Generating Music with Recurrent Neural Networks

Daniel Gareev
University of Luxembourg
Email: daniel.gareev.001@student.uni.lu

Amro Najjar
University of Luxembourg
Email: amro.najjar@uni.lu

Abstract

With the advent of deep learning, researchers are exploring the opportunities to apply neural networks to sequential data such as audio and music. The goal of the project is to use neural networks to generate music of a specific genre. One of the ways for modeling sequence data with neural networks is by Recurrent Neural Networks (RNNs). In this project, we introduce the basic elements of RNNs and its applications in sequence learning. Then, we use MelodyRNN, a model from Google's open-sourced research project Magenta, to generate melodies in a classical music genre. Next, we introduce an interactive web application called MelodyMe, which is built upon the pre-trained MelodyRNN model. The application shows how the neural network responds by letting one play a melody and seeing how the neural network continues it. With MelodyMe, we hope to raise awareness of the black box nature of the deep learning and neural networks.

1. Introduction

The current technological advancements have transformed the way we not only produce, but listen and work with music. Traditionally, music was treated as and was generated manually. However, with the extensive development of deep learning, it has now become possible to generate music without human intervention. This offers artists more creative inspiration and the ability to explore different domains in music.

There are two classes of music generation approaches, symbolic music generation and audio music generation [4]. In this project, we focus on symbolic melody generation, which requires learning from musical notes. Many music genres, such as classical music, consist of melody and harmony. Melody is a sequence of musical notes along time. The notes have different pitches, keys, velocities, and time scales, making the melody generation a difficult task.

Most existing neural network models for music generation use recurrent neural networks (RNNs). However, traditional RNNs fall short when trying to learn long-term structure. Therefore, different types of RNNs have been widely applied to the sequential data to get more meaningful and interesting results. For example, a special kind of RNN, Long short-term memory (LSTM) has the ability to retain the long-term contents of the sequential data better than the traditional RNNs.

In this project, we first introduce the basic elements of RNNs and its specific type, LSTM. Then, we review the state-of-the-art related work in the area of music generation with neural networks. Then, we use MelodyRNN, a model from Google's open-sourced research project Magenta to generate melodies in a classical music genre. To implement this, we first train the model on a large-scale dataset of classical music and then use that pre-trained model to generate new melodies. To evaluate the generated melodies, we analyze the music qualitatively based on a number of criteria.

Finally, we introduce an interactive web application called MelodyMe, which is built upon the pre-trained MelodyRNN model. The application shows how the neural network responds by letting one play a melody and seeing how the neural network continues it. With MelodyMe, we hope to raise awareness of the black box nature of the deep learning and neural networks.

2. Project description

2.1. Domains

2.1.1. Scientific. The scientific domain of the project mainly focuses on two following areas. First, we are exposed to recurrent neural networks (RNNs) and their application in the area of music generation. Second, we focus on the explainability of artificial intelligence.

2.1.2. Technical. The primary technical domain of this project focuses on generating music with Recurrent Neural Networks (RNNs). We use Google's open-source research project Magenta model called MelodyRNN and train it on a large-scale MIDI dataset to generate melodies [10]. The project also extents to areas of web development technologies and prototyping as we aim to develop a web application that demonstrates a process of music generation with neural networks.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables. The main scientific deliverable of the project is to learn about Recurrent Neural Network (RNNs) and Long short-term memory (LSTM), a special type of RNN, capable of learning long-term dependencies. Furthermore, it is important to provide an overview of the applications of neural networks in generating sequential data (such as audio and music) by reviewing the state-of-the-art work in the field. This project is also linked with AI & Art pavilion in Esch 2022 capital for European Culture. Since we are leveraging neural networks to generate music, one of the goals of the project is to improve transparency with human-interpretable explanations of RNNs.

2.2.2. Technical deliverables. The main technical side of the project is using and experimenting with RNNs, which are a specific architecture of neural networks that is often used to generate sequential data (such as music or images). The model will be trained on the MIDI dataset of the specific genre. The model should be able to generate new music by accompanying a MIDI track given a priori by the user. The exact version of the recurrent neural network should be determined as a consequence of the research process.

In this project, the Magenta Python library is used to generate music [10]. The package includes utilities for manipulating source data, using this data to train machine learning models, and finally generating new content from these models [10]. Magenta for TensorFlow features several pre-trained models. In this project, we use MelodyRNN, which is a model that applies language modeling to melody generation using an LSTM. We first train MelodyRNN model with a large-scale MIDI dataset of classical music. Next, we use our pre-trained model to generate melodies by giving it a note sequence to continue.

Furthermore, as the project is expected to be presented in AI & Art pavilion, we will create an application that lets one play a melody and see how neural network responds to it by continuing it. Therefore, one of the objectives of the project is to prototype and design an architecture for the application.

3. Pre-requisites

3.1. Scientific pre-requisites

One of the scientific pre-requisites is a basic understanding of deep learning fundamentals. In addition, basic knowledge in music generation and familiarity with the concepts such as Musical Instrument Digital Interface (MIDI), a digital audio workstation (DAW) is required. Moreover, familiarity with some of the basic musical concepts is needed (such as notes, bars, time signatures, keys, and beat, etc.).

3.2. Technical pre-requisites

The required technical prerequisites to start working on this project are prior programming experience in Python programming language, as well as experience with machine learning. Furthermore, a background in graphic design and web technologies (such as HTML, CSS, Javascript) is required to implement the front-end of the application .

4. A Scientific Deliverable 1: Background on Recurrent Neural Networks (RNNs)

4.1. Requirements

The main objective of this deliverable is to build an understanding of the fundamental elements of recurrent neural networks (RNNs). This section thus provides an introduction to the traditional ”feedforward” recurrent neural networks and a more advanced one, long short-term memory (LSTM).

4.2. Design

The section is produced by reviewing a set of academic articles and literature in the area of deep learning and, specifically, recurrent neural networks in order to obtain background knowledge necessary for implementing the project.

4.3. Production

Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states [1]. RNNs perform the same function for every element of a sequence, with the output being dependent on the previous computations. RNN has looped, or recurrent, connections that allow the network to hold information across inputs. RNNs can also use their memory to process sequences of connections. Neural network algorithms are inspired by the functioning of the human brain as they try to replicate the nature and functioning of the neural network. They are typically structured as follows:

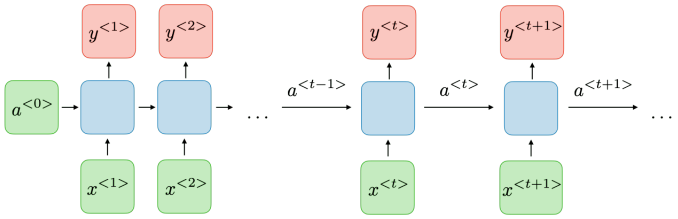

Fig. 1. Architecture of RNN. Figure source: [1]

In traditional neural networks, also called ”feedforward neural networks”, the information moves linearly in one direction, starting from the input layer. Then, it sequentially passes through the hidden layers until it reaches the output layer [1]. RNN can also be thought of as multiple copies of the same network, each passing a message to a successor [20]. As in the human brain, the information that has been computed at the previous state is being carried over time through the hidden state of the neural network. For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{1}$$

and

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y) \tag{2}$$

where W_{ax}, W_{aa}, W_{ya}, b_a, b_y are coefficients that are shared temporally and g_1 and g_2 activation functions [1].

RNN models are mostly used in the fields of natural language processing and speech recognition. Nevertheless, they are also used in the areas of music generation, sentiment classification, name entity recognition, and machine translation [1]. RNNs are particularly useful for learning sequential data like music.

RNN models have a set of advantages, such as the possibility of processing input of any length, RNN model size not increasing with the size of the input, and the RNN computation takes into account historical information [1]. Nevertheless, the computation of the RNN models is often slow, RNN models cannot consider any future input for the current state, and the model has difficulty accessing information from a long time ago [1].

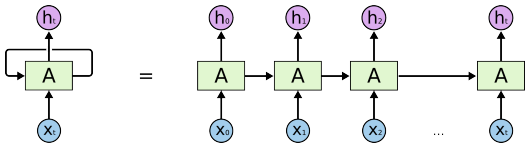

Fig. 2. An RNN's recurrent connection unrolled through time. Figure source: [20]

An RNNs can learn the temporal structure of any length in the input sequence, but as the sequence gets longer, it falls short as the outputs they generate generally lack coherent long-term structure. Unlike traditional feedforward neural networks, long short-term memory (LSTM) recurrent neural network has feedback connections. This type of RNN is able to learn order dependence in sequence data (such as music or speech). They work really well on a large number of problems and are now widely used. LSTMs are explicitly designed to avoid the long-term dependency problem [20].

All recurrent neural networks can be thought of as a form of a chain of repeating modules of neural network [20]. In standard RNNs, this repeating module will have a very simple structure [20] (as seen in Figure 3). LSTMs also have this structure, but the repeating module has a different structure [20] (as seen in Figure 4). Instead of having a single neural network layer, there are several ones that interact in a special way [20].

The key to LSTMs is the cell state, which is the horizontal line that runs through the top of the diagram [20]. The cell state can be thought of as a conveyor belt, which runs down the entire chain, with only some minor linear interactions [20]. The LSTM has the ability to remove or add information to the cell state, carefully regulated by structures called gates. So, it is easy for information to flow along it unchanged [20].

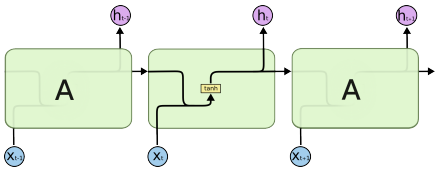

Fig. 3. The repeating module in a standard RNN. Figure source: [20]

4.4. Assessment

We have presented background on RNNs. This section established an overview of traditional neural networks, and

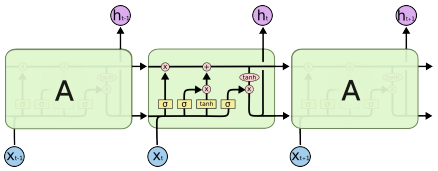

Fig. 4. The repeating module in an LSTM. Figure source: [20]

LSTM network, a more advanced type of RNN.

5. A Scientific Deliverable 2: Music Generation with Neural Networks

5.1. Requirements

The goal of this deliverable is to review the state-of-the-art related work in the field of music generation and machine learning. Specifically, the related work presented in this section focuses on generating music using neural networks. It is also important to analyze the pros and cons of each of the technologies.

5.2. Design

We produce the section by manually researching related work in the area of music generation with neural networks. This section presents an overview of existing models for music generation that produce meaningful content. We also compare the performances and advantages of the mentioned models.

5.3. Production

Music generation using neural networks has attracted much attention. Furthermore, a large number of deep neural network models have been proposed for modeling sequence data (such as music). This includes models for generating a melody sequence or audio waveforms. There are two classes of music generation approaches, symbolic music generation and audio music generation [4]. In this project, we focus on symbolic melody generation, which requires learning from musical notes.

Symbolic melody synthesis is a complex machine learning task, as human perception of the music is sensitive to both global structure and fine-scale coherence. One of the difficult problems in using machine learning to generate sequences, such as melodies, is creating long-term structure. While long-term structure comes naturally to humans, it is hard for machines. For example, basic machine learning algorithms can generate a short melody that stays in the same key, but it is hard to synthesize a longer melody that follows a pattern, chord progression or follows a multi-bar song structure of verses [5]. Without long-term structure, the content produced by recurrent neural networks (RNNs) often seems random [5].

The majority of existing neural network models for music generation use recurrent neural networks (RNNs), which contain loops that allow the multidirectional flow of information between layers. Specifically, Long Short-Term Memory (LSTM) network is often used as it is able to recognize and encode long-term patterns. Recent works have shown success in generating music by using LSTMs as the network is able to remember the sequential order of the information for a more extended period of time than the traditional RNNs. For instance, in one of the recent works, researchers have successfully implemented a model that is able to learn the sequences of polyphonic musical notes over a single-layered LSTM network and produce stellar results in composing new melodies [3]. In another study, researchers focused on developing a hierarchical recurrent neural network for melody generation, which consists of three Long-Short-Term-Memory (LSTM) subnetworks [4]. They claim that the demonstrated structure of the network has an advantage over the single-layer LSTM, which attempts to learn all hidden structures in melodies [4]. The MelodyRNN models proposed by the Magenta Project from the Google Brain team are among the most popular examples of symbolic-domain music generation by neural networks, which aim to improve RNNs' ability to learn longer-term structures [5]. MelodyRNN is best at continuing the note sequences, but it cannot encode long-term structure to produce full compositions [8].

There are other neural networks that have been applied to sequential data, such as Generative Adversarial Network (GANs), which is a state-of-the-art method for generating high-quality images. For instance, in a recent study, the researchers use a generative adversarial network (GAN) to learn the distributions of melodies [6]. They propose that the resulting model, named MidiNet, can be expanded to generate music with multiple MIDI channels [6]. Furthermore, they report that their results show that MidiNet performs comparably with MelodyRNN models in being realistic and pleasant to listen to, yet MidiNet's melodies are much more interesting [6]. In a different study, the authors propose MuseGAN, three models for symbolic multi-track music generation under the framework of generative adversarial networks (GANs) that are able to generate coherent music of four bars right from scratch or generate additional tracks to accompany a given specific track composed by human [7].

Recent works have also shown success in generating music using a Variational Autoencoder (VAE). For example, the Google Brain team developed MusicVAE, a hierarchical recurrent variational autoencoder for learning latent spaces for musical scores [8]. By using the hierarchical decoder, the authors addressed the problem of the existing recurrent VAE models that often have difficulty modeling sequences with long-term structure [9]. The model provides different modes of musical creation, such as random sampling from the prior distribution, interpolation between existing sequences, and manipulation of existing sequences [9]. While GAN and VAE are both generative architectures that generate from random latent variables, VAE is representational of the whole training dataset, and GAN is not.

Some models are designed to generate compositions of multi-track music. One of the leading artificial intelligence research laboratories, OpenAI, recently developed MuseNet, a deep neural network that can generate 4-minute musical compositions with 10 different instruments and can combine different music styles [13]. This advanced model uses the recompute and optimized kernels of Sparse Transformer to train a neural network and is able to remember long-term structure in a musical piece [13].

5.4. Assessment

In this section, the state-of-the-art of music generation with neural networks have been presented. We have provided an overview of existing models for music generation that produce meaningful music content. Furthermore, we also analyzed the pros and cons of the technologies.

6. A Scientific Deliverable 3: Explainable Artificial Intelligence

6.1. Requirements

The main requirement is to introduce the concept of Explainable Artificial Intelligence (XAI). Specifically, we would like to take a look at the concept in the context of the music generation.

6.2. Design

This deliverable is produced by researching the state-of-the-art in XAI and its research directions in the field of music generation.

6.3. Production

Explainable Artificial Intelligence (XAI) has seen growth over the last few years due to the widespread application of machine learning, and specifically, deep learning. This has led to the development of highly accurate models but lack explainability and interpretability.

There are multiple criteria associated with the concept of XAI. First, explainability, a way to explain, in an understandable way, how a model arrives at given predictions or behaves in a certain way. Second, transparency, a way to track how and where Artificial Intelligence has been used to make predictions. is another important aspect of AI development and application. These factors do not only emphasize a need of measuring the existing AI applications but also promote new methods for designing AI systems with these criteria in mind.

In recent years, with the extensive development of deep learning, it has now become possible to generate music without the need of working with instruments or music

software. For example, some of the companies such as Aiva are now producing music for commercial purposes (such as soundtracks) [21]. Therefore, there are multiple implications that are coming along with music generation.

The number of scientific articles and conferences in the field of Explainable AI (XAI) has significantly increased in many domains, with the music generation field being largely unnoticed. In music generation, the explainability of AI can help to address questions of how music pieces were generated by a model. In many cases, the inner mechanisms of neural networks or deep learning are not transparent or understandable because they are considered as a black box, which makes it difficult to explain and justify their decisions. This is an important issue, as we want to understand and explain what (and how) a deep learning system has learned from training data and why it ends up generating a given musical content.

In this project, we hope to raise awareness of the importance of the topic by introducing MelodyMe, an application that shows how the neural network responds by letting one play a melody and seeing how the neural network continues it. It is intentionally designed as a black box, making it difficult to explain why a model arrives at a specific prediction.

6.4. Assessment

In this section, we have introduced the concept of Explainable Artificial Intelligence (XAI). Explainability and transparency of deep learning models is an important issue in the music generation area. Therefore, it is important to continue to deepen our understanding and to explore solutions.

7. A Technical Deliverable 1: Generating Music with MelodyRNN

7.1. Requirements

The goal of the deliverable is to use Recurrent Neural Networks (RNNs) to generate music in a classical music genre. We use Magenta's MelodyRNN model to continue the music sequences.

7.2. Design

We follow Melody RNN docs on Github to train the MelodyRNN model [11]. We also use the Hello Magenta Colab notebook [2] to get started with generating melodies with the MelodyRNN model.

7.3. Production

7.3.1. Introduction to MelodyRNN. Melody-RNN comes from Google's open-source project Magenta that explores the role of machine learning as a tool in the creative process [10]. "The library includes utilities for manipulating source data (primarily music and images), using this data to train machine learning models, and generating new content from these models" [10]. The Magenta project has several machine learning models build with Tensorflow, each with different strengths. A MelodyRNN is an LSTM-based model for musical notes. Currently, there are three types of Melody-RNN models. BasicRNN is a dual-layer LSTM model, which uses one-hot encoding to represent extracted melodies as input to the LSTM [11]. The second model is Lookback RNN, which introduces the lookback feature that makes the model recognize patterns and repeat sequences easier [11]. The last one is Attention RNN, which leverages the use of attention to allow the model to access past information more easily without having to store that information in the RNN cell's state [11]. In this project, we focus on AttentionRNN since it the most sophisticated model that is able to to learn longer-term structures among three types.

7.3.2. Dataset. As MelodyRNN is designed to work with MIDI note sequences, we used a large-scale MIDI dataset for classical piano music called GiantMIDI [14]. The dataset contains 10,854 unique piano solo pieces composed by 2,786 composers [14]. The dataset is collected by applying a convolutional neural network to detect piano solo pieces from the audio recordings from the internet [14]. To obtain the dataset, we acquired a disclaimer to get a link to download GiantMIDI-Piano from the creators as described in [15].

We also attempted to collect our own training dataset, which consisted of classical piano music categorized by composers. The dataset contained 296 MIDI piano compositions, which range in length from 30 seconds to several minutes. However, we were able to achieve more interesting and meaningful music results with the larger dataset GiantMIDI. We also tried to train the MelodyRNN model on other genres of music. For example, we have manually collected over 30 Irish music MIDIs to train the model on. However, as we found out that there is a lack of large-scale Irish music datasets, we focused on training our model with classical music compositions.

7.3.3. Training MelodyRNN. Our first step in training the model is to build the first MIDI dataset. We convert a collection of MIDI files into NoteSequences. NoteSequences are an abstract representation of a series of notes, which is a fast and efficient data format, and easier to work with than MIDI files [11]. We do this by creating a directory of MIDI files and converting them into NoteSequences as shown in Listing 1. Magenta makes it easy to create training datasets from any collection of MIDI files. The NoteSequences are output to `/tmp/notesequences.tfrecord`.

Then, we extract melodies from our NoteSequences and save them as SequenceExamples. SequenceExamples are fed into the model during training and evaluation [11]. Each SequenceExample contains a sequence of inputs and a sequence of labels that represent a melody [11]. The command in the

```
python magenta/scripts/convert_dir_to_note_sequences
    .py --input_dir='MIDIs' --output_file='tmp/
    notesequences.tfrecord'  --recursive
```

Listing 1. Create NoteSequences

Listing 2 below extracts melodies from our NoteSequences and saves them as SequenceExamples. We generate two collections of SequenceExamples, one for training, and one for evaluation. The fraction of SequenceExamples in the testing set is set by the `eval_ratio` parameter. In our case, with the `eval_ratio` of 0.10, 10% of the extracted melodies are saved in the evaluation set, and 90% are saved in the training set.

```
python magenta/models/melody_rnn/
    melody_rnn_create_dataset.py --config='
    attention_rnn' --input='tmp/notesequences.
    tfrecord' --output_dir='tmp/melody_rnn/
    sequence_examples' --eval_ratio=0.10
```

Listing 2. Create SequenceExamples

Next, we train the MelodyRNN model. We start training using the AttentionRNN configuration as seen in Listing 3. We store the checkpoints and TensorBoard data for this training run in `--run_dir` directory. We fed the TFRecord file of SequenceExamples to the model by setting the `--sequence_example_file` parameter. Next, we set the number of training steps `--num_training_steps` to 20,000. In our case, 20,000 update steps are taken before exiting the training loop. We also specify a batch size of 64 instead of the default batch size of 128 to reduce memory usage and prevent out-of-memory errors when training larger models.

```
python magenta/models/melody_rnn/melody_rnn_train.py
     --config='attention_rnn' --run_dir='tmp/
    melody_rnn/logdir/run1' --sequence_example_file=
    'tmp/melody_rnn/sequence_examples/
    training_melodies.tfrecord' --hparams='
    batch_size=64,rnn_layer_sizes=[64,64]' --
    num_training_steps=20000
```

Listing 3. Train and evaluate the model

Finally, we generate a bundle file of the model as we use it when predicting melodies with MelodyRNN. The bundle format merges the model checkpoint, metagraph, and metadata about the model into a single file [11]. We do so by using the the `create_bundle_file` method within SequenceGenerator as seen in Listing 4. The bundle file is output to `tmp/classical_attention_rnn.mag`. After running the training, we can run TensorBoard to view the training and evaluation data.

The whole process of training MelodyRNN model is shown in Figure 5.

It took us over 4 hours to train the AttentonRNN model, with an average of 1.3 seconds per global step. The metrics

```
python magenta/models/melody_rnn/melody_rnn_generate
    .py --config='attention_rnn' --run_dir='tmp/
    melody_rnn/logdir/run1' --hparams='batch_size
    =64,rnn_layer_sizes=[64,64]' --bundle_file='tmp/
    classical_attention_rnn.mag' --
    save_generator_bundle
```

Listing 4. Creating a bundle file

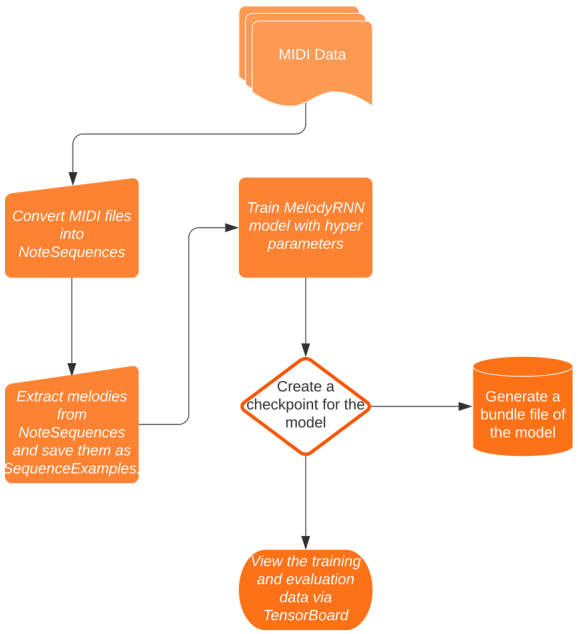

Fig. 5. Training MelodyRNN

are from the run are recorded in the TensorBoard. The accuracy of the model has a very high score of 99% over 20,000 training steps, as seen in Figure 6. The loss of the model is on average 0.03, as seen in Figure 7.

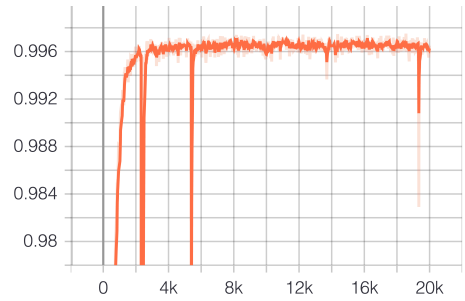

Fig. 6. Model accuracy

We applied the datasets to the model while constantly comparing the generated musical pieces manually. We have also experimented with BasicRNN and LookbackRNN models by training them on the same dataset and manually comparing the generated melodies. Nevertheless, the AttentionRNN model seemed to produce the most interesting music with the ability to retain the long-term structure of the melody. Furthermore, we have tuned the model with different hy-

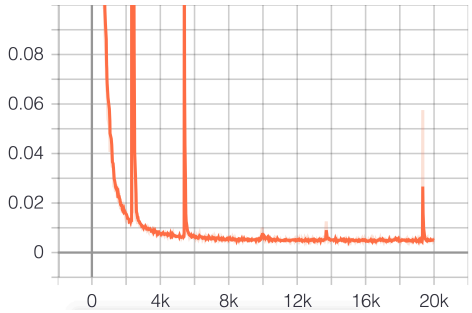

Fig. 7. Model loss

perparameters. We tried a larger batch size, less number of training steps, and the lower length of attention (the number of steps the attention mechanism looks at).

7.3.4. Generating Melodies. Melodies can be generated after training the MelodyRNN model. We can generate a set of melodies using the latest checkpoint file of the trained model. To generate melodies with our pre-trained model, we need to give it a note sequence to continue and the model will return the following sequence.

We initialize the model as seen in Listing 5 by providing the bundle file `classical_attention_rnn.mag` generated before.

```
# Import dependencies.
from magenta.models.melody_rnn import
    melody_rnn_sequence_generator
from magenta.models.shared import
    sequence_generator_bundle
from note_seq.protobuf import generator_pb2
from note_seq.protobuf import music_pb2

# Initialize the model.
print("Initializing Melody RNN...")
bundle = sequence_generator_bundle.read_bundle_file(
    'classical_attention_rnn.mag')
generator_map = melody_rnn_sequence_generator.
    get_generator_map()
melody_rnn = generator_map['attention_rnn'](
    checkpoint=None, bundle=bundle)
melody_rnn.initialize()
```

Listing 5. Initialize the model

Next, we provide the MelodyRNN model with a MIDI file that we want the model to continue. This MIDI file is converted to NoteSequence, which is Magenta's protocol buffer containing different pitches, instruments and strike velocities, much like MIDI. With Melody RNN, we are able to configure the number of steps the new sequence will be, as well as the "temperature" of the result. The higher the temperature, the more random (and less like the input) the sequence will be. The code for continuing the melody with MelodyRNN is seen in Listing 6. The code launches the helper function that prompts to upload a MIDI file to continue the sequence. Next, the visualized sequences are shown with the option to play each of the initial and predicted sequences.

At the end of the execution, the newly generated melody is saved on the machine as a MIDI file.

```
# Model options.
num_steps = 256 # change this for shorter or longer
    sequences
temperature = 2.0 # the higher the temperature the
    more random the sequence.

# Helper functions. Upload a MIDI file and convert
    to NoteSequence.
def upload_midi():
  data = list(files.upload().values())
  if len(data) > 1:
    print('Multiple files uploaded; using only one.'
        )
  return note_seq.midi_to_note_sequence(data[0])

# Visualize the uploaded sequence.
input_sequence = upload_midi()
note_seq.plot_sequence(input_sequence)
note_seq.play_sequence(input_sequence, synth=
    note_seq.fluidsynth)

# Set the start time to begin on the next step after
     the last note ends.
last_end_time = (max(n.end_time for n in
    input_sequence.notes)
                 if input_sequence.notes else 0)
qpm = input_sequence.tempos[0].qpm
seconds_per_step = 60.0 / qpm / melody_rnn.
    steps_per_quarter
total_seconds = num_steps * seconds_per_step

generator_options = generator_pb2.GeneratorOptions()
generator_options.args['temperature'].float_value =
    temperature
generate_section = generator_options.
    generate_sections.add(
  start_time=last_end_time + seconds_per_step,
  end_time=total_seconds)

# Ask the model to continue the sequence.
sequence = melody_rnn.generate(input_sequence,
    generator_options)

note_seq.plot_sequence(sequence)
note_seq.play_sequence(sequence, synth=note_seq.
    fluidsynth)

# This creates a file called `sequence.mid`
note_seq.sequence_proto_to_midi_file(sequence, '
    sequence.mid')

# This is a colab utility method to download that
    file.
files.download('sequence.mid')
```

Listing 6. Continuing a sequence

One of the difficulties of music generation is the evaluation of music quality, as it is largely subjective and can vary widely for a specific piece of music. To evaluate the generated melodies, we have analyzed the music qualitatively in terms of the following three metrics: how pleasing, how real, and how interesting it sounds.

As an example, one of the initial note sequences along with the generated note sequence is presented in Figure 8 and Figure 9.

We release the link to the Google Colab source code and the model bundle file `classical_attention_rnn.mag` at the `https://drive.google.com/drive/folders/1i9HDq6soY8EiLteu_l-lohGnGSfUzl5h`. Note that as the

code contains a number of Google Colab helper functions, it won't be completely runnable outside of the Google Colab environment.

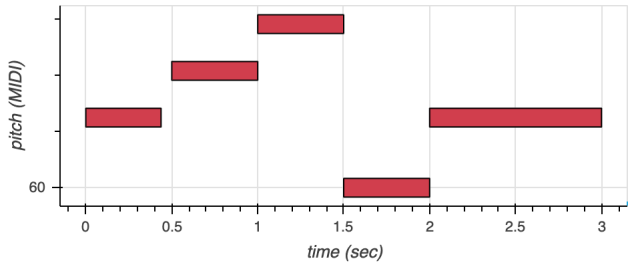

Fig. 8. Initial note sequence

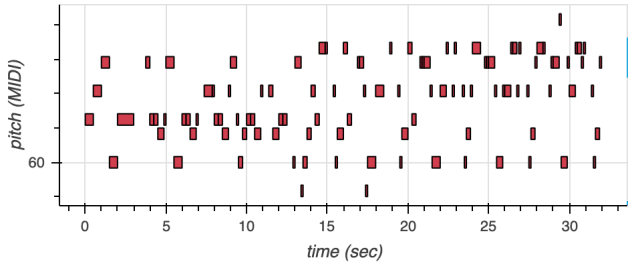

Fig. 9. Predicted note sequence

7.4. Assessment

In this section, we have trained the MelodyRNN model on a large-scale dataset of classical music. Furthermore, we have demonstrated the process of continuing note sequences with the model.

As future work, we can acquire more training MIDI data to enable the model to learn a better note probability distribution to create more interesting and meaningful melodies. Furthermore, to improve the quality of the model, we can conduct the experiments with real participants and ask them to rate the generated melodies. Also, we can extend the model to create multi-track music (such as drums or chords). To increase the speed of the model training, we can migrate the process of model training to a cloud platform's high-performance computing instances with GPU support.

8. A Technical Deliverable 2: MelodyMe - Neural Network that Responds to Melodies

8.1. Requirements

The goal of this deliverable is to design and prototype the application that is able to use the pre-trained MelodyRNN model to continue the musical sequences. In addition, a technical architecture of an application (back-end and front-end) should be proposed. The application is expected to be presented at the AI & Art pavilion in Esch 2022 capital for European Culture. Note that it is not required to have a completely functional application at this stage.

8.2. Design

The deliverable is created by researching the existing applications previously created with Magenta library [16]. The application front-end was prototyped in a web-based design tool called Figma [17]. The architecture of the application is chosen as a consequence of the research process.

8.3. Production

In this section, we introduce an interactive web application called MelodyMe, which is built upon the pre-trained MelodyRNN model. The application shows how the neural network responds by letting one play a melody and seeing how the neural network continues it. The aim of the application is to demonstrate that machine learning can be used to enable and enhance the creative process. This application also reinforces the idea of the explainability of artificial intelligence, as the way the model arrives at predicted melodies is not completely transparent and understandable.

The idea of the experiment is to make music by playing some notes and seeing how the neural network responds to the melody. The application is composed of two parts, the front-end and the back-end. The front-end takes the player's input and plays generated melodies. The back-end takes the MIDI input and continues it using the Magenta's MelodyRNN model that was pre-trained on the large-scale classical music dataset. The player's input is sent to the Flask server, a web framework written in Python [18]. The predicted MIDI is then returned back to the front-end. The complete overview of the architecture is seen in Figure 10.

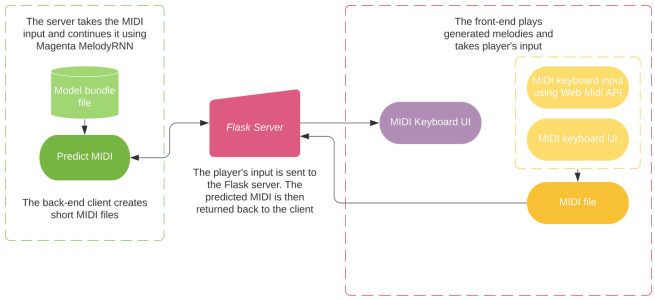

Fig. 10. Application architecture

We propose this architecture as it does not require us to modify our written Python code for generating melodies with MelodyRNN. For instance, it was also possible to use Magenta.js, which is a JavaScript library for generating music with Magenta models [19]. This would require us to modify the initial code and find a way to host our pre-trained model at a server as Magenta.js only provides the models pre-trained by Google.

Next, we prototyped a simple interface for the application. First, we created a welcome screen that quickly introduces an application, as seen in Figure 11. Second, we implemented an actual application screen, where a player can interact with the piano, play some notes and see and hear the newly generated melodies as seen in Figure 12. Here, the green notes are played by a real player and the red ones are generated by the model. As soon as the notes are played, they are scrolled up in a sequence.

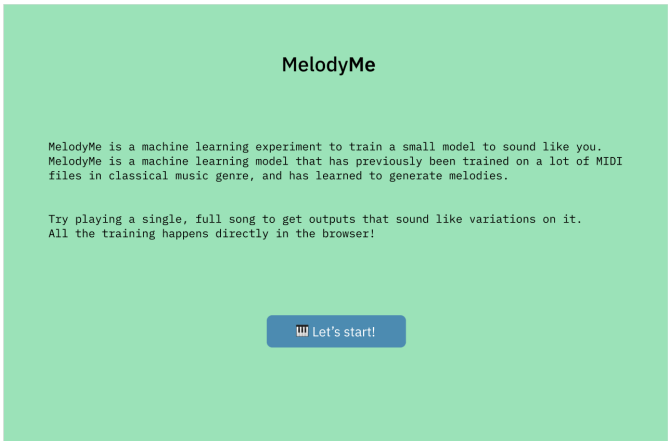

Fig. 11. MelodyMe. Welcome page

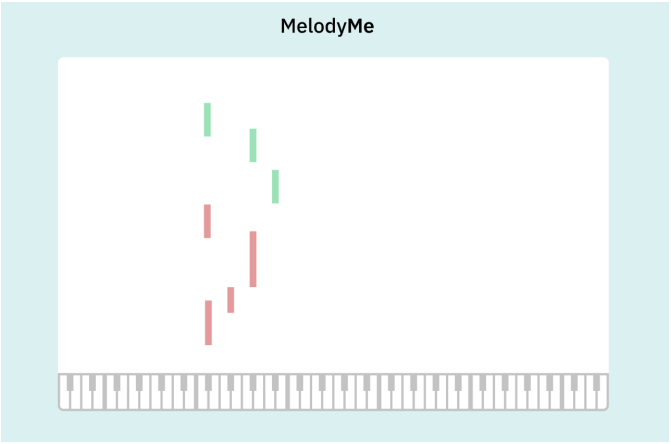

Fig. 12. MelodyMe. Application page

The diagram 13 in the Appendix section shows the link between the MelodyRNN and MelodyMe application. After training MelodyRNN model, we use the pre-trained model to predict the melodies in the MelodyMe application.

With MelodyMe, we hope to raise awareness of the black-box nature of deep learning and neural networks. It is intentionally designed as a black box, which makes it difficult to explain why a model arrives at a specific prediction. In the future, one of the goals of the MelodyMe is to provide an explanation of how a deep learning system has learned from the corpus of data and why it ends up generating a given musical content.

8.4. Assessment

In this section, we have introduced a simple design of an application that is built upon the pre-trained MelodyRNN model. Even though the development of the application still has to be completed to be fully functional, it demonstrates an important role of the need of explainability and transparency in the area of artificial intelligence.

As future work, it is important to develop the application fully as the application is expected to be presented at the AI & Art pavilion. Next, one of the goals is to train multiple MelodyRNN models with different genres of music, extend an application to multiple views, and see how the predicted melodies differ among the models. Furthermore, with the improvement of the MelodyRNN model to support multiple tracks, an application can also be extended to support different instruments (such as drums). Another idea is to explain and show how the model came to particular predictions to allow players to better understand and interpret predictions made by the model. Finally, besides the ability to click the keyboard in the application, the option to use computer keys or even plug in a MIDI keyboard can be added.

9. Conclusion

In this project, we introduced the basic elements of RNNs and its special kind, LSTM. We also reviewed the state-of-the-art related work in the area of music generation with neural networks. Then, we used Magenta's MelodyRNN to generate melodies in a classical music genre. To implement this, we first train the model on a large-scale dataset of classical music and then use that pre-trained model to generated new melodies.

We also introduced an interactive web application called MelodyMe, which is built upon the pre-trained MelodyRNN model. The application shows how the neural network responds by letting one play a melody and seeing how the neural network continues it.

References

[1] CS 230 - Recurrent Neural Networks Cheatsheet https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

[2] GANSynth: Making music with GANs https://magenta.tensorflow.org/gansynth

[3] Mangal, S., Modak, R., & Joshi, P. (2019) LSTM Based Music Generation System. https://arxiv.org/abs/1908.01080

[4] Wu, J., Hu, C., Wang, Y., Hu, X., & Zhu, J. (2019) A hierarchical recurrent neural network for symbolic melody generation. https://arxiv.org/abs/1712.05274

[5] Generating Long-Term Structure in Songs and Stories https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn

[6] Yang, L. C., Chou, S. Y., & Yang, Y. H. (2017) MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. https://arxiv.org/abs/1703.10847

[7] Dong, H. W., Hsiao, W. Y., Yang, L. C., & Yang, Y. H. (2017) Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. https://arxiv.org/abs/1709.06298

[8] MusicVAE: Creating a palette for musical scores with machine learning. https://magenta.tensorflow.org/music-vae

[9] Roberts, A., Engel, J., Raffel, C., Hawthorne, C., & Eck, D. (2018) A hierarchical latent vector model for learning long-term structure in music. https://arxiv.org/abs/1803.05428

[10] Magenta https://magenta.tensorflow.org/

[11] Melody RNN https://github.com/magenta/magenta/tree/master/ magenta/models/melody_rnn

[12] Hello Magenta https://colab.research.google.com/notebooks/magenta/ hello_magenta/hello_magenta.ipynb

[13] MuseNet https://openai.com/blog/musenet/

[14] Kong, Q., Li, B., Chen, J., & Wang, Y. (2020) GiantMIDI-Piano: A large-scale MIDI dataset for classical piano music. https://arxiv.org/abs/2010.07061

[15] GiantMIDI-Piano https://github.com/bytedance/GiantMIDI-Piano

[16] Magenta Demos https://magenta.tensorflow.org/demos

[17] Figma https://www.figma.com/

[18] Flask https://flask.palletsprojects.com/en/1.1.x/

[19] Magenta.js https://magenta.tensorflow.org/js-announce

[20] Understanding LSTM Networks http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[21] Aiva https://www.aiva.ai/

10. Appendix

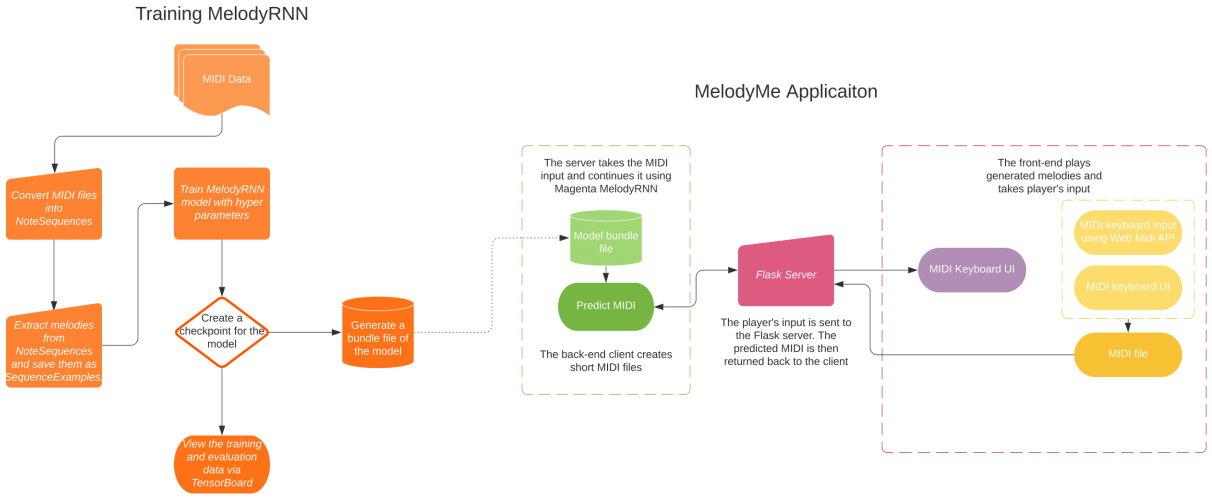

Fig. 13. MelodyRNN and MelodyMe architecture