long short-term memory networks (LSTM), is proving to be the most effective solution. One of the major advantages of LSTM over RNN is its ability to recognize and encode long term patterns. Although RNN models are a great solution when it comes to short sequences, in order for the program to compose a proper musical score, it is necessary that the models are able to understand and remember the context and the relationships behind the sequences, just like the human brain does.

5.4. Selected technology

As a result of this research, it can be concluded that the most appropriate format to choose as input to the LSTM neural network is the ABC music format, since it is textual data and therefore results in the highest efficiency in terms of learning the neural network.

The encoding of music in ABC notation consists of two parts: a header and a body. The header contains information such as title, time signature and key. The body mostly contains the notes, time signatures, etc. The notes are denoted by letters from A to G. The duration is indicated by the numbers next to the letters and the pause is indicated by a 'z'. The bars are separated by the character '|'.



```
C3 E G G3 |F2 G2 A2 G2 |D E3 C2 z2 |C3 E G G3 |
F2 G2 A2 G2 |D E3 C2 z2 |:E3 D C C3 |c c3 B2 A2 |
D2 D2 D2 z2 |B3 A G c3 |B2 A2 G D3 |E C3 C2 z2 ||
```

Figure 1. The ABC notation format

6. Creating the database

6.1. Collection of data

The preparation of the database started with the collection of folk songs, which resulted in 18 songs in MIDI and ABC format and 212 songs in MUS format. (Files with the extension .mus are classified as MIDI formats. The main difference is that it can carry data other than pitch, rhythm and instrument.) By the end of the collection, the database consisted of 231 different folk songs.

6.2. Conversion of data

Data processing started with the conversion of the files. This was done with the help of a program called Harmony Assistant, developed by Myraid SARL. The program can be used to convert, modify and write music for shorter sequences. After the conversion, it became apparent that the .abc files contained not only the necessary information, but also lyrics and comments beginning with “%”. The correctness of the exported files was checked in the input field of the website https://editor. drawthedots.com/, which converts the uploaded .abc data into sheet music and playable sound.

6.3. Preparation of data

The neural network requires a single input text file, but the database consists of 231 ABC files. The goal was to develop a program that could scan the files in a folder with a txt extension, edit them and output them to a single text file. Although the database consists of files with the extension .abc, their contents can be treated as text files.

To develop the software I used JavaScript and Node.js runtime environment. The data preparation program performs the following functions:

- Reads all files in a folder on the specified path.

- Removes comments that follow the comment sign "%".
- Removes the lyrics of the songs.
- Removes unnecessary blank lines.
- Merges file data into one data file.
- Removes commas created by concatenation.
- The X parameter (which is the ID number of the ABC file) are overwritten and re-indexed.
- The final data is written to a .txt file.

7. Building of the neural network

Designing and implementing an AI of this complexity on your own requires a rather high level of knowledge and experience, so it was necessary to look around and see what projects had already been done for this purpose. The project https://github.com/soumya997/Music-Generation-Using-Deep-Learning, in which the LSTM neural network is also trained using an ABC-format database, was a great help in the construction of the neural network.

7.1. Programming language and development environment

To implement the neural network, one of the most suitable and popular programming language and integrated development environment was chosen: Python and Jupyter Notebook. Python is the most widely used language in the field of artificial intelligence and machine learning, and there are also countless libraries available and applicable for a variety of specific purposes. One of Python's most popular IDE is Jupyter Notebook. It has proven to be very powerful for implementing various AI-based developments.

7.2. Layers of the neural network

We start building the neural network by creating weights, which perform transformations on the input data within the hidden layers of the network before passing it on to the next layer. Weights control the strength of the connection between two neurons, i.e. how much influence the input has on the output. After importing the appropriate libraries, we specify the destination folder to save the weights, and then define two functions. The save_weights(epoch, model) function is responsible for saving the weights to the specified folder. Weights are saved in .h5 format by calling the function. Then we define the function load_weights(epoch, model), which loads the weights from the specified folder. These weights are needed for the training, so the functions will be called at that time.

We define the build_model(batch_size, seq_len, vocab_size) function, which is called to build our model for training and generating weights. In the function's body, we build the sequential model using the Sequential() constructor, which is used for layers where each layer has exactly one input and one output tensor.

Embedding layer

First we start with the Embedding() layer, because the model needs to know what input source to expect, so the first layer of the sequential model needs to get information about the size and shape of the input. As parameters, we pass the number of characters of our text data set, the dimension of the vector space in which the words will be embedded, and finally we pass a batch_input_shape argument, which allows us to specify a fixed batch size for the model to accept input data, limiting the creation of a vector of any dimension. We then add three LSTM and three Dropout layers to the model.

LSTM layer
The layer is given the dimension of the output vector space. The return_sequence argument specifies whether a recurrent layer of the network returns either its entire output sequence (a sequence of vectors) to the next layer, or only its last output (a single vector). Since we want it to predict a character for each character in the sequence, we set this argument to True, ensuring that we get the right predictions. In this case, the last state of the samples with index i in one batch will be used as the initial state of the samples with index i in the next batch.

Dropout layer
The layer is given a parameter that has a float value between 0 and 1. This value determines the probability of a neuron leaving the network during a dropout, thus preventing overfitting. During the training process, the layer randomly sets the input units to 0 at each step with the probability we specify.

TimeDistributed(Dense) layer
After the layers created so far, the TimeDistributed(Dense()) layer can be created. In the case of many-to-many LSTM layers, the use of the TimeDistributed wrapper is unavoidable, as it allows the Dense layer to be applied to all outputs at each time step. This way, a fully connected Dense layer can be wrapped with a single output, so that the output layer only needs one connection to each LSTM unit.

Activation layer
The last step in building the model is to specify the activation function in the layer, which in this case is Softmax, which is a normalized exponential function. The Softmax brings the outputs into the range 0-1 so that the sum of the outputs is 1. This way, the result of Softmax will be a percentage distribution. This distribution shows the proportion of the resulting output that is similar to the input.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Figure 2. Softmax formula

7.3. Building of the neural network

We build the model with its function, and then plot the completed model with the Tensorflow plot_model method for illustrative purposes:

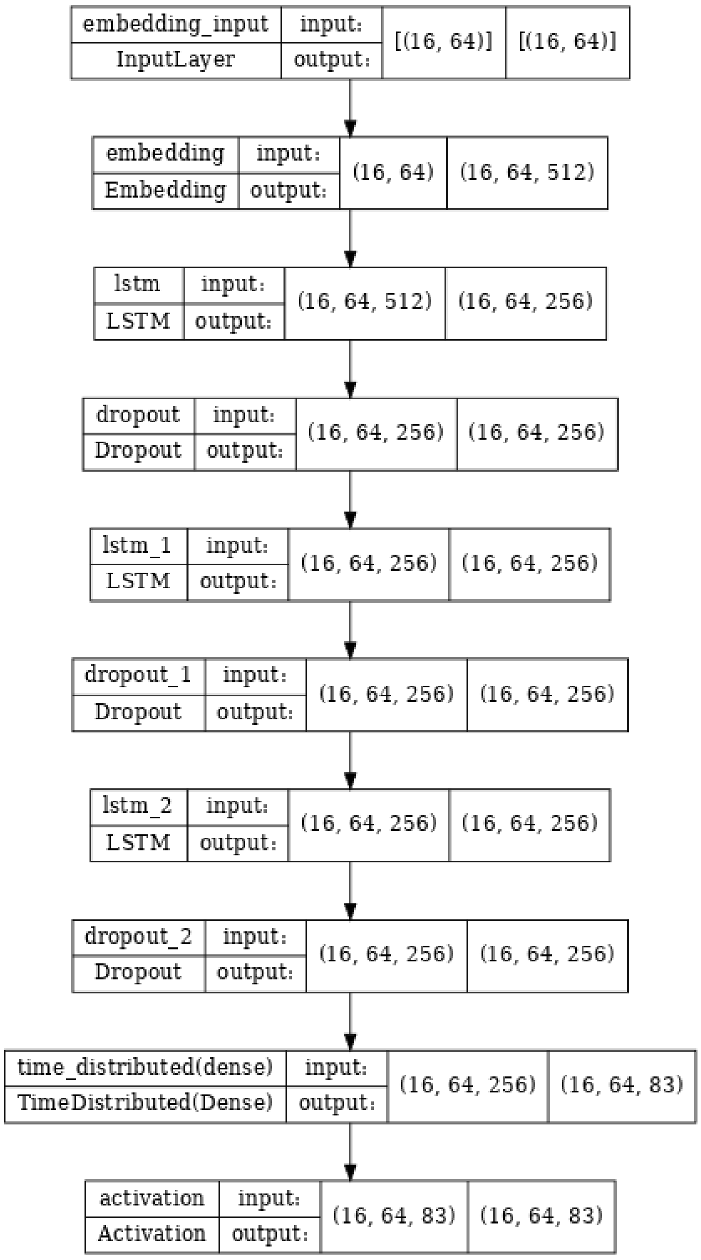


Figure 3. The built model

8. Training of the neural network

8.1. Preparation of the training

The train function we define performs model building, model learning, weights creation and saving. First, the elements of the input textual data are put in ascending order alphabetically, an index is associated with them, and then iterated through the dataset and stored in a JSON structure. The property of the JSON object is the given character, and its value is the index of the character. Finally, we also specify the length of the character set for the training. We then call its building function to build the neural network model, passing it the batch size, sequence length and character set length. Prior to the training, we need to define the loss function, the optimizers, and the prediction metrics. This is an essential step in the compilation process, as it configures the model for the training.

The loss function is used to calculate the amount that the model should minimize during training. For probabilistic classification models, the most commonly used loss function is cross entropy. In this case, we will use the Categorical Cross-Entropy function (also called Softmax Loss), which is a Softmax activation plus a Cross-Entropy loss. It is used for a multiclass classification model with two or more output labels. The function compares the distribution of predictions with the true distribution and computes the loss between these labels. It sets the probability of the real class to 1 and the other classes to 0. Finally, it assigns this category coding value to the output label.

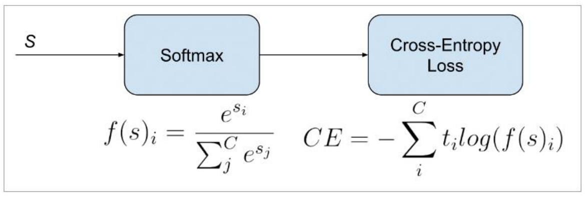

s
Softmax
Cross-Entropy Loss
$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$
$CE = -\sum_i^C t_i log(f(s)_i)$

Figure 4. Categorical Cross-Entropy loss

The optimizer optimizes the input weights by comparing the prediction and the loss function. There are a number of optimizers to choose from, with Adam (Adaptive Moment Estimation) being the preferred option, as it produces the most efficient training in the shortest time. Adam is an algorithm for optimizing a gradient descent algorithm, as it combines the advantages of several gradient descent methods. It computes the exponential moving average of the gradient and the quadratic gradient, and unlike the RMSProp, it not only adapts parameter training rates based on the average of the first moment (the mean), but also uses the average of the second moment (the uncentered variance) of the gradients. Besides speed, the Adam optimizer has the additional advantage of correcting for vanishing training rates and high variance.

The metrics show various calculations, reports and evaluations of the model's abilities and performance during the training process by tracking the learning process. For a neural network, one of the most commonly used metrics is accuracy, which calculates the frequency with which predictions are equal to labels. This will also be used for the model, as it is important to check whether the training rate is good or whether the neural network learns anything at all during the training process.

8.2. Training

Before the training, some concepts need to be clarified. The batch or batch_size refers to the amount of data batches that are transferred to the neural network during a training session, as we cannot transfer the whole data set at once. Epoch means one iteration over the entire data set, i.e. one complete learning phase. The number of iterations an epoch needs to run depends on the size of the batch and the amount of the entire dataset, since within an epoch there will be as many iterations in which the entire dataset can be passed through the network.
Then a cycle is created that iterates over the epoch range specified by the train function parameter. Two arrays are created to store loss and accuracy data. In a subsequent loop, we iterate through the generated batch matrices X and Y returned by the call to read_batches(), and then use the train_on_batch method to generate weights based on the specified collection of samples. Their return values, which are scalar loss and scalar accuracy, are passed to the loss and acc variables. These loss and precision data are stored in the arrays losses and accs, which are the return values of the training function. Subsequently, the total layer weights of the model are saved every 10 epochs. After the function definitions, the train function is called to perform the training and to generate the weights, passing the input.txt file converted to an input string as a parameter. Then the training starts with the weights being saved.

```
for epoch in range(epochs):
        print('\nEpoch {}/{}'.format(epoch + 1, epochs))

        losses, accs = [], []

        for i, (X, Y) in enumerate(read_batches(T, vocab_size)):

            loss, acc = model.train_on_batch(X, Y)
            losses.append(loss)
            accs.append(acc)
```

```
        if (epoch + 1) % save_freq == 0:
            save_weights(epoch + 1,model)
            print('Saved checkpoint to',
            'weights.{}.h5'.format(epoch + 1))
```

```
losses,accs = train(txt)
```

9. Generate a music sample

Once the weights have been created, we can build the model for generating the music sample and the sample generation procedure. For both procedures, we define a function.

The build_sample_model is defined differently from the build_model function in that the batch_input_shape and return_sequence arguments are set to different values. In this case, batch_input_shape is set to (1,1), which is the dimension of the batch. The return_sequence argument is now given (i != 2), which means that only for the first and third LSTM layers the complete output sequence is passed to the next layer; for the second layer only the output of the last hidden state is passed. Since we are building a generative model that intends to generate Softmax probabilities to predict the next character, it is not necessary to return the full sequence at each time step.

The sample function is used to generate our music sample. First, we load the JSON file containing the indexed character set, then we build our model, and load the last generated one from the previously saved weights. We create a sampled array where we will store each predicted and randomized sample. In one loop, we iterate through the specified set of characters to generate, creating a two-dimensional batch null array of the form (1,1) and then assigning a value based on the contents of the sampled array. If the sampled array is not empty, we pass its last element to the batch array. If the array is empty, we fill it with random values between zero and the number of characters in the character set. Then, a prediction is generated in the result variable for the given batch sample, and a character index from the range of the character set is generated using the probability of the result value just obtained and passed to the sample variable. Finally, the generated variable is added to the sampled array. By iterating through the sampled array containing the character indices as the return value of the function, we find and then concatenate the characters associated with each index using a space between each character. Finally, we call the sample function we just defined, which, after building the model and loading the number of epoch weights specified as parameters, produces the specified number of characters specified.

```
def sample(epoch, num_chars):
    with open('data/char_to_idx.json') as f:
        char_to_idx = json.load(f)
    idx_to_char = { i: ch for (ch, i) in char_to_idx.items() }
    vocab_size = len(char_to_idx)
    model = build_sample_model(vocab_size)
    load_weights(epoch,model)
    sampled = []
```

```
for i in range(num_chars):
    print (i)
    batch = np.zeros((1, 1))
    if sampled:
        batch[0, 0] = sampled[-1]
    else:
        batch[0, 0] = np.random.randint(vocab_size)
```

```
        result = model.predict_on_batch(batch).ravel()
        sample = np.random.choice(range(vocab_size), p=result)
        sampled.append(sample)
return ''.join(idx_to_char[c] for c in sampled)
```

The generation took just a minute or two, and the result was a string with the specified number of characters. As a final step, the string is printed out using the print function, so that the new lines are nicely displayed and the generated song is properly segmented. By copying and pasting it into the input field of the website https://editor.drawthedots.com/, we can play and listen to the generated song, and even download it.

10. Evaluation

After a successful generation, we examine the loss and accuracy values collected during the training follow-up. The X-axis represents the number of epochs, i.e. the total number of training stages for both diagrams. The Y axis represents the loss for the first diagram and the accuracy between 0 and 1 for the second. Generally, if the accuracy is high and the loss is low, the model will make few or small errors on some of the data, which is the ideal case for the model to work.

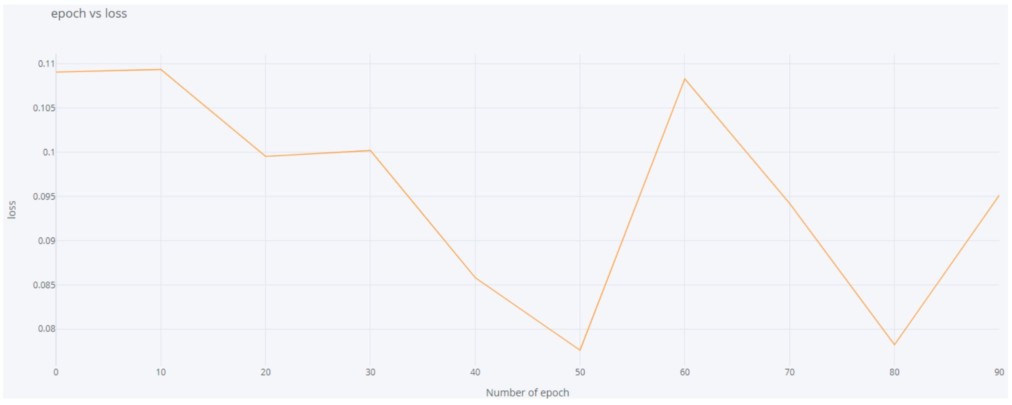


Figure 5. Loss diagram

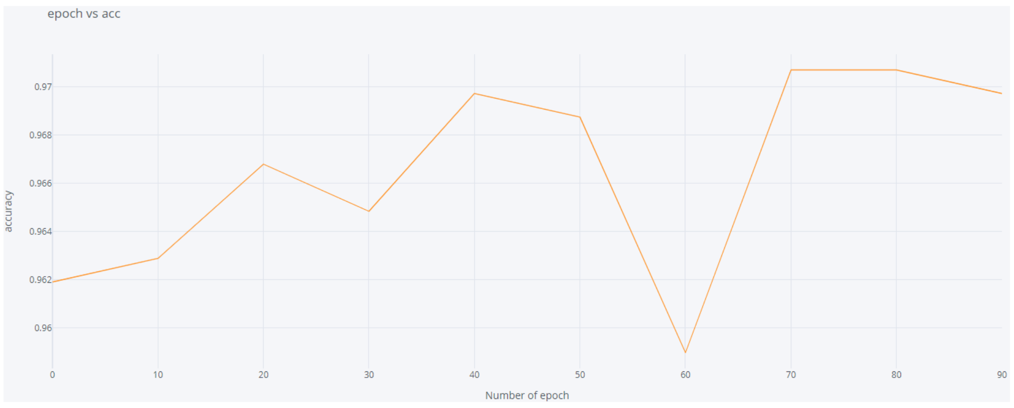


Figure 6. Accuracy diagram

11. Conclusion

The primary goal of the project was to achieve a successful result in the field of music generation by artificial intelligence. It was very interesting to include the genre of Hungarian folk song, which is a very unique blend of folk music from different cultures. However, this objective was only partially achieved, as the neural network did not produce the kind of songs that were expected of it. On the one hand, in some cases there were too many similarities between a particular Hungarian folk song and the generated song, and on the other hand, the network randomly chose a value for the parameter that determines the key of the music and the note. However, the key cannot be separated from the notes, as it can cause dissonance. The problem of too much similarity is presumably due to the small size of the database. In addition, it is worth experimenting with modifying the values and functions of the neural network for better results.

References

[1] Aravindpai Pai. *Want to Generate your own Music using Deep Learning? Here's a Guide to do just that!* 2020. URL: https://www.analyticsvidhya.com/blog/2020/01/how-to-perform-automatic-music-generation/ (Downloaded: 20. 11. 2022.)

[2] Xinyu Yang Shulei Ji, Jing Luo. *A comprehensive survey on deep music generation: Multi-level representations, algorithms, evaluations, and future directions.* 2020. URL: https://arxiv.org/pdf/2011.06801.pdf (Downloaded: 20. 11. 2022.)

[3] Dillon Ranwala. *The Evolution of Music and AI Technology*. 2020. URL: https://watt-ai.github.io/blog/music_ai_evolution (Downloaded: 20. 11. 2022.)

[4] Chong Li. *A Retrospective of AI + Music.* (2019.) URL: https://blog.prototypr.io/a-retrospective-of-ai-music-95bfa9b38531 (Downloaded: 21. 11. 2022.)

[5] Gyula Kovácsy. *Mi a zene?* URL: http://beethovenszimfoniak.hu/Szemelvenyek/14_Mi%20a%20zene.htm (Downloaded: 21. 11. 2022.)

[6] Balázs Török-Szabó. *Értelmes zajok rendszere – mi teszi a zenét zenévé?* (2021.) URL: https://papageno.hu/blogok/animo/2021/06/ertelmes-zajok-rendszere-mi-teszi-a-ze net-zeneve/ (Downloaded: 21. 11. 2022.)

[7] Róbert Roth, Kinga Bernád. *Zenegenerálás, "majdnem" természetes zene.* URL: https://docplayer.hu/3413584-Zenegeneralas-majdnem-termeszetes-zene-bernad-kin ga-es-roth-robert.html (Downloaded: 22. 11. 2022.)

[8] Wikipedia. *Zene.* URL: https://hu.wikipedia.org/wiki/Zene (Downloaded: 20. 11. 2022.)

[9] MTA Nyelvtudományi Intézet. *A magyar nyelv értelmező szótára.* https://www.arca num.com/hu/online-kiadvanyok/Lexikonok-a-magyar-nyelv-ertelmezo-szotara-1BE8B/ (Downloaded: 22. 11. 2022.)

[10] *What is artificial intelligence?* URL: https://azure.microsoft.com/en-us/resources/ cloud-computing-dictionary/what-is-artificial-intelligence (Downloaded: 22. 11. 2022.)

[11] *A magyar népdal.* URL: http://www.magyarnota.com/blog/A_magyar_nepdal.html (Downloaded: 23. 11. 2022.)

[12] Amandaghassaei. *What Is MIDI?* URL: https://www.instructables.com/What-is-MIDI/ (Downloaded: 23. 11. 2022.)

[13] *About abc notation.* URL: https://abcnotation.com/about (Downloaded: 23. 11. 2022.)

[14] Pranj52 Srivastava. *Essentials of Deep Learning : Introduction to Long Short Term Memory.* URL: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/ (Downloaded: 24. 11. 2022.)