Training Sequence Generative Adversarial Nets to Compose Music in the abc-notation.

Bachelor project by O.F. Jansen
Student Bachelor Artificial Intelligence
Universiteit Utrecht, The Netherlands
Email: O.F.Jansen@students.uu.nl

Supervised by dr. A.J. Feelders
Department of Information and Computing Sciences,
Universiteit Utrecht, The Netherlands
Email: A.J.Feelders@uu.nl

Creating new music is traditionally seen as a task performed only by humans. Getting computers to compose music would have advantages such as the ability to create more music in the same amount of time. With recent developments in the field of machine learning, music composition by computers seems closer than ever. In this study we have applied the Sequence Generative Adversarial Nets technique as proposed by Yu et al. to the task of generating tunes in the abc notation, a text-based music notation system. The training set we have used was built by Sturm et al. and consists of 23,636 abc songs. We succeeded in training a model that generates valid abc files. These songs were empirically evaluated with the help of 47 participants. No significant difference between the ratings of real tunes and generated tunes was found. However, we think it is possible that even the real tunes were not well-received by the participants which would make this conclusion less valuable.

1 Introduction

Composing music is very much a creative process of the human mind. As such, we do not yet fully understand how it works and what factors play a role in it, nor can we algorithmically mimic it. It would therefore be quite impressive if we could learn a computer in an unsupervised manner how to generate music that sounds good.

This research focusses on the use of generative adversarial networks (GANs) [11], which are systems of two neural networks that play a minimax game against each other in order to eventually end up with a very good generative model. GANs have been successfully applied to (text to) image generation [11] [16]. They tend to generate sharper images than other generative models [8].

More specifically, this research makes use of a technique that enables the application of the generative adversarial networks framework to sequential data. This technique has recently been developed by Yu et al. in their paper 'SeqGAN' from 2017 [22].

In this research, SeqGANs are trained on a dataset containing songs in a high-level, text-based music notation system called the *abc notation*. The *abc notation* is mainly being used by traditional and folk artists [21]. Files in *abc* are representations of the music to be played but do not actually describe the audio waveforms; they consist of a chronological sequence of notes. This makes it a great and lightweight source of data to train the model on. There are tools available to convert music in the abc notation to a listenable waveform format such as MP3. This process can be seen as

playing the notes described in the abc transcription on a virtual instrument.

The generated music will be evaluated by shuffling real and generated tunes and asking people to rate each tune using an online questionnaire.

1.1 Related work

At the time of writing this paper, there is only one publication that mentions generating music using SeqGANs to be found. Not only do Yu et al. introduce the SeqGAN technique on which the model in this work is based, they also apply it to music generation [22]. They used the Nottingham dataset [7] consisting of 695 folk songs in the MIDI format and modified the songs to focus only on the solo track. BLEU ('bilingual evaluation understudy', normally used for measuring the quality of machine-translated text [14]) was used to evaluate the quality of the piano key patterns and the mean squared error on continuous pitch patterns. The authors report that the SeqGAN model outperforms maximum likelihood estimation significantly on both these metrics. Unfortunately they do not go into detail on how and why these metrics are used for evaluating music.

Sturm et al. use long short-term memory networks to generate Celtic folk songs in the abc notation in their 2016 paper [19]. They have built a large set of training data in the abc notation which they have open sourced and describe the process of building it in their paper. Descriptive statistics such as the number of tokens in the tunes and the occurrences of certain notes are used as an evaluation metric. The authors themselves deem the relevance of this evaluation to the experience of music by the listener highly questionable, and I agree. After this analysis the authors also evaluate just one generated tune in terms of technical musical terms. They do not provide another, more objective evaluation of the quality of the generated songs.

In a paper by Agarwala, Inoue and Sly, the authors try to generate music in the abc notation using various methods such as recurrent neural networks, Seq2Seq and GANs [3]. They also describe the process of building a large dataset of abc tunes for training purposes. The authors created a survey containing fragments of human-composed songs and songs generated by the discussed models and asked the respondents if they thought the songs were human-composed or computer-generated. However, the authors did not succeed in training the GAN model because it was too unstable and they thus did not include any songs generated by it in the survey.

The current research differs from the above. Speaking in terms of the mentioned papers, this research combines the model of Yu et al., SeqGANs [22], with the dataset by Sturm et al. [19] and an empirical evaluation metric comparable with the one used by Agarwala, Inoue and Sly [3].

1.2 Research question

The research question that guided this study and that will be answered at the end of this report is the following:

Can SeqGANs be trained to compose music in the text-based notation system 'abc' that sounds as good as human-composed music?

2 Background

Before expanding on the current research, the following subsections provide some background on generative adversarial nets (GANs), sequence GANs and the abc notation.

2.1 Generative Adversarial Nets

Generative Adverserials Nets were introduced by Goodfellow et al. in 2014 [11]. The idea builds upon the general concept of generative models (models that output new data samples from a distribution that is learned from the training dataset) and the success of deep discriminative models. Instead of training just a generative model, Goodfellow et al. have trained one generative model and one discriminative model at the same time. The objective of this discriminative model is to guess whether a presented data sample is real, or has been generated by the generative model. The objective of the generative model is to generate a sample and maximise the probability that the discriminative model makes a mistake when presented with this sample (in other words: to generate data that the discriminative model thinks is not generated).

We can formalise this a bit. Let $D(x)$ be the probability that the discriminative network predicts sample x is a real sample (i.e. a sample from the dataset) and let $G(z)$ be the data sample generated by the generative model given the random variable z, usually sampled from a Gaussian or Normal distribution. z usually has the same or a lower dimensionality than the training data. The generative model thus maps the latent variable z to a sample from a distribution that will approach the distribution of the dataset after training. Because z is a random variable, the generative model produces a wide variety of data samples given different values of z. Each value of z is a unique seed The objective of the discriminative network is to maximise the probability that it assigns the correct label (1 for a real sample, 0 for a fake sample) to presented samples. The objective of the generative model is to minimise $\log(1 - D(G(z)))$ (the logarithm of 1 minus the probability that the discriminator predicts a generated sample to be real).

This situation can also be interpreted as a two-player minimax game played by the generative and discriminative model. The game has the following value function $V(G,D)$ [11]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \quad (1)$$

In above equation z is the latent variable, sampled from a random noise distribution, and x is a real data sample. $\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]$ is the expected value of the logarithm of the probability that D predicts x is a real sample. $\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$ is the expected value of the logarithm of the probability that D predicts $G(z)$ is a fake sample. So the better D gets at predicting the probability that a sample is real or fake, the higher the value of $V(G,D)$ will get. In this minimax game G tries to minimise the value function and D tries to maximise it.

To algorithmically build the GAN, the generative and discriminative models are trained simultaneously by k times sampling m noise samples and m real samples and using these samples to ascend the discriminator's gradient. Goodfellow et al. used $k = 1$ in their experiments. The generative model then generates m samples and these are used to descend its gradient.

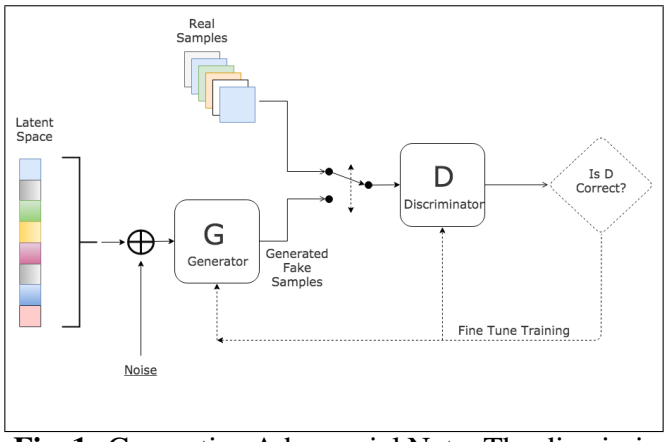

Fig. 1: Generative Adversarial Nets. The discriminator is trained on real samples and generated fake samples. The loss of the discriminator ("Is D Correct?") is used to update both the discriminator and the generator. This figure is made by Al Gharakhanian [9].

In the original paper, a great analogy is made. You can compare the generative model with a team of counterfeiters, trying to produce and use fake currency. The discriminative model would be the police, trying to stop the counterfeiters. The competition between the counterfeiters and the police causes both to improve their methods until eventually the fake money cannot be distinguished from real money by the police. The same goes for the models: the generative model is bound to eventually generate samples that cannot be distinguished from real data samples by the discriminative model.

The adversarial framework has some disadvantages, e.g. the training phase can prove to be tricky because the generator and discriminator must be kept in sync with each other (one model must not significantly outperform the other). This can be done by simply tweaking the hyper parameters but this does not seem to be a very transparent process. The framework also has advantages, for example the generator does not get to 'see' the input data directly but only through gradient updates coming from the discriminator. This prevents parts of the training data to be adopted directly by the gener-

ative model and thus can prevent the model from overfitting.

2.2 Sequence Generative Adversarial Nets

Generative adversarial nets, as described by Goodfellow et al. [11], try to minimise the loss function of the generative model by using stochastic gradient descent and try to maximise the value function of the discriminative model by using gradient ascent (these loss and value functions are the same function, as given in equation 1). The output of the discriminative model on a generated sample (that is: the assigned probability that the sample is real) is being used to guide the generative model in changing its weights a bit so that the next generated sample will be more likely to fool the discriminative model. According to Yu et al. (and also according to Ian Goodfellow, author of the GAN paper, in a Reddit comment [10]) this 'slight change' does not make any sense when working in the domain of sequential data. For example: imagine the oversimplified case in which the discriminator considers only samples containing just the letter 'B' to be real (and samples containing another letter to be fake) and the generative model only generates just the letter 'A'. The output of the discriminator on generated sample 'A' will be about 0. This would lead to an update to the generative model in such a way that it will generate a letter that is a bit more like 'B'. However, this is not possible since there is no such letter in between 'A' and 'B', there is only 'A' or 'B'.

To solve this problem, the authors approach the generative model as an 'agent of reinforcement learning'. Reinforcement learning is a general framework with contributions from research into optimal control problems [5] and animal learning by trial and error [17]. Learning your dog to sit on command by letting him perform random actions and (not) rewarding these actions, instead of showing him examples of how to properly sit, can be seen as an instance of reinforcement learning. In the proposed SeqGAN system, our generative model keeps track of the current state (the sequence of tokens generated thus far) and has a parametrised policy that it uses to decide the next action (the next token to add to the sequence based on the current state) in order to maximise the reward (the probability that the discriminator makes a wrong classification). We use gradient descent to optimise this policy [20]. Because the discriminator can only reward finished sequences, Monte Carlo search with a roll-out policy is used to make up the rest of an unfinished sequence so that scores can be given at intermediate steps during sequence generation. Note that these scores are not always accurate as part of the scored sequence was made up by the roll-out policy. g, d and k are hyper parameters, respectively indicating 1) how many times the generator model should be updated each cycle, 2) how many times a training set with newly trained and real samples should be created and used for the discriminator model to train on each cycle and 3) for how many epochs or full cycles on the created training set the discriminator should be trained.

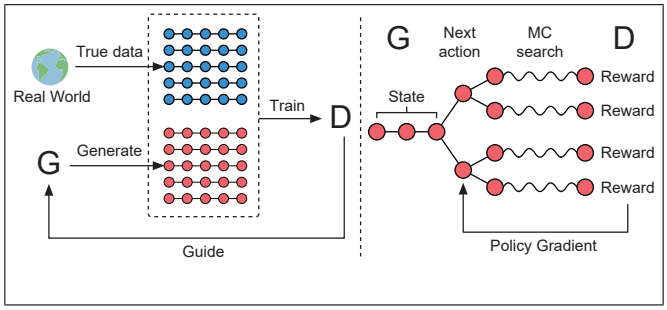

Fig. 2: Sequence Generative Adversarial Nets. The discriminator is still trained on real and generated samples and its loss guides the generator. Inside the generative model, Monte Carlo search is used to roll-out a partial sequence so that the discriminator can judge a sequence and this output can be used as the value for a certain action from the policy. This figure is taken from Yu et al. [22].

Yu et al. have proposed a recurrent neural network as the generative model and a convolutional neural network as the discriminative model. These networks both get pre-trained using maximum likelihood estimation.

Returning to the present study: if we define a tune as a sequence of notes (which are discrete tokens), this solution by Yu et al. to the discrete-token-problem of GANs provides us with a way to train a model to generate music in an adversarial manner.

2.3 The abc music notation system

The abc notation is a text-based music notation system. This means that only ASCII-characters are being used to decode tunes and that it's easy for people to encode their own tunes.

According to abcnotation.com [21] the abc notation is "the de facto standard for folk and traditional music". This is backed up by the numerous websites that distribute free tunes in abc, such as The Session [2], the abc version of the Nottingham Music Database [4], or the tune search on the abc homepage which at the time of writing contains around 560,000 tunes [21].

Every valid abc file starts with a header containing key-value pairs joined by a colon (also called the 'fields' of a tune) describing the song. Not all software for handling abc files demands the same fields to be defined but the K-field, describing the key of a song and marking the end of the header, must always be present. Other common fields are the M-field with which the meter is set, the T and C-fields with which the title and composer of a song are set and the X-field which contains a reference number which is mostly legacy but is also being used by software to tell where one song ends and another song starts.

After the header, the notes section starts. This is a high-level, chronological representation of the song. Notes are notated as one of the letters *CDEFGAB* for the corresponding note. Notes one octave up are notated using these same letters in lowercase. Notes higher than this are denoted by adding apostrophes to the letters. By adding commas to the uppercase letters, we extend the scale an octave below. After a note, a fraction can be added like this: $C/2$. This will result in a C that's half the default note length, which is set in the headers. Other fractions are also possible and result in different relative note lengths. Tunes are segmented into bars by |-signs between the notes.

Examples of abc songs and accompanying explanation are available at the official website [21].

3 Method

3.1 Implementation

Although the authors of the SeqGAN paper have published a Tensorflow implementation on their Github page [1], this study makes use of another implementation [6] because Yu et al. only published the implementation for their synthetic data experiments and the other implementation was found to be better documented and more readable.

Here is a list of all files included in the implementation and a summary of each of them:

model.py
Contains the SeqGAN class which implements most of the ideas of Yu et al. [22]. The class has methods for building the generative model, the discriminative model and the optimizer functions. *train_batch* is a method that trains the system on a single batch and *generate* can be called to generate a sample from the trained model. It also has a *build* method that ties previous mentioned methods together and a *save* method for saving and loading a trained model.

sample.py and **train.py**
Has CLI wrappers for the *generate* and *train_batch* methods of the SeqGAN class in model.py. train.py repeats the batch training for the number of steps times the number of epochs as defined by the user.

utils.py
Defines functions to map the tokens in the data (character in the abc files, in this case) to integers and back again and to store these mappings to a dictionary.

The implementation requires Tensorflow version 1.0.1 to be installed, otherwise it won't run. If you want to train the model or sample from it using a machine without a GPU, you can pass the $-c$ flag to train.py or sample.py.

3.2 Dataset

The dataset of Sturm et al. [19] is of great quality. It contains 23,636 transcriptions of folk songs originating from *thesession.org*. Three variants of the dataset that differ in the amount of cleaning they've gone through have been published.

sessions_data_clean: The raw abc tunes from *thesession.org*. Cleaned to contain only valid

abc and no HTML or comments from the website.

allabcwrepeats_parsed: The same set as above but now without short tunes (fewer than 7 measures), without tunes that have more than one key or meter and with all songs transposed to a key with root C. More details about this cleaning process are described in the original paper.

allabcworepeats_parsed: The same set as the previous set but all songs have now been converted to midi and back to abc. This makes all repeats (which can be denoted with : in abc) explicit.

The second set, *allabcwrepeats_parsed*, is used to train the model discussed in this report. This set was chosen over the first set because of the more thorough cleaning process it went through. The third set wasn't used because I hypothesised that converting the songs to midi and back to abc could lead to some loss of patterns present in the original tune transcription. Implicit repeats also make the tunes more compact which could be an advantage for the model since it gets trained on samples of a fixed length. A more dense representation could result in better recognisable patterns in short transcriptions. Some software requires tunes to have an identifier field so *X* header fields with value 1 have been added to the beginning of all songs to make them valid in all abc-software.

After multiple attempts to train the model on the entire dataset, it was decided to not use the entire dataset because of time concerns. The available hardware wasn't powerful enough for the model to converge in reasonable time. Training on the entire dataset takes extremely long (in the order of weeks) on the used system. This issue could be resolved by training the model on a system with more computational power provided by e.g. a dedicated graphics card.

The dataset used contains the first 1666 songs of the original *allabcwrepeats_parsed* dataset (or: the first 9995 lines). The first and last tunes in this set are displayed in figures 3 and 4.

```
X:1
T:'G Iomain Nan Gamhna
M:9/8
K:Cmaj
G E E E 2 D E D C | G E E E F G A B c |
G E E E 2 D E D C | A D D G E C D 2 A |
G E E E 2 D E D C | G E E E F G A B c |
G E E E 2 D E D C | A D D G E C D 2 D |
E D E c 2 A B A G |E D E A /2 B /2 c A B 2 D |
E D E c 2 A B A G | A D D D E G A 2 D |
E D E c 2 A B A G | E D E A /2 B /2 c A B 2 B |
G A B c B A B A G | A D D D E G A B c |
```

Fig. 3: The first tune in the dataset used.

```
X:1
T:The Beauty Spot
M:4/4
K:Cmix
c A | (3 G B G F A (3 G B G c A | (3 G B G F E D
E F 2 | (3 G B G F A (3 G B G c A | B G F D E C
C A | (3 G B G F A (3 G B G c A | (3 G B G F E D
E F 2 | (3 G B G F A (3 G B G c A | B G F D E C
C e | d B B 2 d B c e | d c B c d e f e | d c B c d c B
G | F 2 (3 A B c d B c e | d B B 2 d B c e | d c B c
d e f 2 | g e (3 f e d c d B G | F 2 (3 A B c d B c A |
```

Fig. 4: The last tune in the dataset used.

4 Results

The songs produced by different trained models were of varying quality. Training with slightly different hyper parameters (g, d and k in the paper by Yu et al. [22]), e.g. training the discriminator 3 times for every time the generator gets trained, sometimes led to completely useless generated samples. An example of such a sample is displayed in figure 5. This tune is of very low quality because it just repeats the same note over and over. Even the time each note should be played, does not vary.

Using the default values for the parameters for the system ($d = g = 1$ and $k = 100$) experimentally proved to be the best option. The system ran about 50 cycles before the samples that were used

| c 2 c 2 c 2 | 2 | 2 | 2 | c 2 c 2 c 2 c 2 c 2 | c 2 c 2 c 2 c 2 | c 2 c 2 c 2 c 2 c 2 c 2 | c 2 c 2 c 2 c 2 |

Fig. 5: Part of a very low quality generated sample.

to evaluate the model were generated. This model generated samples that were in valid abc notation and with a seemingly healthy balance between repetition and variation. Most samples did raise some warnings with the abc parser (some double numbers for indicating note length for example) but the parser software is able to solve these issues by itself. All generated samples that were used in the evaluation section can be found in the appendix.

5 Evaluation

To evaluate the quality of the generated samples, it was decided to take an empirical approach and create an online questionnaire. The following subsection goes into detail on the experimental design and the subsection after that analyses the results of the questionnaire.

5.1 Experimental design

The questionnaire was built using Google Forms because this service allows for music to be embedded in the list (in the form of Youtube videos), it is a responsive service (i.e. it works on mobile devices, this stimulates people to participate) and it allows for total control over the pages, questions and general markup. The survey was distributed among family and friends, mainly using social media. The online questionnaire featured 15 tunes. Ten of these tunes were generated by the model and five were real, human-composed tunes randomly selected from a part of the dataset by Sturm et al. that wasn't used for training the model. Of the ten generated tunes, five were randomly selected (generated and immediately added to the survey) and five were handpicked by me out of a set of 50 generated samples, based on how good I found these songs to sound. These three classes were labeled 'random', 'selected' and 'real'. The order of the tunes is the same for every participant but was randomised before publishing the questionnaire. In the analysis each tune is also labeled with a number that indicates its position in the questionnaire. After a tune has finished playing, the participants are asked to give it a rating on a scale from one ("very bad") to five ("very good"). The accompanying label for this rating was "How do you like this tune?". The results of the survey were exported out of Google Forms and analysed using R [15].

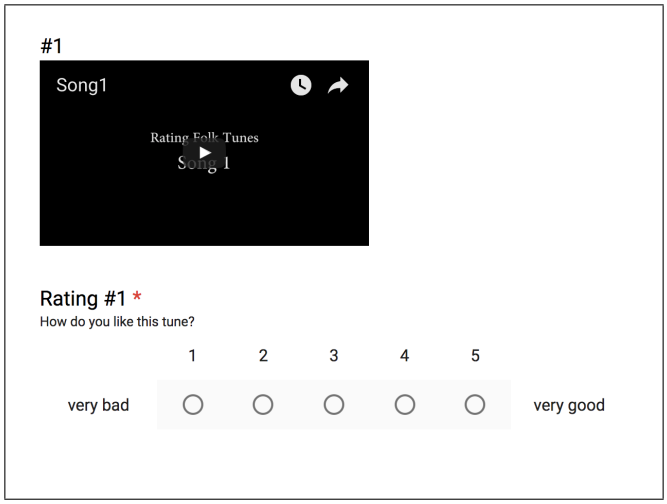

Fig. 6: A screenshot of the questionnaire. All 15 tunes were displayed on a single page.

The results of this experiment will demonstrate 1) how generated tunes are perceived compared to real tunes and 2) if a human selection of generated tunes can boost the quality. The expectation is 1) that generated tunes will be rated lower than real tunes and 2) that human selection can boost the quality.

5.2 Analysis

The questionnaire was completed by 47 people.

As visible in figure 7, it is not immediately clear that songs from the different classes differ in their received rating. Because of the ordinal nature of the data we cannot simply compare the mean ratings of the different classes to draw a conclusion. The distance between two items on an ordinal scale is unknown so the mean would be meaningless. We could have a look at the median of all ratings, which turns out to be 3 for all tunes expect for tune #1 (a random generated one) which has a median rating of 2 and tunes #7 and #10 (real tunes) which have median ratings of 4. This is as

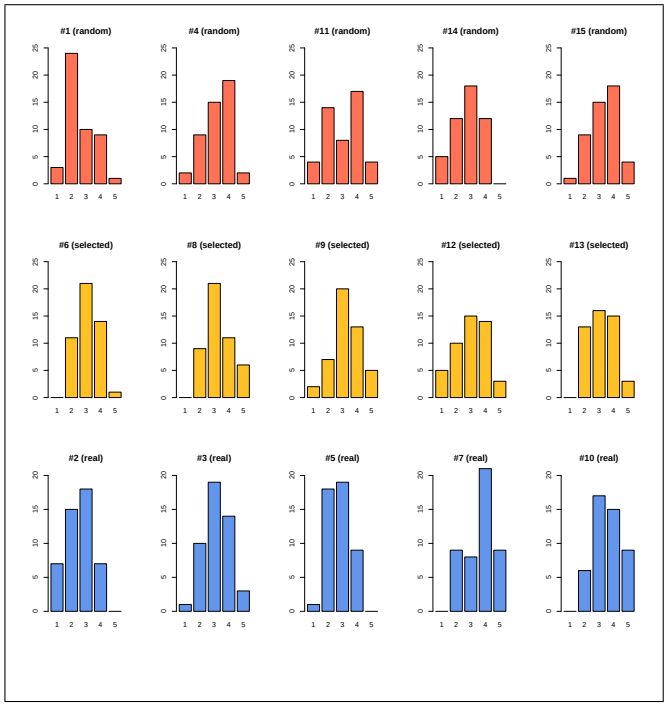

Fig. 7: The distribution of ratings for all featured tunes. The rows contain plots for the tunes that were generated (plots in red, labeled 'random'), the tunes that were selected from a set of 50 generated tunes (plots in yellow, labeled 'selected') and the real tunes respectively (plots in blue, labeled 'real'). All possible ratings are on the x-axis and the height of the bars depicts the cumulative number of times a tune was given this rating.

expected: the real tunes are rated higher than the selected tunes, which are in turn rated higher than the generated tunes. The difference, however, is quite small.

To proceed and test whether the ratings from the different classes actually come from the same distribution (which would imply that there is no significant difference in rating between the songs), the Kruskal-Wallis test [12] was used. The Kruskal-Wallis test is the non-parametric equivalent of the one-way analysis of variance, which means that it does not assume the data is normally distributed and can be performed on ordinal data. The null hypothesis of this test assumes that there is no stochastic dominance between the groups. This would mean that there is no difference between the groups in terms of their received ratings. Variable A is said to stochastically dominate B if $P(A < x) \leq P(B < x)$ for all possible values x of A and B. The alternative hypothesis is that for at least one group i stochastic dominance exists. That would mean that a tune from this group i is less likely to receive a lower rating than a tune from another group.

To compute the test statistic of the Kruskal-Wallis test, let N be the total number of samples (47×15 in our experiment), k be the number of classes or groups (in this case there are three groups) and n_i the sample size of group i (for every group $\frac{N}{k}$ in this case). Now add all samples to a list, sort it and give each item a rank. Tied items receive the average of the ranks they would have received if they were not tied. Let r_{ij} be this rank of sample j from group i. Let $\bar{r}_i$ be the average rank of all samples in group i and $\bar{r}$ the average of all ranks. The test statistic can now be calculated using equation 2, which is a measure for the difference between the groups. The distribution of the test statistic approximates the chi-square distribution, which is used to calculate the p-value.

$$H = (N-1)\frac{\sum_{i=1}^{k} n_i(\bar{r}_i - \bar{r})^2}{\sum_{i=1}^{k}\sum_{j=1}^{n_i}(r_{ij} - \bar{r})^2} \qquad (2)$$

Performing this test on the data using R, led to a test statistic of 2.969 and a corresponding p-value of 0.227. This indicates that, although the observations are somewhat off, the three classes do not differ significantly.

Assuming there really is no difference in distribution between the ratings of the different classes we proceed. Say $random > real$ means the median of the ratings a participant gives to the random tunes is higher than the median of the ratings this same participant gives to the real tunes. Under our assumption, there should be an equal number of participants for which $random > real$ as participants for which $random < real$. In the experiment there were 13 participants who rated the random tunes better ($random > real$) and 15 participants who rated the real tunes better ($random < real$). For 19 participants the median rating was the same for these groups ($random = real$). Using the binomial distribution $B(N = 28, P = \frac{1}{2})$ we calculate the probability of observing this outcome under the null-hypothesis that $P(random >$

$real) = P(random < real) = 0.5$. The p-value of this binomial test is $P(x \leq 13) + P(x \geq 15)$, given $x \sim B(N = 28, P = \frac{1}{2}) = 0.851$. This is far from significant so the null-hypothesis is not rejected. Analysing the difference between the real and selected tunes using this procedure and performing a binomial test lead to a similar p-value of 0.845 which is also far from significant.

This analysis points towards the conclusion that the difference between the real tunes, the random generated tunes and the selected generated tunes as visible in the gathered data could be coincidental. We therefore cannot draw any other conclusion than that the different songs cannot be distinguished from each other based on the class.

6 Discussion

This study was guided by the question whether SeqGANs can be trained to compose music in the text-based notation system 'abc' that sounds as good as human-composed music.

First, a note must be made about the term 'compose'. The Merriam-Webster dictionary [13] defines 'to compose' as 'to create by mental or artistic labor', which doesn't exactly match what the model does. 'Generate', however, is defined as 'to define or originate (..) by the application of one or more rules or operations'. This seems to better describe our 'composer' that just chooses the most likely token based on what it has composed already. This argument can be strengthened by comparing the generative model with the man that cannot speak Chinese from the famous Chinese room argument by John Searle [18]. How well the songs generated by the SeqGAN model may sound, it still does not understand *why* it composed the songs in the way it did and can not reflect on this process. It is for these reasons that in a large part of this study these two terms have been used interchangeably instead of sticking with 'compose' as used in the original research question.

As it turns out, SeqGANs can generate files in the abc notation just fine. All tunes generated by the final model were syntacticly valid and contained all necessary header fields. This is as expected because the SeqGAN technique was developed specifically for handling sequences of discrete tokens, which is what abc tunes are made up of and the syntax rules of abc are not that strict so minor mistakes will be forgiven. The length of the sequences used in the examples from the original paper by Yu et al. mostly lay below 100 tokens. Although the number of tokens in the abc tunes the model in this study was trained on was sometimes almost 10 times larger, this did not seem to be a problem. It would still be a good idea to study how the length of the sequences influences the behaviour of the model.

Determining the quality of generated sequences is a hard problem. It would be great to be able to do this evaluation automatically. The way the BLEU score is being used to evaluate machine-translations [14] inspired a search for such metrics to rate generated abc tunes. Unfortunately a useful and reliable metric was not found. The empirical approach to testing the quality of the tunes, as was used in this study, did lead to a result but a questionable one. Although a difference in ratings was found and it did match the hypothesis, it was not of significant size. A reason for this could be that the participants were not prepared for the way the tunes sound. Compared to mainstream music the tunes from the dataset sound very minimalistic and simple. The automatic conversion of the abc-file to listenable mp3 also made all notes sound very 'staccato'; there is no overlap between the notes and every note is played at the same intensity. This makes the tunes sound somewhat robotic and boring. The difference between the generated tunes and the tunes from the dataset may have gone unnoticed because of this culture shock. The sample size of the experiment could also have been too small to accurately represent the population. Another explanation for the insignificance of the difference found, could be that the generated music really is of comparable quality as the real tunes but the model has not fooled me once so I find that hard to believe. To rule out these options another experiment could be conducted. The sample size should be much larger and the participants should first get used to the 'abc sound', which could be achieved by converting a few famous songs to abc and playing these before the start of the experiment.

References

[1] Implementation of sequence generative adversarial nets with policy gradient.

[2] The session. https://thesession.org/. Accessed: 2017-06-09.

[3] Nipun Agarwala, Yuki Inoue, and Axel Sly. Music composition using recurrent neural networks.

[4] James Allwright. Abc version of the nottingham music database. http://abc.sourceforge.net/NMD/. Accessed: 2017-06-09.

[5] Richard Bellman. Dynamic programming (dp). 1957.

[6] Benjamin Bolte. SeqGAN implementation for generating text using an rnn. https://github.com/codekansas/seqgan-text-tensorflow/. Accessed: 2017-05-18.

[7] Université de Montréal. Nottingham dataset. http://www.iro.umontreal.ca/~lisa/deep/data. Accessed: 2017-05-19.

[8] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[9] Al Gharakhanian. GANs: One of the hottest topics in machine learning. https://www.linkedin.com/pulse/gans-one-hottest-topics-machine-learning-al-gharakhanian?trk=pulse_spock-articles. Accessed: 2017-06-08.

[10] Ian Goodfellow. goodfeelow_ian comments on generative adversarial networks for text. https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/cyyp0nl/. Accessed: 2017-06-11.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[12] William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[13] Merriam-Webster Online. Merriam-Webster Online Dictionary. http://www.merriam-webster.com, 2009.

[14] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[15] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[16] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 3, 2016.

[17] Robert A Rescorla, Allan R Wagner, et al. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical conditioning II: Current research and theory*, 2:64–99, 1972.

[18] John R Searle. Minds, brains, and programs. *Behavioral and brain sciences*, 3(3):417–424, 1980.

[19] Bob L Sturm, João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*, 2016.

[20] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[21] Chris Walshaw. abc notation home page. http://www,abcnotation.com. Accessed: 2017-06-06.

[22] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Appendix A: Generated samples

The first 5 generated samples are of class 'random' (that is: generated and immediatly used without subjective evaluation). The last 5 generated samples are of class 'selected' (they come from a set of 50 generated tunes and were manually picked). The order of the tunes in this section corresponds with the order of the tunes in their classes in the evaluation section.

X:1
T:The Drunk
M:4/4
K:Cmin
|: G C D E G E G | e 2 d c A G A 2 | G A G c e 2 d d | e 2 c e 2 e e 2 2 c e 2 e | e 2 c e 2 c c 2 B | A 2 B F 2 G B c d | e 2 c e 2 c B 2 e | A 2 B c 2 d B A G | E 2 | D E E G 2 | e 2 d c 2 c B 2 G | A 2 G c 2 d | B 2 c G 2 c B 2 B | A 2 G F E E F 2 :| |2 B | G B A B | c 3 G 2 |

X:1
T:Taerinhe Birlin
M:6/8
K:Cmix
f > a c > f f | e > c c > e g | f 3 f 2 :|

X:1
T:The Brue Dort The Bille
M:2/4
K:Cdor
B G C D G 2 | G G F 2 C E | F 2 E F 2 2 C 2 | C D E 2 C 2 | F 2 B, C D | E F F B, 2 A, | , 2 C D | G E C C 2 | E 2 E F D 2 | F D E 2 C 2 | F D E F C 2 | C 3 E, A, 2, | C D D B 2 | A, 3 B, C 2 | A, C D 2 | C 3 :| |: E | C E C 2 C D | E E C E 2 | C 2 E 3 | D E C 4 | C D C 4 | C 2 C 4 | C 2 C 4 | G 2 A 3 G | A B c 2 c 2 | A 3 G 2 c 2 | G 2 G 4 | G 2 A 3 G | A 3 G 2 | G 2 A 3 G | A B c 2 B 2 | A 2 G 4 | G 2 G 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 2 G 2 | A 2 G 4 | G 2 A 4 | |2 D 3 D C 2 | C 2 C 4 | C 2 E 3 D | C 2 G 2 | D 6 | G 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 2 G 2 | A 2 E 2 C 2 | C D E 2 G 2 | C 2 E 2 C 2 | C 2 E 4 | 2 | G 2 E 2 G 2 | A 2 G 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 4 | C D 2< E 2 D | C 2 G 4 | G 2 A 3 G | A 2 c 2 G 2 | A 2 G 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 4 | G 2 E 2 D 2 E 2 | C 2 C 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 2 | 2 | C 2 E 2 G 2 | A 2 G 2 | G 2 G

X:1
T:The Barshe Boss On M:2/4
K:Cmaj
|: C 2 E C C E c 2 | G 2 G 2 E 2 G 2 | F 2 E 2 C 2 G 2 | F 2 E 2 G 2 | F 2 A 2 G 2 | F 2 E 2 G 2 | G 2 A 2 G 2 | A G G 2 | 2 | G 2 E 2 G 2 | A 2 G 4

X:1
T:Batbie Fowsls M:3/4
K:Cmaj
|: A | C E G c e 3 f | g e d e c A G E | G E G c d 2 E 2 | G c e d e 2 f | g e d e c A 3 |1 G c c 2 c 2 :| |: d 2 d 2 c 2 A A | G C D 2 C 2 | C 2 E 2 D 2 | D 2 E 2 C 2 | G 2 A 2 G 2 | F 2 A 2 G 2 | A 2 G 2 E 2 | C 2 G 3 G | A B c 2 B 2 | A 2 G 4 | G 2 G 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 4 | G 2 A 4 | A G A 2 c 2 | A 2 G 4 | E D C 4

X:1
T:The May Waltz
M:3/4
K:Cmaj
G 3 A G 2 | G 2 A 2 c 2 | c 2 G 2 E 2 | F 6 | e 4 B 2 | d 2 c 2 c 2 | c 2 G 2 E 2 | F 6 E F 4 | D 2 G 2 e 2 | e 6 | e 4 B 2 | d 2 c 2 c 2 | c 2 G 2 E 2 | F 6 | F 4 E 2 | D 4 D 2 | D 2 G 2 e 2 | e 2 d 4 | d 2 d 2 d 2 | d c c 2 | d 2 c 2 | G 2 A 2 G | A B c 2 B 2 | A 3 G 2 | G 2 A 2 B 2 | c 2 B3 B c B A 2 | (3 G A G E 2 D 2 | C 6 | E 2 E 2 | D 3 D E 2 | A, 3 B, C 2 | G, 3 C B, 2 | A, 3 B, C 2 | G, 3 C B, 2 | A, 3 B, C 2 | B, 3 C D 2 | E 4 F 2 | C 3 G 3 | A G > E C 3 | D E 2 A, 3 | B, C 2 G, 3 | C B, 2 A, 3 | B, C 2 B, 3 | C D 2 E 3 | E E 2 D 3 | G, C > E G 3 | A G > E C 3 | D E 2 A, 3 | B, C 2 G, 3 | C B, 2 A, 3 | B, C 2 B, 3 | C D > E C 3 | C D 2 C 3 | G, C > E G 3 | A G > E C 3 | D E 2 A, 3 | B, C 2 G, 3 | C B, 2 A, 3 | B, C 2 B, 3 | C D 2 E 3 | E F 2 D 3 | G, C > E G 3 | A G > E C 3 | D E 2 A, 3 | B, C 2 G, 3 | C B, 2 A, 3 | B, C 2 G, 3 | C B, 2 A, 3 | B, C 2 B, 3 | C D 2 E 3

| E F 2 D 3 | G, C > E G 3 | A G > E C 3 | D E 2 A, 3 | B, C 2 G, 3 | C B, 2 A, 3 | B, C 2 B, 3 | C D 22

X:1
T:Anddien The Hor
M:6/8
K:Cdor
| G c c d e d | e d c d c d | e d c B c c | c d c d 3 :| c e g g g c | c d c g f d | f g g g c c | c c d e c c | c d c g c d | c d c g 2 d | c 3 c g 3 :| c' b g g f d | c 3 c c 3 | c b g g f d | f g g g f d | c 3 c c 3 | g c d e c c | c d c g f d | c d c g 2 a | b a g g f d | f g g g c c | c d c g f d | c

X:1
T:The Bllk OotoreenHan
M:6/8
K:Cmaj
|: g > b c' | e > c f 2 :|
X:1
T:The Brill With The Blue Dress
M:2/4
K:Cmin
G C C 2 | C 2 E C F 2 F F | G F E 2 C D | D G E D E D C 2 | C D E D D F | D C D F D C | D D C B, C D | E 2 C D 2 E | F 2 G E E D E C 2 D B, 2 G F 2 | G 3 F F 2 | D 3 D C D | D E | C 3 D C 2 | A 3 G A 2 | G 3 F F 2 | D 3 G C 2 | E 2 C 4 | C 2 C 4 | C 2 C 3 G | A B c 2 B | | B 2 2 | A 3 G 2 2 | A 2 G 2 2 | D 3 G C 2 | A 3 D E 2 | D 3 G B 2 | A 3 G 4 | G 2 A 4 | G 2 G 3 G | A B c 2 B 2 | A 2 G 2 G 2 | G 2 E 2 D 2 | 2 | C 4 | C 2 C 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 4 | G 2 A 3 G | A B c 2 B 2 | A 2 G 4 | E D C 4 | C 2 C 4 | G 2 E 3 D | C 2 C 2 | C 2 C 3 D | E 2 G 4

X:1
T:The Balnk n mont
M:2/4
K:Cmaj
| e c e d c 2 G A | G A G F D 2 G C | c 3 D F G F E F | G A B c A G E D | G A B c A G E D | C 3 D E D A 2 B | A G F G F D F D C | c A F F A F A F | c A F G G F D C | c A F A G F B A | G E D C C C D F | B A G F G 2 C D | D E F D A 2 G c | B c A G A D F D | E D C A, C G, | , C D E D C :| |2 E G D E G E E C | F, C D A, C A, C | A, C D E C B, C A, | G, A, C D A, C A | A, C E, C, C

Appendix B: Screenshot of questionnaire

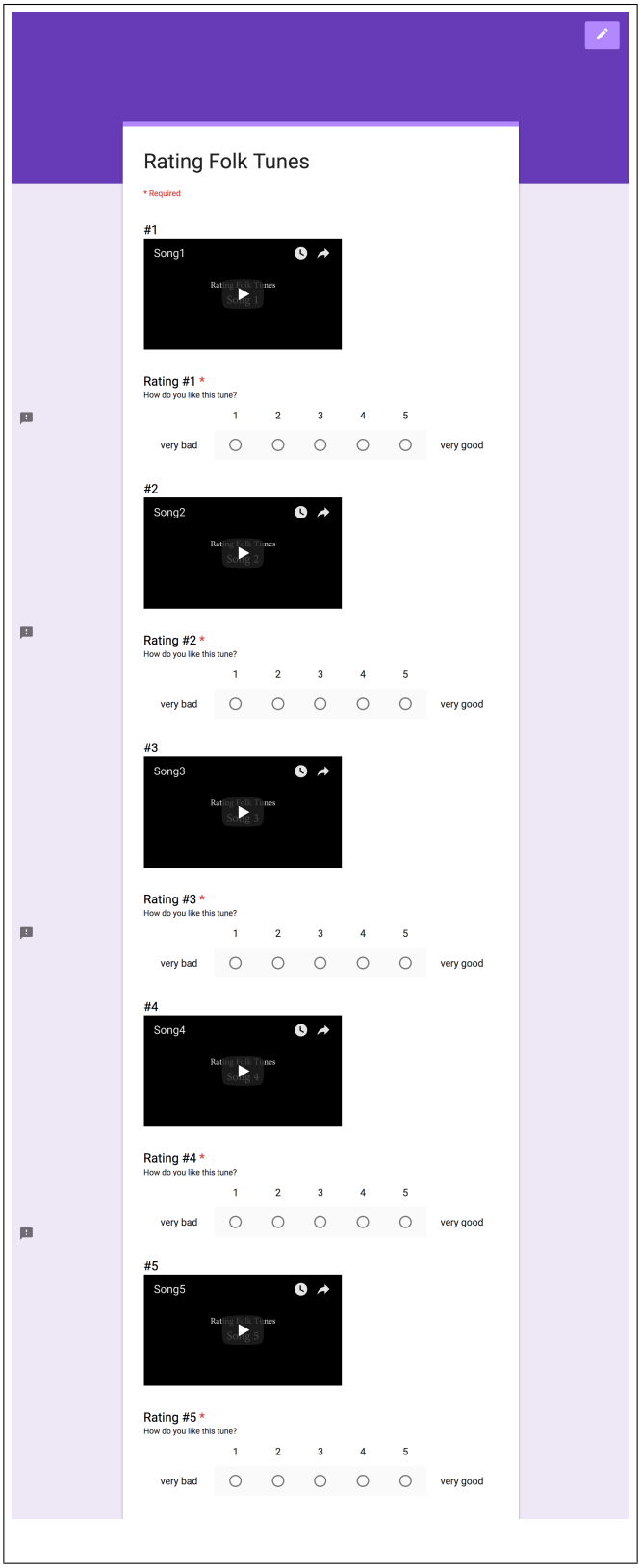

Fig. 8: Top third of the evaluation questionnaire.

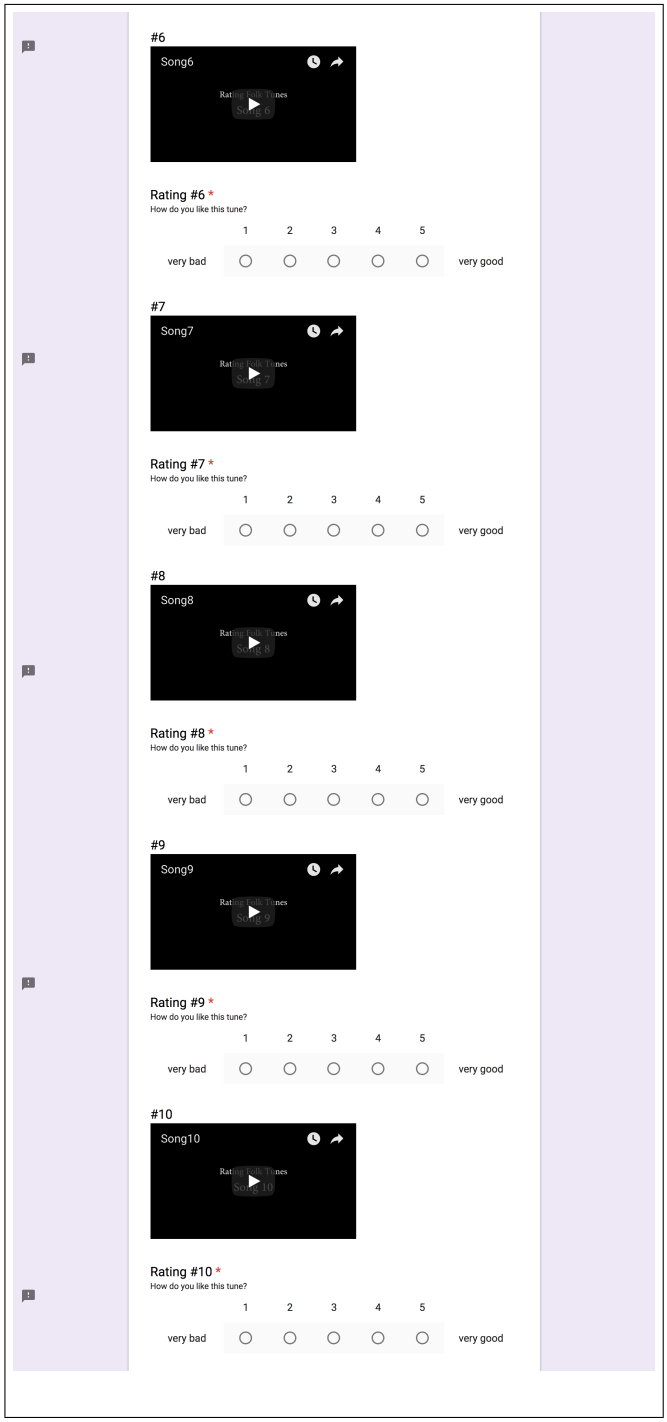

Fig. 9: Middle third of the evaluation questionnaire.

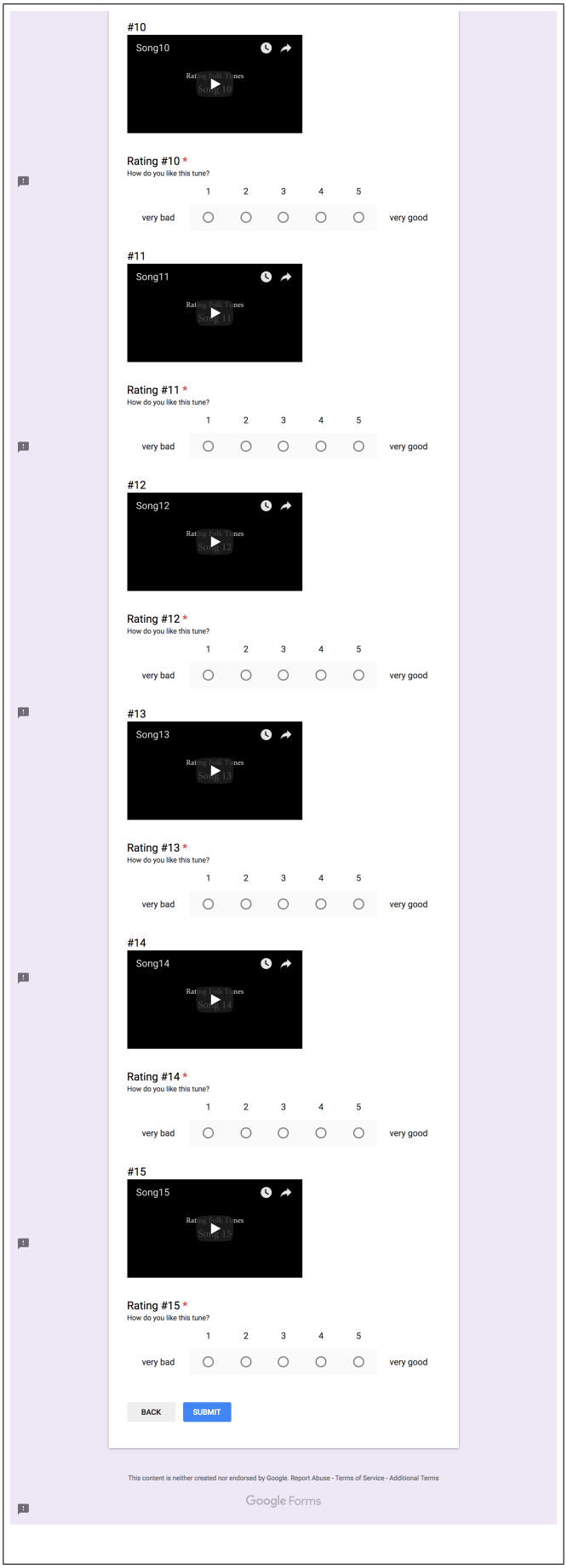

Fig. 10: Bottom third of the evaluation questionnaire.