Completing Bach's Unfinished Contrapunctus XIV With Machine Learning: Linear Regression, Multilayer Perceptron, and Echo State Networks

author 1
author 2
author 3
author 4

ABSTRACT

In this project we set out to complete Bach's famous unfinished fugue (Contrapunctus XIV) using three different machine learning models: linear regression, a multilayer perceptron, and an echo state network. The fugue consists of four voices. A specific input encoding with attention to perceived similarity is used to favourably organize the data points for the learning models. The output for each model is represented as probability distributions over the possible notes for each channel. These probability distribution are post-processed by infusing human knowledge about music in the selection procedure of the best possible note. The linear regression model yielded the highest validation accuracy on the predicted notes (87.5%), but the multilayer perceptron did not give a much worse performance (84.5%). The echo state network gave a significantly lower accuracy of (48.2%). However, such validation accuracy metric is not necessarily correlated to 'good' or Bach-like sounding music. Nonetheless, it appears that more training data is required, beyond a single piece of music, such that the model is able to learn more general musical patterns.

KEYWORDS

machine learning, time series prediction, Bach

1 INTRODUCTION

The last fugue of Johann Sebastian Bach (1685-1750), Contrapunctus XIV from Die Kunst der Fuge, is one of the most enigmatic uncompleted works in music history. This fugue is a summation of Bach's mastermind - a collection of fugal art including 14 fugues, 2 fugal inversions, and 4 cannons, which are linked to a single theme, and gradually undergo subtle developments and variations in rhythm and melody over 4 separate voices. Over the centuries, many have attempted to complete the fugue. More recently in 1992, it had been revisited by the organizers of the Santa Fe Time Series Analysis and Prediction Competition, but this time as a testing ground for methods of statistics, machine learning and artificial intelligence. No specific goal was set for this challenge.

It is generally believed that Bach applied strict musical rules to produce the masterpiece. As remarked by a round table discussion from the Santa Fe Institute, "the art of the fugue is basically an attempt to construct large musical structures based on very rigorous limitations on thematic material… You run the string backwards at great, long expanse; you dilate it; you shrink it; you turn it upside down; you run it backwards. But the whole thing is very self-similar in that, anywhere you look, it's some variant of this thing" [1]. In this line of thought, some modelers have tackled Bach through expert knowledge. An initial collection of context-free rules are summarized from a large musical collection, before machine learning is applied to find out the particular Bach-like probability subspace. A notable example is Kulitta, which applied probabilistic temporal graph grammars for this purpose, and claimed to have passed the Turing test [2].

However, an expert system is after all a coarse space reduction of the multifold that actually exists in reality – for instance, Bach hurried down some notes while the clock cuckooed. For the purpose of demonstrating generalization capacity of machine learning algorithms, it is more interesting to lose the a priori assumption of an expert system, and to apply the same machine learning principles to learn and generate, in this case, a piece of Bach music from scratch. Echoing the original challenge of the Santa Fe institute, in this report, we are working exactly on this purpose - to treat the Bach fugue as a testing ground, to demonstrate and compare generalization capacities of machine learning models.

1.1 Machine Learning Models

We have selected three different models in this report. The first model is based on standard multiple linear regression. In this model, the input vector is directly regressed to the output vector. In the next two models, the lower dimensional input vector is transformed in some way to a higher dimensional state vector. In a multilayer perceptron model, the input vector is fed into hidden layers of artificial neurons; whereas in an echo state network, the input vector is broadcast onto a reservoir of nodes. In the following, we will discuss each one of the models briefly.

1.1.1 Regularized Multiple Linear Regression. Standard multiple linear regression (MR) describes the relationship between an outcome vector paired with a number of regressors. In our case, the regressors are linear inputs. The role of MR is to find the best linear combinations of input regressors to predict the corresponding outcome vector. However, though directly applying standard multiple regression will result in the optimal loss, it would also likely lead to overfitting of the trained pattern. To prevent overfitting and increase expectation in test pattern, some regularization measures need to be included. One popular example is the Ridge regularization, which adjusts the linear function to maximize expectation over cross-validation.

1.1.2 Multilayer Perceptron. The multilayer perceptron (MLP) is a multilayer generalization of the perceptron – one of the simplest architectures in the class of feedforward neural networks. The MLP consists of at least three layers of neuronal units: the input layer, the hidden layer, and the output layer. Each unit is densely connected to the next layer, meaning that its activation is forwarded to every unit in the following layer scaled by some weight. Some units in the MLP typically have a non-linear activation function, which opens the gate to the capacity for separating non-linear data. Regrettably, any linear algebraic procedure to minimize empirical risk, as applicable with the likes of linear regression, will no longer suffice. Fortunately, iterative gradient-descent based methods using the backpropagation algorithm makes the optimization objective computationally tractable. It allows to compute the (stochastic) gradient of a differentiable loss function with respect to the weights, i.e., the trainable parameters, in a supervised setting, by successive applications of the chain rule such that the error is 'back propagated' into the network on a layer-by-layer basis.

1.1.3 Echo State Network. Reservoir computing (RC) is an overarching term that refers to a number of methods and approaches for designing recurrent neural networks (RNN). One of the flavours of RC computing are echo state networks (ESN) [3]. Echo state networks differ from their popular deep learning counterparts (such as long short-term memory networks) as learning the optimal internal weights can be analytically computed using simple linear regression, rather than relying on complicated and expensive gradient descent algorithms. Additionally, a large part of the weights that define an ESN do not have be determined at all, and can be generated randomly. This makes ESN conceptually simple and easy to implement. Nevertheless, squeezing the maximum performance out of an ESN requires some expert insight and practical experience [4].

The basic gist of an ESN is that an input signal is first fed into a reservoir. Signals can then be harvested from the reservoir, and linearly combined to model a target signal. The reservoir adds non-linearity to the learning equation, whilst the readout weights can be computed linearly and analytically. Additionally, the reservoir adds a memory effect.

1.2 General Processing Pipeline

To be consistent in our report, we apply the same preprocessing steps to convert the original data file in MIDI format to an input vector. The input vector is then fed into each of the three optimized machine learning models to generate an output vector. Lastly, the model outputs are fed into the same postprocessing stream to produce playable music. The optimization process is slightly different for MR and MLP as compared to ESN. Standard cross-validation procedure is applied to MR and MLP, since both models are not dependent on their previous states. Nevertheless, a different optimization procedure must be applied in ESN, since the current reservoir node state also relies on the previous node states across a moving window of trained input and output vectors (compared to recurrent networks). Other than necessary differences in optimization, the processing pipelines are kept as similar as possible for comparison across different models.

1.2.1 Preprocessing. The provided data file contains the MIDI note numbers for each channel over time. Each one of these values is converted to a 5-dimensional input vector. This vector encodes the logarithm of the pitch, and the coordinates (x and y) of the notes on the chroma circle and circle of fifths. The data file is also the source of the teacher output vectors. For this, each MIDI note number is encoded into a one-hot vector encoding based on the bounded chromatic range for each channel. With these input and output vectors, the learning task is comprehended as a time series prediction task with a difference in the encoding between the input and output vectors.

1.2.2 Optimization. The individual learning models were optimized by solving for the least squares in the case of MR and ESN with a discount on model complexity through a Ridge parameter. The MLP was optimized on multi categorical cross-entropy with the analogous L_2 regularization using mini-batch gradient descent. Additional hyperparameters for the models were tuned by employing a cross-validation scheme in the case of MR and the MLP. For the ESN a manual tuning approach was exploited.

1.2.3 Postprocessing. The implemented models can generate a sequence of notes by feeding back their output as an input. However, the generated music requires postprocessing to sound pleasant for the human ear. During postprocessing the output vectors of the models are converted into probability vectors for each note, that can be manipulated. The goal during the postprocessing was to adapt the model outputs such that five different patterns of the generated music are the same as for Bach's fugue. These patterns are: The distribution of note lengths, the distribution of note starting positions within a measure, the distribution of pitches, the distribution of pitch changes in regard to the last pitch and the amount of harmonies and disharmonious between voices. The most important adaptation to make produced music sound like Bach, was to have the correct distributions of note length and to reduce disharmony between voices.

2 MUSIC ENCODINGS

2.1 Input Encoding

The data file that was delivered by the Sante Fé competition consists of four channels, each consisting of a sequence of MIDI note numbers (MID), which are related to a unique pitch:

$$f(\text{MID}) = 440 \cdot 2^{(\text{MID}-69)/12} \tag{1}$$

The file does not specify a duration for each note, it is rather encoded through a repeated sequence of the same MIDI value. A single MIDI value (not repeated) relates to a note being played for the duration of a 16th of a measure, or a quarter of a beat for this particular piece. A rest in the music is encoded by a zero entry.

From here, one could decide to provide this representation as an input to the learning models. However, other representations might yield a favourable organization, or separation for that matter, of the structures in the input space. In particular, the MIDI value for each channel $0 \leq c < C$, with $C = 4$, at time step $n \in \mathcal{N} = \{n < N_{max} | n \in \mathbb{N}\}$, with N_{max} the length of the sequence of MIDI values, has been encoded into a 5-dimensional vector $\mathbf{u}_c(n)$, as described by Kuqi [5]. In this representation, the first element encodes the

offset logarithm of the pitch. The second and third elements encode the $(x,\ y)$ coordinates of the position of the note on the chroma circle. Similarly, the fourth and fifth elements encode the $(x,\ y)$ coordinates of the position of the note on the circle of fifths. The appeal of this encoding comes from the conception that 'similarly' sounding notes should be represented by points in a space that lie close to each other considering Euclidean geometry. This is not necessarily the case when one considers an encoding of only the MIDI note numbers.

'Similarly' is put between quotation marks, as it is poorly defined. The original representation distances notes from each other based on the difference in the MIDI value or analogously on the difference in the logarithm of the pitch. However, when one considers tonal fusion due to the perceived consonance of two or more notes, then the similarity between notes is organized differently [6]. Tonal fusion can be associated with a similarity measure, as it quantifies the degree to which tones with different pitches are perceived as unitary. Notably, tones with frequency components that overlap to a greater extent are perceived as more unitary. Notes in unison, octaves, and perfect fifths can be identified as intervals with this property.

The combination of the logarithm of the pitch and the coordinates on the chromatic circle and circle of fifths provides an encoding that takes into account these aspects of similarity when recognizing the euclidean distance between notes in the encoded format as a means of quantifying the similarity. The relative importance of each can be tuned by scaling the individual elements. We define $\mathbf{u}_c(n)$ more precisely:

$$\mathbf{u}_c(n) = \begin{bmatrix} f_{norm,c} & c_{1,x} & c_{1,y} & c_{5,x} & c_{5,y} \end{bmatrix}' \tag{2}$$

with $f_{norm,c}$ the normalized frequency given by:

$$f_{norm,c}(\mathrm{MID}) = 2\log_2 f(\mathrm{MID}) + f_{offset}(c) \tag{3}$$

the offset is defined by the minimum $f_{min}(c)$ and maximum pitch $f_{max}(c)$ for each channel c:

$$f_{offset}(c) = \frac{2\log_2 f_{max}(c) - 2\log_2 f_{min}(c)}{2} - 2\log_2 f_{max}(c) \tag{4}$$

The coordinates on the chroma circle are defined by the following equations:

$$\theta_1 = (2\pi(\mathrm{MID} \mod 12)/12) \tag{5}$$

$$\theta_5 = (2\pi((7\mathrm{MID}) \mod 12)/12) \tag{6}$$

$$c_{1,x}(\mathrm{MID}) = r\cos\theta_1,\quad c_{1,y}(\mathrm{MID}) = r\sin\theta_1 \tag{7}$$

$$c_{5,x}(\mathrm{MID}) = r\cos\theta_5,\quad c_{5,y}(\mathrm{MID}) = r\sin\theta_5 \tag{8}$$

r is a scaling that is set to $r = 1$, i.e., the points are on the unit circle. Furthermore, the points are spaced apart with a radial distance $2\pi/12$. After all, there are 12 semitones in an octave.

2.2 Output Encoding

The task of generating music is best recognized as a time series prediction task. Although there is only an input signal $\mathbf{u}(n)_{n\in\mathcal{N}}$, a corresponding teacher output can be identified by the input signal at the next time step $\mathbf{y}(n)_{n\in\mathcal{N}} = \mathbf{u}(n+1)_{n\in\mathcal{N}}$. With this approach the problem is reorganized to a supervised learning task. However, similar to the representation of the MIDI values that was deemed undesired as a direct input to the statistical models, this representation might likewise be undesired as an output of the model.

Voice	Channel	MIDI min	MIDI max	Range
-	c	$\mathrm{MID}_{min}(c)$	$\mathrm{MID}_{max}(c)$	$\mathrm{MID}_{range}(c)$
Soprano	1	54	76	23
Alto	2	45	71	27
Tenor	3	40	62	23
Bass	4	28	54	27

Table 1: The chromatic range for each voice. The range represents the number of MIDI values (semitones) that can be played by each voice. MIDI min and MIDI max are the minimum and maximum MIDI values for each voice respectively.

The major concern with dealing with the output data, is that it must be in a format that proves to be suitable for post-processing. The problems outlined for the input encoding, namely regarding organization of the data points in space, applies to the output encoding and how it relates to a loss function as well. The encoding that is used for the inputs can in principle be used to encode the output signal. However, it is not natural to decode vectors in this representation, in particular in consideration of a post-processing procedure; let alone finding a suitable loss function for it. When recognizing the limited set of outputs for each channel, that is, the bounded chromatic range for each channel, the problem appears respectable for a reduction to a classification problem. Closer inspection of the channels reveals that different channels have a different chromatic range, albeit with a slight overlap between them. Although there are disagreements on how The Art of Fugue was intended to be played, if at all, the four-voice fugue is typically annotated with a soprano, alto, tenor and bass voice. The exact ranges that were kept in mind for these voices are not known. However, there is little incentive to assume a chromatic range that goes beyond the notes that are played in the uncompleted fugue. These notes will be unrepresented in the dataset and therefore are precarious under the recruitment of statistical models. The range of MIDI note numbers for each channel is shown in Table 1.

To comply with the different chromatic ranges of the channels, a one-hot vector encoding on a per channel basis is employed. This establishes a per channel classification problem and accommodates the model predictions of (pseudo-) probability distributions over the MIDI values. Such distributions over the possible MIDI values are pragmatic, especially as it gives more control over note selection compared to directly produced MIDI values.

The teacher vectors will be encoded from the $\mathbf{y}_c(n)$ from the MIDI values (MID) corresponding to $\mathbf{u}_c(n+1)$. The one-hot vector encoding takes the values:

$$\mathbf{y}_c(n) \in \{0,\ 1\}^m;\quad m := \mathrm{MID}_{range}(c) \tag{9}$$

where $\mathrm{MID}_{range}(c)$ is the MIDI value range for channel c as listed in Table 1.

$$\mathbf{y}_c(n) = \mathbf{e}_i;\quad i := \mathrm{MID} - \mathrm{MID}_{min}(c) + 1 \tag{10}$$

where $\mathbf{e}_i$ is the standard basis vector with 0's everywhere, except a value of 1 at the index i.

In addition, we have experimented with an additional encoding of the duration of the notes, which was treated as a secondary classification variable. This has been implemented and tested for a single

channel signal. Difficulties arise, however, when multiple channels are involved. In particular, not every note starts and ends at the same time steps. Therefore, the model does not need to produce a new output for each channel every time step. This complicates the design. In addition, concerns should be raised with the addition of duration vectors for each channel in light of the curse of dimensionality. Keeping these matters in mind, we omitted the duration encoding and instead allow the model to produce longer duration notes through the repetition of the same note – similar to the format of the raw input data.

3 MACHINE LEARNING MODELS

3.1 Multilayer Perceptron

3.1.1 Model definition. The multilayer perceptron is a densely connected feedforward neural network with atleast one hidden layer. An MLP $h\colon \mathbb{R}^K \mapsto \mathbb{R}^M$ implements a map from input $\bar{\mathbf{u}} \in \mathbb{R}^K$ to output $\hat{\mathbf{y}} \in \mathbb{R}^M$. The input to the network will be a windowed view of the signal $\mathbf{u}(n)$:

$$\mathbf{u}(n) = \mathbf{u}_1(n) \oplus \mathbf{u}_2(n) \oplus \cdots \oplus \mathbf{u}_C(n) \tag{11}$$

where $\mathbf{u}_c(n)$ is the 5-dimensional vector encoding for channel c at time step n, and $C = 4$ because there are four channels. $\oplus$ is the concatenation operator. A sliding window view of length N_w is created from this signal:

$$\bar{\mathbf{u}}(n) = \mathbf{u}(n - N_w + 1) \oplus \mathbf{u}(n - N_w) \oplus \cdots \oplus \mathbf{u}(n) \tag{12}$$

The windowed view $\bar{\mathbf{u}} \in \mathbb{R}^K$ in this flattened representation, is forwarded to set the activations of the units in the input layer of the MLP h. As a result, the number of units in the input layer is $K = N_w \cdot C \cdot 5$.

The output of the network $\hat{\mathbf{y}} \in \mathbb{R}^M$ is comprised of the probability distributions over the MIDI values:

$$\hat{\mathbf{y}}(n) = \hat{\mathbf{y}}_1(n) \oplus \hat{\mathbf{y}}_2(n) \oplus \cdots \oplus \hat{\mathbf{y}}_C(n) \tag{13}$$

with $\hat{\mathbf{y}}_c(n)$ and $0 \leq c < C$ the probability distribution over the MIDI values in the chromatic range of channel c. Therefore, the number of units M in the output layers is:

$$M = \mathrm{MID}_{range}(0) \cdot \mathrm{MID}_{range}(1) \cdot \cdots \cdot \mathrm{MID}_{range}(C) \tag{14}$$

Unlike the input layer that represents the flattened window view of the signal, the output layer constitutes C heads. The concatenation of the heads will yield $\hat{\mathbf{y}}(n) \in \mathbb{R}^M$, as in Equation 13. Although this is not relevant for the actual computation or implementation, it makes the notation more transparent, as will be revealed later.

The activation of the units in the hidden and output layers is computed as follows:

$$x_i^\kappa = \sigma\left(\sum_{j=1}^{L^{\kappa-1}} w_{ij}^\kappa x_j^{\kappa-1} + w_{i0}^\kappa\right) \tag{15}$$

where x_i^κ is the activation of the ith unit in layer $1 < \kappa \leq k$; and w_{ij}^κ the weight between the jth unit in layer $\kappa - 1$ and the ith unit in layer κ; w_{i0}^κ is the weight between the bias unit (activation of 1) in layer $\kappa - 1$ to the ith unit in layer κ. The number of units in the κth layer is given by L^κ. σ is the activation function of the unit.

Parameter	Distribution or set	Selected
N	$\{2^n \mid 4 \leq n \leq 9; n \in \mathbb{N}\}$	32
L^2	$\{n \mid 4 \leq n \leq 2^{10}; n \in \mathbb{N}\}$	410
β	log-uniform$(10^{-8}, 10^{-1})$	4.9×10^{-2}
N_w	$\{n \mid 1 \leq n \leq 400; n \in \mathbb{N}\}$	31

Table 2: The hyperparameters of the multilayer perceptron. The distribution column contains either the set from which was a parameter is uniformly sampled or a distribution from which is sampled.

The units in the hidden layer apply the sigmoid function: $\sigma(a) = (1 + e^{-a})^{-1}$. The output units apply the softmax function per head:

$$(\hat{\mathbf{y}}_c)_i = (\sigma_c(\mathbf{a}))_i = \frac{e^{a_i}}{\sum_{j=1}^{K_c} e^{a_j}}; \quad K_c \coloneqq \mathrm{MID}_{range}(c) \tag{16}$$

where $(\hat{\mathbf{y}}_c)_i$ is the activation of the ith unit in the head c in the output layer k. The softmax activation ensures that each head in the output layer produces a probability vector over the MIDI values for that channel.

3.1.2 (hyper-)parameter optimization. The parameters of the MLP h were optimized on the objective of minimizing the sum of several loss functions. The main loss function is the mean categorical cross entropy of the prediction of each head in the output layer.

$$\mathrm{CCE}(c) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{K_c}(\mathbf{y}_c(n))_i \cdot \log\left(\hat{\mathbf{y}}_c(n)\right)_i \tag{17}$$

In addition, the L_2 loss function is included for flexible regularization:

$$L_2(\theta) = \beta^2 \sum_{w \in \theta} w^2 \tag{18}$$

where θ is the set of all weights of the models.

Combining these losses yields the summed loss as described in Equation 19.

$$\mathrm{Loss} = \sum_{c=1}^{C} \mathrm{CCE}(c) + L_2(\theta) \tag{19}$$

The weights θ were initialized by sampling from the Glorot normalized uniform distribution [7]. The Adam optimization algorithm [8] with the default decay rate for the moments, and a learning rate of $\lambda = 1.0 \times 10^{-7}$. In addition, mini-batch gradient descent was used with a batch size of N. The MLP was kept fixed to one hidden layer (layer 2). Additional hidden layers were not considered, as this would likely contribute to overfitting. One hidden layer is expected to suffice, unless it is very non-linearly organized. The number of hidden units L^2, the batch size N, the L_2 regularization parameter γ, and the window length N_w were optimized by using 5-fold cross-validation. Random search over 100 iterations was applied and the data set was split up into 80% training and 20% validation points by shuffle splitting the data into 5-folds. The MLP was trained for a maximum of 100 epochs, but if the validation accuracy did not increase for 3 epochs it would stop early. Table 2 shows the distributions from which the parameters were sampled and the resulting parameter that showed the lowest accuracy score as described by

Parameter	Distribution or set	Selected
β	log-uniform$(10^{-5}, 10^{3})$	2.6
N_w	$\{n \mid 1 \leq n \leq 400; n \in \mathbb{N}\}$	378

Table 3: The hyperparameters for ridge regression. The distribution column contains either the set from which was a parameter is uniformly sampled or a distribution from which is sampled.

Equation 21 over the validation set.

$$p(c,i) = \begin{cases} 1, & \text{if } \arg\max(\hat{\mathbf{y}}_c(n)) = \arg\max(\mathbf{y}_c(n)) \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

$$acc = \frac{1}{NC} \sum_{n=1}^{N} \sum_{c=1}^{C} p(c,i) \tag{21}$$

3.2 Linear Regression

3.2.1 Model definition. Standard multiple linear regression finds a linear combination of input $\bar{\mathbf{u}} \in \mathbb{R}^K$ (Equation 12) that produces an estimated $\hat{\mathbf{y}} \in \mathbb{R}^M$ (Equation 13), which approximates the actual output $\mathbf{y} \in \mathbb{R}^M$ by solving for least squares on the training data. In our case, we define $\mathbf{U}$ as the concatenation of the vectors $\bar{\mathbf{u}}_b(n) = \bar{\mathbf{u}}(n) \oplus 1$ – the padded 1 gives us an affine linear map (bias component) – as columns into a $K \times N_{max}$ matrix; and $\mathbf{Y}$ as the concatenation of the outputs $\mathbf{y}(n)$ as row vectors into a $N_{max} * M$ matrix, i.e:

$$\mathbf{U} = \begin{bmatrix} \bar{\mathbf{u}}(1) & \bar{\mathbf{u}}(2) & \cdots & \bar{\mathbf{u}}(N_{max}) \end{bmatrix} \tag{22}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}(1)' & \mathbf{y}(2)' & \cdots & \mathbf{y}(N_{max})' \end{bmatrix}' \tag{23}$$

Given the matrices $\mathbf{U}$ and $\mathbf{Y}$, the $(K+1) \times M$ coefficient (or weight) matrix $\mathbf{W}$ can be estimated as follows:

$$\mathbf{W}' = (\mathbf{U}'\mathbf{U})^{-1}\mathbf{U}'\mathbf{Y} \tag{24}$$

However, though this estimation achieves minimal loss on the training data, it may also become overfitted to the trained $\mathbf{y}(n)_{n \in \mathcal{N}}$ biased by random errors, leading to the lower expectation of squared errors. To downregulate overfitting, we add an L_2 term (equation 18), or under linear regression more commonly dubbed Ridge parameter β, as follows:

$$\mathbf{W_{Ridge}}' = (\mathbf{U}'\mathbf{U} + \beta^2\mathbf{I})^{-1}\mathbf{U}'\mathbf{Y} \tag{25}$$

The Ridge parameter is adjusted such that it minimizes the accuracy (Equation 21) during 5-fold cross-validation. In addition, the window length N_w is optimized using the same random search cross-validation procedure as described for the MLP in Subsection 3.1. The resulting sampling distributions and best parameters are shown in Table 3.

3.3 Echo State Network

3.3.1 Model definition. Rather than the windowed input (Equation 12) used by the MLP and the MR, the ESN accepts the input signal $\mathbf{u}$ as defined by Equation 11. For clarity, the definition of this input signal is restated: at every time step n, the input signal $\mathbf{u}$ (Equation 26) is a vector of length N_u, where $N_u = C \cdot 5$. All channels were used. Thus, $C = 4$.

$$\mathbf{u}(n) = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \cdots & \mathbf{u}_{N_u} \end{bmatrix} \tag{26}$$

Given an input signal, the ESN will generate an output signal $\hat{\mathbf{y}}$. This signal is defined by Equation 13. Again, for clarity, the definition of signal $\hat{\mathbf{y}}$ is restated: at every time step n, the output signal $\hat{\mathbf{y}}$ (Equation 27), is a vector of length N_y, where as $N_y = M$ (Equation 14). Note that in this section, the signal $\hat{\mathbf{y}}$ refers to the signal generated by the ESN, wheres the signal $\mathbf{y}$ refers to the teacher signal used during training. Both $\hat{\mathbf{y}}(n)$ and $\mathbf{y}(n)$ are part of $\mathbb{R}^{N_y}$.

$$\hat{\mathbf{y}}(n) = \begin{bmatrix} \mathbf{y}_0 & \mathbf{y}_1 & \cdots & \mathbf{y}_{N_y} \end{bmatrix} \tag{27}$$

The internal state of the networks reservoir is defined by the signal $\mathbf{x}$. At every time step n, the internal signal is a vector of length N_x (Equation 28). Each value $x_i \in \mathbf{x}(n)$ represents the activation level of the i-th neuron, whose value can lie in the range $[-1, 1]$, or $[0, 1]$.

$$\mathbf{x}(n) = \begin{bmatrix} \mathbf{x}_0 & \mathbf{x}_1 & \cdots & \mathbf{x}_{N_x} \end{bmatrix} \tag{28}$$

The ESN is further defined by the following weight matrices:

$$\mathbf{W}^{in} = (w_{ij}^{in}) \quad \mathbf{W} = (w_{ij}) \quad \mathbf{W}^{out} = (w_{ij}^{out}) \quad \mathbf{W}^{fb} = (w_{ij}^{fb}) \tag{29}$$

The matrix $\mathbf{W}^{in}$ holds to input-to-reservoir connections that can feed the signal $\mathbf{u}$ into the reservoir, and is of size $N_x \times N_u$. The weight matrix $\mathbf{W}$ stores the internal reservoir connections, and is of dimensions $N_x \times N_x$. The matrix $\mathbf{W}^{out}$ stores the output weights that can linearly transform the internal signal $\mathbf{x}$ into an output signal $\hat{\mathbf{y}}$. Lastly, the matrix $\mathbf{W}^{fb}$ contains the weights that can feed the generated output signal back into the reservoir, and is of dimensions $N_x \times N_y$.

The transition of internal state vector $\mathbf{x}(n)$ into $\mathbf{x}(n+1)$ is governed by Equation 30. Note that in this equation all signals at time step n are column vectors. The function f denotes the component-wise application of the activation function to each entry in the vector $\mathbf{x}(n+1)$. This activation function is a sigmoid function which will squish each entry into to the range $[-1, 1]$, or $[0, 1]$. The vector $\mathbf{b}$ refers to a bias and is of length N_x.

$$\mathbf{x}(n+1) = f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{fb}\mathbf{y}(n) + \mathbf{b}) \tag{30}$$

Equation 30 can be extended to include a leaking rate α, as specified by Equation 31. The leaking rate can either be a vector $\alpha \in [0, 1]^{N_x}$, or a scalar $\alpha \in [0, 1]$.

$$\mathbf{x}(n+1) = (1-\alpha)\mathbf{x}(n) + \alpha f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{fb}\mathbf{y}(n) + \mathbf{b}) \tag{31}$$

The internal state vector $\mathbf{x}(n)$ can be exploited to produce a output vector $\hat{\mathbf{y}}(n)$ using weight matrix $\mathbf{W}^{out}$, as specified by equation 32.

$$\hat{\mathbf{y}}(n) = \mathbf{W}^{out}\mathbf{x}(n) \tag{32}$$

3.3.2 Producing a model. The learning task for the ESN can be stated as follows: given an input training signal $\mathbf{u}$, and a teacher signal $\mathbf{y}$, determine the weight matrices $\mathbf{W}^{in}$, $\mathbf{W}$, $\mathbf{W}^{out}$, $\mathbf{W}^{fb}$, and bias vector $\mathbf{b}$ such that the output signal $\hat{\mathbf{y}}$ approximates the teacher signal $\mathbf{y}$. The following subsection will highlight the necessary steps to complete this task.

The weight matrices $\mathbf{W}^{in}$, $\mathbf{W}$, and $\mathbf{W}^{fb}$ can be generated randomly by sampling each weight w_{ij} from some distribution. These distributions are typically the Gaussian distribution $\mathcal{N}(0, 1)$, or the uniform distribution $\mathcal{U}(-0.5, 0.5)$ [4]. The entries in bias vector $\mathbf{b}$ can be determined in similar fashion.

However, the weight matrix $\mathbf{W}$ needs to be scaled properly in order for the reservoir to possess the so-called echo state property: that is, for a long enough input signal $\mathbf{u}$, the internal state of the reservoir $\mathbf{x}(n)$ should no longer depend on the randomly initial state $\mathbf{x}(0)$. This property can be ensured by scaling the spectral radius of $\mathbf{W}$ (i.e. its maximum absolute eigenvalue). The spectral radius is denoted by $\rho(\mathbf{W})$, and ensuring that $\rho(\mathbf{W}) < 1$ guarantees the echo state property in most cases [4]. In practice, $\mathbf{W}$ is typically divided by its maximum eigenvalue such that a matrix with unit spectral radius is obtained. $\rho(\mathbf{W})$ can then be scaled with convenience.

In contrast to the previously mentioned matrices, the weight matrix $\mathbf{W}^{out}$ needs to be determined analytically. This can be done by driving the network with input signal $\mathbf{u}$, and collecting all internal state vectors $\mathbf{x}(n)$ as rows into a matrix $\mathbf{U}$. The corresponding output teacher vectors $\mathbf{y}_t(n)$ should be similarly collected into a matrix $\mathbf{Y}$. Next, all rows in matrix $\mathbf{U}$ (and the corresponding rows in $\mathbf{Y}$) should be discarded where $\mathbf{x}(n)$ still depends on the random initial state $\mathbf{x}(0)$ in order not to pollute the weights in $\mathbf{W}^{out}$. The optimal weight matrix $\mathbf{W}^{out}$ can then be computed by minimizing the MSE using ridge regression (Equation 33), as previously described in section 3.2. The parameter β refers to the regularization parameter, and $\mathbf{I}$ to the identity matrix. In order to add a bias, the matrix $\mathbf{X}$ can be extended by an extra column, whose entries have a constant value of one.

$$\mathbf{W}^{out\prime} = (\mathbf{U}'\mathbf{U} + \beta^2\mathbf{I})^{-1}\mathbf{U}'\mathbf{Y} \tag{33}$$

3.3.3 Parameter selection. In contrast to the ridge regression model and the MLP, no automatic parameter search with a K-fold cross-validation scheme was used to determine the optimal parameters. Due to time limitations, it was not feasible to adapt the exiting cross-validation setup such that it respects the memory effects of the reservoir. Instead, the network was tuned by hand according to the method and wisdom as stated in [4]. This section will highlight the influence of a number of parameters on the behaviour of the network, and explain how the optimal parameters were selected.

In order to simplify the tuning task, a number of parameters were kept constant. As such, the randomly generated weight matrices and the bias vector were sampled from the uniform distribution $\mathcal{U}(-0.5, 0.5)$; no other distributions were tested. The activation function was set to *tanh*. Additionally, to simplify the behaviour of the network, the feedback weights were disabled by scaling them to zero. The parameters that remained to be determined were the size N_x of the reservoir and its spectral radius $\rho(\mathbf{W})$, the scaling of weight matrix $\mathbf{W^{in}}$ and bias vector $\mathbf{b}$, the leaking parameter α, and the regularization parameter β.

The size of the reservoir determines the model flexibility. A bigger reservoir results in a better approximation of the teacher signal given that the model is appropriately regularized. A lower limit for the reservoir size can be roughly determined by the number of real values that the reservoir must remember from the input to generate an output [4].

The spectral radius of the reservoir influences the speed of the dynamic of internal signal $\mathbf{x}$. $\rho(\mathbf{W})$ should be set small for tasks where $\mathbf{y}(n)$ depends more heavily on the current input $\mathbf{u}(n)$, and large for tasks that require long memory of the input. However, $\rho(\mathbf{W})$ should be small enough to ensure the echo state property.

	N_x	$s \cdot \mathbf{W^{in}}$	$s \cdot \mathbf{b}$	$\rho(\mathbf{W})$	α	β	$t_{washout}$
Default	500	1.0	1.0	1.0	1.0	0	0
Optimal	2000	0.3	0.9	1.0	0.1	10	140

Table 4: The default and optimal parameters for the Echo State Network. 's' denotes the scalar that scales the proceeding tensor.

The scaling of $\mathbf{W^{in}}$ and bias vector $\mathbf{b}$ determine how non-linear the reservoir responses are. For very linear tasks, $\mathbf{W^{in}}$ and $\mathbf{b}$ should be small. This ensures that the entries of $\mathbf{x}(n)$ operate around the zero point, where their activation is virtually linear (given the *tanh* activation function). For large values in $\mathbf{W^{in}}$ and $\mathbf{b}$, the entries in $\mathbf{x}(n)$ will get saturated and pushed to the boundaries of their activation. $\rho(\mathbf{W})$, also affects the non-linearity of the reservoir, but it can only scaled so much before the network becomes unstable. Ideally, the columns of $\mathbf{W^{in}}$ are scaled individually [4], but for simplicity we scale the matrix with a single value.

The leaking rate α can be seen as the speed of the reservoir update dynamics discretized in time [4]. The leaking rate should be tuned such that it matches the speed of the dynamics of input signal $\mathbf{u}$ and teacher signal $\mathbf{y}$. α can be adapted to be a vector such that each entry in $\mathbf{x}(n)$ has its own leaking rate. However, for simplicity we stick to a single scalar.

The paper [4] recommends changing one parameter at a time when tuning the ESN by hand. As such, this method was adopted. The parameters were tuned in the following order: $\mathbf{W^{in}}$ scaling, $\mathbf{b}$ scaling, $\rho(\mathbf{W})$, and lastly α. The reservoir size N_x was kept at a constant, rather small value, and will only be up-scaled when the optimal parameters are found. Generally, good parameters that apply to smaller reservoir sizes also apply to networks with larger reservoir sizes [4].

Since it was observed in earlier testing stages that the ESN does not obtain a 100% training accuracy (Equation 21) for smaller N_x, all parameters were optimized with respect to the training accuracy rather than a validation accuracy. For each parameter value that was tested, the network was initialized and trained five times. The obtained training accuracy was then averaged over these five runs. This was done to correct for varying performance due to the randomly generated weight matrices. The parameter that yielded the highest averaged training accuracy was selected, and the next parameter was optimized.

Only when all optimal parameters were found, the reservoir size N_x was up-scaled such that the network could obtain a 100% training accuracy. The optimal regularization parameter β was then obtained with an 80/20 validation split. Only in this stage was the washout time determined, and the collected states $\mathbf{x}(n)$ dependent on $\mathbf{x}(0)$ were discarded before training. All default and optimal parameter values can be found in table 4.

4 POST-PROCESSING

For every time-step, the models output a vector for each voice. Every value in the vector corresponds to one possible note that could be played. We convert these vectors into probability vectors by first setting every negative value to 0. Afterwards, we take every value to an power p and normalize the vector afterwards. The reason we

take to values to a power is to have control over the entropy of the vector. For the linear regression model and ESN we choose $p = 2$ and for the MLP $p = 0.5$.

We could always choose the note that corresponds to the highest value in the probability vectors and let the model run while feeding back its output as an input. In this case, we observe that the produced music can run in a loop. Alternatively, we can randomly choose the notes such that the probability to choose a note is the corresponding value in the probability vector. This prevents loops but the produced music usually does not sound very well. In order to improve we can post-process the probability vectors. We do this based on 5 criteria explained in the following sections.

4.1 Distribution of note length

Without post-processing the models produce notes that are very short. A long note would mean that the same note is chosen over multiple time steps, which is unlikely. We can compensate for this by increasing the probability of continuing the same note.

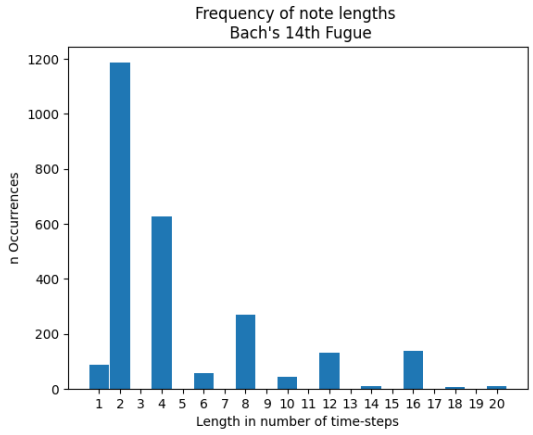
However, some lengths of notes are more or less likely. For example, a note that goes on for exactly 8 time-steps is likely because this corresponds to half a measure. A note going on for 7 time-steps is very unlikely. Ideally, the distribution of the length of the notes should be the same as in Figure 1a. Here we plot the frequencies of notes based on their length in the first half of Bach's 14th fugue. We observe that some lengths are never appearing ($\frac{3}{16}$ or $\frac{5}{16}$ of a measure for example). The ridged regression model without post-processing produced the distribution in Figure 1b. After carefully tuning the post-processing the resulting distribution is shown in Figure 1c, which is almost the same as for Bach. Note that we sometimes ignored the output of the model altogether. For example, we know that a note which goes on for 3 time-steps should not happen. Therefore we set all probabilities for other notes to 0 if a note is already ongoing for 3 time-steps.

4.2 Note beginning positions in measure

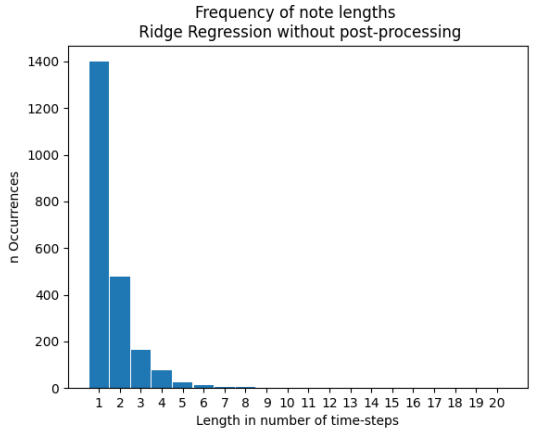
Within a measure there are some positions in which a note is likely to start and other positions for which it is unlikely that a new note is starting. Figure 2a shows the distribution of starting positions in Bach's Fugue. Most frequent positions in which notes often start are the beginning of the measured and positions that are fractions with 4 in the denominator. The same plot for the ridged regression model without post-processing is shown in Figure 2b. All positions within a measure seem to be equally likely to start a new note. We have a similar picture for the ESN or MLP. This is expected, since the models does not know what a measure is or the current position within the measure. However, we can post-process the probability vectors such that time-steps that are unlikely to start a new note have a higher probability to continue with the current note. The post-processed results can be seen in Figure 2c.

4.3 Distribution of notes

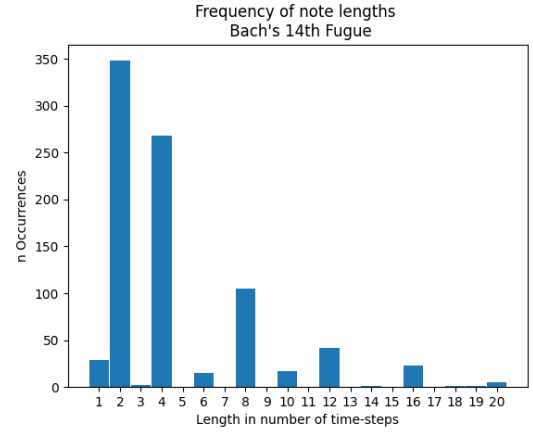
Bach's 14th fugue is written in D-minor which consists of the pitches D, E, F, G, A, $B_\flat$, and C. Therefore we expect that these pitches are more common to appear. This is indeed shown in Figure 3a where the frequencies for each pitch is shown for the first part of the fugue. The models are usually able to generate similar



(a)

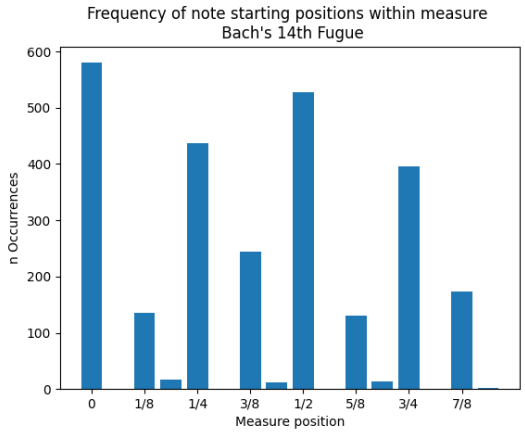


(b)

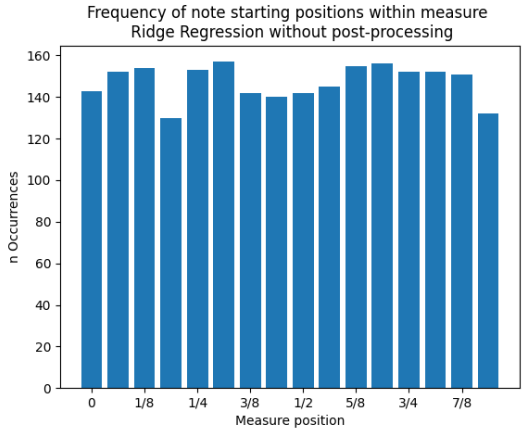


(c)

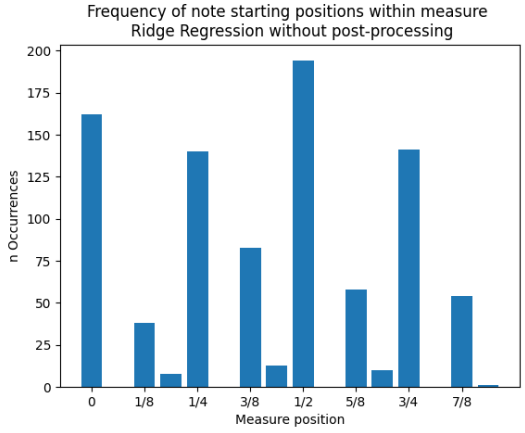
Figure 1: Frequency of length of notes. The x-axis corresponds to number of time-steps, for which there are 16 in each measure. (a): For Bach's 14th Fugue. (b): For the ridge regression model with no post-processing(c): For the ridge regression model with post-processing.



(a)



(b)



(c)

Figure 2: Frequency of the notes starting-positions within a measure. (a): For Bach's 14th Fugue. (b): For the ridge regression model with no post-processing(c): For the ridge regression model with post-processing.

distributions of pitches after training. For example Figure 3b shows the distribution of pitches using ride regression. A difference we observe, is that notes that are not part of D-minor are a little bit too frequent. We can compensate for this by slightly lowering the values in the probability vectors that correspond to non D-minor pitches. With post-processing the pitch-frequencies for regression model is shown in Figure 3c which is a bit more similar to 3a.

4.4 Pitch difference compared to last note

The pitch of a note has a strong connection to the pitch of the note that was played before. Figure 4a shows the frequencies of absolute pitch differences between sequential notes in Bach's fugue. The x-axis encodes the difference in the 12-tone scale (for example from C to B is a difference of 1). Bach's music contains a lot of differences of 1 or 2. Other differences that are common are 5, 7 and 10. Our models that are trained using the first half of Bach's fugue are usually able to generate music with a similar distribution. Figure 4b shows the result using the ridge regression model with no post-processing. What is not the same is that differences of 3, 4 and 9 appear more often than in Bach's fugue. We can compensate this in the post-processing which is shown in Figure 4c.

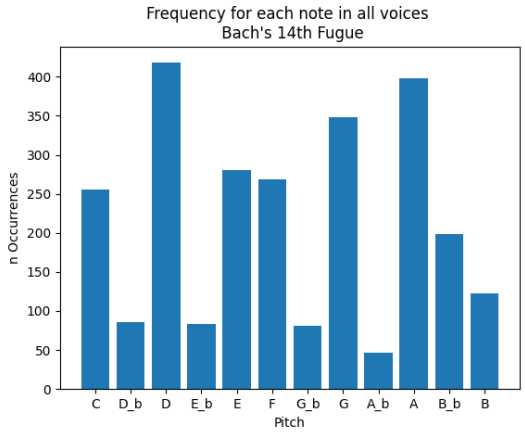
4.5 Pitch difference between voices

When two notes are played at the same time, their difference in pitch can sound harmonic or disharmonic. We used the data of Bach's 14th fugue and show the frequencies of absolute pitch differences between voices in Figure 5a. This plot was created by taking every time-step of the song (16 time-steps per measure) and counted the differences in pitch between all possible pairs of voices (there are 4 voices which means that there are 6 possible ways to compare two voices). Pitches with a difference of 1, 6, 11, 13, 18, 23 and 25 are very unlikely. These differences are disharmonic. For example if one voice plays a C and at the same time another voice plays a B the difference is just 1 which sound very wrong (Try it, if you have a piano!). On the other hand, differences of 3, 7, 9 or 12 are harmonic and sound much better.

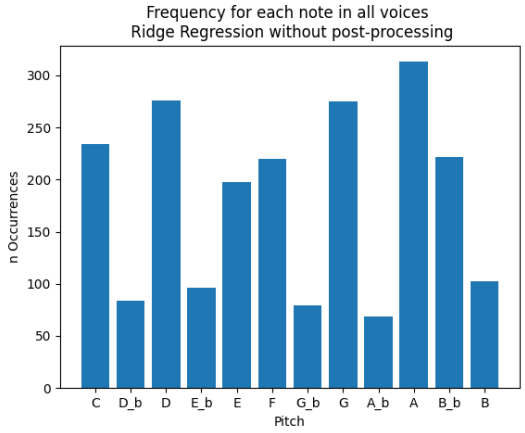
The distributions for the ridge regression model without post-processing is shown in Figure 5b. The model has a tendency to play harmonics more often than disharmonics, but not as much as it should. A key method to make the music generated by the model sound better, is to reduce disharmony during post-processing. To do this we take every combination of notes in regard to the voices. Based on the note ranges discussed in Table 1 there are $23 * 27 * 23 * 27 = 385641$ possible combinations. According to the model, the probability of a combination is the product of corresponding probabilities of the individual notes. During post-processing we can make combinations more likely that contain harmony and less likely if the combination contains disharmony. After post-processing the results for the ridge regression model is shown in Figure 5c.

4.6 Dummy Model

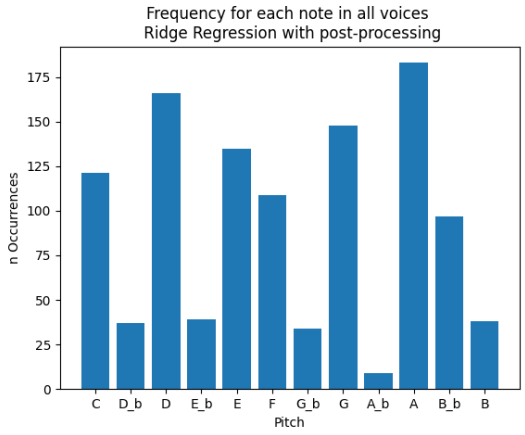
During post-processing the outputs of the models are changed enormously and without these changes the produced music would not sound good. This begs the question if the model outputs are actually important at all. As an experiment we created a Dummy-Model that always predicts equal probabilities for every note. Afterwards,



(a)

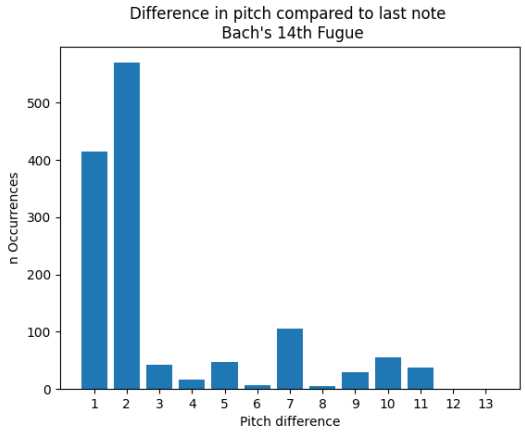


(b)

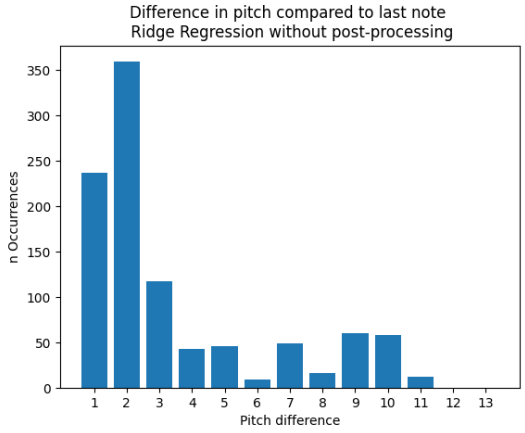


(c)

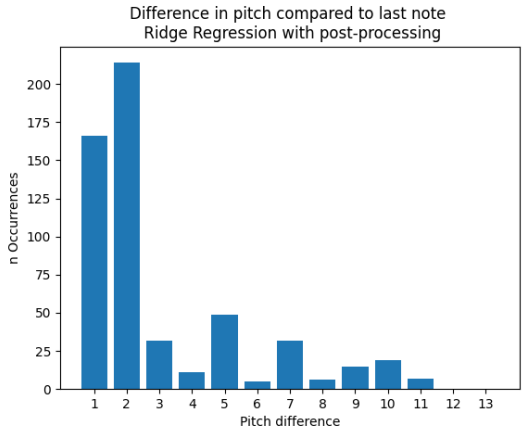
Figure 3: Frequency of each note. (a): For Bach's 14th Fugue. (b): For the ridge regression model with no post-processing(c): For the ridge regression model with post-processing.



(a)

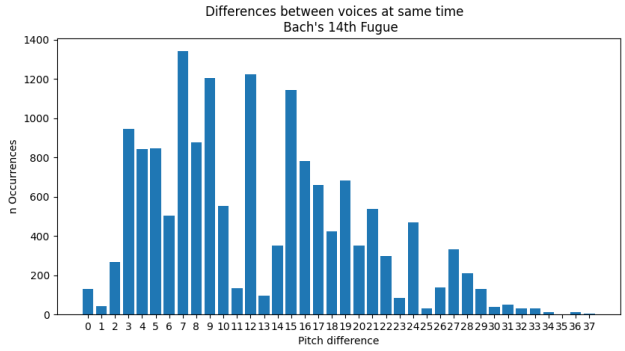


(b)

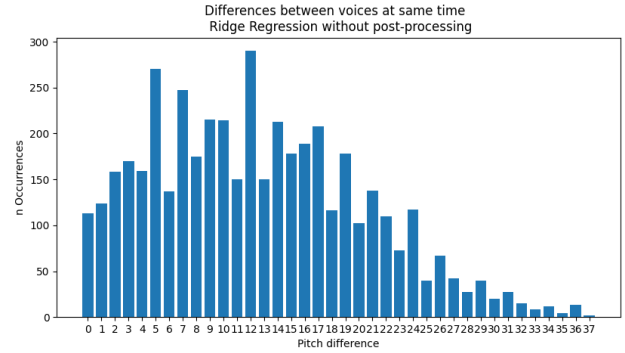


(c)

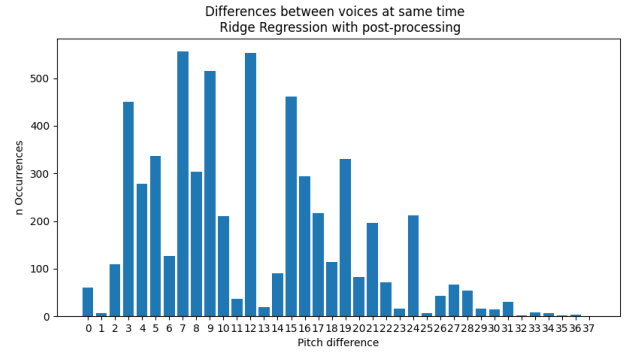
Figure 4: Difference in pitch compared to the last note (a): For Bach's 14th Fugue. (b): For the ridge regression model with no post-processing(c): For the ridge regression model with post-processing.



(a)



(b)



(c)

Figure 5: Difference in pitch between voices at the same time (a): For Bach's 14th Fugue. (b): For the ridge regression model with no post-processing(c): For the ridge regression model with post-processing.

we post-processed the outputs such that the 5 patterns mentioned in sections 4.1 - 4.5 are the same as for Bach's 14th fugue. The resulting music can be found under "dummy_model.mp3" with the corresponding first page of the score sheet in Figure 9. We invite you to listen to it and ask yourself: Does this sound worse than the music produced by the other models? And does it sound more or less like music composed by Bach?

5 RESULTS

5.1 MR and MLP

The ridge regression model using the hyperparameters from Table 3 gave a training accuracy of 100% and a validation accuracy of 87.5%. The MLP with the hyperparameters from Table 2 gave a training accuracy of 88.1% and a validation accuracy of 84.5%.

To generate a sequence of music with these models, as a continuation to the unfinished musical piece, a procedure is used where the MIDI values after post-processing are encoded and concatenated to the window of inputs. The oldest signal in the window is subsequently dropped to keep a window of length N_w. By repeating this for $4 \cdot 121 = 484$ steps, the piece will be extended by a minute, since the piece is supposed to be played at 121 BPM (beats per minute) and there are 4 symbols encoded per beat. The first page of the score sheet corresponding to this generated music is shown in Figure 10 and Figure 11 for the ridge regression and multilayer perceptron respectively. The associated music files are named "pred_ridge_mc.mid" and "pred_mlp_mc.mid" respectively.

5.2 Echo State Network

The obtained model according to the method specified in section 3.3.3, and with parameter values specified in table 4, the model obtained a validation accuracy of 48.2%, and a training accuracy of 66.3%.

Figure 6 shows the activation signals of the first three entries in reservoir signal $\mathbf{x}$, with two randomly generated starting states $\mathbf{x}(0)$, sampled from $\mathcal{U}(-0.5, 0.5)$. The figure shows how the activation signals for each neuron eventually converge. This happens at roughly $n = 140$.

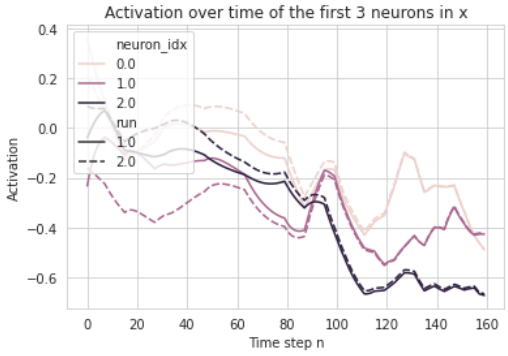


Figure 6: Figure showing the activation signals of the first three neurons from the reservoir. The hue of the lines indicate the index of the neurons, whilst the line style indicates a run with different initial state $\mathbf{x}(0)$.

Figure 7 shows another run with different states $\mathbf{x}(0)$. However, this time the signals are plotted over 500 time steps. The Figure gives an indication of the reservoir dynamics.

Figure 8 shows a histogram of the values $w^{ij} \in \mathbf{W}^{out}$. All values are in the range [-0.047, 0.038], showing the regularization effect of parameter β.

To generate music the trained ESN was driving with the input signal $\mathbf{u}$. When the last entry in the input signal was reached, the

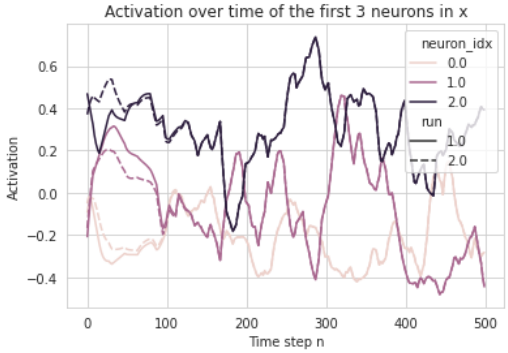


Figure 7: Figure showing the activation signals of the first three neurons from the reservoir.

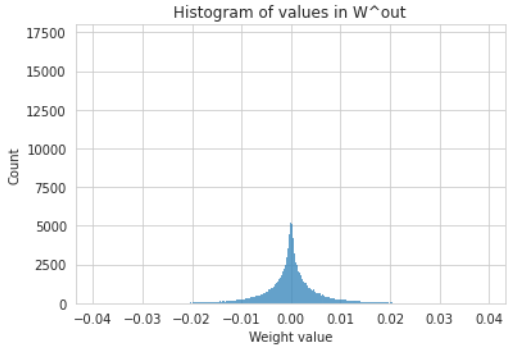


Figure 8: Histogram of the weights in $\mathbf{W}^{out}$.

output vector $\hat{\mathbf{y}}$ of the network was converted to a MIDI value for each respective channel using post processing. The selected MIDI values were then converted into the input vector $\mathbf{u}(n+1)$, and a new output was generated. This process was then continued for a predefined number of time steps. The sheet music for the ESN is shown in Figure 12, and the corresponding music file is named "pred_esn_mc.mid"

What is noteworthy is that the regularization parameter β was set to a value of 1 for the ESN that was used to produce the music. This allowed the model to overfit more to the training data, and thus achieved a lower validation accuracy compared to the aforementioned results. Nevertheless, decreasing the regularization resulted in better music. The parameter setting $\beta = 1$ still had a regularization effect as all values $w^{ij} \in \mathbf{W}^{out}$ were in the range [-0.512, 0.436].

6 CONCLUSION

Generating music using machine learning models is a very complicated task. A music generating model that sound like Bach can not be expected as a result of a student project that only took a few weeks. Nevertheless, we believe that we generated music that is somewhat pleasant for the human ear. We were able to successfully do this with linear regression, an MLP and an ESN. The models could reproduce some of the characteristics of Bach's Fugue. The key aspect to make the models output sound like music was the post-processing step. With post-processing we could control the speed of the generated music and could reduce disharmony. We belief that in order to improve performance, it is necessary to use more training data that includes music example other than Contrapunctus XIV. Incorporating extensive expert knowledge about music theory seems unavoidable for this type of task.

REFERENCES

[1] Murray Gell-Mann. Complex adaptive systems. 1994.
[2] Donya Quick. *Kulitta: A framework for automated music composition*. Yale University, 2014.
[3] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note.
[4] Mantas Lukoševičius. *A Practical Guide to Applying Echo State Networks*, pages 659–686. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi: 10.1007/978-3-642-35289-8_36. URL https://doi.org/10.1007/978-3-642-35289-8_36.
[5] Aulon Kuqi. Art in echo state networks: Music generation. unpublished thesis, 2017.
[6] Gavin M Bidelman. The role of the auditory brainstem in processing musically relevant pitch. *Frontiers in psychology*, 4:264, 2013.
[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

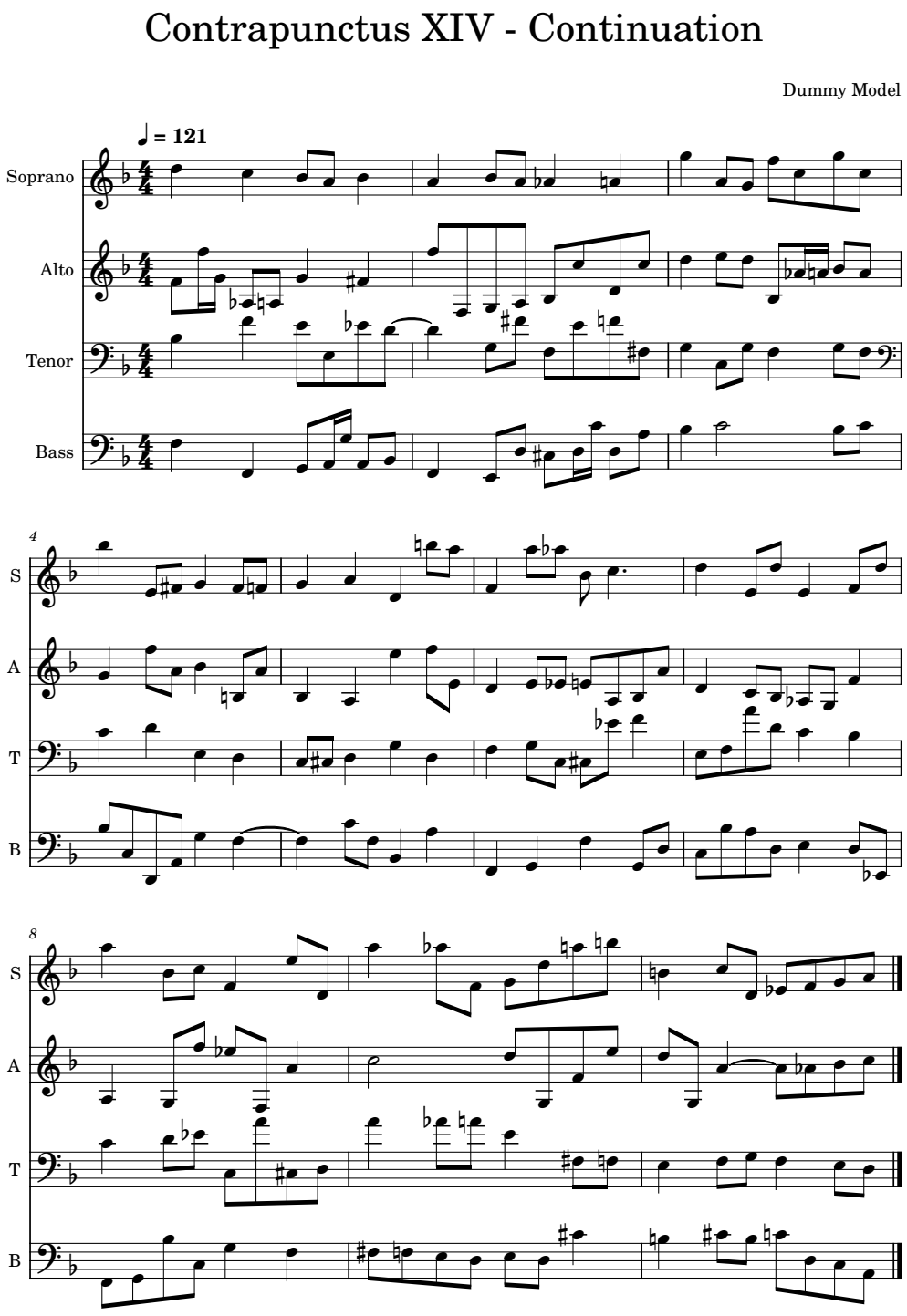


Figure 9: The first page of the score sheet of the generated music by the dummy model as an immediate continuation from the 'ending' of the original piece.

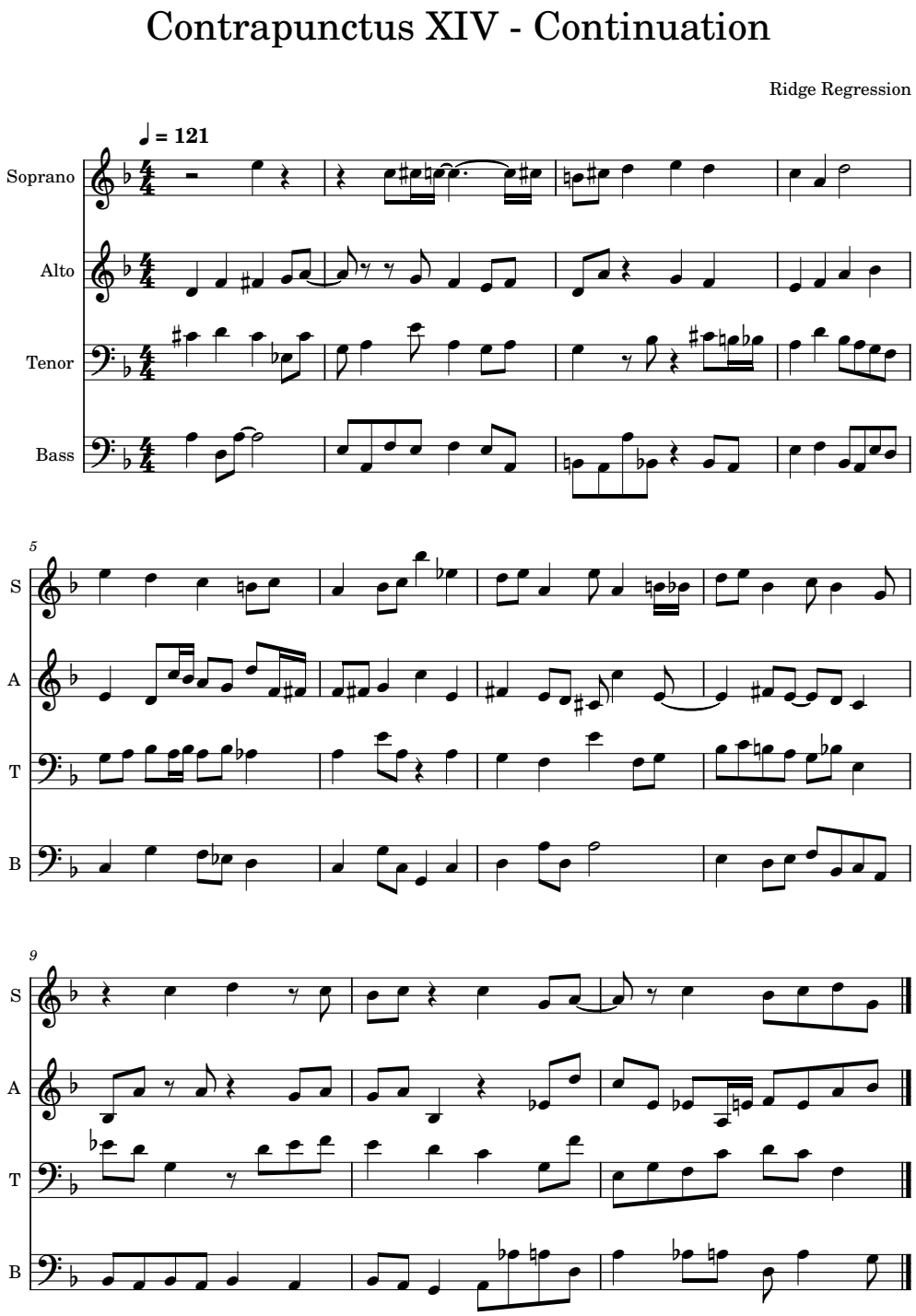


Figure 10: The first page of the score sheet of the generated music by the ridge regression model as an immediate continuation from the 'ending' of the original piece.

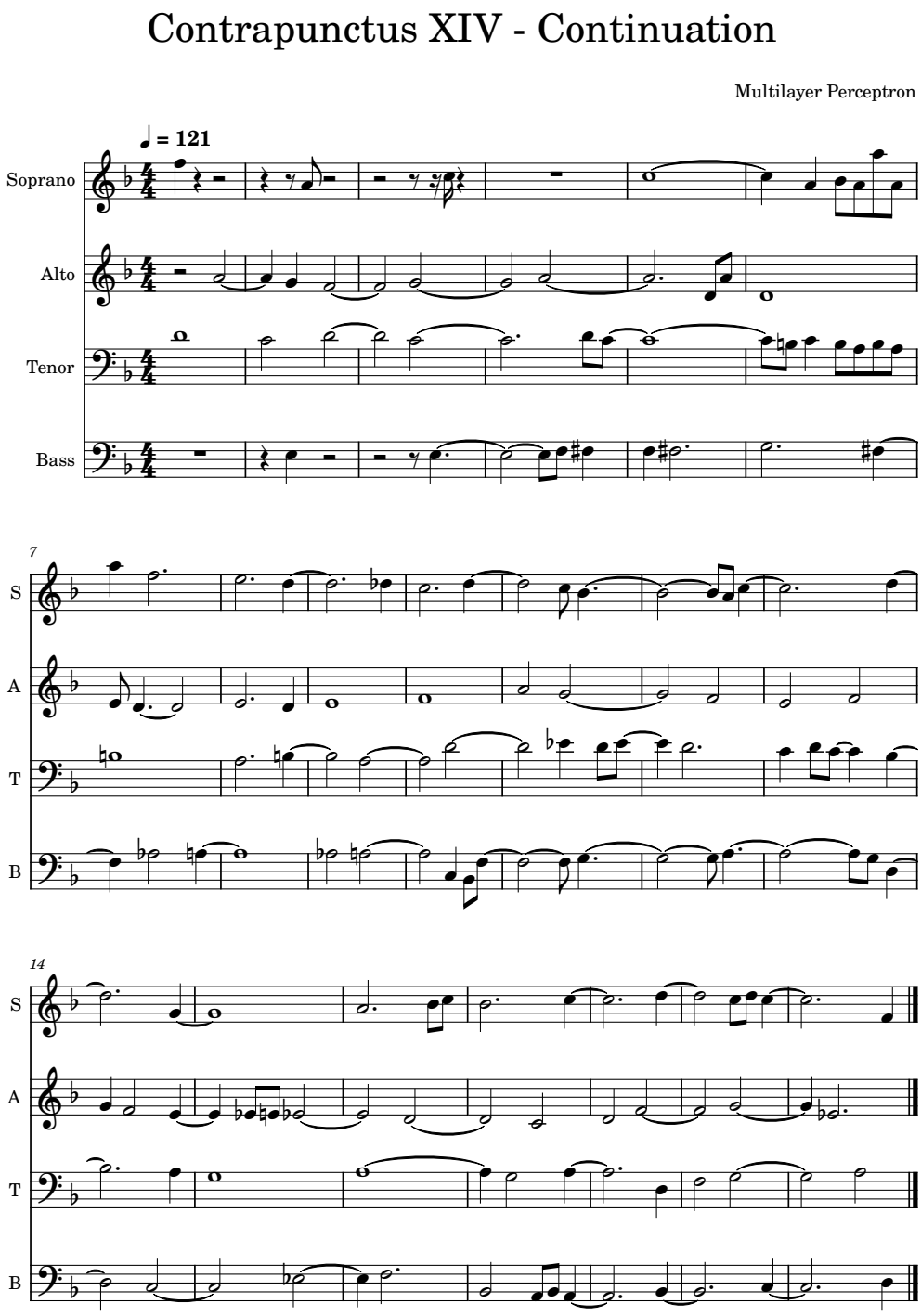


Figure 11: The first page of the score sheet of the generated music by the multilayer perceptron as an immediate continuation from the 'ending' of the original piece.

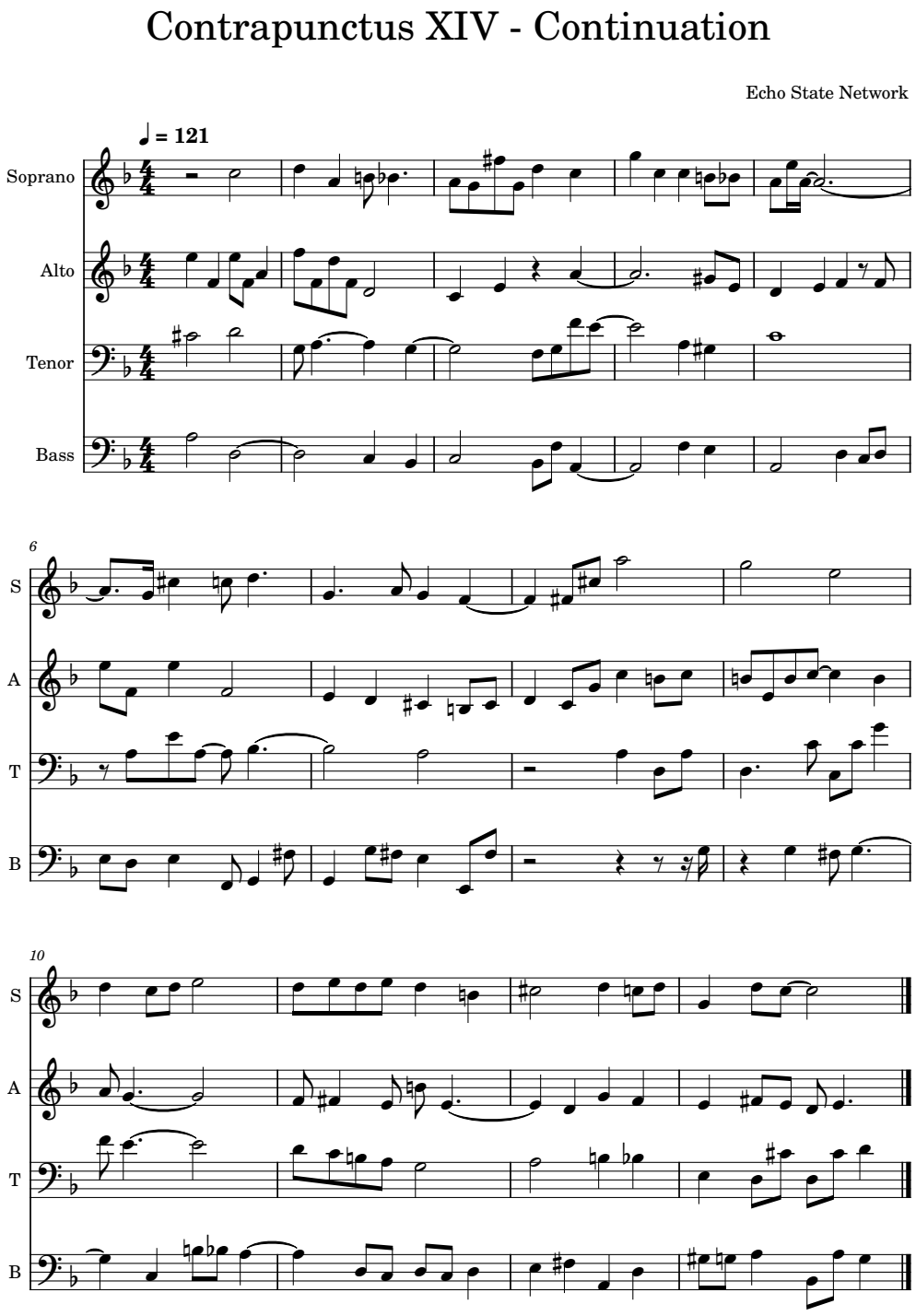


Figure 12: The first page of the score sheet of the generated music by the echo state network as an immediate continuation from the 'ending' of the original piece.