Sample-level sound synthesis with recurrent neural networks and conceptors

Chris Kiefer[1]

[1]**Experimental Music Technologies Lab, Department of Music, University of Sussex, Brighton, UK. BN1 9RG**

Corresponding author:
Chris Kiefer[1]

Email address: c.kiefer@sussex.ac.uk

ABSTRACT

Conceptors are a recent development in the field of reservoir computing; they can be used to influence the dynamics of recurrent neural networks (RNNs), enabling generation of arbitrary patterns based on training data. Conceptors allow interpolation and extrapolation between patterns, and also provide a system of boolean logic for combining patterns together. Generation and manipulation of arbitrary patterns using conceptors has significant potential as a sound synthesis method for applications in computer music but has yet to be explored. Conceptors are untested with generation of long sequences (of more than 20 samples) and multi-timbre pattern generation.

A novel method of sound synthesis based on conceptors is introduced. *Conceptular Synthesis* is based on granular synthesis; sets of conceptors are trained to recall varying patterns from a single RNN, then a runtime mechanism switches between them, generating short patterns which are recombined into a longer sound.

The quality of sound resynthesis using this technique is experimentally evaluated. Conceptor models are shown to resynthesise audio with a comparable quality to a close equivalent technique using echo state networks with stored patterns and output feedback. Conceptor models are also shown to excel in their malleability and potential for creative sound manipulation, in comparison to echo state network models which tend to fail when the same manipulations are applied. Examples are given demonstrating creative sonic possibilities, by exploiting conceptor pattern morphing, boolean conceptor logic and manipulation of RNN dynamics. Limitations of conceptor models are revealed with regards to reproduction quality, and pragmatic limitations are also shown, where rises in computation and memory requirements preclude the use of these models for training with longer sound samples.

The techniques presented here represent an initial exploration of the sound synthesis potential of conceptors, demonstrating possible creative applications in sound design; future possibilities and research questions are outlined.

INTRODUCTION

Machine Learning and Sound Synthesis

Current intersections between sound synthesis and machine learning are evolving quickly. We have seen significant progress in symbolic note generation (e.g. RL Tuner (Jaques et al., 2016), Flow Machines (Ghedini et al., 2016)), parametric control of sound synthesis models (e.g Wekinator (Fiebrink, 2011), automatic VST programming (Yee-King et al., 2018)) and also with current state of the art raw audio generation techniques. These recent advances in raw audio synthesis principally use deep architectures, for example WaveNet (Oord et al., 2016), SampleRNN (Mehri et al., 2016), NSynth (Engel et al., 2017), GANSynth (Engel et al., 2019) and WaveGAN (Donahue et al., 2018), to generate low-level audio representations (sample or spectral level) without using a synthesis engine, working as self-contained models that merge sound generation and control into one.

There is also significant interest from the computer music community in sound synthesis with dynamical and chaotic systems, with strong connections to RNN techniques being used in contemporary deep architectures. This goes back to the earlier work of composers such as Roland Kayn who composed with electronic cybernetic systems, and is reflected in more recent work from, for example, Sanfilippo and

48 Valle (2013) on feedback systems, Ianigro and Bown (2018) on sound synthesis with continuous-time
49 recurrent neural networks, Wyse (2018) on sound synthesis with RNNs and Mudd (2017) on nonlinear
50 dynamical processes in musical tools.
51   The work presented here draws on overlapping research in both machine learning and dynamical
52 systems techniques, in the context of sound synthesis.

53 **Reservoir Computing**
54 While many contemporary developments in machine learning and sound synthesis are based on deep
55 neural network paradigms, pioneering work has also been taking place within the bio-inspired field of
56 reservoir computing (RC) (Schrauwen et al., 2007). Within the RC paradigm, computation is performed
57 using a structure that groups an untrained reservoir with a fixed input layer and a trainable output layer.
58 The reservoir is a complex dynamical system which is perturbed by input signals and transforms these
59 signals into a high-dimensional state space, the current state being dependent on both the current input
60 and on a fading history of previous inputs. The output layer performs a linear transformation of the
61 current reservoir state, and can be trained using supervised methods. RC systems can learn nonlinear
62 and temporal mappings between the input and output signals. A reservoir can be created using both
63 physical systems (e.g bacteria (Jones et al., 2007), a bucket of water (Fernando and Sojakka, 2003) or
64 optics (Duport et al., 2016)) and digital systems. The latter usually take the form of liquid-state machines
65 (Maass et al., 2002) or echo state networks (ESNs) (Jaeger, 2001).

66 *Echo State Networks*
67 ESNs have so far been the primary technique employed for sound and music applications within the RC
68 field. An ESN (see figure 1) uses a randomly generated recurrent neural network (RNN) as a reservoir.
69 This reservoir is connected to inputs and output via single layers of weights. The output layer weights can
70 be trained using linear optimisation algorithms such as ridge regression (Lukoševičius, 2012, p. 10).
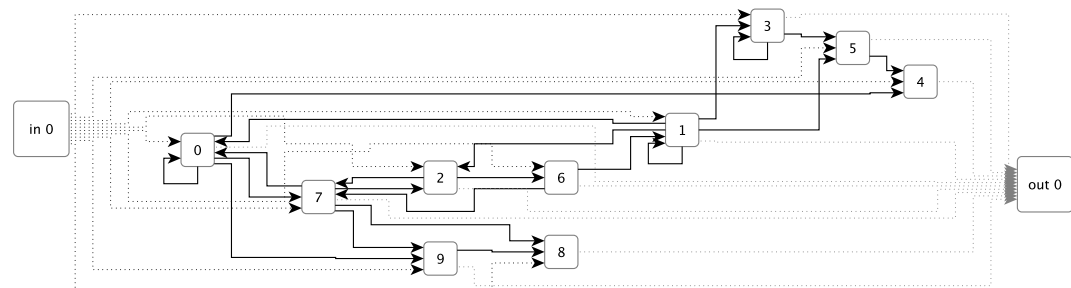
Figure 1. An example of an Echo State Network with ten sparsely connected nodes, single inputs and
outputs, and fully connected input and output layers

71   ESNs are inherently suited to audio applications due to their temporal dynamics. Jaeger's original
72 work with ESNs included examples of models being trained to output discrete-periodic sequences and
73 learning to behave as sine wave oscillators (Jaeger, 2001). Subsequently, ESNs have been applied to
74 a range of creative sound and music tasks. These include symbolic sound generation tasks such as
75 melody generation (Jaeger and Eck, 2006) and generative human-feel drumming (Tidemann and Demiris,
76 2008); direct audio manipulation and synthesis applications bear examples of amplifier modelling, audio
77 prediction and polyphonic transcription (Holzmann, 2009b,a; Keuninckx et al., 2017); they have also
78 been used for modelling complex mappings in interactive music systems (Kiefer, 2014).
79   Under the classical ESN approach, as applied to the task of sound synthesis, ESNs are trained as
80 audio rate pattern generators. A limitation of the classical ESN approach is that it is challenging to learn
81 multiple attractors, corresponding to the generation of multiple patterns on different timescales with a
82 single reservoir, although Holzmann (2009a) offered a solution by decoupling the reservoir with IIR filter
83 neurons.

84 A recent development of the ESN paradigm comes in the form of *conceptors*, an addition to the basic
85 architecture of ESNs that enables the behaviour of the reservoir to be manipulated.

Conceptors

87 Conceptors (Jaeger, 2014a), offer a highly flexible method for generating and manipulating multi-
88 ple patterns within single reservoirs. Conceptors are a mechanism for performing a variety of neuro-
89 computational functions, the ones most relevant to sound synthesis being incremental learning and
90 generation of patterns, morphing and extrapolation of patterns, cued pattern recall, and the use of boolean
91 logic to combine patterns (Jaeger, 2014a). They work by learning the subset of state space visited by
92 an RNN when driven by a particular input. They can then be used to restrict the RNN to operate with
93 this subspace, functioning like an attractor (Gast et al., 2017). The separation of an RNN's state space
94 in this manner allows multiple attractors to be learned using the same network, and for combinations
95 of these subspaces to be used to manipulate the dynamics and output of the RNN. The potential for
96 combination of conceptors is a very powerful feature of this technique, and Jaeger describes boolean logic
97 rules for achieving this (Jaeger, 2014b, p.50). Their strong potential for pattern generation, extrapolation
98 and manipulation, and the combination of continuous and discrete-boolean methods of manipulation are
99 compelling reasons to believe they will have strong applications in the field of audio and creative sound
100 production. Sound generation with conceptors has, however, yet to be explored. Jaeger's original work
101 focuses on generation of short patterns of 20 samples of less, where much longer patterns are required for
102 sound generation. Questions are unanswered concerning whether conceptors will (a) effectively generate
103 longer patterns needed to synthesise audio signals at reasonable sample rates, (b) allow generation and
104 combination of patterns with varied timbres within a single model and (c) produce signals effectively
105 when evaluated with perceptually realistic audio comparison measurements. This paper approaches these
106 questions through the application of conceptor models to a standard sound synthesis technique. It evaluates
107 the effectiveness of conceptors at resynthesising sampled sound, and demonstrates conceptor-based sound
108 synthesis techniques within a granular synthesis paradigm.

New Sound Synthesis Methods

110 A new method of conceptor-based sound synthesis is demonstrated, named *conceptular synthesis*. This
111 is a synthesis method based on granular synthesis. Granular synthesis (Roads, 2004) is based on the
112 sequencing, combination and manipulation of short (typically 20ms - 100ms) windowed segments (grains)
113 of sampled sound. It is a powerful technique for creating and coherently manipulating sound; applications
114 include time and pitch independent stretching of pre-recorded audio. In conceptular synthesis, an RNN
115 model is trained to generate grains, which are recalled by conceptors. The use of conceptor based RNN
116 models allows flexible sound manipulation through creative combinations of conceptors to influence
117 reservoir behaviour. This method is described below; to begin with, a mathematical description of the
118 RNN and conceptor models is presented.

BASIC MODELS

120 This section summarises the fundamental methods used in the creation of the sound synthesis models
121 described below. For a more detailed explanation of these methods, please refer to Jaeger's extensive
122 technical report on conceptors (Jaeger, 2014b). The notation used below will be used throughout the paper.
123 Matrices are represented by capital letters, vectors by lower-case letters, and scalar variables are shown
124 using the greek alphabet. $x(n)$ denotes the state of vector x at timestep n. M' denotes the transposition of
125 matrix M.

126 The basic model is an RNN consisting of ψ nodes, updated according to equations (1) - (3):

$$z(n+1) = Wx(n) + W^{in}a(n+1) \tag{1}$$

127 At discrete time step n, activation levels for each RNN node are stored in state vector x of size ψ.
128 The nodes are sparsely connected such that each node is connected, on average, to 10 other nodes (as
129 recommended in Lukoševičius (2012, section 3.2.2)). Connection weight values are stored in weight
130 matrix W (of size ψ x ψ). Unconnected nodes share a weight value of 0. An input signal vector a (size
131 1) is fully connected to the RNN nodes with the input weight matrix W^{in} (size ψ x 1). W^{in} is generated
132 using linear random values between $-\gamma^{input}$ and γ^{input}. Reservoir weight values are randomly chosen

133 from a normal distribution, scaled according to the spectral radius γ^W (to limit the maximum absolute
134 eigenvalue), and then optimised during training. In equation 1, the activation levels for each node are
135 stored in vector z; in equation 2 these activation values are passed through a nonlinearity, and smoothed
136 using leaky integration.

$$x(n+1) = (1-\alpha)x(n) + \alpha tanh(z(n+1) + b) \tag{2}$$

137 b is a vector of ψ biases, which are generated from a linear random distribution between $-\gamma^{bias}$
138 and γ^{bias}. Scaling in the above cases refers to a tensor being multiplied element-wise by a scalar value,
139 $X = X\gamma$. The *tanh* smoothing function ensures that the reservoir states remain in the range -1 to 1, and
140 introduces a nonlinearity into each node. α is a leaky integration coefficient (Lukoševičius, 2012, section
141 3.2.6). This adds a one-pole lowpass filter to each node; lowering α (between 0 and 1) will slow down
142 reservoir dynamics. This parameter can be fine-tuned to align the temporal dynamics of the reservoir to
143 those of the desired output.

$$y(n+1) = W^{out}x(n+1) \tag{3}$$

144 Output weights W^{out} are a matrix of size 1 x ψ, whose values are optimised during training. The
145 output vector y is a vector of size 1.
146 A model is trained in two phases: (a) audio signals a^j are *stored* (Jaeger, 2014a) in the reservoir, so
147 that they can later be reproduced, (b) a conceptor is calculated for each audio signal. Following training,
148 the model and conceptors are combined and manipulated to synthesise sound.

Storing patterns in the RNN and Calculating Output Weights

150 In this phase of training, a set of randomly generated reservoir weights W^* are adapted so that the model
151 can reproduce an array of driving audio signals a^j, resulting in a new set of weights W. The number of
152 elements in a^j is determined by the sample slicing process detailed below. W^* is optimised such that (a)
153 $Wx(n) \approx W^*x(n) + W^{in}a(n)$, i.e. the reservoir can simulate the driving inputs in their absence, and (b) the
154 magnitudes of weights W are minimised. The resulting weight matrix W is used in all further calculations.
155 W^{in} is no longer required after this step.
156 Training starts with a *washout* phase of length λ where the reservoir dynamics are allowed to settle,
157 reducing influence from any transients that might result from the initial randomised state and that may
158 adversely affect training. The model is subsequently driven by an input for length ϕ. The size of ϕ is
159 task dependent, but should by large enough to collect the reservoir states that are likely to occur when
160 perturbed by the input sequence. ϕ is calculated as an integer multiple of the length of training signal a^j.
161 The training process works as follows: for each pattern a^j, the reservoir with weights W^* is driven from
162 an initial randomised state for $\lambda + \phi$ steps using equations 1 and 2, and the resultant reservoir states are
163 collected.
164 Beginning at timestep 0, the states x from timesteps $\lambda - 1 \ldots \lambda + \phi - 2$ are stored in ψ x ϕ matrix,
165 $\widetilde{X}^j$; states x from timesteps $\lambda \ldots \lambda + \phi - 1$ are stored in ψ x ϕ matrix, X^j; states z from timesteps
166 $\lambda \ldots \lambda + \phi - 1$ are stored in ψ x ϕ matrix, M^j. The remaining states from the washout phase are discarded.
167 The driving signals a^j from timesteps $\lambda \ldots \lambda + \phi - 1$ are stored in 1 x ϕ matrices P^j.
168 These collections are concatenated into matrices $\widetilde{X} = [\widetilde{X}^1 | \widetilde{X}^2 | \ldots \widetilde{X}^n]$, $X = [X^1 | X^2 | \ldots X^n]$, $M =$
169 $[M^1 | M^2 | \ldots M^n]$ and $P = [P^1 | P^2 | \ldots P^n]$
170 W and W^{out} can now be calculated using linear regression. The regression could be solved using a
171 number of techniques; in this case, following Jaeger's initial work on conceptors (Jaeger, 2014b), ridge
172 regression is used:

$$W = ((\widetilde{X}\widetilde{X}' + \rho^W I_{\psi x \psi})^{-1}\widetilde{X}M')' \tag{4}$$

$$W^{out} = ((XX' + \rho^{out}I_{\psi x \psi})^{-1}XP')' \tag{5}$$

173 In both of the above, I is an identity matrix and ρ^W and ρ^{out} are regularisation factors.

Calculating Conceptors

Conceptors can take several forms, the form used in this study is the *alloconceptor* (Jaeger, 2017, p18), a matrix conceptor that is calculated after patterns are stored in the network, and inserted into the update loop of the network at runtime. To calculate a conceptor which will influence the RNN to reproduce audio signal a^j, the reservoir state correlation matrix R is initially calculated:

$$R = \frac{X^j(X^j)'}{\phi} \tag{6}$$

The singular value decomposition (SVD) of R is found

$$U^j S^j (U^j)' = R^j \tag{7}$$

S^j is modified as follows, and used to calculate the conceptor C^j:

$$S^{new} = S^j (S^j + \beta^{-2} I_{\psi x \psi})^{-1} \tag{8}$$

$$C^j = U^j S^{new} (U^j)' \tag{9}$$

β is the *aperture* of the conceptor (Jaeger, 2014b, p43). The aperture is a scaling factor for the amount of energy that is allowed to pass when the conceptor filters the reservoir state in equation 11. The optimal value for β can be found programatically (see below).

The new conceptor can now be inserted to the runtime loop of the RNN, as follows:

$$z(n+1) = Wx(n) \tag{10}$$

This is a modification of equation 1, with the audio signal input removed, as it is no longer needed. $z(n+1)$ is then passed through the leaky integration filter from equation 2, and result is multiplied by the conceptor, as follows:

$$x^*(n+1) = (1-\alpha)x(n) + \alpha tanh(z(n+1)+b)$$
$$x(n+1) = C^j x^*(n+1) \tag{11}$$

Following this step, equation 3 is used to calculate the output signal.

An optimal value for β can be found by observing the *attenuation* $a_{C,\beta}$, which is the level of reservoir signal energy suppressed by the conceptor when the network is updated using equations 10 and 11 (Jaeger, 2014b, p47). Attenuation can be calculated as follows:

$$a_{C,\beta} = \frac{E[||z(n) - x(n)||^2]}{E[||z(n)||^2]} \tag{12}$$

The optimal value for β corresponds to the minimum value of attenuation, calculated by collecting states from the model using conceptors calculated with varying values of β.

The result of this training process is an RNN coupled with a set of conceptors; this model is referred to as a conceptor-controlled recurrent neural network (CCRNN).

These basic methods are used in the experiments below, and expanded on with new techniques that allow training and exploitation of the models for sounds synthesis.

METHOD AND MATERIALS

This project asks how the pattern generation ability of CCRNNs can be applied to the field of sound synthesis. It aims to establish and evaluate the fundamental capabilities of CCRNNs to be trained to reproduce arbitrary audio signals, and to explore their creative affordances. The next section evaluates the potential of CCRRNs to resynthesise sampled sounds.

The ability of trained models to resynthesise the training signal is used as a measure of basic success in sound synthesis. Resynthesis ability is the core indication of sound synthesis quality, although this evaluation only tells part of the story, as the techniques outlined in this project are intended for open-ended use in creative sound synthesis applications. To this end, the project maps out key methods for parameterising and manipulating CCRNN sound synthesis models to create new sonic variations of the original training material, establishing the technical strengths and limitations of CCRNN sound synthesis, and identify open questions for future research in this area.

There is some discussion of processing time to indicate the scale of computation involved with these techniques. The conceptular synthesis experiment was run on a machine with 4.2 GHz i7 CPU and an NVidia GTX 1080 Ti GPU using TensorFlow.

Python 3 source code in Jupyter notebooks for all experiments is provided at `https://github.com/chriskiefer/conceptorSoundSynthesis`. Source code for a working implementation of conceptular synthesis in the form of a drum synthesiser can be found at `https://github.com/chriskiefer/conceptularBeatSynth`.

Error and Similarity Metrics

The experiments in this project focus on the quality of audio signals resynthesised with trained models in comparison to the original training material. In wider literature in reservoir computing, Normalised Root-Mean-Square Error (Lukoševičius, 2012, p2) (NRMSE) is commonly used to measure similarity between signals. NRMSE does not reflect perceptual aspects of sound similarity; these are crucial to understanding the results therefore a different metric is required. Mel-Frequency Cepstral Coefficients (MFCCS) have been shown to be a robust measure of timbral similarity (Jensen et al., 2009), and a good model of perceptual timbre space (Pampalk et al., 2003). They are widely used for music information retrieval tasks (Hamel and Eck, 2010) across a variety of use cases (e.g. (Yee-King et al., 2018; Khunarsal et al., 2013).

For the purpose of audio signal comparison, MFCCs were calculated from a 2048 point windowed FFT. The sounds being analysed were short (typically between 200 and 5000 samples); in order to capture the detail of the timbral envelope at all stages, a short 64 sample hop-size was used. The FFT results from each window were used to calculate 20 MFCCs. The first MFCC coefficient from each window was discarded as it does not give information about timbre (Jensen et al., 2009). The remaining coefficients from each window were appended to create a single feature vector. The feature vectors from two sounds were compared using NRMSE to arrive at an error value that reflects the similarity between the two sounds. This will be referred to as the *MFCC error*, with lower values indicating higher similarity.

Where relevant, waveforms and spectrograms are displayed for visual comparison, and audio is included in the dataset accompanying this paper.

CONCEPTULAR SYNTHESIS

Conceptular synthesis is a new sound synthesis method based on CCRNNs, and expanding on an established method of sound synthesis, *granular synthesis*. In granular synthesis, sound is broken into small parts (grains), which are recombined in varying ways to produce new sounds. The theoretical roots of this method lie in Gabor's (1947) theory of acoustic quanta, and in the compositional theory of Xenakis (1971). Digital implementations of the technique were developed by Roads (1978) and Truax (1986). Granular synthesis offers methods for further sound manipulation techniques including timestretching (Truax, 1994) and corpus-based concatenative synthesis (Schwarz, 2006).

Jaeger's demonstration of the ability of CCRNNs to be trained to generate arbitrary sequences suggests that they could become powerful sound synthesis tools, as they can theoretically reproduce arbitrary audio waveforms. However they are pragmatically limited to playing relatively short sequences; the reason for this is that the computational complexity of the model increases with ψ, and the desired size of ψ increases with the length of training sequences. However, if a model is trained to reproduce a set of

250   shorter sound sequences, then granular synthesis techniques can be used to recombine these sequences to
251   produce longer sounds. Conceptular synthesis therefore expands upon granular synthesis, by dynamically
252   generating grains using conceptors rather than replaying grains from sound sample data. Grain patterns
253   are *stored* in an RNN, and conceptors force the RNN to replay specific grains. A granular synthesis-style
254   control mechanism is used to switch conceptors so that the model generates a sequence of short signals,
255   which are combined into a longer waveform. The use of dynamic models instead of static sample data
256   broadens the sonic potential of conceptular synthesis in comparison to classical granular synthesis, as the
257   models provide further possibilities for creative manipulation.
258   This section begins by describing conceptual synthesis techniques in detail, illustrated by an example
259   demonstrating resynthesis of a kick drum sample. Resynthesis quality is then explored in an experiment
260   comparing conceptual synthesis to a baseline method using feedback induced oscillation. Following this,
261   extended sound synthesis options are described.

Conceptular Synthesis Techniques

263   Conceptual synthesis works by subdividing the audio training data into a set of sub-sequences, and
264   learning an RNN and set of conceptors that can regenerate these sub-sequences, with the intention of
265   resynthesising the audio sample by recombining the model-generated sequences. The audio training data
266   may be a single audio sample, or it may be composed of multiple audio samples to create a model that
267   will create variation between different samples.

Slicing Training Data

269   Two methods of slicing the audio were explored: using constant or variable values for the sub-sequence
270   size μ. Using constant μ, the sample is divided into a set of equal length signals a, of length μ samples
271   each. The optimal value of μ for a particular sound sample can be determined programmatically through
272   a grid search. The constant μ slicing method runs into problems, particularly when there are dominant
273   frequencies in the source audio whose wavelength is longer than μ. Slicing lower frequency waveforms at
274   non-zero points creates high-frequency artefacts in the training data, which can distort the training process
275   because these artefacts are not present in the original training material. For example, consider the kick
276   drum waveform in figure 2. The sample has low frequency components with varying long wavelengths,
277   and there is no constant value of μ that will avoid slicing at non-zero points. To avoid this, the sample can
278   be analysed using a zero-crossing detector, resulting in a set of points i at which the sample is sliced to
279   create a set of driving audio signals a.

$$i = \{n | y(n) \geqslant 0 \wedge y(n+1) < 0\} \tag{13}$$

Hyperparameters and Training

281   A key parameter is the reservoir size ψ. ψ correlates with the memory capacity of the reservoir; it should
282   be at least equal to the number of independent variables needed for the task the model is being trained
283   for (Jaeger, 2002). As we increase the quantity and length of training signals, we need to increase ψ to
284   give the model the capacity to learn them. However increasing ψ makes computation more expensive (at
285   approximately $O(N^2)$), so there are practical limits to this value.
286   The other key parameter is the leak rate α. Lowering α has the effect of filtering out high frequencies
287   in the reservoir activation levels x, and therefore slowing down the behaviour of the RNN. α needs to
288   be chosen such that the model can reproduce the frequency content from the source audio, for example
289   a sound with dominant low frequencies like the kick drum in figure 2 will need a network with slowly
290   changing activations, and therefore a low α value. Optimal values can be found using a grid search.
291   After choosing hyperparameters, a model is trained using the techniques set out earlier in the paper.
292   The output is a set of conceptors, calculated to reproduce each audio signal a^j.

Resynthesis

294   To resynthesise the training signal, the CCRNN is initially run for λ steps with the first conceptor C^0.
295   The model is then run with each conceptor C^j inserted into the update loop, as described in equations 10
296   and 11, to create a set of output signals q. The number of samples for which each conceptor is used in
297   the runtime loop corresponds with the length of the audio signal from which the conceptor was trained.
298   The algorithm makes a short linear crossfade between conceptors, over a small percentage of the pattern
299   length. This prevents artefacts appearing from instantaneous switching of conceptors. Finally, output
300   signals are appended to create the waveform $k = [q^0|q^1|\ldots|q^n]$.

301   *Example: Resynthesis of Kick Drum*
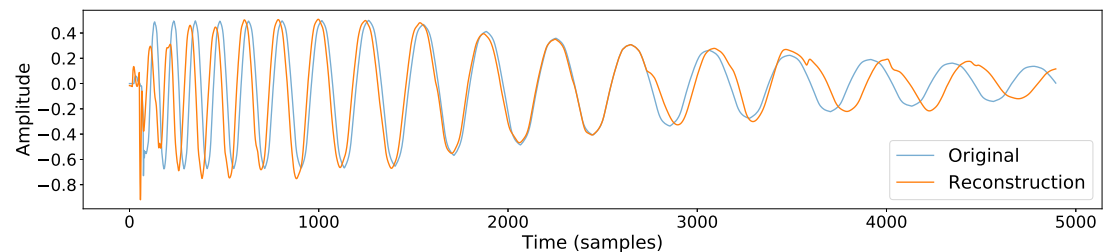
Figure 2. A comparison of the original kick drum sample and the output of the trained CCRNN model.

302   The output of a trained conceptular synthesis model is now shown, to illustrate how this technique can
303   be applied. A CCRNN model was trained to resynthesise a kick drum. The original audio (see figure 2
304   and supplemental audio S5) was re-sampled at 22050 Hz (half of *CD-quality*), in order to reduce the CPU
305   load of training.
306   The sample was segmented using the zero-crossing method, and the model was trained with the
307   parameters as shown in table 1, with leak rate $\alpha = 0.15$. The kick drum sample was resynthesised with
308   the crossfade length set at 5% of signal length. The result is shown in figures 2 and 3, and included in
309   supplemental audio S7. Both show a close reconstruction of the original, with the addition of some high
310   frequency artefacts.
311   This example is not compared quantitatively to the original, instead this is done systematically in the
312   experiment below.
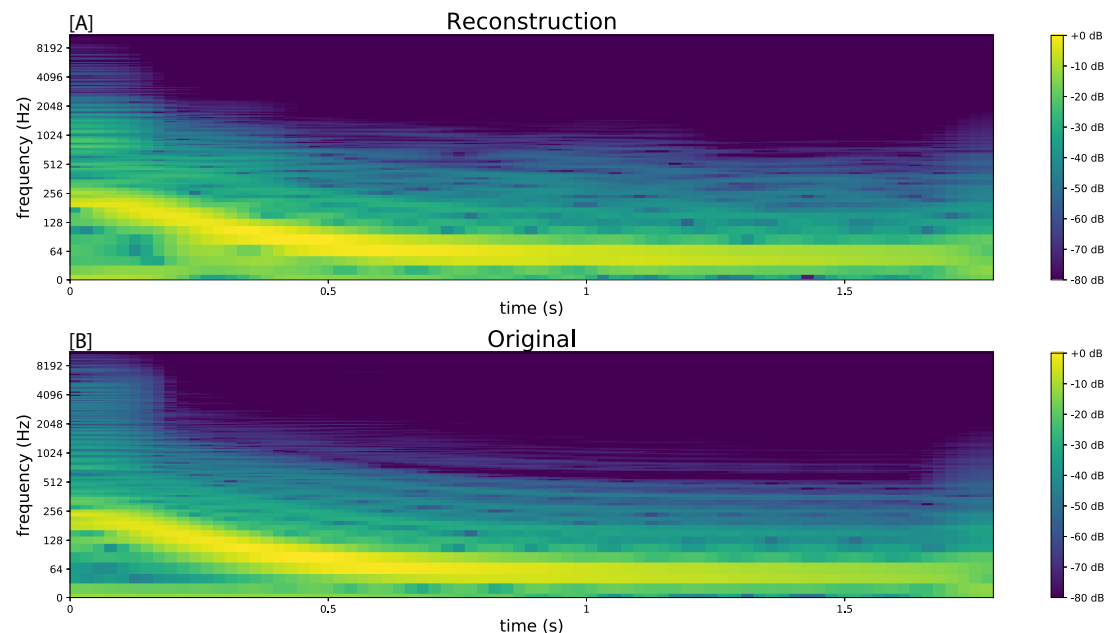
Figure 3. Spectrograms comparing the resynthesised kick drum [A] to the original sample [B].

313   **Measuring Resynthesis Quality**
314   We have seen the potential capabilities for conceptual synthesis to resynthesise audio from trained
315   CCRNN models. These trained models offer extensive creative possibilities for sound synthesis, which
316   are detailed later in the paper. Before exploring these avenues, the quality of resynthesis method should
317   be evaluated, in comparison to an existing baseline.

Baseline Models: Echo State Networks with Stored Patterns and Feedback

The closest comparable method to conceptual synthesis is to use ESNs with output feedback as trainable oscillators. Feedback is used to guide reservoir dynamics to a stable limit cycle where the model is outputing a desired signal. Early research on ESNs (Jaeger, 2001) showed their capability for self-driven oscillation using output feedback. More recent research has focused on increasing the effectiveness of output feedback by adapting the weight matrix W to increase the accuracy of reproduced patterns. Jaeger (2014a, p2) summarises this family of techniques including *self-prediction* (Mayer and Browne, 2004), *self-sensing networks* (Sussillo and Abbott, 2012) and Jaeger's own method for storing patterns which has already been detailed in this paper. The reservoirs for both techniques are created using the same stored pattern method, but then the methods diverge. These baseline models will be referred to as ESNSPFs.

The training method for ESNSPFs works as follows: a set of audio signals is *stored* in an RNN, using an identical approach as detailed earlier with equations 1, 2 and 4. The result of this process is to adapt the randomly initialised weight matrix W^* to trained matrix W.

This model is now used for training a set of output weight matrices, such that each matrix $W^{out\,j}$ will be used to recreate audio signal a^j. To train an ESNSPF using output feedback, we train an output layer that will predict the the next sample from the input signal. The model is driven with signal a^j using equations 1 and 2. Following a washout period, states x(n) from timesteps λ to $\lambda + \phi - 2$ to are stored in $\psi \; x \; \phi - 1$ matrix X. Samples from the audio signal a^j from $\lambda + 1$ to $\lambda + \phi - 1$ are stored in 1 x $\phi - 1$ matrix P. The output weight matrix $W^{out\,j}$ can then be calculated using equation 5. Feedback models are sensitive to initial conditions; a brute force search can be used to find a good value for the initial state $x(0)$, stored to use later as x_{cue}.

At runtime, the model is run by feeding the output back into the input with a modification of equation 1, starting from initial state x_{cue}. To reproduce audio signal a^j, the model is updated as follows:

$$z(n+1) = Wx(n) + W^{in}y(n)$$
$$x(n+1) = (1-\alpha)x(n) + \alpha tanh(z(n+1)+b) \tag{14}$$
$$y(n+1) = W^{out\,j}x(n+1)$$

To recreate a sequence of patterns or grains that comprise a longer audio signal, each output matrix $W^{out\,j}$ is sequentially inserted to the runtime loop for a period matching the length of the audio signal a^j.

Dataset

Sounds from the Ixi Lang data set (Magnusson and Kiefer, 2019) were used to compare the two methods. This is a collection of the sounds that accompanies the Ixi Lang live coding environment. The collection represents a wide variety of short samples that are used for live performance. There are 127 audio clips, lasting between 5ms and 36.3s (mean: 1.75s).

Method

Each sample was resampled at 22050Hz, normalised and scaled by half, and truncated to a maximum of 5000 samples (or 0.23s); this was to create a balance between computation demands on the model, and providing enough material to make a useful comparison. The sample was then sliced; the zero-crossing method (equation 13) was used as this is more widely applicable to a range of timbres. A maximum of 150 patterns were kept from the slicing process, again to reduce demands on computation time and memory for storing conceptors. This process resulted in a set of patterns extracted from each sample in the dataset. ESNSPF and CCRNN models were trained to resynthesise each pattern set, and then used to resynthesise the corresponding samples.

Table 1 shows the hyperparameters used for both models types, chosen to match the models as fairly as possible for comparison. Some were fixed, others were searched for within the ranges shown. Fixed hyperparameters were chosen based on experimental reports in wider literature (Lukoševičius, 2012; Jaeger, 2005, 2014b), and through extensive manual experimentation. Some parameters were deemed to be sensitive to the training material, for example α needs to be tuned to match the frequencies in the source, in which case the values were optimised through automatic search as detailed below.

The use of randomly determined weight values is fundamental to the design of both models, however this leads to variance in training results. To mitigate against this variance, the training process was run multiple times at each hyperparameter setting and the best models chosen. It is acknowledged that, as

366 with all non-deterministically generated models, it is possible that the most optimal model will not be
367 found, but running the training process multiple times will increase the possibility of finding a better
368 model. Further to this, variances in this process will be averaged out across the 127 samples in the dataset.

369 **CCRNN Models** The key hyperparameter for this model type was the leak rate α, which needed to be
370 tuned to match the frequency response of the model with the frequencies needed for resynthesis
371 of the training material. For each sample in the dataset, α was determined through a grid search
372 (as recommended in (Lukoševičius, 2012, section 3.3.4)) of values $\{0.05, 0.15, 0.25 \ldots 0.95\}$. The
373 grid search evaluated models at each of these settings at a lower model size ($\psi = 600$) in order to
374 save computation time, evaluating 5 different models and recording the best MFCC error. This
375 process resulted in an optimal value of α which was used to evaluate 10 models at a higher model
376 size ($\psi = 900$). The highest score of these models was recorded.

Table 1. Hyperparameters in the resynthesis quality experiment

Model Type	ψ	α	γ^W	γ^{input}	γ^{bias}	λ	ρ^W	ρ^{out}
CCRNN	900	$\{0.05, 0.15 \ldots 0.95\}$	1.5	1.2	0.3	50	$1e-5$	$1e-5$
ESNSPF	$20 \ldots 900$	$0 \ldots 1$	1.0	$0 \ldots 1$	$0 \ldots 1$	50	$1e-5$	$1e-5$

377 **ESNSPF Models** Models trained with output feedback showed sensitivity to four parameters: model
378 size ψ, leak rate α, bias scaling γ^{bias} and input scaling γ^{input}. Interestingly, this technique showed
379 sensitivity to ψ, while CCRNN models show consistent improvement with larger ψ. A four-
380 dimensional grid search was impractical due to computational demands, instead a microbial genetic
381 algorithm (Harvey, 2009) was used to find optimal values, conducting a stochastic evolutionary
382 search of parameters in the ranges shown in table 1. The fitness function selected the best model
383 based on evaluations of 10 models created with identical hyperparameters. Each model was then
384 evaluated with 10 randomised starting states x_{cue}, and the x_{cue} with the lowest error was chosen.

385 ***Results***
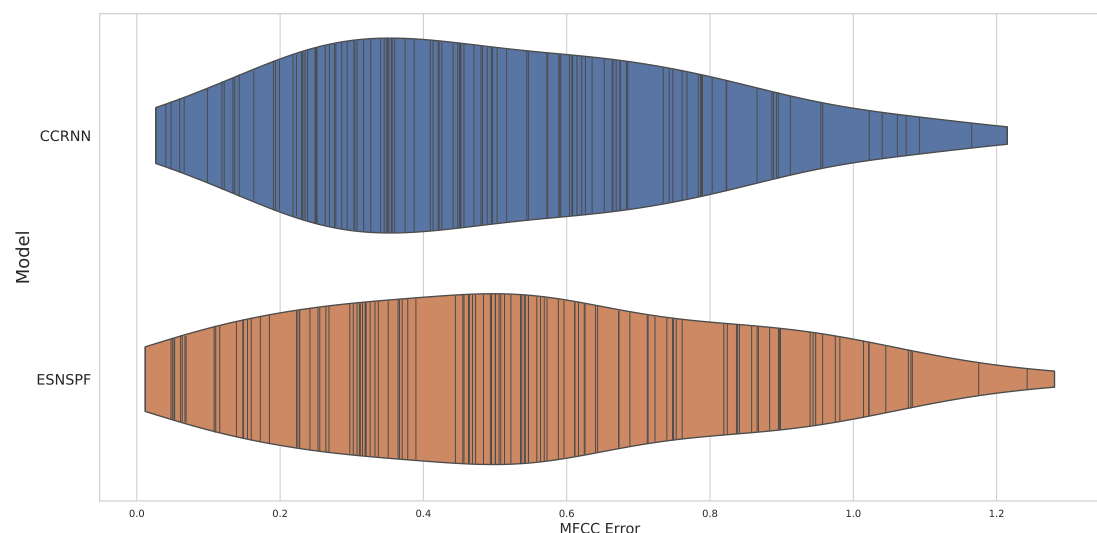
Figure 4. A violin plot of MFCC error scores for each model type, for resynthesis of samples in the Ixi
Lang dataset. The plot shows the distribution of scores, with real values marked by vertical lines.

386 The experiment resulted in an error score for each model type for each of the 127 samples in the dataset.
387 The two methods gave comparable results (figure 4) (CCRNN: mean 0.542, median 0.482, ESNSPF:
388 mean 0.568, median 0.536). There was no significant difference between the model types (wilcoxon
389 signed rank test: $W = 3872, p = 0.644$). These results are discussed further below, after contextualisation
390 within extended synthesis techniques.

Extended Conceptular Synthesis Techniques

The above experiment demonstrates that CCRNN models are capable of sound synthesis quality comparable to ESNSPFs. Moving beyond this baseline, they offer a wider range of creative sound synthesis possibilities.

Extended Sound Synthesis Parameters

The sound generation algorithm can be manipulated with three key parameters: speed, leak rate scale, and weight scaling. The speed parameter changes the amount of time in which the algorithm waits until a new conceptor is plugged in to the RNN update loop. For example, a speed of 0.5 results in two cycles of a pattern being played for each conceptor C^j and resulting in a rendered sample that is twice the length of the original. At negative speeds, a sample can be crudely reversed by playing the patterns in reverse sequence. This parameter can have the effect of timestretching, i.e. extending or compressing the length of a sound, independent from its pitch. Supplemental audio S8 presents an example of timestretching from 50% up to 800% in 50% steps with a CCRNN model trained on a kick drum sample.

The leak rate α can be scaled during resynthesis, by updating equation 2 as follows:

$$\alpha^{scaled} = \alpha \xi^{\alpha}$$
$$x(n+1) = (1 - \alpha^{scaled})x^{target}(n) + \alpha^{scaled}tanh(x^{target}(n+1) + b)$$

(15)

ξ^{α} becomes an useful parameter in resynthesis for control over timbre and pitch (which is explained later when discussing pitch-controlled oscillators). It should be limited such that α stays between 0 and 1.

Weight scaling can be introduced by updating equation 1 so that the weight matrix is linearly scaled by scalar ξ^W:

$$W^{scaled} = W \xi^{W}$$
$$z(n+1) = W^{scaled}x(n) + W^{in}a(n+1)$$

(16)

This causes timbral changes in the rendered sample whose characteristics are based on the random make up of the RNN. There is some consistency in this parameter in that when raised, more high frequency content tends to be introduced. At higher values, the RNN can behave in *musically interesting* non-linear ways. Below a lower limit (model dependent), the model output tends towards silence.

Further manipulations of sound can be achieved by manipulating conceptors.

Extending Sound Synthesis with Conceptor Logic

The use of conceptor logic and conceptor manipulation is where this mode of sound synthesis significantly moves on from standard granular synthesis features, and brings its own unique possibilities. New conceptors can be created using boolean logic rules; Jaeger (2014b, p.52) defines formulae for AND, OR and NOT operations. Boolean operations provide a system of logic with which to combine conceptors, with applications in classification and memory management. In the case of conceptular synthesis, logic operations provide a wide range of creative possibilities. Conceptors can be logically recombined to create new timbral variations. Two examples are now given:

Example 1

A CCRNN model was trained to reproduce a snare sample (supplemental audio S1). Each conceptor in the snare model C^j was combined with the subsequent three conceptors to make a new set C_2, using the rule $C_2^j = C^j \vee C^{j+1} \vee C^{j+2} \vee C^{j+3}$ This resulted in a variant on the original snare sound shown in figure 5 (Supplemental Audio S9).

Example 2

A new set of conceptors C_3 was made by combining each conceptor in the set with a random choice of two other conceptors in the set $C_3^j = C^j \vee C^{random1} \vee C^{random2}$. This is designed with the intention of keeping the main structure of the sample but introducing random variations. Figure 6 shows the resulting waveforms from 4 separate iterations of this process using the snare model detailed above (Supplemental Audio S10).

In both these examples, the variations are subtle, and the renderings suffer from some audible artefacts, however this does point to generative possibilities that are worthy of further research.
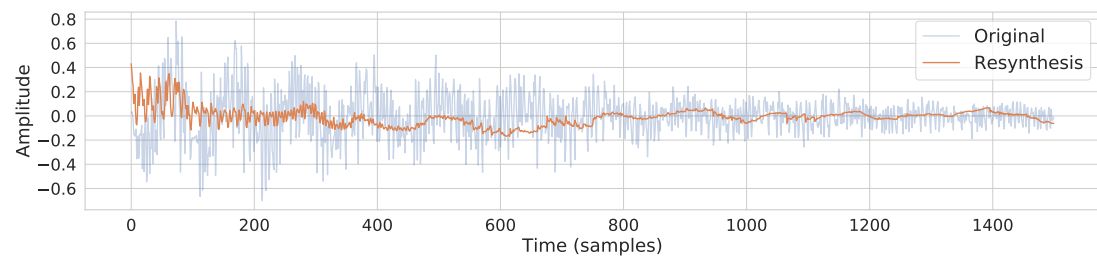
Figure 5. The waveform of a variant of a snare sample, produced using boolean logic
$C_2^j = C^j \vee C^{j+1} \vee C^{j+2} \vee C^{j+3}$
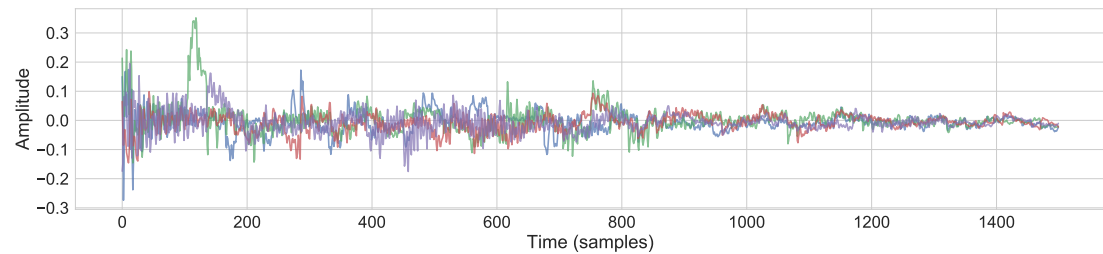
Figure 6. Waveforms showing generative variants of the snare sample, using the boolean logic rule
$C_3^j = C^j \vee C^{random1} \vee C^{random2}$

Sound Morphing with Interpolated Conceptors

Jaeger (2014b, p.42) demonstrated shape morphing between heterogeneous patterns using conceptors. This same technique can be applied within conceptular synthesis to morph between sounds. It is not within the scope of this paper to analyse the quality of sound morphing using conceptular synthesis in comparison to other techniques, rather just to demonstrate that sound morphing is a creative possibility with CCRNNs, and to outline how it can be achieved.

Morphing can be implemented by creating a linear combination of conceptors to interpolate between the two patterns the conceptors were trained to recreate. Equation 17 shows how this can be done with two conceptors, where ω is the morphing factor.

$$x(n+1) = ((1-\omega)C^i + \omega C^j)tanh(Wx(n)+b) \tag{17}$$

Varying ω between 0 and 1 forces the RNN to create a morph between the patterns represented by the two conceptors. When $0 \leq \omega \leq 1$, the mix of conceptors will interpolate between patterns. However, when ω is outside of this range, the mix of conceptors can extrapolate between patterns.

The intention of morphing between sounds is to create a new mixture of sounds that retains the shared perceptual properties of the original sources (Slaney et al., 1996). Morphing was investigated with conceptular synthesis by training a CCRNN recreate patterns from two different samples: the snare sample already discussed, and a short bongo sample (Supplemental Audio S12). An 800-node network was trained for recreation of 100 individual patterns (fixed length 15) for each sample, resulting in two sets of conceptors, C_{snare} and C_{bongo}. Morphing was achieved by creating a new set of conceptors based on a linear mixture of the trained conceptors, for each pattern segment. The results demonstrate a morph between samples that is different from a linear mixture of the two samples. Supplemental figure S1 shows how the time-domain waveform result varies over an 11 point morph from $\omega = 0$ to $\omega = 1$, and the result can be heard in supplemental audio S3. For comparison, supplemental figure S2 and audio S4 demonstrate a linear mix between amplitude values with the same two samples.

Boolean conceptor logic can also be used for sound morphing. For example, a set of conceptors $C_{bongoSnare}$ was created, with each element combining elements from the snare and the bongo $C_{bongoSnare}^j = C_{bongo}^j \vee C_{snare}^j$. A sample rendered with this conceptor set contains characteristics of both sounds (Supplemental Audio S11).

Equivalent Techniques with ESNSPF Models

Where possible, equivalent extended sound manipulation techniques were attempted with ESNSPF models. There is no parallel using ESNSPF models for conceptor logic. However, it is possible to manipulate ξ^α and ξ^W during resynthesis, and also to introduce the same timestretching mechanism as for CCRNN models. Attempts at all of these methods however did not lead to satisfactory results. Timestretching with ESNSPFs only works when slowing down playback at integer subdivisions of the original speed. At other ratios, the models tend to quickly converge to silence. When altering ξ^α and ξ^W, the models tend towards producing high-amplitude artefacts. To demonstrate this, all models trained in the experiment above were tested with either ξ^α or ξ^W set to values $\{0.7, 0.75 \ldots 1.3\}$. Figure 7 shows the average percentage change in standard deviation of amplitudes of resynthesised waveforms compared to a waveforms generated with $\xi^\alpha = 1$ and $\xi^W = 1$. For ESNSPFs, this change in standard deviation is high compared to the same measurement for conceptular synthesis models, reflecting high amplitude artefacts introduced by the change in these values. CCRNN models however remain close to the original amplitude range (although with some very small variation).
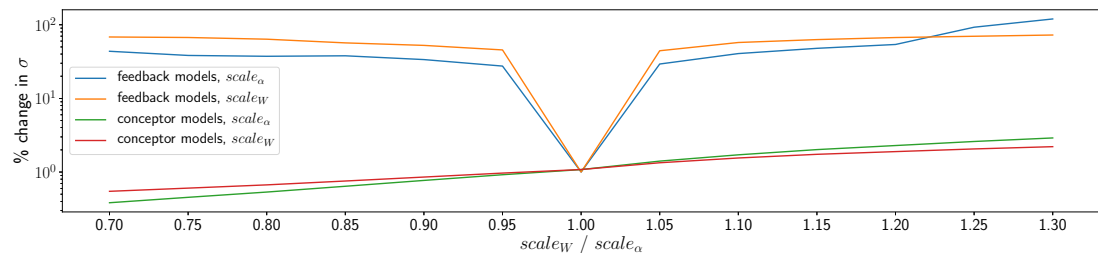
Figure 7. Demonstration of the effects of ξ^α and ξ^W on waveform amplitude, reflecting high-amplitude artefacts in the ESNSPF models.

Discussion

This work demonstrates how CCRNNs can be used to resynthesise short samples by dividing the sample up into short signals and training a conceptor for the reproduction of each one. An experiment on resynthesising the Ixi Lang dataset show that conceptular synthesis can achieve comparable resynthesis quality to echo state network models that use stored patterns and feedback, when measured using MFCCs. Both types of model could not perfectly reproduce the training samples, but were able to make reasonable reconstructions. There was some variance in resynthesis quality across the results, the causes of which are the topic of future investigation.

CCRNN models offer malleable sound synthesis possibilities when manipulated using inherent runtime parameters and through conceptor combinations created either by interpolation or by boolean logic. These techniques provide unsatisfactory results when attempted with equivalent implementations in ESNSPF models. This is likely to be due to the high sensitivity to initial conditions of models that use output feedback. While they can provide good quality resynthesis, ESNSPF models are brittle in nature, while CCRNNs show themselves to be highly robust to manipulation during resynthesis, making them extremely valuable as creative tools.

There is natural variability between models, due to random initialisation of the RNN. This variability is minimised when using the network within normal constraints, however when pushed into non-linear modes of behaviour by, for example, changing the value of ξ^W, a higher variability between different CCRNNs can be observed. This behaviour for a particular network may turn out to be *musically interesting*, lending conceptual synthesis potential for serendipitous discovery of new sounds, and a level of generative unpredictability that is often valued by musicians (McCormack et al., 2009).

Is it possible that, by using the reconstruction error as the evaluation metric, this experiment has produced models that are flawed because of overfitting? In this context, overfitting would be the production of generative models that are only useful for reproducing the training data, but do not function well as *malleable* creative tools when manipulated using the techniques described above, ie. they would have a limited *aesthetic state space* (Eldridge, 2015). Conventionally in ESN research, the complexity of reservoirs, approximately determined by the size of N, has been established as a factor in overfitting in discriminative models (Wyffels and Schrauwen, 2010), however there is very little research on the

nature of overfitting in generative ESNs and related models, especially with regards to creating *malleable* models. Jaeger acknowledges that there are open questions around the concept of overfitting (Jaeger, 2005). The experimental results do not indicate that N is a factor in overfitting in this experiment. In the search for ESNSPF models, N was optimised through evolutionary search. The resulting values of N follow an approximately flat distribution across the search range (mean 379, std: 256). If higher N resulted in higher scoring (but overfitted) models, these values would have been skewed towards larger model sizes. Further investigation revealed no significant correlation between N and reconstruction error. The measurement of high amplitude artefacts produced by manipulation of ξ^α and ξ^W (as detailed above) could be taken as a metric for basic malleability. There was no significant correlation between N and the level of artefacts, overall showing a lack of evidence for a connection between N and the possible effects of overfitting in ESNSPFs. CCRNN models were produced with a fixed N, and have shown themselves to be highly malleable in the examples detailed in this project. When producing generative reservoir models, increasing N could give the model more opportunity for varied dynamical behaviours rather than limiting scope; it's certain that overfitting in generative reservoir models is a nebulous concept that warrants future attention.

An audio synthesis process would ideally run in realtime. In this example, the rendering was carried out on a CPU, and was around 10 times slower than realtime. While this leaves much room for improvement, it should be noted that this version was not optimised for speed, and a dedicated C++ or GPU renderer is expected to be faster than the python version used here. It does however show the scale of computation involved in this method of sound synthesis, and indicates that computational resources are a challenge in this area.

CONCLUSIONS AND FUTURE WORK

The experiments presented here show how recurrent neural networks under conceptor control, as originally described in (Jaeger, 2014b), can be configured, trained and run as sample-level sound synthesisers. *Conceptular synthesis* is an extension of granular synthesis, where a CCRNN is trained to reproduce very short segments of a sound sample using conceptors to recall the different patterns. It is controlled at runtime to recombine these short segments into a longer continuous sound. The models were not limited to straight-forward sound reproduction; CCRNNs presented a large variety of creative options for synthesising new sounds based on the training materials. Techniques included classic granular synthesis methods: timestretching and compression, and creative recombination of grains. Techniques were extended by the new possibilities of combining conceptors, using boolean conceptor logic, and using linear combinations of conceptors to morph between signals. The leak rate of RNN nodes and the RNN spectral radius can be manipulated at runtime to create new sonic possibilities.

CCRNNs were shown to have similar resynthesis quality to baseline ESNSPF models, when compared using MFCCs. CCRNNs however excelled in their possibilities for creative sound manipulation, compared to ESNSPFs which produced either significant artefacts or silence when manipulated.

The experiments outlined common limitations of CCRNN models for sound synthesis. There was always some high-frequency loss in the reproduction of the original driving audio signals, some further experimentation is needed to discover the source of this issue. Issues with high frequencies are affected by the choice of leak rate α, which needs to be chosen carefully to slow down RNN dynamics for reproduction of low frequency patterns, while also preserving enough high-frequency dynamics. It's possible that oversampling may help, although the efficiency impact of oversampling could be significant considering the high computational cost of training and running these models.

This work helps to answer the questions posed at the beginning of the paper concerning the fundamental capability of conceptors to synthesise audio signals. Conceptors are able to generate longer patterns needed for audio signals at reasonable sample rates, as demonstrated by resynthesis of simple sine-like patterns up to 890 samples long in the example of the kick drum, and resynthesis of varied audio materials in the Ixi Lang dataset. These pattern lengths are relatively short but useful enough for sound synthesis. As pattern lengths extend, pragmatic limits on computational resources limit further exploration. The resynthesis quality experiment established that when training models with multi-timbre sounds, there is variance in the ability of CCRNN models to reproduce these sounds accurately, indicating sensitivity in CCRNNs to sonic qualities in the source materials. It's not clear yet what this relationship is; this should be the topic of further investigations.

Conceptual synthesis required large models (of approximately 800 nodes upwards) to produce

reasonable results, resulting in slow resynthesis times. The large size of these models was required for them to be able to learn either long patterns or high volumes or short patterns. The technique ran at around 10 times slower than realtime. The memory requirements for conceptular synthesis were particularly large, as a conceptor was needed to reproduce each training signal, resulting in model sizes between 0.5 and 1 GB in the experiments above. These computation requirements still may be considered lightweight compared to some deep learning sound-synthesis techniques, nevertheless it would be a considerable success if these models could be optimised to reach realtime at reasonable sample rates. Recent research into deep architectures in echo state networks may offer promise for increasing computational efficiency, as they have been shown to have better memory capacity compared to classical ESNs with similar numbers of nodes (Gallicchio et al., 2018). More broadly, the relationship between memory capacity and computation time will be a limit on sound synthesis with CCRNNS and their potential to move beyond short sound samples, until methods are found to change architectures and reduce this dependency.

This initial demonstration of the potential of sound synthesis with CCRNNs stimulates further questions. Future research should establish:

1. how these techniques can be scaled upwards to facilitate learning models of longer sound samples

2. the causes of variance in resynthesis quality when training models with signals of varied timbre

3. whether high-frequency loss in resynthesis can be resolved

4. how to optimise the RNN leak rate α for sounds with wide frequency ranges

5. how the techniques identified in this paper can be extended for the purpose of generative sound synthesis.

6. how to optimise network architectures to achieve closer-to-realtime performance

7. how to conceptualise overfitting, in the context of producing creatively malleable generative models

8. potential for use with analysis and resynthesis in the spectral domain (as we are seeing with systems such as NSynth (Engel et al., 2017), GANSynth (Engel et al., 2019))

Conceptually, CCRNN architectures could be creatively compelling for computer musicians; it can sometimes be challenging to create believable and coherent complexity with standard digital sound generation and editing tools. With CCRNNs, complexity comes for free and needs to be managed instead of created. The models presented here are inherently variable, and can be easily encouraged towards unpredictability and nonlinearity, creating sometimes surprising and serendipitous results. The models offer plenty of entry points for creative manipulation, with a potentially wide *aesthetic state space* (Eldridge, 2015). The musician must interact *with* these models, rather than control them.

The experiments presented here have mapped out initial explorations into sound synthesis with CCRNNs. They extend a classical sound synthesis method, bringing boolean logic, pattern morphing and non-linear modulation possibilities into granular-style synthesis. The techniques exhibit some limitations that require further investigation, but also show unique creative possibilities for musicians, and rich potential for further research in this area.

ACKNOWLEDGMENTS

Thank you to Sussex Humanities Lab for generous access of their computing facilities.

REFERENCES

Donahue, C., McAuley, J., and Puckette, M. (2018). Synthesizing Audio with Generative Adversarial Networks. *ArXiv e-prints*.

Duport, F., Smerieri, A., Akrout, A., Haelterman, M., and Massar, S. (2016). Fully analogue photonic reservoir computer. *Scientific reports*, 6:22381.

Eldridge, A. (2015). You pretty little flocker: Exploring the aesthetic state space of creative ecosystems. *Artificial life*, 21(3):289–292.

Engel, J., Agrawal, K. K., Chen, S., Gulrajani, I., Donahue, C., and Roberts, A. (2019). Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*.

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., and Norouzi, M. (2017). Neural audio synthesis of musical notes with wavenet autoencoders. *arXiv preprint arXiv:1704.01279*.

Fernando, C. and Sojakka, S. (2003). Pattern recognition in a bucket. *Advances in Artificial Life*, pages 588–597.

Fiebrink, R. (2011). *Real-time Human Interaction with Supervised Learning Algorithms for Music Composition and Performance*. PhD thesis, Princeton University.

Gabor, D. (1947). Acoustical quanta and the theory of hearing. *Nature*, 159(4044):591–594.

Gallicchio, C., Micheli, A., and Silvestri, L. (2018). Local lyapunov exponents of deep echo state networks. *Neurocomputing*, 298:34–45.

Gast, R., Faion, P., Standvoss, K., Suckro, A., Lewis, B., and Pipa, G. (2017). Encoding and decoding dynamic sensory signals with recurrent neural networks: An application of conceptors to birdsongs. *bioRxiv*, page 131052.

Ghedini, F., Pachet, F., and Roy, P. (2016). Creating music and texts with flow machines. In *Multidisciplinary Contributions to the Science of Creative Thinking*, pages 325–343. Springer.

Hamel, P. and Eck, D. (2010). Learning features from music audio with deep belief networks. In *ISMIR*, volume 10, pages 339–344. Utrecht, The Netherlands.

Harvey, I. (2009). The microbial genetic algorithm. In *European Conference on Artificial Life*, pages 126–133. Springer.

Holzmann, G. (2009a). Echo state networks with filter neurons and a delay and sum readout. *Neural Networks*.

Holzmann, G. (2009b). Reservoir computing: a powerful black-box framework for nonlinear audio processing. In *DAFx*.

Ianigro, S. C. and Bown, O. (2018). Exploring continuous time recurrent neural networks through novelty search. In Luke Dahl, Douglas Bowman, T. M., editor, *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 108–113, Blacksburg, Virginia, USA. Virginia Tech.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13.

Jaeger, H. (2002). Short term memory in echo state networks. Technical report, Fraunhofer Institute for Autonomous Intelligent Systems.

Jaeger, H. (2005). A tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the "echo state network" approach. Technical report, Fraunhofer Institute for Autonomous Intelligent Systems.

Jaeger, H. (2014a). Conceptors: an easy introduction. *arXiv preprint arXiv:1406.2671*.

Jaeger, H. (2014b). Controlling recurrent neural networks by conceptors. *arXiv preprint arXiv:1403.3369*.

Jaeger, H. (2017). Using conceptors to manage neural long-term memories for temporal patterns. *Journal of Machine Learning Research*, 18(13):1–43.

Jaeger, H. and Eck, D. (2006). Can't get you out of my head: A connectionist model of cyclic rehearsal. In *ZiF Workshop*, pages 310–335.

Jaques, N., Gu, S., Bahdanau, D., Hernández-Lobato, J. M., Turner, R. E., and Eck, D. (2016). Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. *arXiv preprint arXiv:1611.02796*.

Jensen, J. H., Christensen, M. G., Ellis, D. P., and Jensen, S. H. (2009). Quantitative analysis of a common audio similarity measure. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(4):693–703.

Jones, B., Stekel, D., Rowe, J., and Fernando, C. (2007). Is there a liquid state machine in the bacterium escherichia coli? In *Artificial Life*.

Keuninckx, L., Danckaert, J., and Van der Sande, G. (2017). Real-time audio processing with a cascade of discrete-time delay line-based reservoir computers. *Cognitive Computation*, 9(3):315–326.

Khunarsal, P., Lursinsap, C., and Raicharoen, T. (2013). Very short time environmental sound classification based on spectrogram pattern matching. *Information Sciences*, 243:57–74.

Kiefer, C. (2014). Musical instrument mapping design with echo state networks. In *NIME '14: Proceedings of the 12th international conference on New interfaces for musical expression*.

Lukoševičius, M. (2012). A practical guide to applying echo state networks. In *Neural networks: tricks of*

658   *the trade*, pages 659–686. Springer.

659   Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new
660   framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

661   Magnusson, T. and Kiefer, C. (2019). Dataset of sounds used with the ixi lang live coding environment.
662   `https://doi.org/10.25377/sussex.7764845.v1`.

663   Mayer, N. M. and Browne, M. (2004). Echo state networks and self-prediction. In *International Workshop*
664   *on Biologically Inspired Approaches to Advanced Information Technology*, pages 40–48. Springer.

665   McCormack, J., Eldridge, A., Dorin, A., and McIlwain, P. (2009). *Generative algorithms for making*
666   *music: emergence, evolution, and ecosystems*. na.

667   Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y.
668   (2016). Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint*
669   *arXiv:1612.07837*.

670   Mudd, T. (2017). *Nonlinear Dynamics In Musical Interactions*. PhD thesis, The Open University.

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

674   Pampalk, E., Dixon, S., and Widmer, G. (2003). On the evaluation of perceptual similarity measures for
675   music. In *of: Proceedings of the sixth international conference on digital audio effects (DAFx-03)*,
676   pages 7–12.

677   Roads, C. (1978). Automated granular synthesis of sound. *Computer Music Journal*, 2(2):61–62.

678   Roads, C. (2004). *Microsound*. MIT press.

679   Sanfilippo, D. and Valle, A. (2013). Feedback systems: An analytical framework. *Computer Music*
680   *Journal*, 37(2):12–27.

681   Schrauwen, B., Verstraeten, D., and Van Campenhout, J. (2007). An overview of reservoir computing:
682   theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial*
683   *Neural Networks. p. 471-482 2007*, pages 471–482.

684   Schwarz, D. (2006). Concatenative sound synthesis: The early years. *Journal of New Music Research*,
685   35(1):3–22.

686   Slaney, M., Covell, M., and Lassiter, B. (1996). Automatic audio morphing. In *Acoustics, Speech, and*
687   *Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference*
688   *on*, volume 2, pages 1001–1004. IEEE.

689   Sussillo, D. and Abbott, L. (2012). Transferring learning from external to internal weights in echo-state
690   networks with sparse connectivity. *PloS one*, 7(5):e37372.

691   Tidemann, A. and Demiris, Y. (2008). Groovy neural networks. In *Proceeding of the 2008 conference on*
692   *ECAI 2008: 18th European Conference on Artificial Intelligence*.

693   Truax, B. (1986). Real-time granular synthesis with the dmx-1000. In Berg, P., editor, *Proceedings of the*
694   *International Computer Music Conference,*. Computer Music Association, The Hague.

695   Truax, B. (1994). Discovering inner complexity: Time shifting and transposition with a real-time
696   granulation technique. *Computer Music Journal*, 18(2):38–48.

697   Wyffels, F. and Schrauwen, B. (2010). A comparative study of reservoir computing strategies for monthly
698   time series prediction. *Neurocomputing*, 73(10):1958 – 1964. Subspace Learning / Selected papers
699   from the European Symposium on Time Series Prediction.

700   Wyse, L. (2018). Real-valued parametric conditioning of an rnn for interactive sound synthesis. In
701   *6th International Workshop on Musical Metacreation, International Conference on Computational*
702   *Creativity (ICCC)*.

703   Xenakis, I. (1971). *Formalized Music*. Indiana U. Press, Bloomington.

704   Yee-King, M. J., Fedden, L., and d'Inverno, M. (2018). Automatic programming of vst sound synthesizers
705   using deep networks and other techniques. *IEEE Transactions on Emerging Topics in Computational*
706   *Intelligence*, 2(2):150–159.