AI for Human Music Generation

Rodrigo Ferreira
rodrigolf080@gmail.com
Middlesex University

Abstract

This project investigates the application of AI technology for creating more expressive and human-like electronic music. Specifically, the project utilizes both Neural Networks and Genetic Algorithms to generate music that incorporates imperfections in pitch, tempo, and mood, resulting in more dynamic and adaptable compositions that can better align with the listener's and the musician’s individual personalities. By exploring these techniques, this project aims to push the boundaries of what is possible in AI-generated music and contribute to the development of more emotionally engaging musical experiences.

Keywords: Artificial Inteligence, Genetic Algorithms, Neural Nets, Music Theory

Table of Contents

- Cover → p.1
- Abstract → p.2
- Table of Contents → p.3
- Background →p.4
- Literature Review → p.6
- Requirement Specifications → p.10
- Methodology →p.11
- Results and Evaluation → p.20
- Limitations and Further Work→p.21
- Conclusion →p.22
- Resources and Acknowledgements → p.23

Background

Problem definition

The main difference between human-generated music and electronic music generated by a machine is the presence of imperfections in the former. When a human plays an instrument or sings, there are natural variations in pitch, timing, and expression that create a unique and personal quality to the performance. These imperfections make the music sound more natural, emotional, and alive. On the other hand, electronic music generated by a machine is often perfect in terms of pitch and timing, lacking the variation and nuances of human expression. While this precision can make the music sound technically impressive, it can also make it sound sterile and robotic to some listeners.

Furthermore, humans are capable of improvisation and adapting their performances to the mood and environment, creating music that is spontaneous and dynamic. Machines, on the other hand, are programmed to follow specific rules and patterns, making it challenging to create music that is truly unpredictable and original. Although recent advances in machine learning and artificial intelligence have enabled machines to generate music that sounds more natural and human-like, they still lack the creativity, intuition, and emotional depth that come naturally to humans.

Aim

The goal for this project was to build test the usage of AI for music generation. For this, Neural Nets (NN) and Genetic Algorithms (GA) were used and the difference in methodology and results are discussed further into the paper.

The real goal, can AI generate midi sounds for you. Essentially if you can generate the sounds, simply change the soundfont of some intruments, mix it together with a drum and/or beat and loop for AI assisted DJ experience, keeping the human in the loop the whole time.

The problem at hand is the difference between electronically generated music and human generated music. One can imagine this contrast in a simple EDM sound where notes and chords are accuractly played where in contrast a human playing a guitar will not generate perfect notes and chords most of the time - this is a variation in pitch. A human player will unlikely keep perfect timing during performance where a machine will – this is a variation in tempo. The fundamental difference between human generator music e.g. guitarist playing an electric guitar and computer generated electronic music is the imperfections the first has and the latter lacks.

Trying to make progress towards electronic music generation the usage of AI for generating notes and chords in a more human way by trying to generate small flaws in pitch and tempo as randomly as possible.

The first approach was to use Human Feedback Genetic Algorithms (HFGA) were used to generate notes and chords randomly from an input set and allow a human to review and rate the generated sounds. By using human feedback eventhough more labour intensive it is expected to be more accuratly adaptable to the musician’s personality.

The second approach was to Reinforcement Neural Networks (RNN) trained on complex EDM music from professional producers who can make an electronic music sound human during production. After some testing Classical music was added to this mix brings together good music composition from Classical and some noise from complex EDM sounds, and contrasted with the same architecture trained on accoustic guitar studio recordings for a set of chords. This approach is mainly impacted by the

dataset used to train the model, making those decisions from the musician the ones affecting the output the most. For this we use a softmax activation function plus a Cross-Entropy loss function, along with RMSprop optimize to to increase our learning rate. LSTM in music was pioneered by the work of Lewis and Todd [https://papers.cnl.salk.edu/PDFs/Proceedings%20of%20the%201988%20Connectionist%20Models%20Summer%20School%201989-3370.pdf] [https://ieeexplore.ieee.org/document/23933] in the 80s to the work of Eck and Schmidhuber [https://ieeexplore.ieee.org/document/1030094] where we have traced a long way. Their work first utilized LSTMs in music generation. In [https://www.cs.hmc.edu/~ddjohnson/tied-parallel/johnson2017tiedparallel.pdf], authors stated that LSTMs are able to capture the medium-scale melodic structure in music pretty well. When trained on sufficient audio data, they are also able to generate novel melodies. Due to the recent success in speech synthesis models, particularly with WaveNet [https://www.academia.edu/65267864/Hyper_parameter_optimization_with_REINFORCE_and_Masked_Attention_Auto_regressive_Density_Estimators] raw audio files are increasingly used in music generation. We will be using a bidireitional LSTM network with softmax loss ending for more accurate not tracking and faster training.

In music it's very difficult to design a reward function as generally the rule is, if it sounds good it's good, making it a human heuristic that is different between individuals. Therefore the goal here was to try and generate inperfect but accurate sounds and provide the methods and results for tool development and further work in this area as groups with larger resource allocation such as Google work on developing their own Neural Networks for music generation.

By using midi files we allow for an extra level of style editing in a DAW software for changes in pitch and tempo and further customisation.

Ethical Considerations

All the code including source code and weights files, this excludes any MIDI files either from input or output for the system, are under GPL2 (GNU General Public License, version 2) License. Please refer to [https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html] for legal details.

Literature Review

The purpose of this literature review is to give some context on pitch, tempo and mood in music theory and production, midi file format, Neural Nets and Genetic Algorithms used in the context of this project.

Music Theory and production

Pitch, tempo, and mood are three fundamental elements of music that play crucial roles in shaping the overall emotional impact of a piece of music. In music theory and production, pitch refers to the perceived highness or lowness of a sound, which is determined by the frequency of its vibrations. Different pitches can evoke different emotions, with higher pitches often associated with feelings of excitement or joy, and lower pitches associated with sadness or melancholy. Tempo, on the other hand, refers to the speed or pace of a piece of music, and can also have a significant impact on mood. Fast tempos are often used in music to convey feelings of excitement or energy, while slower tempos can create a sense of relaxation or contemplation. Finally, mood refers to the emotional quality or atmosphere of a piece of music, which can be influenced by a variety of factors, including pitch, tempo, instrumentation, and harmonic structure. By understanding how pitch, tempo, and mood work together, music producers can create compositions that are more effective at evoking specific emotional responses in their listeners.

Genetic Algorithms with Human Feedback

Genetic algorithms (GAs) are a class of optimization algorithms that use principles inspired by biological evolution, such as selection, crossover, and mutation, to search for optimal solutions to a problem. While GAs are powerful and flexible optimization techniques, they are not always efficient in finding the best solutions, especially in complex or multi-objective optimization problems. One way to improve the performance of GAs is by incorporating human feedback into the optimization process. According to [https://www.researchgate.net/publication/222514150_Design_and_optimization_using_genetic_algorithms_of_intensified_distillation_systems_for_a_class_of_quaternary_mixtures], human feedback can be used to guide the GA search process towards desirable solutions, reduce the search space, and improve the optimization speed. Human feedback can take many forms, such as subjective evaluations, expert knowledge, or user preferences. By using human feedback to adjust the parameters of the GA, the optimization process can be guided towards solutions that are more desirable to the user or are more likely to meet specific requirements. This can lead to better solutions being found more efficiently than with traditional GA methods alone.

Human feedback as a reward function in the musical context seems like the most accurate approach as good music is subjective between individuals.

MIDI

The Musical Instrument Digital Interface format (MIDI or .mid) is used to store message rules which contain note pitches, their volume, speed, start and end timestamp, phrases and so forth. It doesn't store songs like sound formats, however it stores data that is equipped for producing future melodic notes. These rules can be deciphered by a sound card which uses a wavetable (table of recorded sound waves) to make an understanding of the MIDI messages into genuine stable information. It very well may be deciphered by midi player studio, for example, Fruity Loops (FL) Studio or standard sequencers like Synthesia. Musical notation software like MuseScore or Finale can make a translation of midi into editable sheet music; this empowers customers to make musicin regular music documentation on their PCs and they may listen to it by MIDI players.

RNN with softmax loss and RMSprop GDO

Recurrent Neural Networks (RNNs) are a type of neural network that are designed to process sequential data by introducing feedback connections that allow information to be passed between time steps. RNNs are able to maintain a memory of past inputs by using the output from the previous time step as input to the current time step, allowing them to capture long-term dependencies in the data. However, traditional RNNs suffer from the vanishing and exploding gradient problems, which can make it difficult for them to learn long-term dependencies in the data. This has led to the development of specialized types of RNNs, such as Long Short-Term Memory (LSTM) networks, that are designed to overcome these issues.

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is designed to handle the vanishing and exploding gradient problems in traditional RNNs. According to Hochreiter and Schmidhuber [https://www.bioinf.jku.at/publications/older/2604.pdf] , LSTMs can learn to bridge time intervals of unknown size between relevant events. LSTMs are particularly useful for processing sequential data where there are long-term dependencies between the inputs, as they are able to selectively forget or remember information over time. This is achieved through the use of specialized memory cells that are able to maintain information over long time periods. According to Neurwirth [https://arxiv.org/abs/2203.12105], LSTM network structures have proven to be very useful for making predictions for the next output in a series. We sought to demonstrate an approach of music generation using Recurrent Neural Networks (RNN). More specifically, a Long Short-Term Memory (LSTM) neural network. Generating music is a notoriously complicated task, whether handmade or generated, as there are a myriad of components involved. Taking this into account, we provide a brief synopsis of the intuition, theory, and application of LSTMs in music generation, develop and present the network we found to best achieve this goal, identify and address issues and challenges faced, and include potential future improvements for our network.

Softmax is a widely used activation function in machine learning and deep learning, especially in classification tasks where the goal is to predict the probability distribution of an input belonging to each of the possible classes. According to DeepAI's Research Team [https://deepai.org/machine-learning-glossary-and-terms/softmax-layerh , the softmax function transforms the output of a neural network into a probability distribution over the possible classes. Softmax takes a vector of arbitrary real values as input and maps it to a probability distribution over the possible classes.
Softmax is known for its ability to handle multi-class problems, as it allows the output values of the function to be interpreted as probabilities of each class. The function outputs values that are always

non-negative and sum to 1, which makes it suitable for probabilistic interpretation of the predicted outcomes. Furthermore, the differentiability of Softmax makes it suitable for use in backpropagation algorithms for training neural networks. Backpropagation involves computing the gradient of the loss function with respect to the model parameters, and Softmax provides a smooth and differentiable mapping that facilitates the gradient computation.

Categorical cross entropy (CCE) is a widely used loss function for classification tasks in deep learning, and it has been studied extensively in the literature. According to Garcia-Valencia [https://arxiv.org/abs/2006.02217] CCE is defined as the average of the logarithm of the predicted probability distribution over the true probability distribution for each class label. This means that CCE measures the difference between the predicted probabilities and the true probabilities of each class. During training, the model's parameters are adjusted to minimize this difference.
The effectiveness of CCE in multi-class classification problems has been noted by Kingma and Ba [https://arxiv.org/abs/1412.6980] (for ADAM instead of RMSprop), who pointed out that it allows for efficient computation of the gradient during backpropagation. CCE is particularly useful when there are multiple classes to classify, as it allows for a simpler and more efficient computation of the loss function as compared to other loss functions. Moreover, the use of CCE in conjunction with the softmax activation function has been highlighted by Goodfellow [https://www.deeplearningbook.org/] . The softmax function ensures that the predicted probabilities are normalized and sum to one. This is an important property, as it allows the predicted probabilities to be interpreted as class probabilities. . For instance, CCE is used in various neural network architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

In conclusion softmax loss is adequate for the task at hand, which has also been proven to be very effective for MIDI datasets according to [https://arxiv.org/abs/1810.12247].

RMSprop (Root Mean Square Propagation) is an optimization algorithm for stochastic gradient descent (SGD) that seeks to overcome the diminishing learning rate problem by adapting the learning rate for each weight based on the average of the magnitudes of recent gradients for that weight. It was introduced by Geoff Hinton in a lecture in 2012 [https://www.scirp.org/(S(czeh2tfqyw2orz553k1w0r45))/reference/ReferencesPapers.aspx?ReferenceID=1911091] and has been widely used in deep learning.. RMSprop is particularly effective for dealing with non-stationary, noisy or sparse gradients, which can cause standard SGD to converge slowly or even to diverge. It does so by using a moving average of the squared gradient values to normalize the learning rate. The algorithm is computationally efficient and has been shown to perform well on a variety of deep learning tasks, including image recognition, natural language processing and speech recognition. As stated in a paper by Tijmen Tieleman and Geoffrey Hinton [https://www.scirp.org/(S(czeh2tfqyw2orz553k1w0r45))/reference/ReferencesPapers.aspx?ReferenceID=1911091] , "RMSprop is a generalization of Adagrad that deals with its radically diminishing learning rates. It is motivated by the problem of Adagrad that the learning rates of some parameters are not updated for long periods of time, which can slow down the learning significantly. RMSprop solves this by using a moving average of the squared gradient, which scales the learning rate differently for each parameter and thus adapts to the changing conditions of the optimization problem.

As a result, RMSprop can converge faster and better than Adagrad on some problems." Another paper by Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov [https://jmlr.org/papers/v15/srivastava14a.html] shows that RMSprop is effective in improving the convergence rate and accuracy of deep neural networks. They state that RMSprop can efficiently optimize large-scale neural networks and is robust to the choice of hyper-parameters. The proposed algorithm converges faster and attains higher accuracies than the commonly used stochastic gradient descent with momentum (SGD+M) algorithm on several benchmark datasets, including CIFAR-10, CIFAR-100 and ImageNet.

Conclusion

Attention in LSTM is a later advancement that really takes care of our center issue. This enables to take care of specific segments of the contribution at some random moment and utilize those segments to help produce portions of the yield as opposed to simply the last output of the LSTM layer. Bidirectional LSTMs are an expansion of conventional LSTMs that can improve model execution on arrangement order issues. In issues where all timesteps of the information arrangement are accessible, Bidirectional LSTMs train two rather than one LSTM on the information grouping To handle complex structures such as pitch, a cross-entropy loss function and softmax activation function can be utilized to minimize loss. Additionally, an RMSprop gradient descent optimizer can be used with an increased learning rate to improve training efficiency by adapting to the complexity of the musical composition more accurately. Incorporating a personalized touch to music generation can be achieved model, which can adapt to the musician's unique style more accurately. Moreover, this model can generate related samples closely resembling the input while still allowing for creative randomness during the generation process.

Requirement Specifications

For approach one he software should be easy to bundle with dependencies and distribute to personal workstations, as well as train from scratch ou personal records or datasets. Therefore it should be quick to train and demonstrate results while still allowing further training for improved results regardeless of diminishing returns over time. For this Tensorflow was used and alongside with an NVIDEA GPU, a company currently leading AI training and inference modules, being only rivaled in training by Google and Qualcom's own TPUs not for commercial sale and Graphcore's inference SoCs also not for commercial sale. NVIDEA, Google and Graphcore provide us with cloud access to their chip infrastructure but for this project decentralised AI access was taken in consideration and thus NVIDEA GPUs were the selected hardware for training as the author doesn't have access to Apple hardware. Tensorflow was choosen due to it's easier and quicker use for developing a neural network from mathematically proven models on NVIDEA GPUs, just not on Apple's hardware.

Inference and training depend directly on the size of the dataset. Inference time should not matter as much if the results generated are consistently positive. Training time should be kept to a minimum, as we aim of training this in a personal workstation we need a balance of time to train and usefulness of the results generated.

The purpose for this requirements in reality is so that anyone can download the code and train the model themselves on their own musical style.

For approach two the genetic algorithm runs simply on CPU, it is hardcoded with notes, chords and scales, as scales are known to be sequences that sound good to the human ear and requires human feedback to perfect mutation into a customised and well sounding sample.

Methodology - Neural Network

For the project multiple datasets two datasets were user, a combination of EDM and Classical music, and multiple compositions of minor chords. The results should be a random looking set of notes and chords that sound good together, atleast in subsets, we will be generating 500 output notes and chords (so a little over 500 notes) and there should be usable subsets of the output that can be sliced, edited with a soundfont instrument and integrated into a sound with other samples, a beat and/or drums and loops.

Dataset description
Dataset1, a set of midi files from www.crapto.com of EDM and Classical music genres, providing lists of pitch and tempo for training. Dataset2 a set of midi files from accoustic recording of minor chords with different emphasis.

Dataset 1:
'Alan Walker - Alone (midi by Carlo Prato) (www.cprato.com).mid'
'Alan Walker - Faded (Original Mix) (midi by Carlo Prato) (www.cprato.com).mid'
'Alan Walker - Sing Me To Sleep (midi by Carlo Prato) (www.cprato.com).mid'
'Calvin Harris - Blame (midi by Carlo Prato) (www.cprato.com).mid'
'Calvin Harris - My Way (midi by Carlo Prato) (www.cprato.com).mid'
'chopin_nocturne_9_2 copy.mid'
'chopin_Waltz_Opus_69_No_2_bmin copy.mid'
'deadmau5 - 2448 (midi by Carlo Prato) (www.cprato.com).mid'
'deadmau5 feat. Grabbitz - Let Go (midi by Carlo Prato) (www.cprato.com).mid'
'deadmau5 - Hyperlandia (midi by Carlo Prato) (www.cprato.com).mid'
'deadmau5 - No Problem (midi by Carlo Prato) (www.cprato.com).mid'
'deadmau5 - So There I Was (midi by Carlo Prato) (www.cprato.com).mid'
'Galantis - Love On Me (midi by Carlo Prato) (www.cprato.com).mid'
'Galantis - No Money (midi by Carlo Prato) (www.cprato.com).mid'
'Galimatias - Noelle'\"s Eloquence (midi by Carlo Prato) (www.cprato.com).mid'
'J_S_Bach_-_Cello_Suite_1007_complete copy.mid'
'Marshmello - Alone (Original Mix) (midi by Carlo Prato) (www.cprato.com).mid'
'Marshmello - Ritual (Original Mix) (midi by Carlo Prato) (www.cprato.com).mid'
'Marshmello - Summer (midi by Carlo Prato) (www.cprato.com).mid'
'Martin Garrix & Bebe Rexha - In The Name Of Love (midi by Carlo Prato) (www.cprato.com).mid'
'Martin Garrix & Dua Lipa - Scared To Be Lonely (midi by Carlo Prato) (www.cprato.com).mid'
'Martin Garrix ft. The Federal Empire - Hold On & Believe (Original Mix) (midi by Carlo Prato) (www.cprato.com).mid'
'moonlight_sonata copy.mid'
'mozart_sonata_9_2ndmvt_k311_PNO copy.mid'
'Nero - The Thrill (Porter Robinson Remix) (midi by Carlo Prato) (www.cprato.com).mid'
'pachelbel_canon copy.mid'
'Pegboard Nerds - Emoji (midi by Carlo Prato) (www.cprato.com).mid'
'Pegboard Nerds - Melodymania (midi by Carlo Prato) (www.cprato.com).mid'

'Pendulum - Salt In The Wounds (midi by Carlo Prato) (www.cprato.com).mid'
'Pendulum - Witchcraft (midi by Carlo Prato) (www.cprato.com).mid'
'Porter Robinson & Madeon - Shelter (Original Mix) (midi by Carlo Prato) (www.cprato.com).mid'
'Porter Robinson - Sad Machine (midi by Carlo Prato) (www.cprato.com).mid'
'schubert_serenade_VLN copy.mid'
'Vanic x K.Flay - Make Me Fade (midi by Carlo Prato) (www.cprato.com).mid'
'vivaldi_autumn copy.mid'
'vivaldi_spring copy.mid'
'Wax Motif & Vindata - Crazy (midi by Carlo Prato) (www.cprato.com).mid'
'Zedd & Hailee Steinfeld Grey - Starving (midi by Carlo Prato) (www.cprato.com).mid'

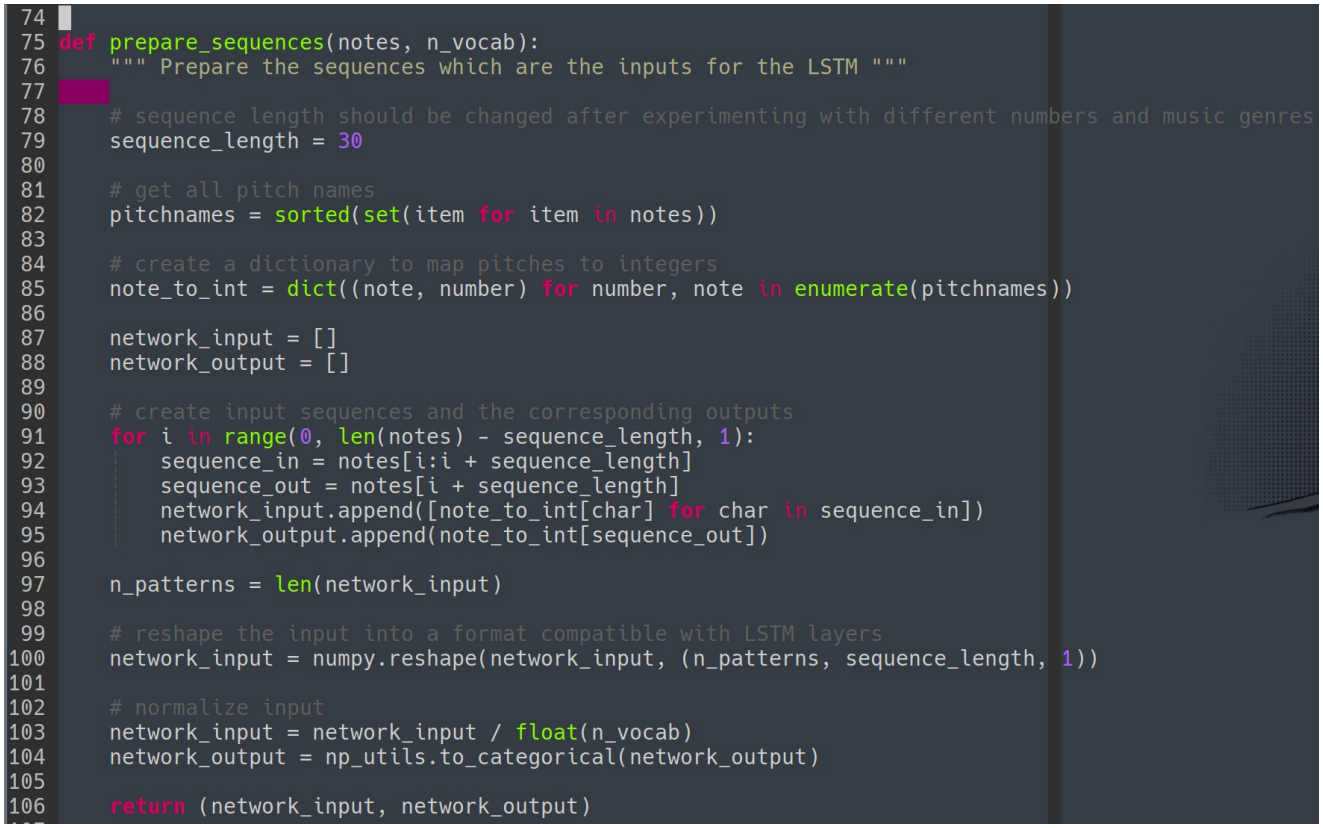
Dataset 2:
'BLA - A Minor 10.mid'
'BLA - A Minor 3.mid'
'BLA - A Minor 8.mid'
'BLA - D# Minor 1.mid'
'BLA - E Minor 3.mid'
'BLA - A Minor 1.mid'
'BLA - A Minor 4.mid'
'BLA - A Minor 9.mid'
'BLA - D Minor 2.mid'
'BLA - F Minor 1.mid'
'BLA - A# Minor 1.mid'
'BLA - A Minor 5.mid'
'BLA - C# Minor 1.mid'
'BLA - D# Minor 2.mid'
'BLA - F Minor 2.mid'
'BLA - A Minor 2.mid'
'BLA - A Minor 6.mid'
'BLA - C# Minor 2.mid'
'BLA - E Minor 1.mid'
'BLA - F Minor 3.mid'
'BLA - A# Minor 2.mid'
'BLA - A Minor 7.mid'
'BLA - D Minor 1.mid'
'BLA - E Minor 2.mid'
'BLA - F Minor 4.mid'

Training and Inference

First we extract all the notes and chords and create a raw dump file for later usage for translation between midi and network.

```
47
48 def get_notes():
49     """ Extracts all notes and chords from midi files in the ./midi_songs
50     directory and creates a file with all notes in string format"""
51     notes = []
52
53     for file in os.listdir(MIDI):
54         midi = converter.parse(f"{MIDI}/{file}")
55         print("Parsing %s" % f"{MIDI}/{file}")
56         notes_to_parse = None
57
58         try:
59             s2 = instrument.partitionByInstrument(midi)
60             notes_to_parse = s2.parts[0].recurse()
61         except:
62             notes_to_parse = midi.flat.notes
63
64         for element in notes_to_parse:
65             if isinstance(element, note.Note):
66                 notes.append(str(element.pitch))
67             elif isinstance(element, chord.Chord):
68                 notes.append('.'.join(str(n) for n in element.normalOrder))
69
70         with open(NOTES, 'wb') as filepath:
71             pickle.dump(notes, filepath)
72
73     return notes
```

This data still has to be preprocessed one more time before being fed into the network into simplier numbers that represent each note.



```
74
75 def prepare_sequences(notes, n_vocab):
76     """ Prepare the sequences which are the inputs for the LSTM """
77
78     # sequence length should be changed after experimenting with different numbers and music genres
79     sequence_length = 30
80
81     # get all pitch names
82     pitchnames = sorted(set(item for item in notes))
83
84     # create a dictionary to map pitches to integers
85     note_to_int = dict((note, number) for number, note in enumerate(pitchnames))
86
87     network_input = []
88     network_output = []
89
90     # create input sequences and the corresponding outputs
91     for i in range(0, len(notes) - sequence_length, 1):
92         sequence_in = notes[i:i + sequence_length]
93         sequence_out = notes[i + sequence_length]
94         network_input.append([note_to_int[char] for char in sequence_in])
95         network_output.append(note_to_int[sequence_out])
96
97     n_patterns = len(network_input)
98
99     # reshape the input into a format compatible with LSTM layers
100    network_input = numpy.reshape(network_input, (n_patterns, sequence_length, 1))
101
102    # normalize input
103    network_input = network_input / float(n_vocab)
104    network_output = np_utils.to_categorical(network_output)
105
106    return (network_input, network_output)
```

Then we can initiate a the LSTM neural net with the data, a softmax loss algorithm (categorical crossentropy with softmax activation) and RMSprop optimiser. The constant values are the best results found from testing both the datasets on 100 epochs.

```
108
109
110 def create_network(network_input, n_vocab):
111     """ Creates the structure of the neural network """
112     model = Sequential()
113     model.add(LSTM(
114         512,
115         input_shape=(network_input.shape[1], network_input.shape[2]),
116         return_sequences=True
117     ))
118     model.add(Dropout(0.3))
119     model.add(LSTM(512, return_sequences=True))
120     model.add(Dropout(0.3))
121     model.add(LSTM(512))
122     model.add(Dense(256))
123     model.add(Dropout(0.3))
124     model.add(Dense(n_vocab))
125     model.add(Activation('softmax'))
126     model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
127
128     return model
129
```

For training using batches of size 64 demonstrated the best results for dataset1 for 100, 256 and 512 epochs. We save the weights everytime there is an improvement in compression in the network and save it as a Hierarchical Data Format (hdf5) file which is loaded by the network during inference.

```
130
131 def train(model, network_input, network_output):
132     """ train the neural network """
133
134     filepath = WEIGHTS+"/weights-{epoch:02d}-{loss:.4f}-bigger.hdf5"
135     checkpoint = ModelCheckpoint(
136         filepath,
137         monitor='loss',
138         verbose=0,
139         save_best_only=True,
140         mode='min'
141     )
142     callbacks_list = [checkpoint]
143
144     # experiment with different epoch sizes and batch sizes
145     model.fit(network_input, network_output, epochs=1000, batch_size=64, callbacks=callbacks_list)
146
```

```
94
95     # Load the weights to each node
96     model.load_weights(WEIGHTS+"/weights-prediction.hdf5")
97
```

The output is generated using the loaded weights for n amount of chords and or notes, preprocessed again from neural net compressed data to notes and chords into a MIDI file.

```
100
101 def generate_notes(model, network_input, pitchnames, n_vocab):
102     """ Generate notes from the neural network based on a sequence of notes """
103
104     # pick a random sequence from the input as a starting point for the prediction
105     start = numpy.random.randint(0, len(network_input)-1)
106
107     int_to_note = dict((number, note) for number, note in enumerate(pitchnames))
108
109     pattern = network_input[start]
110     prediction_output = []
111
112     # generate 500 notes
113     for note_index in range(500):
114         prediction_input = numpy.reshape(pattern, (1, len(pattern), 1))
115         prediction_input = prediction_input / float(n_vocab)
116
117         prediction = model.predict(prediction_input, verbose=0)
118
119         index = numpy.argmax(prediction) # numpy array of predictions
120         result = int_to_note[index] # indexing the note with the highest probability
121         prediction_output.append(result) # that note is the prediction output
122
123         pattern.append(index)
124         pattern = pattern[1:len(pattern)]
125
126     return prediction_output
127
```

```
128
129 def create_midi(prediction_output):
130     """ Converts the output from the prediction to notes and create a midi file
131         from the notes """
132
133     offset = 0
134     output_notes = []
135
136     # create note and chord objects based on the values generated by the model
137     for pattern in prediction_output:
138         # pattern is a chord
139         if ('.' in pattern) or pattern.isdigit():
140             notes_in_chord = pattern.split('.')
141             notes = []
142             for current_note in notes_in_chord:
143                 new_note = note.Note(int(current_note))
144                 new_note.storedInstrument = instrument.SnareDrum()
145                 notes.append(new_note)
146             new_chord = chord.Chord(notes)
147             new_chord.offset = offset
148             output_notes.append(new_chord)
149         # pattern is a note
150         else:
151             new_note = note.Note(pattern)
152             new_note.offset = offset
153             new_note.storedInstrument = instrument.SnareDrum()
154             output_notes.append(new_note)
155
156         # increase offset each iteration so that notes do not stack
157         offset += 0.5
158
159     midi_stream = stream.Stream(output_notes)
160
161     midi_stream.write('midi', fp='./output_prediction.mid')
```

Methodology – Genetic Algorithm

This approach is a lot more labour and attention intensive as instead of providing a dataset for the neural net to adapt, the user must have music theory knowledge and programming knowledge to edit the configurations for musical composition, provided through execution flags.

```
@click.command()
@click.option("--num-bars", default=8, prompt='Number of bars:', type=int)
@click.option("--num-notes", default=4, prompt='Notes per bar:', type=int)
@click.option("--num-steps", default=1, prompt='Number of steps:', type=int)
@click.option("--pauses", default=True, prompt='Introduce Pauses?', type=bool)
@click.option("--key", default="C", prompt='Key:', type=click.Choice(KEYS, case_sensitive=False))
@click.option("--scale", default="major", prompt='Scale:', type=click.Choice(SCALES, case_sensitive=False))
@click.option("--root", default=4, prompt='Scale Root:', type=int)
@click.option("--population-size", default=10, prompt='Population size:', type=int)
@click.option("--num-mutations", default=2, prompt='Number of mutations:', type=int)
@click.option("--mutation-probability", default=0.5, prompt='Mutations probability:', type=float)
@click.option("--bpm", default=128, type=int)
def main(num_bars: int, num_notes: int, num_steps: int, pauses: bool, key: str, scale: str, root: int,
         population_size: int, num_mutations: int, mutation_probability: float, bpm: int):
```

```
BITS_PER_NOTE = 4
KEYS = ["C", "C#", "Db", "D", "D#", "Eb", "E", "F", "F#", "Gb", "G", "G#", "Ab", "A", "A#", "Bb", "B"]
SCALES = ["major", "minorM", "dorian", "phrygian", "lydian", "mixolydian", "majorBlues", "minorBlues"]
```

Using user ratings as feedback from 0 to 5 for the selection function we mutate the best rated results and keep on repeating the process until the user is satistied and exits the program, all results are saved in disk but we only keep 3 out of 10 of the best results for the next stage.

```
def fitness(genome: Genome, s: Server, num_bars: int, num_notes: int, num_steps: int,
            pauses: bool, key: str, scale: str, root: int, bpm: int) -> int:
    m = metronome(bpm)

    events = genome_to_events(genome, num_bars, num_notes, num_steps, pauses, key, scale, root, bpm)
    for e in events:
        e.play()
    s.start()

    rating = input("Rating (0-5)")

    for e in events:
        e.stop()
    s.stop()
    time.sleep(1)

    try:
        rating = int(rating)
    except ValueError:
        rating = 0

    return rating
```

```
 83
 84 def run_evolution(
 85         populate_func: PopulateFunc,
 86         fitness_func: FitnessFunc,
 87         fitness_limit: int,
 88         selection_func: SelectionFunc = selection_pair,
 89         crossover_func: CrossoverFunc = single_point_crossover,
 90         mutation_func: MutationFunc = mutation,
 91         generation_limit: int = 100,
 92         printer: Optional[PrinterFunc] = None) \
 93         -> Tuple[Population, int]:
 94     population = populate_func()
 95
 96     i = 0
 97     for i in range(generation_limit):
 98         population = sorted(population, key=lambda genome: fitness_func(genome), reverse=True)
 99
100         if printer is not None:
101             printer(population, i, fitness_func)
102
103         if fitness_func(population[0]) >= fitness_limit:
104             break
105
106         next_generation = population[0:2]
107
108         for j in range(int(len(population) / 2) - 1):
109             parents = selection_func(population, fitness_func)
110             offspring_a, offspring_b = crossover_func(parents[0], parents[1])
111             offspring_a = mutation_func(offspring_a)
112             offspring_b = mutation_func(offspring_b)
113             next_generation += [offspring_a, offspring_b]
114
115         population = next_generation
116
117     return population, i
NORMAL  ⎇master  src/genetic.py
```

Results and Evaluation

For the first approach 100 epochs we tested different batch sizes of 32, 64 and 128 samples resulting in the conclusion that 64 is the best size. The same was performed for dropout values of 0.1, 0.3, 0.5 and 0.7 resulting in a best performance with 0.3 for the best learning outcome using dataset1.

Using the same dataset1 it was tested running for 100 epochs, 256 and 512 epochs and 1000 epochs. There is a huge difference between 100 and 512 epochs and a slight improvement from 512 to 1000 demonstrating dimishing returns.

Running 1000 epochs of training we were able to generate 500 notes and chords that sound good and not out of place for about 99% of the sample where for 100 epochs the generated output almost sounds like a random set notes and chords put together for dataset1. For dataset2 the conjuntion of chords does not generate a nice sounding sample but we were able to conclude that the model does not deviate from the training input, as in pitch in results in similar pitch out as thanks to the softmax loss.

Using dataset1 each round on an NVIDIA T1000 takes about 3 to 4 minutes to train and less than 2 minutes for inference, with positive and usable results from 1000 rounds it is sustainable for running on a personal workstation. For dataset2 it takes about 20s per epoch and the same time for inference.

In conclusion we were able to reproduce a system for generation of music samples in MIDI format that is easy and practical to edit and append to produce a final sound that is sustainable to run on a personal workstation thus maintaining the goal of decentralisation.

For the second approach we were able to generate good sounding MIDI samples after two iterations of ten sized populations from human feedback, this process can be extended as much as needed to generate multiple good sounding samples.

In conclusion this process is faster and generates better results short term and on slower hardware with much simpler software but lacks the automation processed provided from the neural net after enough training cycles which is a higher initial investement in time and hardware along with the need for a custom training set that is not necessary for the second approach. Without enough human feedback the Genetic approach struggles to generate complex samples, runs quickly on CPU but requires human labour and attention while the Neural Network approach requires the human to handpick a favourable dataset and train the model for over 512 epochs before being ble to generate any samples that sound like music. Both processes proved fitted to solve the problem with different pros and cons in cost of time, hardware and attention.

Limitations and further work

For approach one as demonstrated from running 100 to 1000 epochs the model shows diminishing returns therefore it could not be worth it to run for longer than 1000 epochs considering the improvement in results but more tests would need to be performed in order to conclude the optimal training values for different user's systems and the viability of the model for distribution. Neural Networks as a blackbox method seem to be unfit for full cycle music production as they lack the sense of mood and dynamic adaptability to enviroment that only a human can provide. As music is a form of art and expression it seems to be the wrong application for a model like this. Nontheless the application of Neural Networks for sound generation outside of the music field are still promising.

For approach two the program is fit for distribution and usage to generate sample sounds at the cost of human selection for this sounds, lacking the level of automation of approach one, therefore a selection function incorporated into this program could potentially prove to be more useful that the neural network approach but we were not able to define one that provides better or equal to human feedback results.

Conclusion

Artificial Inteligence is able to generate human music that sounds good but is imperfect by keeping the human in the loop. Either that be with Neural Networks by giving the user the option of providing custom training sets or by using Genetic Algorithms with human feedback and the selection function. Neural networks seem to struggle to capture the flaws in human music being mostly well fitted for sound reproduction rather than creative music creation as this element is brough by the musician and eventough we can customise pitch and tempo for the Neural Network as a blackbox method it has limitations in adapting to mood and environment which is where a human is needed. In the future it doesn't seem like AI is fitted for a form of art and human expression of emotion and story telling. Eventough recent AI models seem to have a good graps of the last one such as LLMs, using this approach in music seems counterproductive appart from increasing music production and composition automation for a human musician.

As we can generate all possible musical patterns using AI, it lacks the fundamental understanding of musical expression that makes it sound real and the sense of connection.

Resources and acknowledgements

Tensorflow community forums and discord channels - [https://www.tensorflow.org/community]

W&B meetup, everyone who taught me on AI software and hardware during the in person London meetup 2023, particularly on softmax loss, batching and inference latency

AI papers dissected - [https://www.youtube.com/@TwoMinutePapers] [https://gombru.github.io/2018/05/23/cross_entropy_loss/] [https://deepai.org/machine-learning-glossary-and-terms/softmax-layer]

AI development in Python and lessons - [https://github.com/geohot]

Genetic algorithms for music theory - https://github.com/kiecodes

Access to academic papers in AI, Cornell University Archive - [https://arxiv.org] [https://arxiv.org/pdf/2002.03854.pdf] [https://arxiv.org/pdf/1812.01060.pdf] [https://arxiv.org/abs/1810.12247] [https://arxiv.org/abs/2203.12105] [https://arxiv.org/pdf/2002.03854.pdf]

Reading material on final year project development and AI genetic algorithms and neural nets - [https://www.mdx.ac.uk] [https://cwa.mdx.ac.uk/cst3170/cst3170.html]

Music theory - https://www.musictheory.net/lessons]

MIDI dataset - [www.cprato.com]

MIDI file format - [http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html]