# A CUSTOMISABLE GENETIC ALGORITHM FOR MELODY GENERATION BASED ON USER PREFERENCE

Kevin Kossipos -
40270313

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc (Hons) Computing

School of Computing

July 30, 2021

# Authorship Declaration

I, Kevin Kossipos, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

*Signed: Kevin Kossipos - 40270313@live.napier.ac.uk*

*Date: 30/07/2021*

*Matriculation no: 40270313*

# General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

**Abstract**

As music and its creation are becoming more accessible, musicians look for new ways to create it, and to be inspired. One of the main ways this is done is through the use of artificial intelligence. More specifically, genetic algorithms and neural networks. Previous research in this field have either tried using objective fitness functions for their genetic algorithms, or some arbitrary ways to let users score the generated pieces of music. This paper proposes a novel, genetic algorithm-neural network hybrid. A 2D grid of MIDI notes is generated based on user preference, where individuals of the population could move around and "collect" notes. A MIDI file is then generated from the best individual, which the user can rate from 1 to 7 (1 being the worst-, and 7 being the best score). This rating, along with the MIDI sequence then gets written to a file, which is used as the dataset to train the neural network. Based on initial experiments, the algorithm was able to improve on the quality of the generated melodies based on user feedback. However, the experimentation was not extensive enough to concretely conclude whether the algorithm can produce perceivably better melodies than its competitors.

# Contents

# List of Tables

# List of Figures

# 1 Introduction

The ever increasing demand for new music and the ease of access to the creation of it for the general population means that the market is becoming much more competitive. While there are a vast number of melody lines that can be written, the over-saturation of new artists can put a bottleneck on the perceived creative freedom for many musicians. Melody generation in specific was chosen, as it is a key element of music, and tends to be the most memorable part of any composition, considering its place in the foreground. One of the ways to achieve this, and to create new, unique, and interesting music is through the use of machine learning.

Machine learning is a subdomain of artificial intelligence. The main idea of machine learning is that a machine learning model could learn and improve itself automatically, through experience. This is done using a dataset, without the need for human intervention. While this is the case with machine learning, it is still inseparable from human input. All data processed by these algorithms need to be fed from human experience. The use of machine learning has been implemented in many different areas from software defect detection, to stock market price predictions. The vast number of applications for machine learning is out of the scope of this paper, and instead the focus will be on the use of machine learning within the area of music-, more specifically, melody generation.

The usage of machine learning in the domain of music composition can be dated as far back as 1991, with the use of a genetic algorithm (a form of machine learning) (Horner & Goldberg, 1991) to transform a musical pattern using thematic bridging. In 1994 a genetic algorithm was used more specifically for music generation according to a target music style(Laine & Kuuskankare, 1994). By 2001, machine learning methods were used which resembles that of more modern approaches such as in (Chen & Miikkulainen, 2001).

However, since purely using neural networks to generate music has become more popular in recent years, a novel model which incorporates both a genetic algorithm and a neural network could prove useful when it comes to furthering the research in this area. One of the biggest challenges when it comes to automatic composition of music, is having a good fitness function, which can properly evaluate the outcome of the algorithm according to the user's needs. Another inherent problem that comes with music generation, or the generation of art in general, is the difference between each individual's taste. Therefore, it could be said that there is no one universally perfect fitness function. However, since there seems to be a consistency of taste among

individuals (North & Hargreaves, 2007), it is likely that fitness functions on a per person basis are a good solution to this problem. This study proposes a combination of algorithms using user input to determine a fitness score which gets fed into a neural network that attempts to score future outputs of the genetic algorithm based on the user's preferences.

The hope of this study is to not only further the field of using artificial intelligence to create music, but also to provide artists and composers with more creative ideas, which are tailored to their own tastes and vision.

## 1.1 Musical Terminology

This section gives a brief introduction to some very basic musical terminology, which will later be discussed and referenced.

At the most basic level, there is pitch. Pitch allows for the judging of sounds to be higher or lower. Sounds moving up in pitch are considered to be getting sharper, while sounds moving down are considered to be getting flatter. This is a subjective experience which can be objectified via frequency (for example, the note A4 is usually quantified as 440Hz). After increasing or decreasing the pitch of a note by a certain amount, playing them simultaneously would produce a psychoacoustic phenomenon where it would seem that only one note is being played at the same time. This distance between two pitches is called an octave.

In modern European music theory, there are 12 semitones in an octave, each of which represents a certain pitch. These pitches are represented by notes, therefore it could be said that there are 12 notes in each octave (this is called twelve-tone equal temperament). When representing specific notes, a number is assigned to the note, for example A4. This is used to distinguish different octaves from each other. A lower number represents lower octaves, while a higher one higher octaves.

The 13th note would just be a repetition of the 1st note, however with its frequency having doubled due to the rise of pitch (for example if A4 is 440Hz, A5 would be 880Hz, and A3 would be 220Hz). Semitones are the shortest distance represented between each note. The 12 notes in the octave are "A", "A♯/B♭", "B", "C", "C♯/D♭", "D", "D♯/E♭", "E", "F", "G♯/A♭", after which the notes repeat again, starting with "A".

Some notes, such as "A♯" can be represented as "B♭", as they are considered to be enharmonically equivalent. For simplicity's sake this representation generally depends on the movement of the piece. For example, if the previous note is a B, and the current note is a semitone below that, it would be called "B♭"; whereas if the previous one was an A, and the current note would be called "A♯".

Melodies are different notes arranged in a time-series, meaning notes are played one after the other. These notes can also vary in terms of duration, or even expression (how loud the note is played). The duration of a note is relative, and is dependant on the beats per minute (bpm) of the piece, as well as the time signature. These note durations are such that their length is always half of the previous, longer duration (whole note, half note, quarter note, eighth note, sixteenth note, etc.). The rhythm of a piece of music is dependent on the arrangement of these note durations.

When selecting a number of notes (most of the time 7) from the 12 notes in an octave, scales can be generated. The most popular scales are the major, and minor scale, however there are a number of other variations (modes). Scales are based on specific patterns, and depending on where the scale starts, the tonic (the note from which the scale pattern is started) represents the tonal centre.

When multiple notes are played at the same time, they form what is called a "chord". Generally, chords can be made up of at least 3 notes, for example, playing the "C", "E", and "G" notes together to make a C major chord.

## 1.2 Genetic Algorithms and Neural Networks

This section describes the function of genetic- and machine learning algorithms to a brief extent.

A genetic algorithms (GA) is a search based algorithm inspired by natural selection. The process involves generating a population of individuals, usually with completely random values, which are stored in each individual's "chromosomes". Depending on the representation, a chromosome can have a number of different values (the value which a chromosome holds is referred to as an allele), such as integers or floating point numbers.

Once these random individuals are generated, they are scored on how well they perform in the given problem. This score is called the fitness function, and it ranks each individual within the population, placing them relative to each other. Depending on the type of problem, the goal could either be to maximise, or to minimise the fitness of individuals. There are also multi-objective GAs, which aim to optimise for multiple values.

Once every individual is generated and ranked, the iterative process of selection, reproduction, mutation, and replacement begins. This cycle can run until there is human intervention, a certain fitness function is met, or a specific number of generations is reached.

Parents are selected to produce offspring, which takes (usually) half of the chromosomes from one parent, and half from the other. The way in

which the chromosomes are selected for the child can differ depending on the set-up, but a popular way is to create two children, one with the first half of the first parent, and the second half of the second; and one with the second half of the first parent and the first half of the second.

Once the children are produced, they are mutated in some way. This is done so that the population does not stagnate, and helps introduce some new genes to the pool which can be used in finding solutions which could not have existed with the genes the base population had.

These children will then have to replace individuals within the population. This can also be done through various methods, such as replacing the parents themselves, or replacing the worst functioning members of the population.

While genetic algorithms try to mimic natural selection and evolution, neural networks try to imitate how the brain works. These models are composed of artificial neurons, which "light up" depending on what kind of input they receive. These neurons have certain weight values which are acquired during the training of the neural network. These networks may be used for the purpose of classification, where they are usually trained on a dataset which can be compromised of certain expected inputs, as well as expected outputs based on those inputs. After training, the neural network is expected to be able to take an input, and accurately produce a classification or regression of said input.

The network is usually compromised of an input layer, hidden layer(s), and an output layer. The input layer takes in some sort of input, which affect the neurons in the hidden layers, and based on the weights set up during training for the hidden layers, they end up activating a neuron in the output layer. This lights up the neuron corresponding (in this case) to a classification of the input.

# 2 Literature Review

This section discusses the previous work in music generation using machine learning. Some of the early contributions to the research in this area has been mentioned before. The area of research is also reduced to studies which are mainly about evolutionary algorithms to better focus on the topic at hand.

One of the most famous examples of using genetic algorithms for generating music is GenJam (Biles et al., 1994). They used different modes in order to generate music: learning mode, in which random phrases are selected and presented to the user for feedback, but no genetic operators are used; breeding mode, where genetic operators are used, and half of the population is replaced with new individuals, after which a solo is presented to the user for feedback; and demo mode which is aimed at performance, meaning user feedback is ignored once a piece of music is presented.

The user feedback is given in real-time, by having the listener press "g" or "b", if they think the piece they are hearing at the moment sounds good or not. The magnitude of feedback was determined by the frequency of key presses and had an upper limit. Considering the user feedback had to be given every run of the algorithm, they proposed that to improve it, a preliminary fitness function using a neural network should be utilised, which would have some data already from previously conducted runs.

A 1998 paper (Papadopoulos, Wiggins, et al., 1998) went in a different route compared to GenJam. Their algorithm generated a melody given an input chord progression. The fitness function used was objective, rather than user tailored. They argued it would be too hard to find a consistent evaluation of fitness within a reasonable period of time, given the "fitness bottleneck" (Biles et al., 1994) which was a big drawback with GenJam.

The fitness function consisted of large interval restriction, pattern matching, suspensions, downbeat notes, half-bar notes, long notes, contour, and speed. Their representation of an individual used chromosomes which contained degrees of a scale, instead of absolute pitches, which gave flexibility to be used in a multitude of scales, but restricted the use of chromatic notes. The melodies generated were not particularly groundbreaking.

In 2007 AMUSE(Özcan & Erçal, 2007) used a similar way to represent their individuals; with chromosome values which represented the order of a given note in the scale, restricting chromatic notes. Similarly, they used an objective fitness function which included core- and adjustable objectives (by the user).

The core objectives included checking whether notes are in the chord that is played on the same beat (only when a chord file is used), various

relationships between notes as well as their directions, starting and ending note, the prevalence of fifths, and any drastic changes in pitch. Adjustable objectives were the prevalence of rests, the proportion of hold events, and pattern matching. This algorithm performed fine in a Turing test between it, and solos generated by an amateur musician, and received better scores for solos which had a higher fitness function defined by the aforementioned objectives than the ones which were lower.

In 2011 an algorithm was created which not only generated melodies, but a whole composition, with harmonies and rhythms as well (Donnelly & Sheppard, 2011). Each of the chromosomes were split into 4 parts, which contained soprano, alto, tenor, and bass voices (different ranges of pitch).

Various melodic, harmonic and rhythmic rules were used for the fitness function. The melodic rules included having the opening and closing chord to be tonic, having a final cadence (end to a phrase), leap resolution, stepwise motion; restrictions on leap height, the lowest and highest note, voice crossing and range, as well as repeating pitch. Harmonic rules limited parallel octaves and fifths, as well as the use of seconds, fourths, or sevenths. Rhythmic rules positioned the piece to have variable note durations, as well as limiting note repetition. While they were able to generate a four part composition, the parts were not coherent and moved independently.

The 2013 paper (Koga, Inoue, & Fukumoto, 2013) proposed a method which allowed users to not only score, but make edits to generated melodies, to better fit user preferences. Their fitness score was based on users giving a score from 1 to 7, 1 being extremely dislikable, 4 being neither bad nor good, and 7 being extremely likeable. They represented individuals via a series midi notes with each chromosome containing a note, and presented eight of the best individuals from each generation to eight males testing the algorithm.

The subjects participated in two conditions, A and B. With condition A, the users were able to alter the presented piece of melody, with no more than 3 changes to the notes via a MIDI Sequencing Software. In condition B, the users were only able to score the generated melody. Both of these conditions ran for ten generations of the algorithm.

Another experiment was conducted by having the same subjects listen to four melodies, two of which were picked from the 10th generation from the same user, and two from the result of another user's preferences. In both cases a melody from condition A, and one from B was picked (as well as best individuals).

In the first experiment, the average fitness value of every participant was higher on condition A (where editing as allowed) than in condition B consistently. In the second one the average of fitness values given to the melody

pieces were higher for the melodies generated based on the original user's preference, as opposed to the other user's, with condition A still performing better.

A multi-objective genetic algorithm was used in 2015 (Jeong & Ahn, 2015), with the idea to optimise for more than one objective function, as opposed to simple genetic algorithms, which can only optimise for one. Due to this, the fitness function was compromised of two parts:

- One part was trying to guide the composition toward "stability", by using chord tones (notes within the chord being played), and by motioning towards lower pitches.
- The other part was focused on giving the piece more tension, motioning upwards, while awarding more dissonant notes (which are not part of the chord) better scores.

They used a set rhythm, meaning all notes stayed a constant duration, and only pitch changes were employed by the algorithm. By their description, the produced melodies were pleasant.

A method which distributed the scoring of generated melodies was proposed in 2017 (Fukumoto & Hatanaka, 2017). This allowed for the creation of music that multiple users would like. Suggested applications for such a system were common areas, such as hospital waiting rooms, or shopping centres.

Sixteen subjects were divided into pairs, where the score of each user would affect the fitness value for the other. Similarly to (Koga et al., 2013), the subjects scored each melody from 1 to 7, depending on how good they perceived the piece to be. Twenty-one generations worth of melodies were evaluated all together, while exchanging them at the end of the 3rd, 7th, 11th, 15th, and 19th generation between users. The experiments showed a weak upwards trend in terms of mean and maximum fitness values, indicating that the proposed method has a possibility to find better melodies.

Recently, (Nam & Kim, 2019) developed an algorithm which uses some user input to improve itself, and is able to generate jazz melodies. They achieved this by allowing users to score the generated piece of melody using a survey on the best individual for the 1st, 30th, 100th, 200th, 500th, and 1000th generations. They used a novel geometric crossover method, so as to allow individuals to have variable length chromosomes, and to better preserve good traits from parents. The melodies were generated from a fixed pattern of chord progression, to better focus on the generation of melodies. The algorithm in this study was able to create melody lines which according to their questionnaire completed by humans preformed quite well.

Since in recent years the research around using neural networks instead of genetic algorithms have gotten much more popular, it seems like a good idea to revisit the use of genetic algorithms, while also implementing the use of more modern approaches.

# 3 Methodology

This section will discuss the implementation of the algorithm, as well as the purposes of specific design choices.

The algorithm is implemented using python (Foundation, n.d.). A third-party library called MIDIUtil (MarkCWirt, n.d.) was used to generate the MIDI files for listening and rating purposes. The generated MIDI files were listened to via Window's default Windows Media Player (*Microsoft*, n.d.). The neural network was built using TensorFlow's (TensorFlow, n.d.) Keras (Team, n.d.) library.

While there are a great number of ways in which sound or music can be synthesised, using neural networks, such as (Oord et al., 2016; Hadjeres, Pachet, & Nielsen, 2017; Mehri et al., 2016; Yang, Chou, & Yang, 2017), there also seems to be a problem with not being able to give users the ability to alter the outcome of the generation of the music. While users are in control of the input data to an extent, these networks are usually trained on already existing music, which might be preferred by the users, but is not "created" by them. And when it comes to creating a fitness function from scratch (Papadopoulos et al., 1998; Özcan & Erçal, 2007; Jeong & Ahn, 2015), we can likely not reach the intricacies and accuracy of human preference.

Therefore, for the design of this algorithm, a way in which both user interaction (a way for the user to alter the way the melodies are generated) and user preference (a way in which the generated melody is shaped) is employed. This is done by letting the user specify settings for the algorithms such as the search space size, the key, the mode, ratio of notes, starting and range of octaves, various genetic operator settings, and some settings for the neural net. The flow of the algorithm is explained below, however some important details about it must be understood first, therefore the following sections will explain this required information.

## 3.1 The Search Space

Firstly, the user is able to specify how the search space looks, by choosing

- The key in which the melody is generated in eg. "C".
- The mode in which the melody is generated in eg. Ionian (also known as the major scale).
- The percentage of diatonic notes (notes within the scale), chromatic notes (notes outside of the scale), rests, and holds.

- The starting octave as well as the range in which the melody is generated.
- The size of the search space.

The algorithm will then generate a 2D grid where notes are placed based on these settings. While other methods, such as the one presented in (Özcan & Erçal, 2007) can allow the user to specify the rest density, and the rhythmic variety, this generally means very little to the end user. Therefore it would be much better for the user to allow some number of rests, and holds; but ultimately decide it themselves how much they want these to be utilised, by rating the given pieces. If a user prefers melodies with more pauses, or a lower variety of notes, the system should adapt to this based on the user's ratings. In each coordinate, either notes (integers corresponding to MIDI values), rests (represented as a 1), or holds (represented as a 0) are placed. The genetic algorithm part of the program will then use this search space to form melodies. Currently the algorithm generates a 4 bar melody in $\frac{4}{4}$ time, however the amount of bars or time signature can easily be changed. Due to how the neural network is set up though, these changes would mean that the network would have to be retrained from the ground up.

A randomly generated search space

- in the key of C,
- in Ionian mode,
- with 55% diatonic notes, 0% chromatic notes, 15% rests, and 30% holds,
- starting at octave 3, with a range of 2 octaves,
- of size 5

is shown in Table 1.

| | | | | |
|---|---|---|---|---|
| 67 | 71 | 1 | 0 | 0 |
| 50 | 69 | 71 | 0 | 0 |
| 0 | 0 | 65 | 67 | 50 |
| 48 | 1 | 57 | 67 | 0 |
| 0 | 0 | 0 | 71 | 55 |

Table 1: A randomly generated search space

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| DO5 | UL2 | DL5 | DR1 | UR3 | LE2 | UR5 | RI3 | ST0 | LE3 |

| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|
| DR5 | DO2 | RI2 | DL2 | ST0 | ST0 | UR5 | LE5 | ST0 | RI2 |

| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| UL1 | LE3 | UR4 | UR4 | UR4 | DL5 | DL2 | DL5 | ST0 | UL3 | UL4 |

Table 2: The representation of an individual

As these grids are constructed programmatically, the top left of the grid (in this case 67) if considered to be [0,0]. Moves on the $x$ axis correspond to increasing or decreasing the second value, while moves on the $y$ axis correspond to increasing or decreasing the first value; such that [$y$,$x$].

## 3.2 Representation of Individuals

While other genetic algorithms tend to represent each individual's genes as MIDI values, or actual musical notes; here they are represented as moves they make within the search space. This is done to allow a user to create a customisable environment for the notes to be generated in. Once the algorithm starts, a population of individuals are generated, each with a random list of moves they will perform.

Any one gene can represent a move in any of eight directions as well as the ability to stay where they are, giving nine different operations for travel. Additionally each gene also has a move value associated to them (0 for staying in place), which has a maximum value given by the user. Once generated, an individual with a maximum jump value of five might look like the one in Table 2.

The numbers on the upper row represent the gene number. Each direction is encoded as:

- UP = Up direction

- UR = Up right direction
- RI = Right direction
- DR = Down right direction
- DO = Down direction
- DL = Down left direction
- LE = Left direction
- UL = Up left direction
- ST = Stay (no travel)

Every individual starts at a random point on the generated grid, and that starting point (no matter what it originally is), is converted to the tonic note. For example, if the starting point is chosen to be at coordinates [1,0] of the previously described grid, the "0" would be replaced with a 48 (MIDI value for C3). This will always be the first value of the individual, meaning not matter what every melody line will start with the tonic. This is why there are only thirty-one genes in every individual instead of thirty-two.

Once these individuals need to be converted to MIDI, and to train the neural network, a function calculates the moves these individuals take within the search space to generate an array of size thirty-two, filled with the MIDI notes "collected" along the path. After this conversion, the above mentioned individual would look as shown in Table 3. Each MIDI number represents an eighth note. Since there are thirty-two of these eighth notes, they add up to 4 bars in $\frac{4}{4}$ time signature.

## 3.3 Genetic Operators

This section will briefly discuss the genetic operators used.

The selection is done through the use of "tournament selection". A set of random individuals (size of set determined by user) are selected from the population, and the best individual from this selection is passed onto being a parent. This tournament is ran twice so that two parents are generated.

Uniform crossover is used to generate the children. This is done by going through the chromosomes of the parents, then with a probability (user setting) deciding which child gets which chromosome. For example, the first chromosome from the first parent could either be passed onto the first child or the second. If it is passed onto the first child, the first chromosome from the second parent will be passed onto the second child, and vice versa if decided otherwise. The user is able to set the chance for crossover to happen. If it does not, the parents are passed onto the mutation operator.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 48 | 0 | 0 | 0 | 0 | 0 | 71 | 0 | 0 | 0 |

| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|
| 71 | 55 | 55 | 55 | 0 | 0 | 0 | 0 | 67 | 67 |

| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 71 | 67 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 48 | 67 |

Table 3: An individual's moves converted to MIDI notes from search space

Uniform mutation is used to further explore the search space. With a probability (user setting) the genes of each child is either mutated or not. If they are mutated, a random direction and move number will be chosen, with the maximum move number specified by the user.

After mutation, tournament replacement is employed to place the children into the population. Elitism is used to keep a percent (user set) of the best performing individuals in the population. This is done so as to keep good individuals, as well as to improve the performance of the algorithm (Parks & Miller, 1998; Zitzler, Deb, & Thiele, 2000). Similar to selection, a random portion of the population that did not get saved by elitism get selected, but this time the worst performing individual will be replaced. This is also done twice, for each child.

## 3.4 Fitness Function

The fitness function is perhaps one of the most important aspects of any genetic algorithm. It makes or breaks the usefulness of the application. Since it is nigh impossible to create a perfect objective fitness function for the purposes of generating music, the next obvious option is to create a subjective one. There are many ways to go about this, however, the focus is set on using a neural network to achieve this. This is done in the hopes that it can learn user preference based on the scores given to the MIDI sequences generated

by the genetic algorithm.

Following some of the aforementioned methods for scoring melodies (Koga et al., 2013; Fukumoto & Hatanaka, 2017), the user will be generated a MIDI file which they have to listen to and rate from 1 to 7, 1 being the worst, and 7 being the best possible score. These scores and MIDI sequences are then saved to a file which is then fed into the neural network for training. The MIDI sequences are as described in Table 3.

The neural network (Figure 1) itself is a modified version of the TensorFlow 2 implementation (Fawaz, Forestier, Weber, Idoumghar, & Muller, 2019; Hfawaz, n.d.) of the model proposed in (Wang, Yan, & Oates, 2017). While this model was originally used for the purpose of classification, it would not make sense to utilise it the same way in this scenario. Since the scoring works from 1 to 7, there would be no way to distinguish individuals with the same score. Instead the model is used for regression, which means the output will be a floating-point number between 1 and 7. This way, the differentiation between individuals' fitness value is possible. Another change is the addition of a dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) layer with a rate of 0.5, which should help mitigate over-fitting issues.

## 3.5 Algorithm Flow

This section will describe how the whole algorithm flows (Figure 3) to generate melodies, and improve upon this generation via user input.

First, the search space is generated based on user input. This will include MIDI notes, as well as rests, and holds, as described before. The population will then be initialised based on parameters set by the user, with individuals being represented by the moves they make within the search space. If it is the first generation, a random individual will be chosen, and a MIDI file will be generated based on the interpretation (the moves will be translated into the MIDI notes "picked up" from the search space). Figure 2 shows what the above mentioned individual would look like. This is done 10 times on the first generation to build up a base for the neural network. This same action of MIDI generation is also done if it is the generation to rate, but instead of choosing a random individual, the best one in the population will be chosen.

The user is able to set how often they would like to score melodies. If the same melody would show up as before at the time of scoring, the algorithm will wait until the next opportunity to allow the user to score. This is done so that users do not have to score the same melody twice, which would result in multiple entries of the same sequence and potentially score in the training data which might cause over-fitting issues.

Once the user scores the melody, it gets recorded along with the MIDI

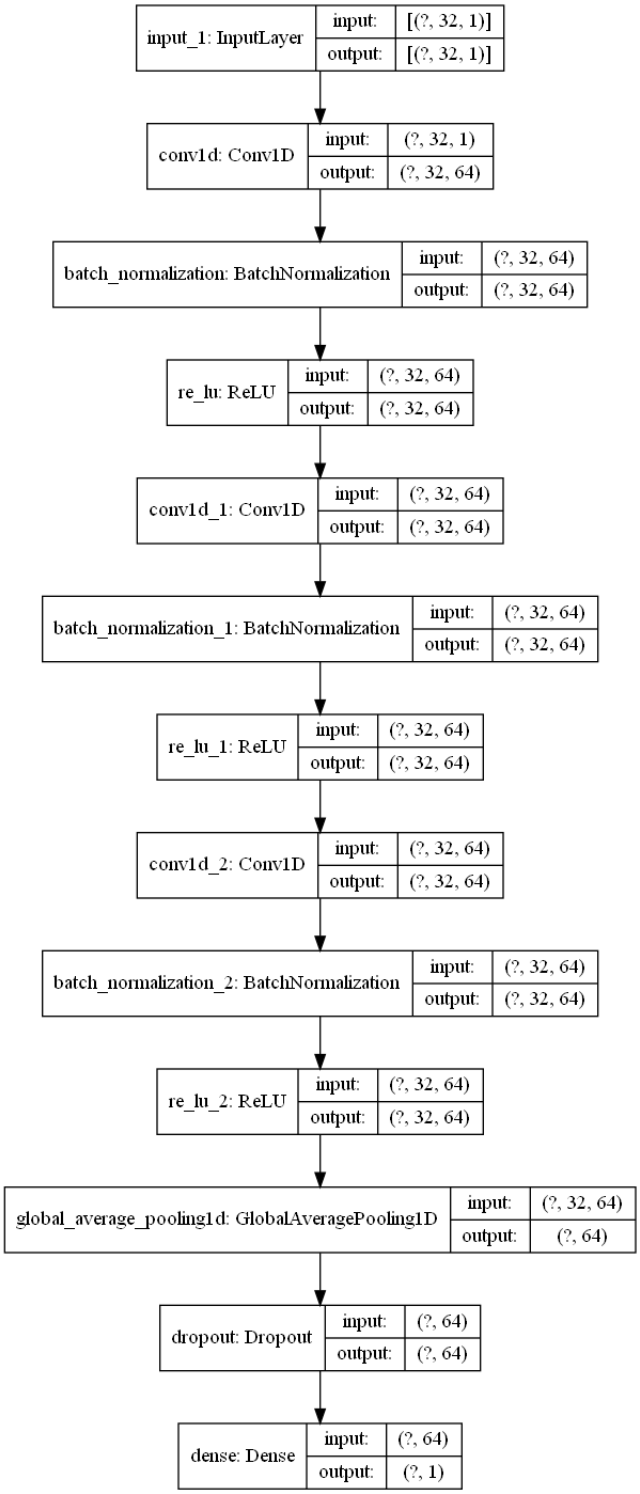


Figure 1: The neural network used to generate fitness values

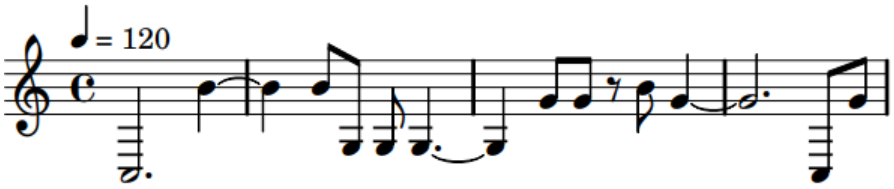


Figure 2: A random individual (score generated from MIDI)

sequence to a file, which is then used to train the neural network to produce fitness values. To speed up training and provide consistency, the best existing weights for the model are loaded (only after the network has been trained at least once). When the network is trained, the population is passed onto it to score, and fitness values for each individual are given.

If it is the last generation, a MIDI file for the best performing individual is generated. Otherwise the process of selection, crossover, mutation, and replacement are utilised, as described before.

This cycle continues until the last generation, or user intervention (by stopping the program for example).

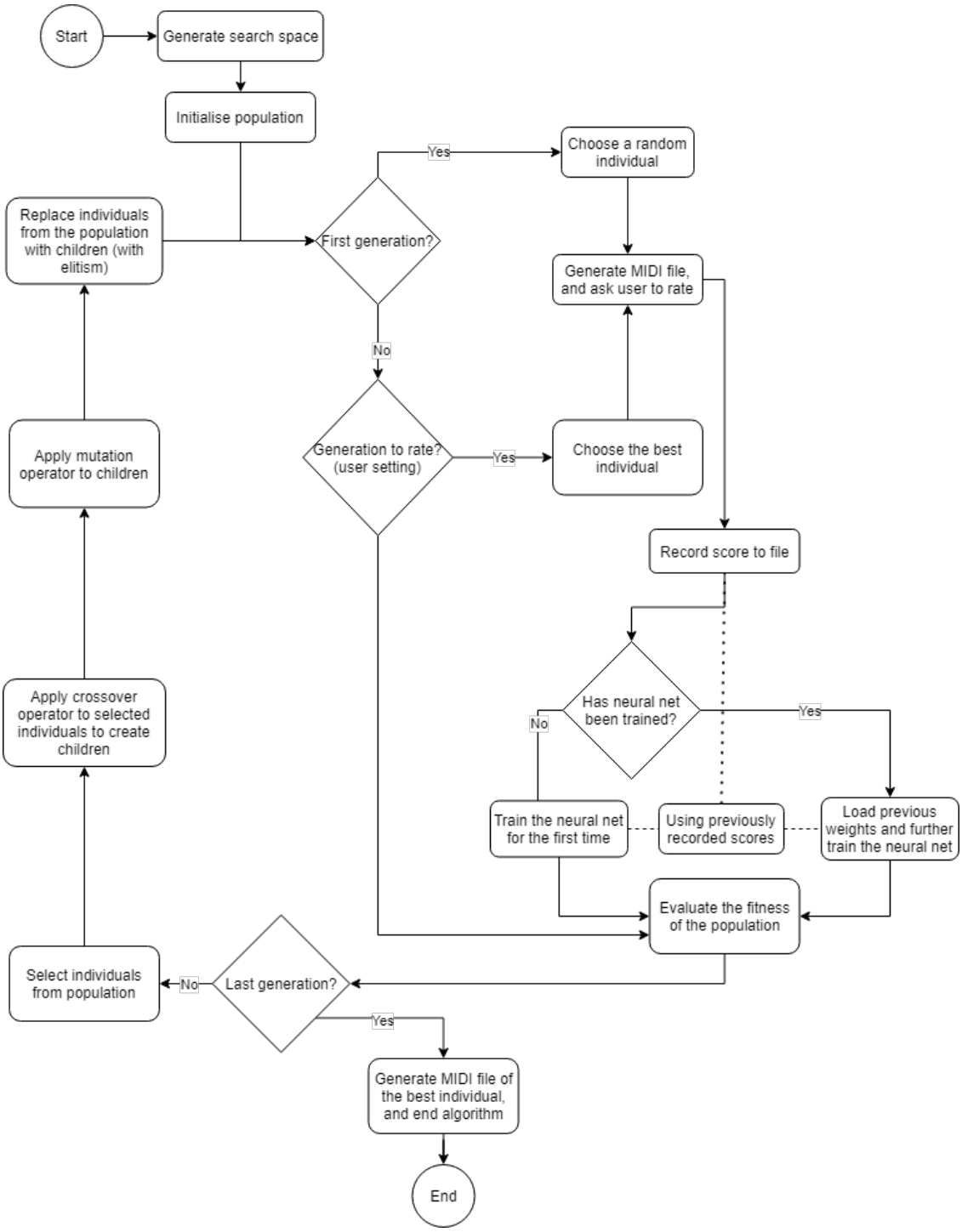


Figure 3: The flow of the algorithm

| | |
|---|---|
| Search space size | 50 |
| Maximum number of moves | 5 |
| Key | C |
| Mode | 0 (Ionian/Major) |
| Starting octave | 3 |
| Range of octaves | 2 |
| Diatonic note percent | 55 |
| Chromatic note percent | 0 |
| Rest percent | 15 |
| Hold percent | 30 |
| Population size | 300 |
| Number of generations | 300 |
| Generations per training | 10 |
| Tournament size | 8 |
| Crossover chance | 90% |
| Mutation chance | 10% |
| Elitism | 5% |
| Tempo | 120 |
| Epochs | 200 |

Table 4: Original experiment parameters

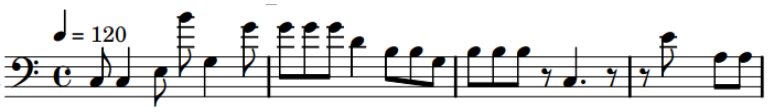


Figure 4: A randomly generated individual from the first experiment

# 4 Experimental Results

A number of experiments were run to test the algorithm. Unless stated otherwise, the parameters used for the experiments are as described in Table 4.

## 4.1 Initial Experiment

As expected, the first set of 10 randomly generated melody lines were extremely random, and did not sound good. An example from this set can be seen in Figure 4.

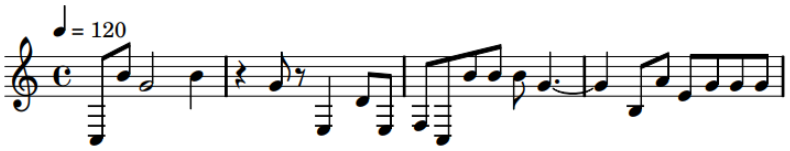


Figure 5: The result of the first experiment

| | |
|---|---|
| Search space size | 50 → 100 |
| Maximum number of moves | 5 → 100 |
| Number of generations | 300 → 500 |

Table 5: Parameter changes in the second experiment

At the end of the first experiment, the fitness of the best individual was 1.7576112, a rather low score. However, this was expected, as the there were only twenty-three melodies rated so far, which is not a lot for the neural network to go off of. The end result of experiment one can be seen in Figure 5

## 4.2 Increasing the Search Space and Move Number

For the second experiment, the search space size was increased to 100, and the maximum number of moves was increased to match this, to 100. Along with this, the number of generations was also increased to 500. This was in hopes that with a larger area for individuals to cover, and more opportunities to build the network, better melodies can be generated. Since the network had some initial ratings of random notes, the amount of random individuals to rate at the start of the algorithm was reduced from ten to three. This is still a useful interaction, as these randomly generated sequences could potentially help gauge what not to create. From this point on this number will not change. The changes are summarised in Table 5.

## 4.3 Narrowing the Move Number

By the end of the second experiment, the best fitness reached 2.757979. This is still not a good score, however the melody took a shape (Figure 6) , which is not as unpleasant as the one generated at the end of the first one. By this point, fourty-seven ratings were indexed. Throughout the second experiment a lot of the same note patterns started propping up, indicating a premature convergence. To mitigate this, the mutation chance was increased to 0.25.

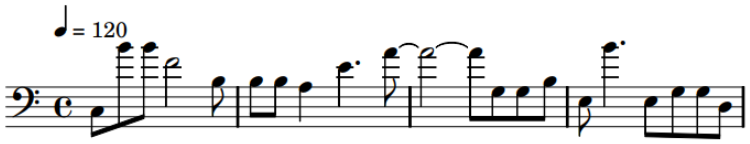


Figure 6: The result of the second experiment

| | |
|---|---|
| Maximum number of moves | 100 → 10 |
| Mutation chance | 10% → 25% |

Table 6: Parameter changes in the third experiment

The number of allowed jumps was also reduced to 10, to see if it makes any effect. These changes are summarised in Table 6.

## 4.4 Change of Key, and Narrowing of Octave Range

The third experiment also improved the melodies generated by a little bit (Figure 7), with a score of 4.5674295 by the end. Though the piece generated was not particularly pleasant, and the use of perfect octaves (playing the same note again, but one octave higher), as well as the rate of jumping to higher octaves seemed to be high. For the fourth experiment (Table 7), the range of octaves was lowered to one. To introduce more variety into the training data, the key and mode were also changed to E Aeolian (natural minor). The mutation chance was also lowered to twenty percent.

## 4.5 Experiment on the So-Far Built Dataset

Reducing the number of octaves seemed to have produced more stable individuals. The final score was 4.308205, lower than in the third experiment, however the melody (Figure 8) sounded much better. For the fifth, and final experiment, the key and mode was once changed back to the original C Ionian. This time the generations per training was not set, and the algorithm

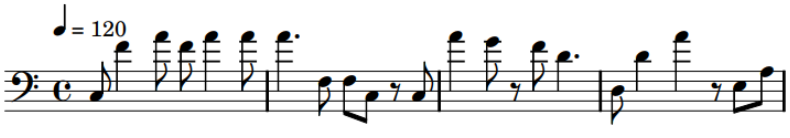


Figure 7: The result of the third experiment

| | |
|---|---|
| Key | C → E |
| Mode | 0 → 5 (Aeolian/Natural Minor) |
| Range of octaves | 2 → 1 |
| Mutation chance | 25% → 20% |

Table 7: Parameter changes in the fourth experiment

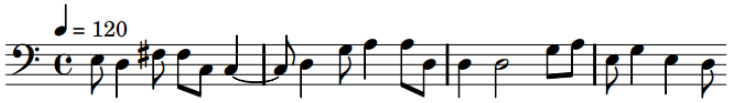


Figure 8: The result of the fourth experiment

was let to run, to try and produce a melody line based on the training data collected so far. This was done 10 times, to see what the best produced melody would be.

After running the fifth experiment, the highest fitness score reached was 4.693455 (Table 9), which produced the melody in Figure 9. A total of 143 outputs were rated by the end of the experiment, which is still relatively low for the neural network to accurately predict user preference. However the model exhibited improvement in terms of melody generation and score prediction.

| | |
|---|---|
| Key | E → C |
| Mode | 5 → 0 (Ionian/Major) |
| Generations per training | 10 → not set |

Table 8: Parameter changes in the fifth experiment

| | |
|---|---|
| First run | 4.693455 |
| Second run | 3.9166205 |
| Third run | 4.0125384 |
| Fourth run | 3.9604008 |
| Fifth run | 4.3725057 |
| Sixth run | 4.2411494 |
| Seventh run | 3.2873302 |
| Eighth run | 4.315959 |
| Nineth run | 3.4854746 |
| Tenth run | 2.5659797 |

Table 9: Best individual fitness scores from each run for the last experiment

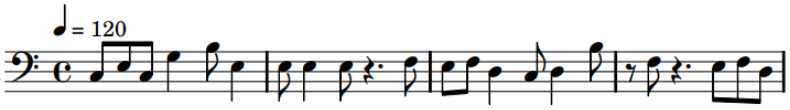


Figure 9: The best scoring individual in the final experiment

# 5 Discussion

While slow to start, it seems the algorithm was able to generate melody lines that fit to the user's requirements. It would be hard to compare the findings of this paper to other papers' results without many users. And it would take much more experimentation and data for the neural network to be able to conclude how well the algorithm performs in the long run. There is also a lack of data on the performance of the algorithm outside of the author's experience.

To be able to distribute this algorithm, and to test it with multiple users, a way to train the neural network when the user does not have TensorFlow installed has to be figured out. This could be done through sending the users' scores to a central server, which can be used to train the network, and score the melodies generated by the users. This might be computationally expensive, however.

There is also a lack of UI for the program, and the user would have to set each parameter within the python code, which for some would introduce problems, as users may not have the required knowledge to edit the code. If there was a UI, the search space could also be visualised for the user, along with showing them how the best performing individual looks on that search space. This would allow the user to tweak settings according to how they think the melodies could be improved within that search space. There is also a considerable amount of settings, which some users might feel intimidated by. A lot of the genetic algorithm part of the settings could be partially hidden from the user, so that they would have to open a separate "advanced" menu to access these parameters, making it easier to use for general users.

There is also many more avenues in which the customisability of the algorithm could be improved. This includes allowing the user to change the selection, crossover, mutation, and replacement operators. While the current operators perform well, this is just one aspect in which users can generate different melodies. For the genetic part of the algorithm, to prevent premature convergence, if there are multiple individuals with the same genes, they could be taken out at the replacement stage, and new, perhaps random individuals could be introduced. This would also bring in new genes for the population, and could potentially produce better results. When it comes to the representation of the individuals, improvements can be made such that each gene could hold multiple step values. This would allow for the movement to be more free within the search space, potentially resulting in better suited melodies.

Along with this, the user should be able to save the search space, so that

it may be used in future runs of the algorithm. If the user feels like one particular search space is producing melodies which they like, this would allow for them to test this same space with different parameters.

Multiple searches could be run simultaneously so that polyphonic music would be generated. If running 3 searches at once for example, this could be in the form of having a separate search space for each population, while starting them on the tonic, the third, and the fifth, respectively. This would mean that the start will always play the first chord of the scale that it is in, and from then the melody could take shape from the user's preferences. The search space could even be modified so that the algorithm generates drum patterns.

Since the range of octaves only go in the up direction, the feeling of the pieces aren't as dynamic as they could be. This can easily be solved by adding a lower octave bound as well, so that the piece would have a greater customisability in terms of range. This range could also be made partial, so for example only half of the lower (below the tonic note) octave can be utilised, while one and a half above.

The way the fitness function is calculated can also be improved, by utilising a range of neural networks instead of just one. The user could pick which one they would like, and train their preferences on that particular model.

While compared to other genetic algorithms which use an objective fitness function such as in (Papadopoulos et al., 1998; Özcan & Erçal, 2007; Jeong & Ahn, 2015) the output is not While there a lot of changes that could be made to improve the algorithm. It has performed the originally set out goal to produce melodies according to the user's preference, while being customisable.

# 6 Conclusion

As the creation of music becomes more accessible, people look for new ways to improve their workflow, or to get inspiration. To help solve this issue, a genetic algorithm/neural network hybrid is presented in this paper, which uses grid-based search space filled with MIDI notes. The population of individuals are initialised, each with a random set of moves that pertain to their movement within the previously generated search space. At the point of interpretation, the moves of these individuals are processed, and the "collected" MIDI notes along their moves are written to a MIDI file. This file is then rated by the user with a score of 1 to 7. This score, along with the MIDI sequence that was rated, is then added to the dataset which is used to train the neural network. The neural network used in this paper is a 3 layer 1D convolution network with a dense layer as its output. It is a modified version of the TensorFlow 2 implementation (Fawaz et al., 2019; Hfawaz, n.d.) of the model proposed in (Wang et al., 2017). The only changes being the addition of a dropout (Srivastava et al., 2014) layer before the final dense layer, and the output layer being utilised for regression instead of classification.

The results show improvement in terms of the generation of melodies and their quality. Though the algorithm should be tested for a wider range of individuals, and for a longer time-frame, there are promising qualities for the proposed way of generation.

While there are a great number of improvements that could be made to the system, it does provide appropriate outputs, and does improve based on the input. If ran for long enough, this could be a powerful tool for musicians to incorporate into their workflow. Especially as the code can be integrated into a DAW (Digital Audio Workstation) as a plugin, streamlining the process of melody generation.

# References

Biles, J., et al. (1994). Genjam: A genetic algorithm for generating jazz solos. In *Icmc* (Vol. 94, pp. 131–137).

Chen, C.-C., & Miikkulainen, R. (2001). Creating melodies with evolving recurrent neural networks. *IJCNN01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*. doi: 10.1109/ijcnn.2001.938515

Donnelly, P., & Sheppard, J. (2011). Evolving four-part harmony using genetic algorithms. In *European conference on the applications of evolutionary computation* (pp. 273–282).

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data mining and knowledge discovery*, *33*(4), 917–963.

Foundation, P. S. (n.d.). *Python.org.* Retrieved from `https://www.python.org/`

Fukumoto, M., & Hatanaka, T. (2017). A proposal for distributed interactive genetic algorithm for composition of musical melody. *Information Engineering Express*, *3*(2), 59–68.

Hadjeres, G., Pachet, F., & Nielsen, F. (2017). Deepbach: a steerable model for bach chorales generation. In *International conference on machine learning* (pp. 1362–1371).

Hfawaz. (n.d.). *hfawaz/dl-4-tsc: Deep learning for time series classification.* Retrieved from `https://github.com/hfawaz/dl-4-tsc`

Horner, A., & Goldberg, D. E. (1991). Genetic algorithms and computer-assisted music composition. *Proceedings of the 1991 International Computer Music Conference*, *1991*, 479–482.

Jeong, J. H., & Ahn, C. W. (2015). Automatic evolutionary music composition based on multi-objective genetic algorithm. In *Proceedings of the 18th asia pacific symposium on intelligent and evolutionary systems-volume 2* (pp. 105–115).

Koga, S., Inoue, T., & Fukumoto, M. (2013). A proposal for intervention by user in interactive genetic algorithm for creation of music melody. In *2013 international conference on biometrics and kansei engineering* (pp. 129–132).

Laine, P., & Kuuskankare, M. (1994). Genetic algorithms in musical style oriented generation. *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. doi: 10.1109/icec.1994.349942

MarkCWirt. (n.d.). *Markcwirt/midiutil: A pure python li-*

*brary for creating multi-track midi files.* Retrieved from `https://github.com/MarkCWirt/MIDIUtil`

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., ... Bengio, Y. (2016). Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*.

*Microsoft.* (n.d.). Retrieved from `https://windows.microsoft.com/en-UK/windows/products/`

Nam, Y.-W., & Kim, Y.-H. (2019). Automatic jazz melody composition through a learning-based genetic algorithm. In *International conference on computational intelligence in music, sound, art and design (part of evostar)* (pp. 217–233).

North, A. C., & Hargreaves, D. J. (2007). Lifestyle correlates of musical preference: 2. media, leisure time and music. *Psychology of Music*, *35*(2), 179-200. doi: 10.1177/0305735607070302

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Özcan, E., & Erçal, T. (2007). A genetic algorithm for generating improvised music. In *International conference on artificial evolution (evolution artificielle)* (pp. 266–277).

Papadopoulos, G., Wiggins, G., et al. (1998). A genetic algorithm for the generation of jazz melodies. *Proceedings of STEP*, *98*.

Parks, G. T., & Miller, I. (1998). Selective breeding in a multiobjective genetic algorithm. In *International conference on parallel problem solving from nature* (pp. 250–259).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Team, K. (n.d.). *https://keras.io/.* Retrieved from `https://keras.io/`

TensorFlow. (n.d.). *Tensorflow.* Retrieved from `https://www.tensorflow.org/`

Wang, Z., Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *2017 international joint conference on neural networks (ijcnn)* (pp. 1578–1585).

Yang, L.-C., Chou, S.-Y., & Yang, Y.-H. (2017). Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*.

Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, *8*(2), 173–195.

# Appendices

## A Project Management Evidence

Original project plan:

- 17/05 - 23/05: Finding a new project title
- 24/05 - 14/06: Literature review and initial research
- 15/06 - 28/06: Implementation
- 29/06 - 12/07: Evaluation

This way, there would be a 2 week leeway if the project would get sidetracked.

Unfortunately, due to external circumstances, as well as responsibilities for other modules, the project plan ended up looking as such:

- 17/05 - 3/06: Finding a new project title
- 4/06 - 19/07: Literature review and initial research
- 20/07 - 30/07: Implementation and evaluation