An Application for Evolutionary Music Composition Using Autoencoders

Robert Neil McArthur(✉) and Charles Patrick Martin

Australian National University, Canberra, Australia
{u6956078,charles.martin}@anu.edu.au

Abstract. This paper presents a new interactive application that can generate music according to a user's preferences inspired by the process of biological evolution. The application composes sets of songs that the user can choose from as a basis for the algorithm to evolve new music. By selecting preferred songs over successive generations, the application allows the user to explore an evolutionary musical space. The system combines autoencoder neural networks and evolution with human feedback to produce music. The autoencoder component is used to capture the essence of musical structure from a known sample of songs in a lower-dimensional space. Evolution is then applied over this representation to create new pieces based upon previously generated songs the user enjoys. In this research, we introduce the application design and explore and analyse the autoencoder model. The songs produced by the application are also analysed to confirm that the underlying model has the ability to create a diverse range of music. The application can be used by composers working with dynamically generated music, such as for video games and interactive media.

Keywords: Algorithmic music composition · Interactive evolutionary computation · Autoencoder neural networks

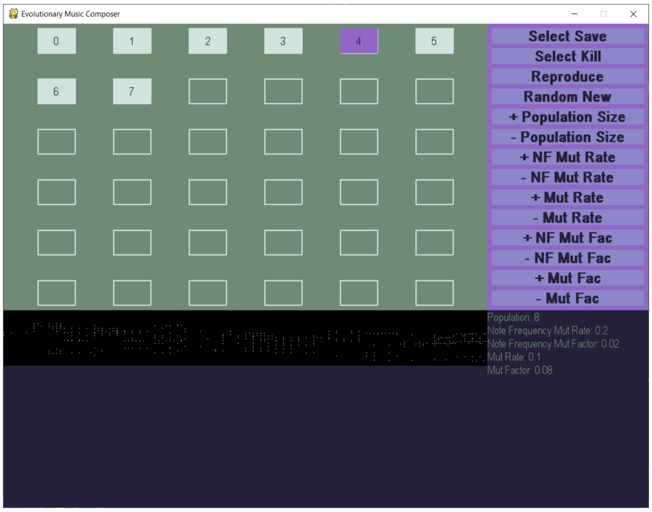
1 Introduction

Evolutionary art (where art is generated through computation, usually with human feedback) has had a long history. There have been many successes in applying evolutionary algorithms over the years to generate, more commonly visual, but also aural forms of art [18]. This paper presents an interactive application that allows a user to create new songs inspired by the process of biological evolution, as shown in Fig. 1. The application repetitively generates sets of songs which the user can choose from so that the underlying algorithm can create new, similar pieces (each song is represented by a box in the upper left area of the application). Once the user reaches a tune they enjoy, they can save the music.

The application is based upon a novel method for algorithmic music composition, combining autoencoder neural networks and evolution with explicit human feedback to produce music. This method is related to latent variable evolution

J. Romero et al. (Eds.): EvoMUSART 2021, LNCS 12693, pp. 443–458, 2021.
https://doi.org/10.1007/978-3-030-72914-1_29

which was introduced in a paper presenting how the latent input variables of a trained generative adversarial network (GAN) could be evolved to create fingerprints [2]. Despite the successes autoencoders and evolution have had in making music, little work has been done in combining them for interactive generative composition.



Population: 8
Note Frequency Mut Rate: 0.2
Note Frequency Mut Factor: 0.02
Mut Rate: 0.1
Mut Factor: 0.08

Fig. 1. The evolutionary music composition application. Each numbered square represents an individual song.

For the purposes of algorithmic music composition with human feedback, the algorithm should know some sense of musical structure. Otherwise, it can quickly become possible that the algorithm would become stuck forever in a random mess of notes. This leads to the question - what actually is music? The answer is much deeper than saying music is some ordering of groups of notes. After all, the vast majority of people would not classify something as structured as playing two major scales on a piano simultaneously (with the second scale starting a major seventh above the first - a notoriously dissonant combination) as music. Despite this, there exist some fantastic atonal masterpieces in the world (where the notes need not conform to any scale). The true definition of music is complex, to say the least.

Though often not perfect, autoencoders work exceptionally well at finding hidden structures within data. If we have a dataset of music, we can then use an autoencoder to learn some hidden structure within the songs represented at a much lower dimension. Following this, each of the features can be analysed to find the latent space of the encoding (the range of input which produces meaningful outputs - the features need not be 0-centred for example). Principal component analysis can then be applied to find a meaningful mapping from a standard normal distribution (the principal components with dimension equal to the feature size) to this latent space. For our purposes, the use of an autoencoder solves the problem of defining music. It provides a lower-dimensional representation of music, forming a genotype to which evolution can then be applied.

This application could be used by composers working with dynamically generated music, such as for video games and interactive media. Further, modifications to the underlying algorithm could allow for music to automatically change while it is being played. This could be applied in games so that the accompanying music transforms dependent on metrics such as mood.

In the next section, we discuss related works in the fields of evolutionary art and algorithmic music composition. We then describe our dataset, the autoencoder model, and how evolution has been applied before putting it together in the application. We perform experiments to confirm that the application has the capacity to create a diverse range of musical outputs.

2 Related Works

One of the most well-known works in the field of evolutionary art is the online collaborative service called Picbreeder (an application allowing its users to create a wide range of pictures) [20]. The mechanism behind Picbreeder is the usage of NEAT (NeuroEvolution of Augmenting Topologies) to successively add layers of complexity to each generation of generated images. This is achieved by the use of composition pattern producing networks (CPPNs), mapping the coordinate of a pixel to an intensity value (using a variety of functions to add regularities found in nature). The combination of NEAT and CPPNs allows users to select from a variety of lower complexity images and progressively add more detail through evolution. One of the primary outcomes of this work was the ability to overcome user fatigue (creating good images takes time). Their method countered this by allowing users to start generating new images from others which were made by the online community.

A similar work has shown promise for images by combining GANs with interactive evolution [3]. The paper shows how the latent input variables (to a trained GAN for some class of image) could be evolved to produce new images. The work intended to make it easy for users to evolve images to some conceived target as opposed to Picbreeder, which is more exploratory. Users were able to do as such more successfully for when the GAN was trained on shoes as opposed to faces. The paper gives merit to the idea of combining autoencoders with interactive evolution for generating music, though audio presents its own unique challenges compared to images.

In the field of algorithmic music composition, one recent paper [17] has started looking into this combining autoencoders with evolutionary algorithms. This work achieved success by using a variational autoencoder and evolutionary algorithm to generate new music (with genotype equivalent to the latent space of the variational autoencoder). The variational autoencoder used was Magenta's MusicVAE [16], an autoencoder which on its own can make a wide range of music. Instead of relying on human feedback, however, the fitness function for evolution is done by finding the distance from a generated song to some predetermined target state. The combination successfully composed music, albeit with some memory loss regarding tonality for longer compositions.

In the field of evolutionary music-making, there have been many successes in generating music. Two works [1,12] go into detail in explaining the mechanisms behind evolutionary music-making and potential solutions to problems which may arise. This includes dealing with issues surrounding musical structure (how elements including key, melody patterns and motifs could be embedded within a model). A more specific example is a paper which presents an interactive melody generation tool [5]. The paper breaks its technique down into three phases. In the first phase, a spectrum of melodies is generated in ABC notation using a genetic algorithm based on the similarity to a database of compositions. These results are passed on to a human who rates the songs from 0 to 100 for the second phase. These ratings are fed into a bidirectional LSTM which is used to mimic the human's scoring. The same genetic algorithm, as described as the first phase, runs again to maximise the trained bidirectional LSTM instead. The paper reported the results to be pleasurable and consistent, though, through personal experience from prototypes of our work, it is hard for a human to rate songs in such a manner.

Another important work for rhythmic evolutionary music-making was the creation of evoDrummer [13]. In the paper, the authors introduce the notion of rhythmic divergence (calculated as the mean relative distance between two feature vectors of drum music, for example, syncopation). A user selects a base rhythm and a desired divergence factor. Rhythms are initialised either randomly, copying the base rhythm, or anywhere in between (taking parts from each). The rhythms and intensities of the percussive elements are then evolved over time to reach the desired divergence. Another well-known work within evolutionary music-making is MaestroGenesis, an application for composing music built on a technique called functional scaffolding for musical composition. In a work extending the technique to produce multi-part accompaniments [9], two CPPNs are used to generate the accompaniment based on the rhythm and pitch information of the melody. One CPPN controls the pitch information, while the other controls rhythmic information for each part. The CPPN pairs are randomly initialised, and they combine and mutate through user selections. Further, the paper presents a layering technique whereby previously generated parts can serve as inputs to new CPNNs to relate separate layers more closely. Other proposed and successful techniques in evolutionary music-making worth mentioning include using particle swarm optimisation to evolve musical features to user ratings [14]; performing random walks on common chord progressions while evolving the pitches of a melody with lightly randomised rhythms [19], and evolving measures of music at a time with random fitness functions [21].

There have also been many methods regarding the use of autoencoder neural networks for composing new music. Choi et al. created an autoencoder with a transformer model to capture musical style from a sample and embed it in a base melody [4]. Tikhonov and Yamshchikov applied a variational recurrent autoencoder supported by history which is trained to compress a dataset of music into features which can then be used to generate original music [23]. Magenta's MusicVAE [16] is based on a multi-layered bidirectional LSTM to encode an input sequence of music. In this system, the decoder used a novel hierarchical recurrent neural network in order to separately produce a drum, bass and melody output from the compressed representation. The hierarchical method showed definite improvements over the flat counterparts for the reconstruction accuracy metric for each part that was produced. The hierarchical method was shown to outperform the flat model in a human-centred listening study consistently. These previous works show that autoencoder neural networks are an effective way of encoding and generating music.

3 Methodology

3.1 Dataset

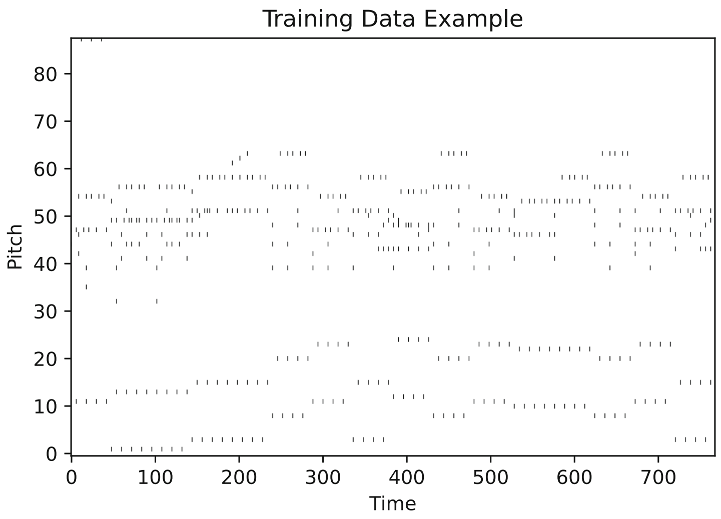


Fig. 2. An example of the training data expressed in piano roll notation. Note that all notes have the same very short duration, a limitation of our model.

The dataset consisted of symbolic music in MIDI format. Initially, MIDI files were downloaded from the MAESTRO dataset [8] which contains over 200 h of piano performances. Unfortunately, our model did not perform well with this dataset, potentially due to the expressive timing that occurs in this dataset. Our second training attempts applied a dataset of video game music, which was more repetitive and included simpler rhythmic information. A dataset of more than 30000 MIDI files was scraped from VGMusic [15]. The dataset was further simplified by changing all notes to have the same very short duration; equivalent to instantly pressing and releasing a note on a piano.

The files need to be processed so that they could be fed into an autoencoder. The files were converted to piano roll notation, as demonstrated by Fig. 2. The horizontal axis represents time, and the vertical axis represents pitch - black indicating a note being played and white being the absence of a note. The range of the pitch was limited to be 88 notes, matching the range of a piano. Further, to restrict the horizontal axis, we decided that the autoencoder would produce 16 bars of music at a time, with a bar broken down into 48 ticks (which was enough to represent faster moving passages). This limited the horizontal axis to 768 across.

The 16-bar length training samples are represented by a 768×88 matrix of binary values. The VGMusic dataset yielded 75136 non-overlapping 16-bar segments for training.

3.2 Autoencoder Model

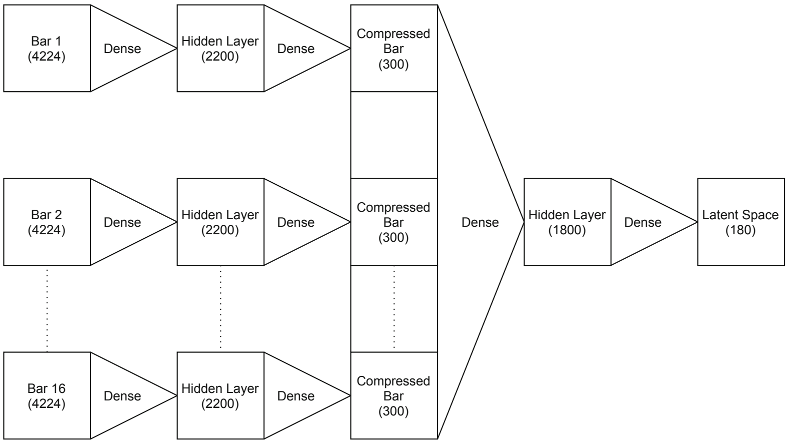


Fig. 3. The encoder model. The numbers in the brackets indicate the size of each part. The decoder is a mirrored version of the encoder. Bars 3–15 are represented with a dotted line.

We applied a process of trial-and-error to construct an autoencoder that would be useful for evolutionary music composition. This included attempts to use variational autoencoders [6], dense autoencoders, bidirectional LSTM autoencoders [16] and transformer models [10, 11].

Ultimately, inspired by similar work by HackerPoet or CodeParade [7], we applied a standard (i.e., not variational) autoencoder. Figure 3 shows the encoder model layout. Each bar is compressed from a size of 4224 (pixels in each bar using piano roll notation) to a size of 300 with the assistance of a hidden layer.

The compressed bars are then concatenated (to a size of 4800) before being compressed again to the latent space representing the song with a size of 180. The decoder is an exact mirror of the encoder. Further, each transition between layers is done with a dense layer and the ReLU activation function. This is with the exception of the last layer which uses a sigmoid function, and the latent space layer which uses no activation function. In addition to this, dropout and batch normalisation layers were applied to negate overfitting.

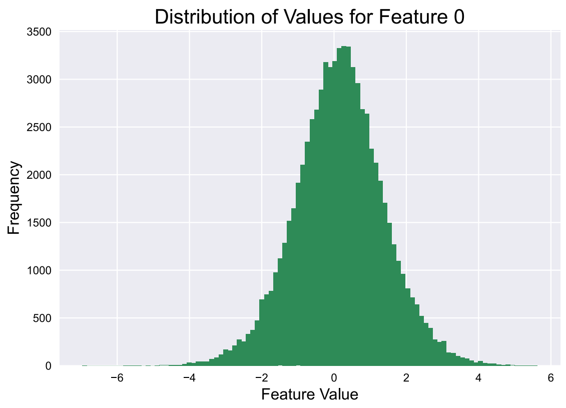


Fig. 4. Each feature in the latent space was found to follow a normal distribution.

Preliminary experiments in randomising the latent variables to generate new music led to "noisy" and hence unpleasant results. As such, the training set was fed into the encoder to investigate the nature of the latent space. The analysis showed that although each feature followed a normal distribution (an example of which is shown in Fig. 4), they, for the most part, did not follow a standard normal distribution (i.e., with $\mu = 0$, $\sigma^2 = 1$) as Fig. 5 reveals.

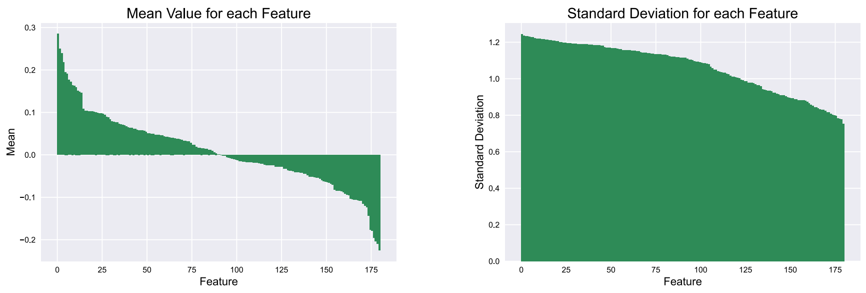


Fig. 5. The mean value and standard deviation of features in the latent space (ordered) from the training data.

Further analysis revealed that some features of the latent space were correlated with one another. This is illustrated in Fig. 6.

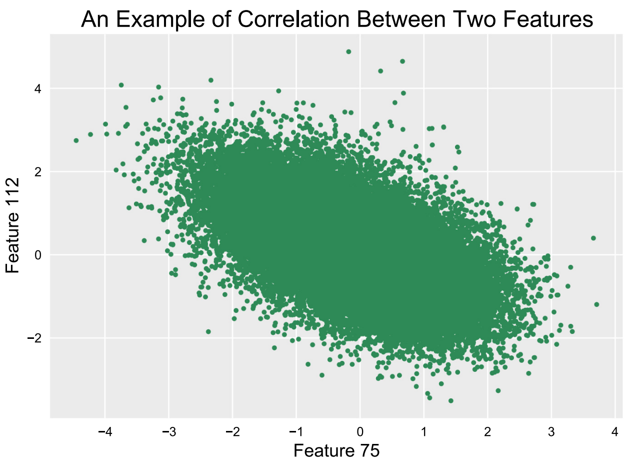


Fig. 6. There is a negative correlation between features 75 and 122 of the latent space.

In combination, this implied that a randomised vector for the latent space was likely to lie outside the space the decoder had been trained on. To rectify this issue, Principal component analysis [22] was applied to map a standard normal distribution to the latent space. The importance of each principal component can be visualised by looking at its associated eigenvalue in Fig. 7.

Diminishing returns are observed for the principal components with the first 15 or so being the most important before holding steady and dipping off again soon after 125. Applying principal component analysis allows us to map a standard normal distribution to the latent space of the autoencoder, forming a basis for evolution.

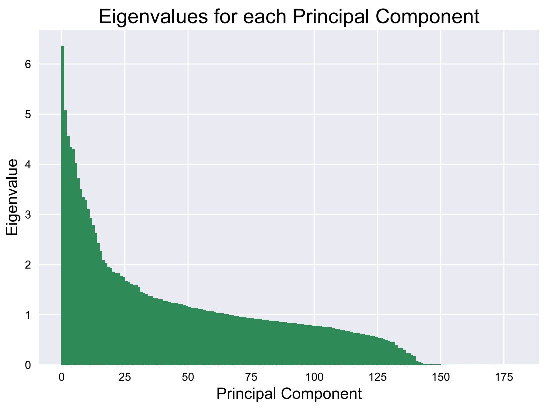


Fig. 7. The ordered eigenvalues associated with each principal component.

3.3 Evolutionary Mechanism

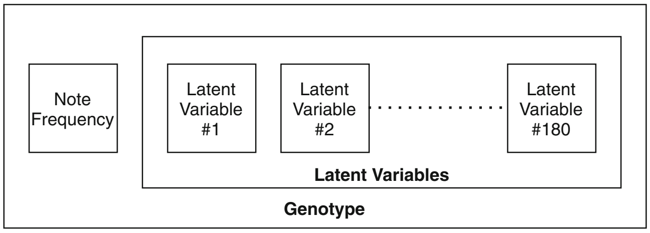


Fig. 8. The latent variables together with the note frequency threshold form the genotype for evolution.

To generate new music, a random vector of dimension 180 (the dimension of the latent space) can be sampled from a standard normal distribution. This is mapped to the latent space using principal component analysis before being sent through the decoder component of the autoencoder. Unlike the input, however, which is discrete for the elements of the piano roll (0 and 1 corresponding to the absence and presence of a note respectively), the decoder produces continuous values in the range $[0, 1]$. This can be thought of as the likelihood of a note being played—how confident the autoencoder is in thinking a note of a given pitch should be played at a given time. With this in mind, another value can be used for generating music which can be thought of as a threshold representing note frequency. This value, in $[0, 1]$, represents how confident the autoencoder must be about a note for it to be played; otherwise, it is ignored. Through this method, the decoder can now generate music. The threshold (referred to later as note frequency) in combination with the 180 latent variables form the genotype for evolution depicted in Fig. 8.

The evolutionary process can be defined as follows:

1. A population is initialised by creating individuals with random genotypes
2. The genotype of each individual is fed through the decoder to generate music
3. The user listens to individual songs and removes any that they do not like
4. New members of the population are created by applying Algorithm 1 to the remaining songs
5. Repeat the evolutionary process from step number 2

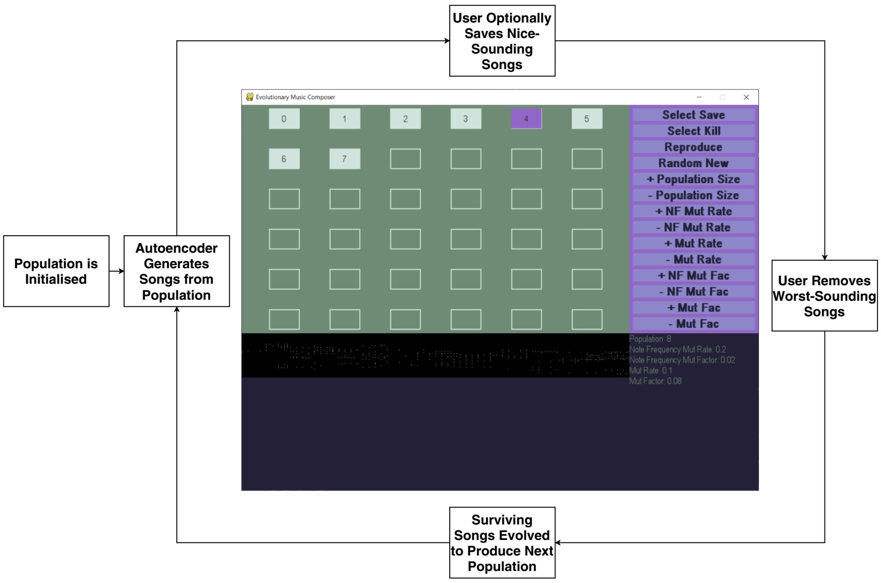
This process completes when the user is satisfied with the evolved population.

```
Algorithm 1: Evolutionary Process
Default Parameters: populationSize ← 8, nfMutChance ← 0.2,
nfMutFac ← 0.02, mutChance ← 0.1, mutFac ← 0.08

Input  : A list, $S$, of songs (each represented by an array of 181 values)
         selected by the user to evolve with size greater than 1
Output: A list of songs of size equal to the population size formed by
         adding newly evolved songs to $S$
1  newSongs ← $S$;
2  while newSongs.length < populationSize do
       // Get unique pair of songs to evolve, pairs not yet
          chosen will be selected where possible
3      song1, song2 ← getUniquePairOfSongs($S$);
4      crossover ← 181-dimensional array, each position filled with
        Bernoulli(0.5);
5      newSong ← song1 * crossover + song2 * (1 - crossover);
6      mutations ← 181-dimensional array, each position filled with
        Bernoulli(mutChance) * $\mathcal{N}$ (0, mutFac);
7      mutations[0] ← Bernoulli(nfMutChance) * $\mathcal{N}$(0, nfMutFac);
8      newSong ← newSong + mutations;
9      newSongs.append(newSong);
10 end
11 return newSongs
```

3.4 The Application

A significant constraint with initial prototypes of our method was how time-consuming it is for humans to judge songs manually. As such, it was decided to create an application to help streamline this process and increase usability.


User Optionally Saves Nice-Sounding Songs
Population is Initialised
Autoencoder Generates Songs from Population
User Removes Worst-Sounding Songs
Surviving Songs Evolved to Produce Next Population
Evolutionary Music Composer
0
1
2
3
4
5
6
7
Select Save
Select Kill
Reproduce
Random New
+ Population Size
- Population Size
+ NF Mut Rate
- NF Mut Rate
+ Mut Rate
- Mut Rate
+ NF Mut Fac
- NF Mut Fac
+ Mut Fac
- Mut Fac

Population: 8
Note Frequency Mut Rate: 0.2
Note Frequency Mut Factor: 0.02
Mut Rate: 0.1
Mut Factor: 0.08

Fig. 9. The evolutionary music composition application shown inside a flowchart of its core operations.

The application, seen in Fig. 9, is divided into four main sections. The upper left area is used to display the population of songs generated by the algorithm (each number represents the ID of a song). A user can select a honeydew green box to hear the respective song played. At the same time, it is displayed in piano roll notation inside the black rectangle on the screen. The upper right purple area displays the controls of the application. The first four of which are the most important. Once the user evolves a song they enjoy, they can use the "Select Save" option to save the song. For each generation, the user can remove songs that they do not enjoy with the "Select Kill" button. After this is done, they can click the "Reproduce" control to generate new songs from those that survived. In the event that the user enjoys none of the songs, or just wishes for something new, the "Random New" button can be pressed. This introduces a new random song to the population.

The remaining ten controls either increment or decrement the parameters used for evolution. The parameters are displayed in the bottom right zone of the screen. "± Population Size" modifies the size of the population—the number of songs in each generation. The next four boxes modify two different types of mutation rates for evolution. "NF Mut Rate", short for note frequency mutation rate, modifies how likely the note confidence threshold is to change. The mutation is performed by adding Gaussian noise to the value with standard deviation equal to the "Note Frequency Mut Factor" (which is changed with the "$\pm$ NF Mut Fac" buttons). The "Mut Rate" button works similarly, instead represented how likely each principal component is to mutate. This is again done by adding Gaussian noise to the values with standard deviation equal to the "Mut Factor". The note frequency and principle component mutations were kept independent because the equivalent change in the mutation of the principal components would have a drastic effect on the note frequency, creating poor songs. The intention is for the user to modify the mutation values if they feel generated songs are either too similar or too different from previous songs.

The initialisation of the population of songs is done by, for the latent space genotype part, drawing randomly from a standard normal distribution, and for the note frequency part, drawing from a uniform distribution between 0.2 and 0.4. From here, the user simply interacts with the application in the above-stated manner to create music they find appealing. The application, along with pre-generated samples is available online[1]. Through personal experimentation, the reproduction phase appears to create diverse new songs with each still sounding inspired by its parents.

3.5 Experiments

An experiment was conducted to verify that the application can create a variety of music with respect to tonality. The application was run several times, with selections made by the first author, to compose a dataset of music containing 49 songs. For each of these songs, a pitch class histogram was created [24]. Pitch-class histograms measure the frequency of types of notes. In standard music, there are 12 unique notes, and these are repeated through octaves. These 12 unique notes form our pitch-classes.

The distribution of the pitch-classes for four songs in the dataset is shown in Fig. 10. These were selected to demonstrate that a variety of pitch-class distributions is possible between separate runs. As can be seen, the distribution of pitches is quite different between the songs. Consider songs 1 and 2. Even though pitch-class 4 is the most common, the second most common for each seldom appears in the other (pitch-classes 9 and 12). This is among other differences, such as the presence of pitch-class 5. Additionally, 2 is the most common pitch-class in song 4 but is rare in all of the others. This essentially means that the four excerpts from the dataset have quantitatively different tonalities and keys. This implies that the application can create a variety of music tonality-wise.

[1] Source code for the music composition application: http://doi.org/10.5281/zenodo.4497829.

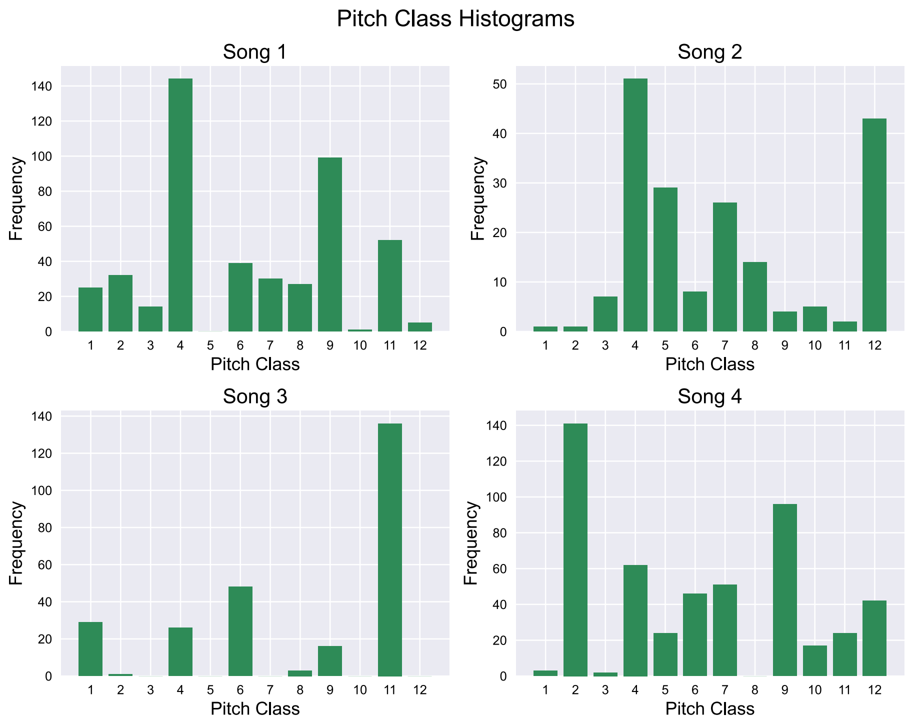


Fig. 10. The pitch-class histograms of four generated songs.

Another experiment was performed to establish that the application can also produce a range of rhythms. To quantify this, the average inter-onset-interval (IOI) [24] was calculated over the same dataset. The average IOI measures the mean time in seconds between two consecutive notes. All generated music is played at the default MIDI tempo of 120 bpm. The IOI can be used as a measure of how fast the rhythms are moving inside each song.

Figure 11 shows a histogram created to measure the distribution of the average IOI between the songs. From the numeric results, the IOI varies from less than 0.04 to more than 0.16 s in our dataset. This effectively means that the average time between notes of the slowest moving piece is more than four times that of the fastest. It is noted that there is a skewing in the histogram towards faster moving pieces. This was likely caused by the first author preferring faster-moving compositions to the slower moving ones when evolving the music. Regardless, the histogram shows that the application can indeed generate a range of rhythmic compositions.

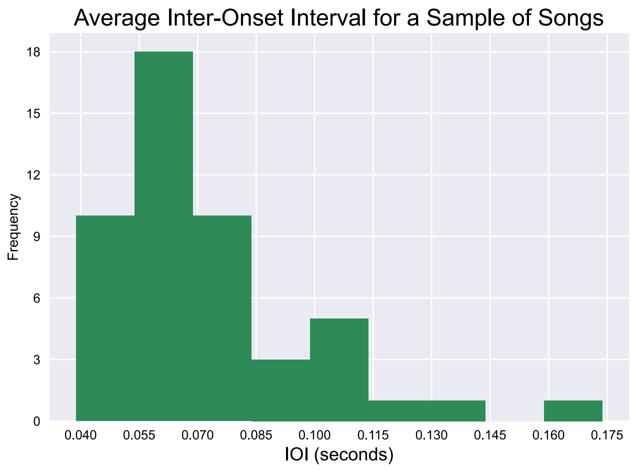


Fig. 11. The distribution of the average inter-onset-interval for the generated songs.

4 Discussion and Future Work

The application successfully allows users to create music through evolution. The novel underlying method was found to have worked, generating quantitatively diverse music between different runs. The autoencoder component successfully finds a lower-dimensional structure of music for the evolution part to run, albeit with simplified data input.

One clear limitation of the system is the simplification of the data to ignore duration so that notes are instantaneously played and released. In practice, this leads to situations where nothing is played for a long time as long notes in the training set translate into long rests. This can sometimes lead to occurrences of empty bars in the output (though usually only early on in the evolutionary process). Applying an autoencoder that can represent duration in our system would likely improve the quality, continuity and diversity of the composed music.

While our evaluation has examined the autoencoder's latent space and characterised the musical output of our system quantitatively, we have not undertaken a human-centred evaluation. Future user studies could examine both the quality of the music generated as well as the affordances of our evolution interface.

Although our system streamlines the process of human evaluation, a potential extension could be to include a discriminator neural network to supplement these decisions. This could allow the system to automatically evolve towards a target feature such as mood or genre. Multiple autoencoders could also allow a user to change the style. These features could be used for dynamic music generation as is used in video game soundtracks.

5 Conclusion

This paper presented an interactive application which generates music to a user's liking inspired by the process of biological evolution. The application uses a novel combination of autoencoder neural networks and evolutionary algorithms driven by explicit human feedback. The autoencoder was successfully used to capture musical structure from pre-existing music and represent this at a much lower dimension. Principal component analysis was then used to normalise the latent space and find a meaningful encoding of features. This provided a genotype which, for the purposes of evolution, could easily be edited to gradually change the music which is generated based off of other songs. The results show that the application is capable of creating a diverse range of music with regards to tonality and rhythm.

References

1. Biles, J.A., Miranda, E.R.: Evolutionary Computation for Musical Tasks, pp. 28–51. Springer, London, London (2007). https://doi.org/10.1007/978-1-84628-600-1_2
2. Bontrager, P., Roy, A., Togelius, J., Memon, N., Ross, A.: Deepmasterprints: generating masterprints for dictionary attacks via latent variable evolution*. In: 2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS), pp. 1–9 (2018). https://doi.org/10.1109/BTAS.2018.8698539
3. Bontrager, P., Lin, W., Togelius, J., Risi, S.: Deep interactive evolution. In: Liapis, A., Romero Cardalda, J.J., Ekárt, A. (eds.) EvoMUSART 2018. LNCS, vol. 10783, pp. 267–282. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77583-8_18
4. Choi, K., Hawthorne, C., Simon, I., Dinculescu, M., Engel, J.H.: Encoding musical style with transformer autoencoders. In: Proceedings of the 37th International Conference on Machine Learning, ICML. Proceedings of Machine Learning Research, vol. 119, pp. 1899–1908. PMLR (2020). http://proceedings.mlr.press/v119/choi20b.html
5. Farzaneh, M., Mahdian Toroghi, R.: Music generation using an interactive evolutionary algorithm. In: Djeddi, C., Jamil, A., Siddiqi, I. (eds.) Pattern Recogn. Artif. Intell., pp. 207–217. Springer International Publishing, Cham (2020)
6. Foster, D.: Generative Deep Learning: Teaching Machines to Paint. Compose, Write and Play. O'Reilly Media, Newton (2019)
7. HackerPoet: Composer (2018). https://github.com/HackerPoet/Composer
8. Hawthorne, C., et al.: Enabling factorized piano music modeling and generation with the MAESTRO dataset. In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=r1lYRjC9F7
9. Hoover, A.K., Szerlip, P.A., Norton, M.E., Brindle, T.A., Merritt, Z., Stanley, K.O.: Generating a complete multipart musical composition from a single monophonic melody with functional scaffolding. In: Proceedings of the Third International Conference on Computational Creativity (ICCC-2012), pp. 111–118 (2012)
10. Huang, C.Z.A., et al.: Music transformer: generating music with long-term structure. In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=rJe4ShAcF7

11. Huang, Y.S., Yang, Y.H.: Pop music transformer: beat-based modeling and generation of expressive pop piano compositions. In: Proceedings of the 28th ACM International Conference on Multimedia, pp. 1180–1188. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3394171.3413671
12. Husbands, P., Copley, P., Eldridge, A., Mandelis, J., Miranda, E.R., Biles, J.A.: An Introduction to Evolutionary Computing for Musicians, pp. 1–21. Springer, London (2007). https://doi.org/10.1007/978-1-84628-600-1_1
13. Kaliakatsos–Papakostas, M.A., Floros, A., Vrahatis, M.N.: evoDrummer: deriving rhythmic patterns through interactive genetic algorithms. In: Machado, P., McDermott, J., Carballal, A. (eds.) EvoMUSART 2013. LNCS, vol. 7834, pp. 25–36. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36955-1_3
14. Kaliakatsos-Papakostas, M.A., Floros, A., Vrahatis, M.N.: Interactive music composition driven by feature evolution. SpringerPlus **5**(1), 1–38 (2016). https://doi.org/10.1186/s40064-016-2398-8
15. Newman, M., Evans, S., Carroll, M., Hill, J., Camarena, D.: Vgmusic (2020). https://vgmusic.com/
16. Roberts, A., Engel, J., Raffel, C., Hawthorne, C., Eck, D.: A hierarchical latent vector model for learning long-term structure in music. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning Proceedings of Machine Learning Research, pp. 4364–4373. PMLR Stockholmsmässan, Stockholm, Sweden (2018). http://proceedings.mlr.press/v80/roberts18a.html
17. Rogozinsky, G., Shchekochikhin, A.: On VAE latent space vectors distributed evolution driven music generation. In: Proceedings of the 11th Majorov International Conference on Software Engineering and Computer Systems. MICSECS (2019). http://ceur-ws.org/Vol-2590/paper22.pdf
18. Romero, J.J.: The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music. Springer Science & Business Media, Berlin (2008)
19. Scirea, M., Togelius, J., Eklund, P., Risi, S.: MetaCompose: a compositional evolutionary music composer. In: Johnson, C., Ciesielski, V., Correia, J., Machado, P. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design, pp. 202–217. Springer International Publishing, Cham (2016)
20. Secretan, J., Beato, N., D Ambrosio, D.B., Rodriguez, A., Campbell, A., Stanley, K.O.: Picbreeder: evolving pictures collaboratively online. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1759–1768. CHI 2008, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1357054.1357328
21. Waschka II, R., Miranda, E.R., Biles, J.A.: Composing with Genetic Algorithms: GenDash. Springer, London (2007). https://doi.org/10.1007/978-1-84628-600-1_6
22. Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemom. Intell. Lab. Syst. **2**(1), 37–52 (1987). https://doi.org/10.1016/0169-7439(87)80084-9
23. Yamshchikov, I.P., Tikhonov, A.: Music generation with variational recurrent autoencoder supported by history. SN Appl. Sci. **2**(12), 1–7 (2020). https://doi.org/10.1007/s42452-020-03715-w
24. Yang, L.-C., Lerch, A.: On the evaluation of generative models in music. Neural Comput. Appl. **32**(9), 4773–4784 (2018). https://doi.org/10.1007/s00521-018-3849-7