Representation Learning with No Strings Attached?

Investigating Unsupervised Time-Domain Representation Learning for Digital Instruments

Author

Thomas Kaplan

Supervisor

Dr Andrew McPherson

MAT Advanced Placement Project Report 2019

School of Electronic Engineering and Computer Science
Queen Mary University of London

September 2019

Abstract

Digital musical instruments (DMIs) are typically constructed with highly engineered signal models, using physical models and general-purpose signal processing. Despite providing low-latency and intuitive feature spaces which are convenient for performance applications, these approaches often lack the flexibility and complexity needed to model the vast range of achievable performance nuance. In this report we instead harness unsupervised learning methods, with the goal of learning complex and salient feature representations of DMI signals. Variational Recurrent Autoencoders (VRAEs) are deep generative models which learn to compress data in high-dimensional spaces into low-dimensional latent spaces, and to decompress latent encodings. Constructed using Recurrent Neural Networks (RNNs), VRAEs learn to capture the most salient features of a time series, which is appealing for modelling high-resolution DMI signals. We demonstrate that VRAEs are capable of encoding time-domain signals of bowed string vibrations in only 3-dimensions; with a compression ratio of 100:3 (at 11.025kHz). Resynthesis of the latent encodings was greatly improved in a novel model, a VRAE-ST, which introduces a Sequence Transformer (ST) in order to remove distributional shifts in the form of magnitude and temporal perturbations. With reversible transformations before encoding and after decoding, the VRAE-ST applied transformations resembling downsampling and amplitude normalisation. Given the VRAE's unintuitive latent geometry, we also introduce a LatentRNN that is capable of learning the complex trajectories in latent space. Supported by the improved synthesis quality of the VRAE-ST, a LatentRNN might be used for various generative applications with DMIs. An exciting opportunity for future work will be to meaningfully map from latent encodings (or trajectories) to synthesis parameters in some DMI.

Acknowledgments

I would like to thank Dr Andrew McPherson for his unwavering support and continuous stream of exciting and invaluable ideas throughout this project - it was truly a pleasure. I would also like to thank Jack Armitage for his countless useful suggestions in realising this project as more than just a bundle of algorithms. I look forward to being able to present them with an actual digital instrument in the future.

Table of Contents

Abstract i

Acknowledgments ii

Table of Contents iii

1 Introduction 1
- 1.1 Contributions . . . 2
- 1.2 Report Structure . . . 2

2 Background 3
- 2.1 Time Series Modelling in Digital Instruments . . . 3
- 2.2 Music Information Retrieval . . . 3
- 2.3 Physical Models . . . 5
  - 2.3.1 Bowed Strings . . . 5
  - 2.3.2 Percussion . . . 5
- 2.4 Probabilistic Models . . . 6
  - 2.4.1 Hierarchical Hidden Markov Models . . . 6
- 2.5 Neural Models . . . 8
  - 2.5.1 RNNs, LSTMs and GRUs . . . 8
  - 2.5.2 Anomaly Detection . . . 9
  - 2.5.3 Latent Representations . . . 10
  - 2.5.4 Variational (Recurrent) Autoencoders . . . 10
  - 2.5.5 Latent Regularisation and Traversal . . . 11
- 2.6 Trade-offs in Model Complexity for Digital Musical Instruments . . . 13

3 Anomalies in Prediction as Salient Information 14
- 3.1 Multi-output LSTMs for Anomaly Detection . . . 14
  - 3.1.1 Dataset . . . 15
  - 3.1.2 Implementation and Training . . . 16
  - 3.1.3 Evaluation . . . 17
- 3.2 Summary . . . 20

4 Learning a Latent Space of Salient Features 21
- 4.1 Variational Recurrent Autoencoders . . . 21
  - 4.1.1 Dataset . . . 22
  - 4.1.2 Implementation . . . 22

4.1.3 Training . . . 23
4.1.4 Evaluation . . . 24
4.2 Sequence Transformer . . . 34
4.2.1 Learning Invariance . . . 34
4.2.2 Implementation . . . 35
4.2.3 Training . . . 37
4.2.4 Evaluation . . . 37
4.3 LatentRNN . . . 41
4.3.1 Learning Latent Sequences . . . 41
4.3.2 Implementation . . . 42
4.3.3 Training . . . 42
4.3.4 Evaluation . . . 43
4.4 Summary . . . 44

5 Conclusions **45**
5.1 Future Work . . . 46

Bibliography **47**

Appendix A Custom Electronic Drumhead **51**
A.1 Design . . . 51
A.2 Sensor Technology . . . 52
A.3 Example Output . . . 52

Appendix B VRAE-ST Latent Embeddings **53**

Appendix C Model Training Framework **55**
C.1 Module Organisation . . . 55
C.2 Data Collection . . . 55
C.3 Model Training . . . 56
C.3.1 Model Definition . . . 56
C.3.2 Configuration . . . 57
C.3.3 Training and Logging . . . 57
C.4 Evaluation Tools . . . 60

Chapter 1

Introduction

Digital musical instruments (DMIs) face a significant challenge in providing and enhancing the level of performance nuance achievable with acoustic musical instruments. Key to this challenge is developing intuitive *and* detailed schemes to control sound synthesis, beginning with computationally modelling the signals produced by the DMI. State-of-the art approaches typically share a common thread in extracting fixed features, as this design supports intuitive control. However fixed features can be inhibitive in capturing of performance detail, if they (or their combination) don't reflect the most perceptually salient properties of a signal. These models might also fail to meaningfully interpret rare or novel performance nuance.

For example, Buys et al. [1] developed a model which expects a uniform signal with features reflecting bowed string vibrations. This was reported by Pardue et al. [2] to behave chaotically when a violin produced raucous tones, likely due to unanticipated string dynamics. Sensory Percussion by Sunhouse [3, 4] uses neural models with spectral features to perform strike (onset) detection on a drumhead, and to alter timbre of triggered audio samples. Triggering discrete pre-prepared audio samples won't reflect the subtle acoustic properties of the drumhead, vastly reducing a performer's control and potentially masking nuances. This report is therefore concerned with developing alternative approaches to modelling DMI signals, avoiding fixed features.

Generative models and unsupervised learning techniques offer an exciting opportunity to learn meaningful representations of features in signals, without defining fixed features. One popular generative model being explored in various musical applications is the Variational Autoencoder (VAE). VAEs include encoder and decoder neural networks: the encoder learns to compress input data into a latent space (dimensionality reduction); and the decoder uses the latent vectors to reconstruct the input. The construction of a latent space is appealing in providing a reduced representation of signals from DMIs, which might reflect the most salient features for resynthesis. This report is mostly concerned with this opportunity for detailed latent analysis of a signal, but the network can also be used to synthesise novel signals (generation). We assert that the reduced but rich feature representation learned by VAEs might resemble something closer to expertise transfer.

Typical VAE applications in music involve transformation of audio samples into the frequency domain [5] or into symbolic representations [6, 7, 8]. However, we are concerned with high-resolution time-domain analysis in order to retain full performance detail, which poses a challenge in data dimensionality. Harnessing continuous and high-resolution sensors has been key to developing new hybrid and electronic instruments, such as The Magnetic Resonator Piano by McPherson [9], and The MagPick by Morreale et al. [10]. Additionally, time-domain models avoid latency costs associated with pre-processing, such as transforming large audio buffers in-and-out of the frequency domain.

This report investigates unsupervised learning as a technique for expertise transfer in DMIs. Instead of designing and extracting fixed features, we learn reduced representations of high-resolution time-domain sensor signals. We evaluate how these modelling techniques might enable the development of DMIs with intuitive control alongside performance nuance.

1.1 Contributions

Two instruments are explored in this work using unsupervised learning techniques: (1) a custom electronic drumhead, as electronic percussion is plagued by discrete representations of performance (drumhead strikes as discrete 'onsets') and (2) a violin with electronic pickup, as contrasting approaches in literature use fixed-feature extraction techniques. Although the models developed weren't evaluated in real-time use, we were mindful of the training and performance costs, for future use in performance applications. Instead, we largely focused on a high quality (re)synthesised output of the models - a mandatory feature for performance applications. Generative use of the models was explored, as a lesser focus, as it illustrated whether models were intuitive. The technical contributions of this work are as follows:

1. In *Chapter 3* we develop a multi-output LSTM forecast model capable of anomaly detection for time-domain drumhead oscillation signals. By providing a continuous signal reflecting forecast error, a richer signal is provided to characterise exogenous events (e.g. drum strikes) than simple onset markers. Harmonic properties are harnessed to make accurate forecasts over numerous oscillation cycles ($\sim$18ms), treating windows of samples as multiple-features of a single timestep.

2. In *Chapter 4* we develop Variational Recurrent Autoencoders (VRAEs) that construct a latent space characterising time-domain bowed string velocity signals. In order to improve quality of the resynthesised signal, we (1) apply an overlap-add method to reconstructed windows (2) introduce Sequence Transformer (ST) layers that learn task-irrelevant variances in the input distribution (Section 4.2). Together, these form a novel approach that minimises audible noise resulting from inconsistent window reconstructions. We demonstrate the challenge in using the VRAE's latent space for generative ends, before introducing a more rigorous method to circumvent unintuitive feature representations (Section 4.3).

1.2 Report Structure

This report is organised as follows: in Chapter 2 we provide a background on computationally modelling features in time series and DMI signals; in Chapter 3 we develop RNNs as predictive models for DMI signals; and in Chapter 4 we develop VRAEs which learn a latent representation of DMI signals. Within Chapter 4: in Section 4.2 we develop a model that harnesses task-irrelevant invariances in DMI signals; and, in Section 4.3, we develop a model to learn meaningful sequences of latent vectors.

Chapter 2

Background

In this chapter we describe various approaches to computationally modelling features in time series, which might be applied to DMI sensor signals. It should become evident that some approaches start with a process of (sometimes rigorous) feature engineering or modelling - i.e. developing fixed features. While being computationally malleable and interpretable, the surveyed approaches may not develop feature representations allowing for the level of nuance demanded by an experienced musician. As an alternative, we finally introduce unsupervised machine learning algorithms at the end of this chapter. In doing so, we consider some of the challenges in using these models in musical applications, alongside the exciting benefits.

2.1 Time Series Modelling in Digital Instruments

Digital musical instruments (DMIs) crucially demand not only low but consistent latency, in order to seamlessly translate performer actions to some audible outputs. Jack et al. [11] demonstrated that the aim should be 10ms, and that variance shouldn't exceed 1ms. This poses a significant challenge for systems trying to extract information from time series, which is often complex and high-resolution sensor data with a sampling frequency around 44.1kHz ($46\mu s$). In order to address the limitations faced when processing complex sensor data for interactive music systems, McPherson et al. [12] developed a new embedded platform, Bela[1]. Bela allows audio processing to preempt its kernel, facilitating audio buffers as small as 2 samples at 44.1kHz [13].

Bela has been applied in *many* interactive applications. Morreale et al. [10] created 'MagPick', an augmented guitar pick that can detect nuanced ancxillary gestures, which is used to modify (and extend) the sound of the guitar. This is made possible by electromagnetic induction - the pick contains loops of wire, and the induced signal is passed through a preamplifier before processing in Bela. In terms of data modelling, simple signal transformation and thresholding is used to activate some pre-defined effects, devised through manual data exploration. Pardue et al. [2] developed a hybrid acoustic-electric violin, where a Bela Mini is used per-string to process the output of an electrodynamic pickup. Discussed in 2.3.1, a linear segmented regression model is harnessed for real-time analysis of string signals. Somewhat similarly to MagPick, this also relies on relatively predictable signal dynamics. There is much room for work to explore how Bela can harness less-traditional time series modelling techniques being introduced in fields such as machine learning.

2.2 Music Information Retrieval

Before looking at probabilistic and neural approaches to time series modelling, its worth introducing a typical approach taken: techniques under the umbrella of Music Information Retrieval (MIR). MIR has developed a vast range of features reflecting various properties of

[1] Bela: beautifully interactive sensors and sound. http://bela.io

sound, which can be used as building blocks for sound processing algorithms such as modelling. The typical MIR toolset will be illustrated via a relevant example for brevity - a state-of-the-art product in electronic percussion, 'Sensory Percussion' by Sunhouse [3].

Sensory Percussion is an electronic music suite that combines proprietary high-bandwidth sensors that measure drumhead vibrations, and an impressive software pipeline using machine learning and MIR techniques. We can speculate on the implementation through the patent [4]. Most notably, although continuous drumhead vibrations are recorded, they may not be directly used in synthesis. Instead, the user configures a range of sound samples that are played back with varied timbre based on discrete strikes of the drumhead ('onsets'). This commonly requires buffering samples from the sensors, and performing frequency-domain transformations such as the Fast Fourier Transform (FFT) and Constant-Q frequency transform. Although useful in reducing data dimensionality, these frequency-domain transformations are not always robust over short time frames, harming resolution [14]. Despite this, in Sensory percussion, spectral analysis of the buffered frames allows (1) a pooled representation of sensory activity/energy to be created, that might reflect an onset (2) timbral properties of the sensor signals to be inferred, which can modulate a sample's playback (3) inference of spatial properties, namely discrete onset locations on the drumhead, such as 'centre' or 'edge'.

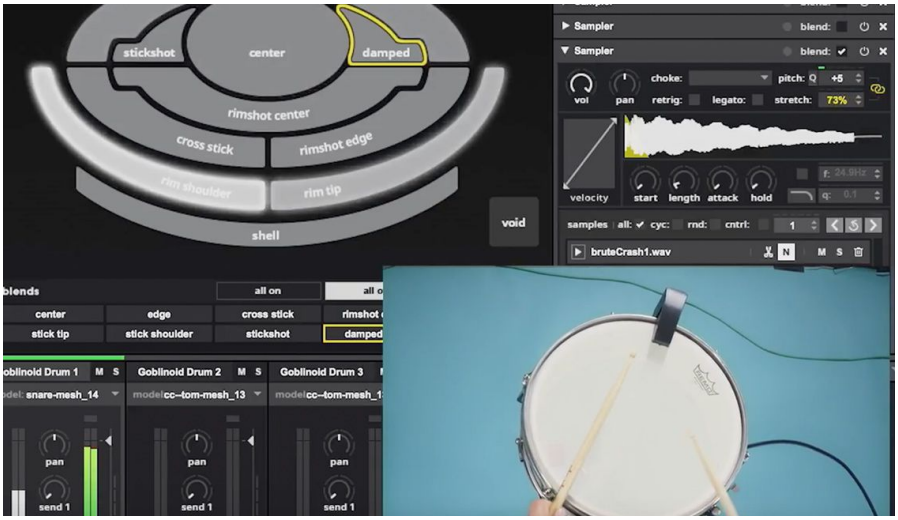


Figure 2.1: Sensory Percussion GUI and sensor hardware, DJ TechTools, 12/10/17.

It is common in the world of MIR to focus on 'onsets' (individual strike actions on the drumhead) as these events typically correlate with the start of sounds. However, a drumhead is a continuously oscillating surface where nuanced details of gesture subtly influence timbre. Certain techniques such as using brushes, or high-speed stick control involving flams and (buzz-)rolls might fail to get captured by an algorithm chasing onsets; as they happen rapidly and get absorbed into the nuances of drumhead oscillation. If frequency-domain transformations fail to surface detail in these gestures, a drummer may rapidly develop bad technique that translates poorly to acoustic drumkits - such as poor volume control, timing or stick positioning. Fundamentally, the MIR approach can enforce a constraint on the possible feature space that can be used to construct meaningful representations of musical performance. This might be a limiting factor in the expressive capabilities of DMIs.

2.3 Physical Models

One alternative to relying on generic frequency domain transformations for sound modelling is the construction of physics-based models of musical instruments (an overview was presented by Välimäki et al. [15]). These models are realistic simulations of instruments, demanding rigorous technical implementations informed by significant knowledge of physical dynamics. Models of bowed strings and percussion instruments are briefly introduced, as a reference point for the contrasting techniques investigated in this report.

2.3.1 Bowed Strings

Helmholtz motion is a well understood phenomena in bowed strings where the string vibrates in a V-shape [16], discovered in the nineteenth century and later developed. Theoretically this could be likened to a time-domain triangular signal, but practically there are deviations introduced by complex interaction between the bow and string, string stiffness etc (see Figure 2.2 for an example). Despite this, Buys et al. [1] harnessed the simplicity in modelling basic string motion to develop of a real-time regression technique allowing detailed analysis of a string velocity signal. This has been used by Pardue et al. [2] as a reference signal in a tone regulating violin, however it was noted that harsh tones cause chaotic model behaviour that demand either extending the model or filtering high variance during sustained oscillation regimes. This illustrates the inherent challenge in developing specialised physical models - even for better understood phenomena, rare nuances or non-linearities challenge the fundamental assumptions that ensure general purpose efficacy of the model.

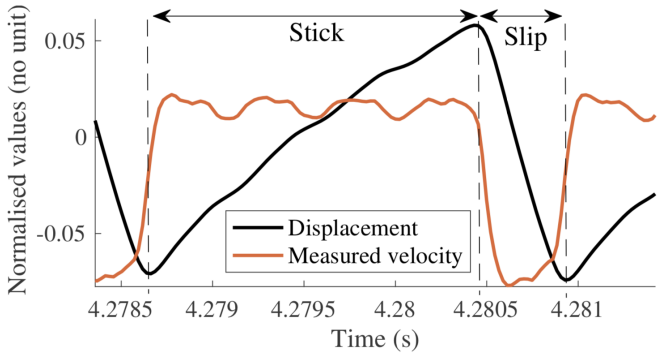


Figure 2.2: Example string velocity cycle measured in [1].

2.3.2 Percussion

Percussion instruments pose significant challenges for physical simulations, for several reasons: (1) the drumhead membranes must be modelled as 3D objects which interact with air in the shell's cavity (2) drums often have two coupled membranes on the top and bottom, and snare drums additionally have wires under tension on the lower membrane (3) when struck, or vibrating under certain conditions, complex non-linearities are observed. Examples of these non-linearities include pitch glides, where perceived pitch varies under large strike amplitudes. Torin [17, 18] rigorously simulated these dynamics using finite difference methods. Despite this, separate work has pursued simpler means of mimicking non-linearities of percussion instruments. Hus et al. [19] notably introduced a 'loop back' in frequency modulation

(FM) synthesis, where the carrier modulates its frequency. In any case, these models lack the full complexity of an acoustic drum, and there are still questions around how these may be used in a meaningful way as part of a digital instrument. For example, how sensors might activate certain aspects of a percussion model; in light of the fact that non-acoustic drumheads might exhibit different physical behaviour.

2.4 Probabilistic Models

In contexts not limited to music, significant work in human activity and gesture recognition has harnessed probabilistic models of time series. This has prompted valuable work in motion-sound mapping, where physical gestures are used to manipulate meaningful synthesis schemes. Compared to developing physical models or MIR feature-engineering, this process importantly begins with data reflecting the activity in question. Unsurprisingly there are also numerous efforts applying deep learning, as surveyed by Nweke et al. [20].

2.4.1 Hierarchical Hidden Markov Models

There have been many interactive music systems developed involving gesture, but Françoise [21, 22] notably developed and applied probabilistic models of sequences for continuous-time mapping between gesture and sound parameters. It is data driven, introducing an approach called 'mapping by demonstration', where the user builds a one-shot multimodal model that can be harnessed in real-time for new gestures.

One of the models used is a Multimodal Hierarchical Hidden Markov Model (MHHMM) [23]. The Hidden Markov Model (HMM) is a probabilistic time series model that treats a sequence as a noisy Markov process, with discrete states over discrete time steps - further information can be found in a popular tutorial by Rabiner [24]. The Hierarchical HMM (HHMM) [25] creates a multi-level structure where transitions take place between HMM structures. In other words, HMM structures form segments (gestural phrases), and the HHMM models higher order sequences in terms of these lower level segments. An example is shown in Figure 2.3. The MHHMM assumes a gesture and sound sequence share an underlying Markov process, such that the respective feature vectors can be concatenated (as a multimodal sequence) during training and modelled as a joint Gaussian Distribution. Therefore, a single training example consisting of parallel gestural and synthesis features (i.e. some sound sample) is required for 'mapping by demonstration'.

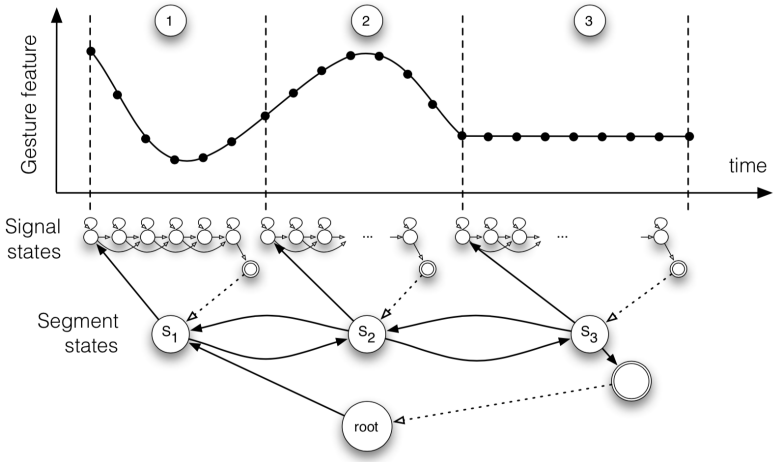
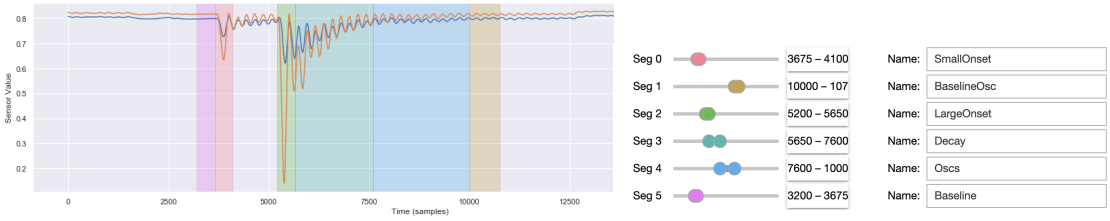


Figure 2.3: An example of a HHMM with two levels from [22].

These probabilistic techniques allow control over the temporal dimension of gestures, however variations in gesture might not follow the training interpretation. Model generalisation can be controlled using a *static* 'variance offset' parameter, which trades off robustness (phrase recognition accuracy) and lower reliability (inconsistent phrase detection). Unlike this static parameter, dynamic variance/confidence bounds have been introduced through dynamic simulation - Manitsaris et al. [26] used it in modelling of pottery and Volioti et al. [27] in musical gestures. However, in any case, these gestural tracking solutions pose problems in how they learn by example. We experimented with HHMM tracking for high-bandwidth sensor data using a custom drumhead which measures surface oscillations (detailed in Appendix A), and the `XMM` library [28]. Interestingly it was possible to use a single gesture (see Figure 2.4), to track various low level signal phrases for other examples (e.g. Figure 2.5), but it generalised to other gestures poorly. It should be evident that this not only requires a crude classification of sensor phrases, but clearly cannot scale to the realm of all gestural variations that may occur in performance. This is particularly problematic for complex time series.



(a) (b)

Figure 2.4: Example flam used in HHMM training, coloured by phrase. The two lines reflect different sensors as detailed in Appendix A.

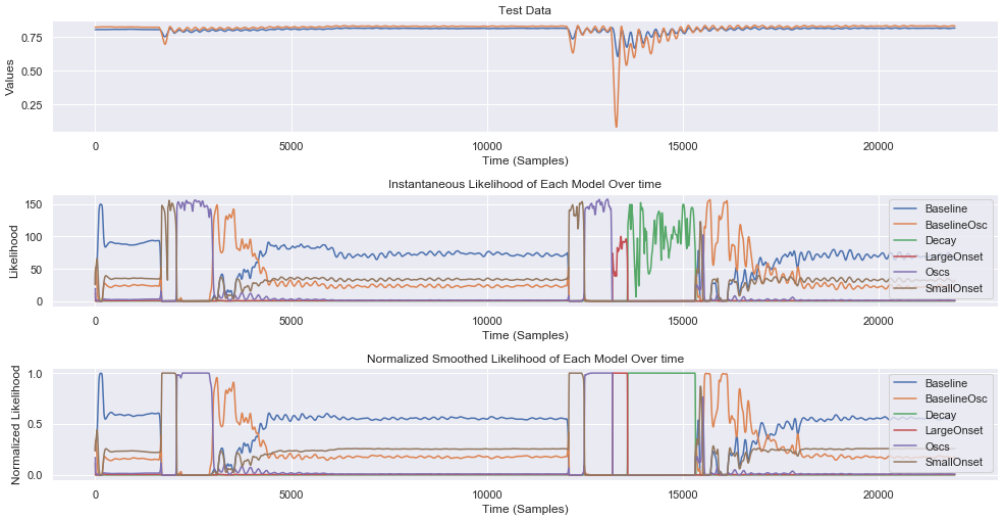


Figure 2.5: HHMM tracking performance for an unseen example flam (top subplot). The phrase templates are defined in Figure 2.4.

2.5 Neural Models

Developing a data-driven approach even further, deep learning is a popular option for learning high-complexity features from time series with minimal hand-holding (e.g. discrete labelling of training data). This is made possible through technical developments in deep learning, and readily available compute capacity. New neural network units such as Long Short-Term Memory units (LSTMs) and Gated Recurrent units (GRUs) have enabled new approaches to many sequence to sequence learning tasks [29], where a network learns to encode, transform and decode sequences according to input and target output. This can be a symbolic sequence (e.g. text sentences or MIDI music), but equally a raw time series. In this section we first detail relevant neural network architecture, before exploring how they can be harnessed for anomaly detection and representation learning of sequences.

2.5.1 RNNs, LSTMs and GRUs

Recurrent Neural Networks (RNNs) [30] are a class of deep neural network used in tasks involving sequences, that learn 'hidden' representations of some data for each step in time. Basic RNNs consist of a 2D grid of 'hidden state' vectors, $h_{1..t}^{1..l}$, with t corresponding to a time step and l the layers (depth). The first hidden layer h^1 consumes input vectors and the final h^l is used as an output; where h_j^i is recurrently updated by h_{j-1}^{i-1}, according to weights w_j^i that are optimised in training. The hidden state of RNNs is able to model highly complex and non-linear features of time series, and can easily scale to handle multivariate data. Additionally, RNNs can be configured to take variable length input sequences. This is useful in cases where input signals may have different sampling frequencies (such as in a DMI). Following challenges in training RNNs for learning long-term dynamics in data (e.g. the 'vanishing gradient problem' [31]), variants were introduced - most notably, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) cells.

LSTMs [32] were designed to mitigate the issue of vanishing gradients. In addition to a hidden vector, each cell contains a memory vector c_t^l, which via gating structure can be read,

written or reset on a given time step. Unless cell state interacts with a forget gate, it can be propagated through time for a long period of time. GRUs [33] have more recently been introduced as simplified memory cells. They propagate state through the hidden vector via smooth interpolation, where some amount of information is forgotten via a reset gate on each time step. LSTM and GRU gating is demonstrated in Figure 2.6. Both LSTMs and GRUs can prove effective for sequence modelling, and should often both be evaluated.

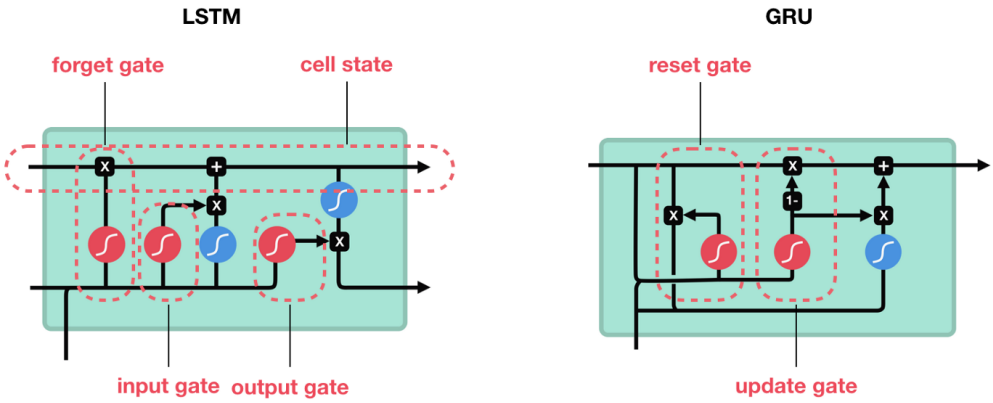


Figure 2.6: LSTM and GRU unit illustrations [34] - the red and blue gates are $sigmoid$ and $tanh$ functions respectively, arrows are vector concatenation and the operators are pointwise.

LSTMs have been used widely in sequence modelling tasks involving symbolic musical data such as MIDI (e.g. by Eck et al. [35]). Low-level sensor data in digital instruments is continuous and high-bandwidth, posing a much greater challenge. Hantrakul et al. [36] demonstrated real-time use of LSTMs to generate X-Y gestural sequences based on a 2D musical surface[2]. Notably this was trained on a small data-set of merely 3000 time steps, sampled at only 33Hz. Martin et al. [37] suggested LSTMs were insufficient for complex performance data, not supporting stochastic sampling. They instead introduced a mixture density networks (MDN) layer after LSTMs to build a multi-modal distribution. Separately, large networks of dilated convolution layers (CNNs) were used to create 'WaveNet' [38], a model allowing synthesis of high-resolution raw audio (e.g. generative piano synthesis [39]) without RNNs.

Unfortunately RNNs are generally challenging to interpret, and their performance is poorly understood. Several studies have attempted to understand how these models actually represent long-term sequence characteristics, but largely offer graphical techniques for analysing symbolic sequences of text [40, 41]

2.5.2 Anomaly Detection

RNNs are appealing candidates for anomaly detection, given the ability to perform complex time series modelling. They can be trained to model some historic nominal data; and then the optimised forecasting capabilities can be used to infer time steps where large errors might correspond to anomalous data, in some (real time) data stream. This is interesting in the case of DMIs, as reconstruction error might not simply correspond to sensor errors, but to nuances reflecting original performance.

[2]The Roli Lightpad Block: https://roli.com/products/blocks/lightpad-block

Alternative anomaly detection techniques include construction of a Matrix Profile [42], where crude Euclidean distance-measures construct a similarity profile of windows across a time series. This distance analysis can be very computationally demanding for high-resolution sensor data, and insufficient for high complexity data.

Developing an anomaly detection strategy with RNNs requires an approach to handling reconstruction error. Malhotra et al. [43, 44] demonstrated that anomaly likelihood estimation can be performed using LSTMs, when modelling predictions errors at a given time step using some probability distribution. This technique can incorporate overlapping residuals from sequential time steps, prior to the future observation time. They applied techniques to numerous data sets, including electrocardiogram (ECG) signals. Hundman et al. [45] used LSTMs to detect anomalies in sensor data from Curiosity, the Mars Science Laboratory (MSL) rover, trained using expert-labelled anomalous data. Instead of assuming some Gaussian distribution of past anomalies, they developed a dynamic and non-parametric thresholding approach that is sensitive to non-stationarity and noise. Cowton et al. [46] also avoided hand-crafted error features, using expert-labels and particle swarm optimisation (PSO) for a threshold-based detector; applied to analysing environmental sensor data that had been linked to respiratory disease in pigs.

2.5.3 Latent Representations

In light of the opaqueness of LSTM hidden state (refer back to 2.5.1), it is often useful to create an explicit Latent Variable Model (LVM). An LVM specifies a compressed (i.e. basic) distribution of a higher dimension distribution, allowing complicated variables to be represented in low-dimensional 'latent' variables. These latent variables $z \in \mathcal{Z}$ represent an efficient and useful distribution of the underlying data manifold ($x \in \mathcal{X}$) as a generative model with function $f : \mathcal{Z} \to \mathcal{X}$. Unlike previously discussed techniques, these models can theoretically develop representations of data in a continuous space without significant assumptions. For example, a bundle of MIR features reflecting sound properties in a time series are not required, and the model doesn't heavily rely on a few template samples like HMMs.

Note however that this latent mapping isn't trivially interpreted, as it can be highly non-linear, i.e. it may provide a distorted perspective of the input space $\mathcal{X}$ (discussed in detail by Arvanitidis et al. [47]). This means that the Euclidean distance between two latent points z_i and z_j may not reflect some meaningfully equivalent distance in the input space between corresponding points $f(z_i)$ and $f(z_j)$.

Taking a probabilistic modelling approach, an LVM is constructed by maximising the probability of drawing existing/observed data points, given a model with parameters θ. This is the product of $p_\theta(x_i)$, equivalent to the sum of $log\ p_\theta(x_i)$. The two components of the objective are therefore (1) a fixed prior, a simple latent distribution $p(z)$ and (2) a decoder, a parameterised family of conditional distributions $p_\theta(x|z)$. For a single data point x_i, this can be written as:

$$\log p_\theta(x_i) = \log(\int p_\theta(x_i|z_i) \cdot p(z_i)\, dz) \tag{2.1}$$

2.5.4 Variational (Recurrent) Autoencoders

Variational Autoencoders (VAEs) [48] are deep generative models that learn a LVM. They are composed of an 'encoder' and 'decoder'. The encoder learns the LVM, representing the D-

dimension input space ($\mathcal{X} = \mathbb{R}^D$) in d-dimension latent encodings ($\mathcal{Z} = \mathbb{R}^d$). The decoder learns to reconstruct D-dimension data from the d-dimension encodings. The latent space, as a compressed representation of the input space, requires a bottleneck in the network where smaller sub-networks absorb salient information from the input space. Relevant applications of VAEs will largely be introduced in the following section - for now we further detail VAEs.

The encoder and decoder networks wrapping the latent sub-network can be constructed in numerous ways, most simply as linear mapping (fully-connected, FC) layers with nonlinear activation functions. An appealing variant for time series modelling is the Variational *Recurrent* Autoencoder (VRAE) [6], which uses RNNs (equally LSTMs and GRUs) for the encoder and decoder. The encoder is initialised with a zero-vector as hidden state, h_0, and at later time steps h_t will reflect previous state $h_{(t-1)}$ and data on step x_t. The final hidden state for a sequence h_{end} is used for the latent distribution $\mathcal{Z}$. The decoder's hidden state is then initialised with weights sampled from a given latent encoding z, before use as a regular RNN. By harnessing memory in the network, as introduced in 2.5.1, the issue of scale faced by a network composed of linear layers may not be faced with larger time series sequences. The VRAE when introduced by Fabius et al. [6] was interestingly evaluated against a musical corpus - 8 MIDI files (with one dimension per pitch) of 80s-90s video game music. Although symbolic in nature, whereas we are concerned continuous time series, this demonstrated an ability to meaningfully model similarity among segments of these files; whilst also providing a somewhat-stable generative model.

Doersch [49] provided a thorough tutorial on VAEs, which explains the underlying mathematics. For a brief gist, the bottleneck consists of two functions (1) $\mu_\phi : \mathcal{X} \to \mathcal{Z}$ generates a surface in $\mathcal{Z}$ and (2) $\sigma_\phi : \mathcal{X} \to \mathbb{R}^d$ records uncertainty in reconstruction. These approximate the unknown posterior distribution $p_\theta(z|x)$, via a variational distribution $q_\theta(z|x)$. The parameters ϕ and σ are optimised by maximising the variational lower bound (or 'evidence lower bound', ELBO) of $p(x)$ as per Equation 2.2.

$$\{\theta^*, \phi^*\} = \underset{\theta,\phi}{\operatorname{argmax}} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\sigma(\mathbf{x}|\mathbf{z})] - \mathbf{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z})) \tag{2.2}$$

This directly informs a training loss function $\mathcal{L}(x; \theta, \phi)$, where the first term in Equation 2.2 refers to a 'reconstruction' loss and the second term a 'KL regularizer'. The former encourages latent variables to support reconstruction, and the latter encourages $q_\theta(z|x)$ towards the prior distribution $p(z)$ (it measures the quality of approximation).

2.5.5 Latent Regularisation and Traversal

Variational LVMs aren't commonly used in problems involving musical audio modelling and synthesis, and haven't been used in the development of digital musical instruments. As suggested in 2.5.3, VAEs can suffer to oddity in their highly non-linear latent variables [3]. Harnessing a VAE for generative purposes or as a feature model therefore can therefore be challenging, as continuous attributes (e.g. amplitude) of interest in the input space might not be interpretable in the latent space. When applying VAEs in musical problems, strategies are commonly pursued to improve the malleability of the latent space.

One commonly trodden approach to encouraging meaningful latent structure is through regularisation. Hadjeres et al. [7] devised 'Geodesic Latent Space Regularisation' (GLSR-VAE),

[3] This is well illustrated by an interactive demo on [50].

where an additional regularisation term in the loss function links changes in attribute functions to changes in latent space. This was explored for chorale melodies stylised after J.S. Bach, regularising by the number of notes played in a given sub-sequence of the training set. Despite allowing more meaningful interpolation in the latent space, for generative ends, this work was fortunate to operate on a symbolic input space which facilitated a simple regularisation scheme.

Similarly Esling et al. [5] developed a regularisation scheme, but instead it is perceptually motivated. Previous studies have used Multi-Dimensional Scaling (MDS) to create 'timbre spaces', based on (perceptual) dissimilarity matrices between instruments. The topology of these timbre spaces is used to shape the latent space of a VAE built on spectral samples from the respective instruments, using an 'additive penalty' regularisation term in the loss function. The resulting VAEs develop a highly convincing and smooth evolution in timbre when used generatively. Unfortunately it is challenging to generalise this approach in the case of DMI design, as the input space might not be easily regularised according to existing perceptual studies. Engel et al. [51] notably introduced a similar network for synthesising sounds that morphed between instruments, using a different architecture - a WaveNet Autoencoder.

Outside of music, in order to improve explanatory power of the latent space for capturing Brownian dynamics, Hernandez et al. [52] introduced a 'Variational Dynamics Encoder' (VDE). They introduced an autocorrelation loss term, successfully demonstrating that the network better represents the long-term kinetics of a time series. This might be valuable for time-domain based feature representations in the musical domain, as there may not be intuitive continuous attribute vectors, but expected correlation in latent representations over time.

A different approach is adopted by Fortuin et al. [53], where a Self-Organising Map (SOM) is used to discretise the VAE latent space; as latent encodings are mapped to nodes with meaningful neighbours. This is conceptually similar to approaches that use vector quantisation of the latent space [54]. These techniques would be interesting to explore in musical problems, as they allow a somewhat-unsupervised approach to latent regularisation. Fasciani [55] also used SOMs to create mappings between synthesis and control spaces, informed in training by a notion of vocal gestures and postures. This is also a meaningful distinction for digital instrument design - postures define steady acoustic characteristics, and gestures define sounds with dynamic acoustic characteristics that might transition between postures.

Instead of encouraging the latent space to be organised in some meaningful way, it is possible to train separate networks to harness the latent space. For sequential data tasks, this can involve training an RNN to traverse the complex non-linear manifold $\mathcal{Z}$. One notable example of this is for 'World Models' by Ha et al. [56], an unsupervised reinforcement learning model (e.g. for video game playing). In this a VAE is used as a 'vision' model to encode input observations from the environment, and an RNN acts as a 'memory' model, learning the pattern of latent encodings such that it can predict future states. This technique was similarly applied to musical score inpainting by Pati et al. [57], where a 'LatentRNN' was trained to fill the gap between latent encodings of past and future states, such that the missing segment of score could be decoded. This is an appealing strategy for cases where little is known about meaningful attributes in the input space, such that regularisation schemes may be challenging. Additionally, it provides a further auto-regressive model that can be used generatively.

2.6 Trade-offs in Model Complexity for Digital Musical Instruments

In light of 2.5.5, why might someone prefer to leverage unsupervised learning techniques in DMIs? There is clearly a significant challenge in ensuring a learned feature space is spatially useful, and instruments inherently demand intuitive mappings from performer actions to audible output.

As an alternative, consider the segmented linear regression developed by Buys et al. [1] (described in 2.3.1) for bowed strings, which produces explicit features that reflect known characteristics: fundamental frequency, amplitude, relative corner position, RMSE. Immediately, frequency and amplitude are available for use by some synthesis engine. Additionally the model provides a consistently low latency, which might be compromised slightly based on precision demands, but should in any case provide a detailed view of the specified features. However, this hinges on a crucial assumption - "the bowed string vibration captures almost all the player's musical intentions...". It goes without saying that the string forms a large component of violin sound production, yet it is not apparent that a bundle of (highly detailed) physical features might be used to model musical expertise in a perceptually meaningful way.

It is not unreasonable to assume that the signals of interest possess unobvious features, beyond known physical non-linearities, that are highly salient. These features may be complex, entangled, inexplicable and highly personal. Unsupervised learning techniques offer an opportunity to develop feature spaces of this nature, as introduced in 2.5. For example, VRAEs with a LVM of N-dimensions expect that there is a complex relationship between N variables over time for an input signal. Crucially, using a low number of variables to represent a given signal demands learning the crux of it. Therefore traversing a manifold produced by a VRAE, and decoding the path of latent vectors, might naturally result in perceptually salient changes in an audio signal. These changes might be continuous, transitioning through unimaginable or unachievable dynamics in the signal - yet remaining uniquely nuanced and recognisable. By virtue, this might encompass a level of achievable performance nuance that cannot be baked into a model with manually-curated fixed features. In 2.5.5 we highlighted that this path might be highly complex, posing a technical challenge to harness it - but this seems reasonable given the challenge of expert musical performance. Unlike a segmented or windowed approach to signal analysis, a VRAE can consume varying length input signals over time (this simply updates encoder/decoder hidden state); which is valuable in that an it doesn't enforce feature representation at a predetermined time scale. Finally, it is worth noting the non-trivial latency of interacting with large and complex neural models, including simple mapping tasks from the input domain through the VRAE. Hardware acceleration (e.g. GPU or FPGA) may assist in minimising latency if necessary.

Chapter 3

Anomalies in Prediction as Salient Information

In this chapter we briefly present RNNs that can make forecasts of high-resolution time-domain sensor data from DMIs. Forecast error can be cast as anomaly, producing a continuous signal of salient information such as rare motifs (nuances) and exogenous events (performer interaction such as drum strikes). This is demonstrated as a novel approach to facilitate onset detection in percussion instruments. More importantly, we establish that RNNs alone are insufficient as a model for analysing performance, due to complexity of their hidden state.

3.1 Multi-output LSTMs for Anomaly Detection

Several of the anomaly detection methods presented in 2.5.3 forecast at a single time step. This is not useful for high-resolution sensor data whereby each sample at e.g. 44kHz reflects only 0.02ms. Further to this, recursively using this initial forecast to construct further forecast samples would cause accuracy to rapidly deteriorate. This demands multi-output forecasting. Fox et al. [58] demonstrated a range of networks for multi-output forecasting which improve on a recursive baseline. These center around an RNN, enhanced with: fully-connected (FC) output layers per forecast time-step; an additional decoding RNN prior to the FC layers; and polynomial function learning per forecast time-step.

Fortunately, this design might be simplified for certain digital instruments with harmonic signals. For example, the continuous oscillation of a drumhead might resemble a sinusoidal function, where cycles spanning many samples might be forecast at a given sample. It is feasible that even a basic RNN might capture such a relationship in a signal. Instead of passing low-dimension vectors (e.g. 1 dimension per sensor channel) for a time step, high-dimension vectors may be fed into the RNN (e.g. N dimensions reflecting a window/frame size of the N previous samples). This latter setup is shown in Figure 3.1. Multiple hiddens cells may capture the evolving oscillatory dynamics in a signal over multiple cycles, and hidden state could be mapped to an N-step forecast using a single FC layer. The immediate benefit of this is a multi-step forecast of size equal to that of the input window, without introducing a significant number of extra layers for each forecast time-step.

This makes sense for the modelling of a drumhead, as anomalies might only be registered for complex non-linearities and the most immediate moments of drum strikes. In both of these cases, typical oscillatory dynamics will break down, affecting forecast performance. Therefore a continuous anomaly signal reflecting forecast error might be used directly to infer events of interest, or serve as a malleable signal that could be mixed into a synthesis engine.

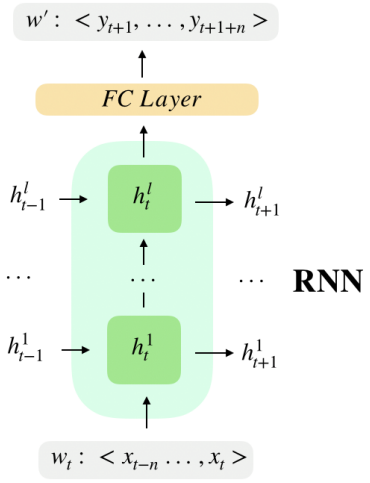


Figure 3.1: RNN with l layers and n-dimensional input and output vectors.

3.1.1 Dataset

In order to harness a lightweight RNN for multi-output forecasting of a drumhead signal, we suggested the potential importance of harmonic properties in a signal. Most commercially available electronic drumheads use piezoelectric sensors as onset triggers, as they are sensitive to spiking input-forces. These are not suitable for this task, so we created a custom electronic drumhead using electromagnetic induction to record surface oscillations. This is detailed in Appendix A.

For this modelling task, we recorded 60s of flam-taps[1] performed across the drumhead. These are challenging gestures for typical onset detection algorithms, as strike events occur in quick succession (almost perceptually indistinguishable) at contrasting amplitudes. Only the drumhead's central sensor channel was used, resulting in a 1D time series. To reduce noise and center the signal around a 0-valued baseline (supporting neural network convergence), the signal was filtered between 10Hz and 250Hz using 5th order Butterworth filters. The signal was decimated from 44.1kHz to 11.025kHz for these preliminary assessments, demanding a smaller network and simplifying initial parameter tuning - it was empirically observed that this retained reasonable signal quality. We used 80% of the data for training, 20% of the data for testing, and 20% of the training data as validation (used to assess learning progress). The data is shown in Figure 3.2.

[1] Flam-taps are basic drum rudiments combining flams with single taps in double strokes. Flams consist of two strokes in quick succession, where one is typically a quieter 'grace' note.

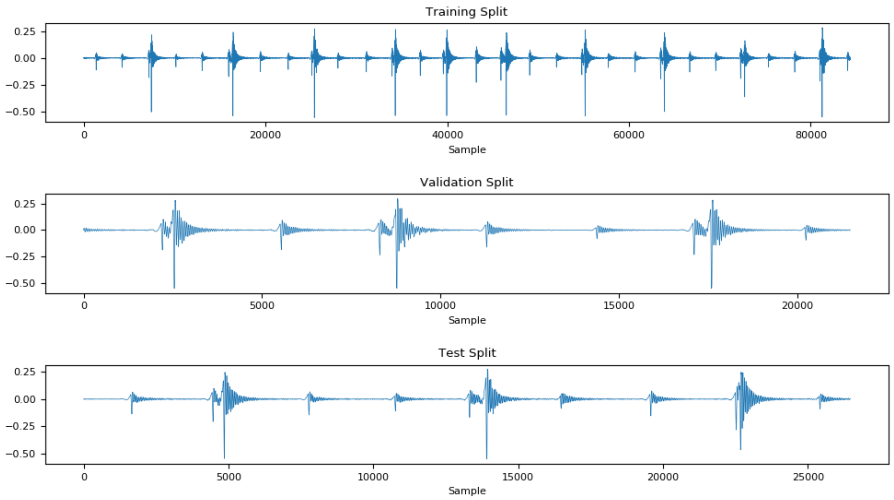


Figure 3.2: Training, test and validation splits used to train a multi-output forecasting RNN. The data corresponds to 60s of flam-tap rudiments on the custom drumhead presented in Appendix A, from the central pickup.

3.1.2 Implementation and Training

Note that for all neural implementations in this work, including later sections, we used the PyTorch [59] library. Additionally, we empirically tuned network parameters, and it is worth noting that a significant amount of future work might explore hyperparameter optimisation schemes. The final configurations crudely followed from a position where we were able to discuss some level of success in the modelling task. For example, for this section, this reflected accurate forecasting. The window size was notably chosen to encompass a number of cycles, and was equivalent to 18ms of the decimated signal. This could also be adjusted during hyper parameter optimisation, but will ultimately depend on latency and the model application.

The final configuration is shown in Table 3-A. Training was performed without mini-batching, and terminated after 324 epochs of 500, due to the early stopping criteria (performed on validation loss). The server used for training was shared among a number of research staff, equipped with: 2 $\times$ Xeon 5122 CPUs at 3.60GHz, 2 $\times$ P100 Tesla GPUs with 16GB RAM and 192GB of host RAM. In total, training took 8.1 hours, notably extended by the shared server usage.

The loss curve is shown in Figure 3.3. The lack of complete convergence between training and validation accuracy suggests an un-representative training set, which may be fixed with a larger sample in future work.

Hyperparameter	Value	Note
Hidden Cells	220	-
No. Layers	2	-
In/Out Dims.	200	Window size of 200, steps of 1.
Dropout	0.3	-
Activation	ReLU	-
Epochs	500	Early stop w. patience 20, min. delta $1e^{-6}$.
Learning Rate	$1e^{-3}$	-
Loss	MSELoss	-
Optimiser	Adam	Default parameters.

Table 3-A: Configuration for multi-output LSTM forecaster.

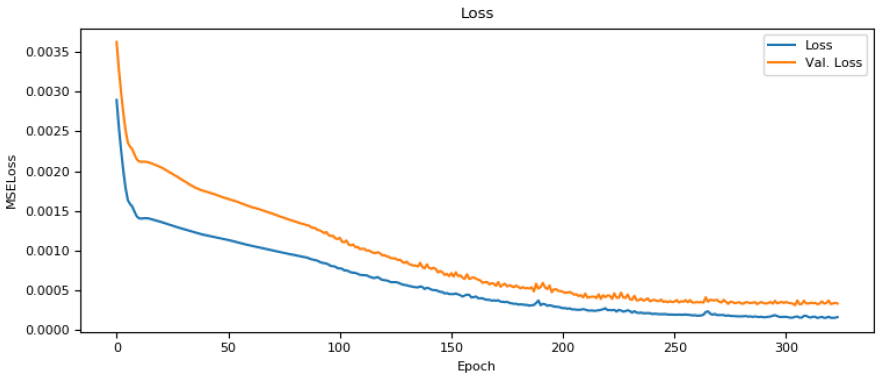


Figure 3.3: Loss curve for multi-output LSTM forecaster, as detailed in Table 3-A.

3.1.3 Evaluation

The proposed model developed the desired tendency to make forecasts with harmonic properties, whilst remaining sensitive to periods where the drumhead was inactive (not oscillating) or transitioning between modes (strike events). In Figure 3.4, we demonstrate how the forecast trajectories evolves at various positions of the rolling window, for a given flam-tap. Forecasts following the initial strike clearly incorporate harmonic properties closely resembling the amplitude and period of the actual drumhead oscillation. Although the forecast window is small in human terms, at ∼18ms of the decimated signal, it remains accurate over a non-trivial and promising 200 time steps. For example, applications might leverage the fact this exceeds the ∼10ms latency stability requirement introduced in 2.1. As hypothesised, this might be made possible due to the harmonic properties of the custom and continuous electromagnetic signal. It is unclear whether this might work for other instruments with simple functions defining their low-level dynamics, or even at the original sampling frequency (44.1kHz).

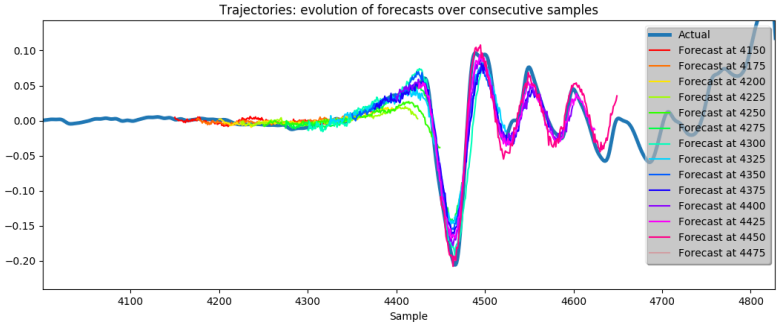


Figure 3.4: Forecasts made by LSTM for the first strike of a flam-tap event. For a given forecast, the previous 200 samples were used to project 200 at the given sample.

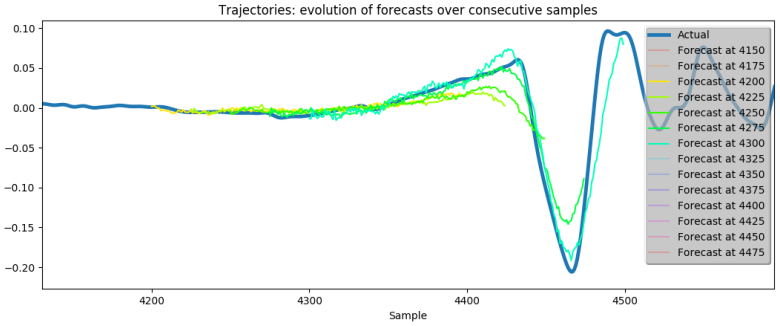


Figure 3.5: Forecasts made by LSTM for the first strike of a flam-tap event, as a subset of Figure 3.4.

Figure 3.5 focuses on the beginning the same flam-tap. Between sample 4250 and 4275, a period equivalent to only ∼2.27ms, the model is able to rapidly adapt the forecast to an evident strike action. For an observer there is no obvious signal event that might indicate an incoming strike, suggesting a subtle and complex signal property is detected. Therefore manually developing a fixed feature reflecting this property would be challenging.

In Figure 3.7, additional curves are introduced for the same strike event to indicate forecast errors. Although cumulative error noticeably rose in forecasts leading up to the flam-tap, it rose more significantly around sample 4700, prior to the second strike. This might reflect the complex transition between oscillatory modes as further energy is injected into the system - yet unfortunately this error signal offers little explanation. The LSTM's hidden state underpinning forecasts is equally impenetrable, illustrated in Figure 3.6. Hidden cell values certainly resemble plausible aspects of the signal at that point in time, but their combination for a forecast is unobvious. Further work is needed to consider how analysis of an expanded dataset might make sense of this state; and whether they even reflect performance nuances of interest. However, this signal might already be sufficient to develop a highly responsive onset detector, if anomaly detection techniques such as dynamic-thresholding were leveraged (refer back to 2.5.2).

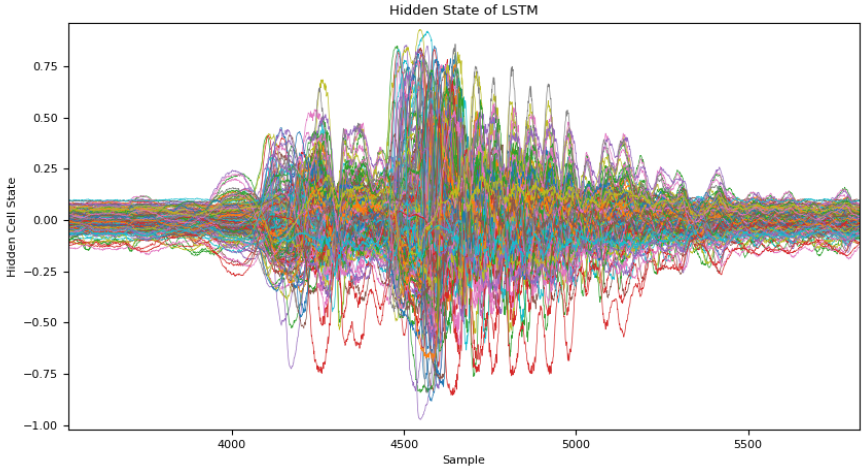


Figure 3.6: Output hidden state vectors of LSTM for a flam-tap, where each line corresponds to a different cell. Note that on a given timestep, these cells are mapped via a FC layer to form a forecast.

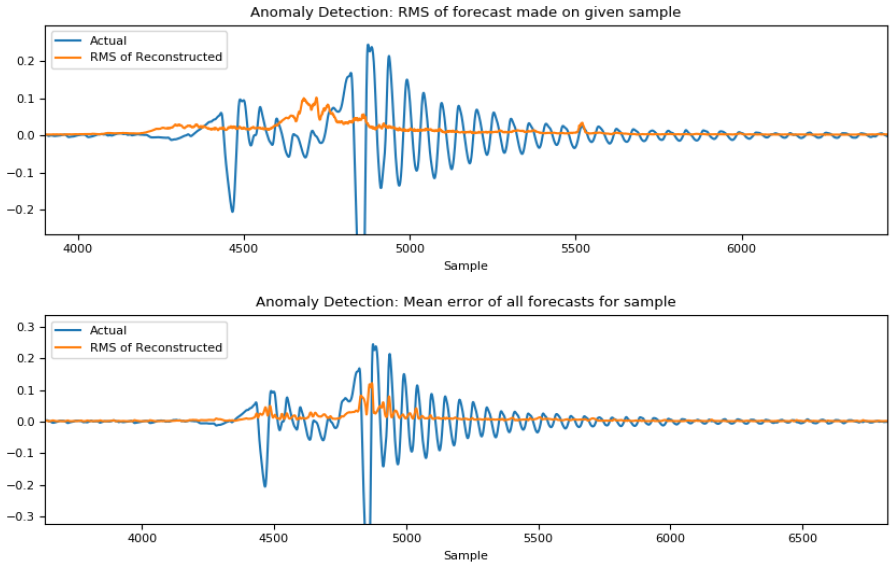


Figure 3.7: Forecast error signals, for a given flam-tap event. Top: the orange curve depicts RMS of forecasts made at the given sample. Bottom: the orange curve represents mean of accumulated RMS forecast errors, for the given sample.

In terms of model compute time, preliminary measures on the aforementioned server are promising for supporting a real-time system. It took 6.02s to process the test set (single batch) as a mean of 5 trials, which equates to 0.231ms per window. This involved windowing in steps of 1 sample (0.09ms), so it is quite possible a real-time system running on basic hardware (e.g. a laptop) might use steps closer to $\sim$10 or $\sim$25 samples. Steps of this size reflected the duration by which forecasts undergo meaningful changes, according to the examples studied. Reassuringly, neural models like this have already seen real-time usage - Martin et al. [37] demonstrated an interactive and generative neural music system using LSTMs for single time-steps. Notably however, this time-step could represent a value at a greater time delta than 1 sample.

Finally, as this solution relies on treating segments of a 1-D signal as an N-D feature vector, it is unclear how this approach might be adapted to N-D signals. This might be problematic for many instruments. For example, oscillations will vary across locations on a drumhead, demanding multiple signals. An obvious solution involves using a separate processor per channel, similar to Pardue et al. [2], but performance nuance might be reflected in the relationship between these signals. This should be considered in future work.

3.2 Summary

In this chapter basic LSTMS were adapted to make accurate forecasts of drumhead oscillations, over non-trivial window sizes, likely by virtue of the harmonic properties in the continuous sensor signal. Rapid adjustments of forecasts following strike actions suggested the model was already considering more than meets the eye.

The RNN alone seems insufficient in explaining more about the percussion performance than a simple onset detector. This is not due to lacking complexity in the model, but instead an ambiguous forecast-error signal and overly complex hidden state. This is a problem for developing DMIs which want to analyse model features, to harness performance nuance. The next chapter will explore how RNNs similar to this model might form building blocks of a larger and more malleable and interpretable model.

Chapter 4

Learning a Latent Space of Salient Features

In this chapter we use RNNs to construct neural networks that perform dimensionality reduction on time-domain sensor data from DMIs. Instead of compressing sensor data into some anomaly signal, the model creates a feature space which represents segments of the signal as smaller vectors. The goal of this is therefore to produce a model that represents features of performance with lower complexity than RNN hidden-state alone. This is explored for bowed string signals from an electric violin pickup, and challenges in using this model are addressed: (re)synthesising signals clearly, and interpreting latent representations.

4.1 Variational Recurrent Autoencoders

Referring back to 2.5.4, RNNs can be used to construct a Variational Recurrent Autoencoder (VRAE). VRAEs are trained to extract the most salient features of an input signal, such that they can compress the input signal into a latent feature space, before regenerating the signal. Similar to the GRU-based Autoencoder developed by Cowton et al. [46], a VRAE can be constructed using GRUs for encoding and decoding as per Figure 4.1. We focus on GRUs as opposed to LSTMs in this chapter simply due to their reported faster training, and LSTMs should be evaluated in future work.

Considering the physics of a bowed string, as introduced in 2.3.1, there are a few basic dimensions by which the process of Helmholtz motion could be characterised: period (pitch), amplitude, phase, direction (of bowing) etc. It is feasible that nuances in performance might involve unusual combinations of these features, or more complex features, over time. It would be valuable if a VRAE was able map windows from a sensor signal into a feature space reflecting some combination of similarly explainable properties, providing a richer means of analysing performance. This analysis wouldn't necessarily require fully understanding the features extracted; as it would still be valuable for synthesis if altering latent vectors in some dimension(s) resulted in perceptually salient changes to resynthesised signals.

In 2.5.5 we outlined a number of existing V(R)AE applications in modelling audio signals, which relied on frequency or symbolic representations. This work is concerned with retaining detail from the original time-domain signal, posing a greater challenge in input data dimensionality at the expense of resolution. Engel et al. [51] circumvent this problem in their *large* WaveNet Autoencoder by performing large strides of input chunks (32ms) as part of their CNN-based 'temporal encoder', and upsampling latent codes as biases for a powerful decoder. Their application was quite different, in learning a manifold that allows synthesising new musical notes by meaningfully interpolating between timbres of real sounds. However this may not be desired for a DMI; it is possible that the dilation over input samples might cause a loss of meaningful information for the LVM, that would otherwise be accounted for by a powerful decoder alone.

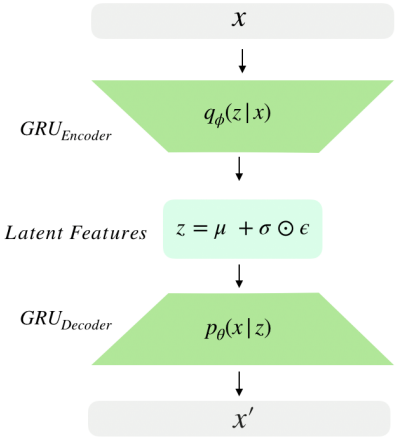


Figure 4.1: Variational Recurrent Autoencoder (VRAE) with GRU encoder and decoder.

4.1.1 Dataset

We used extracts from an existing dataset, where an electrodynamic string pickup was used to continuously measure string velocity on a violin. This is detailed by Pardue et al. [2] §2.1, and generally uses similar technology to the custom drumhead sensors detailed in Appendix A. As a starting point, we used a 150s segment of performance, with open string bowing of increasing rate on each string, for both regular and a slow bouncing technique. Beyond capturing a reasonable pitch range[1], this sample naturally contained some variation in bow position and pressure. Figure 4.2 shows samples of this training data.

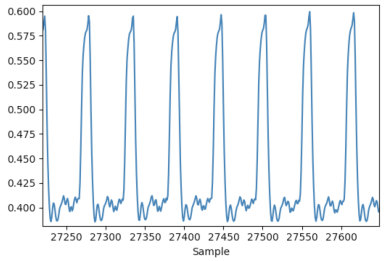
Similar to the last chapter, the signal was decimated from 44.1kHz to 11.025kHz for these preliminary assessments, and a 100Hz high pass filter was applied (not masking the lowest open string frequency). No validation split was used, as it was unclear what smaller sample would be representative of the broad training set - this would be easily solved in future work using a custom dataset.

4.1.2 Implementation

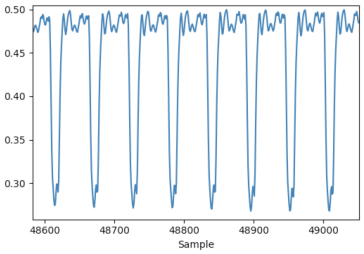
The final configuration for the GRU-based VRAE is shown in Table 4-A. As per the last chapter, note that hyperparameter optimisation wasn't performed and should be considered for future work. This would include testing LSTMs instead of GRUs, and alternative loss and activation functions (e.g. MSELoss and ReLU respectively).

We attempted to maximise re-synthesised signal quality with few latent dimensions, as the chosen 3D latent space might be more directly interpretable than something with higher dimensionality. For example, a 4D latent space or above would demand dimensionality reduction techniques for visualisation, such as t-SNE or PRCA, which would not perfectly preserve the original latent geometry. We used 100-sample windows at 25% steps, allowing overlap-add (OLA) reconstruction for synthesis (applying Hann function to decoded segments). This proved a valuable first step in improving the quality of signals resynthesised by the model. Notably, we perform compression at $\sim 33\times$ for 100-sample windows of 11.025kHz audio; whereas Engel et al. [51] report $\sim 32\times$ compression of 4s segments of 16kHz audio.

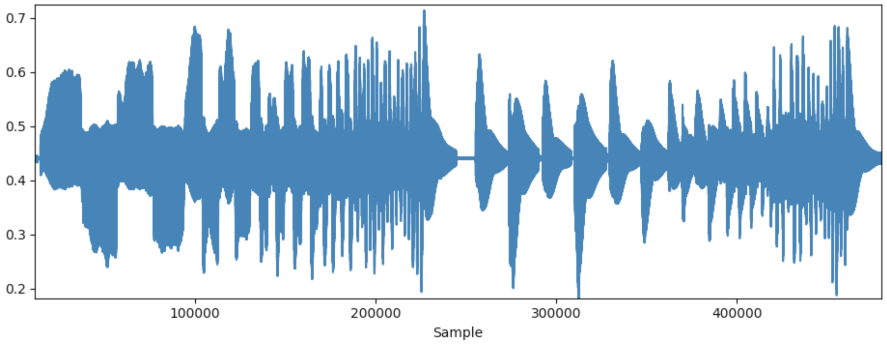
[1] The frequencies of violin strings are ~196Hz G3, ~293Hz D4, ~440Hz A4 and ~659Hz E5.



(a) Regular up bowing.



(b) Regular down bowing.



(c) All open bowing, with a gap between regular and slow bouncing.

Figure 4.2: Snippet of training data used for the VRAE - open G string velocity for bowing with increasing rate.

Hyperparameter	Value	Note
Enc/Dec Hidden Cells	128	-
Enc/Dec Layers	2	-
In/Out Dims.	1	Window size of 100, steps of 25.
Latent Dims.	3	-
Dropout	0.3	-
Activation	Tanh	-
Batches	256	Stateful training (no GRU state resets).
Epochs	500	Early stop w. patience 25 after epoch 200, min. delta $1e^{-6}$.
Learning Rate	$5e^{-4}$	-
Loss	L1Loss	-
Optimiser	Adam	Default parameters.

Table 4-A: Configuration for GRU-based VRAE.

4.1.3 Training

Training was performed on the same shared server as the last chapter, taking ∼18.8 hours. Unlike the last chapter, minibatch training was used - the dataset was chunked and gradients were updated between each chunk. As noted in Table 4-A, stateful training was used, where GRU (encoder/decoder) hidden state was not reset between consecutive batches. As the dataset was large, we felt this was important to capture long-term signal state (unlike

stateless approaches).

Training terminated due to the early stopping criteria after 302 of 500 epochs. The loss curve is shown in Figure 4.3. The noise can be explained by jumps between local minima in the learned function, during gradient descent - in moderation this seems expected and harmless, and any issues would likely fall out during later evaluation.

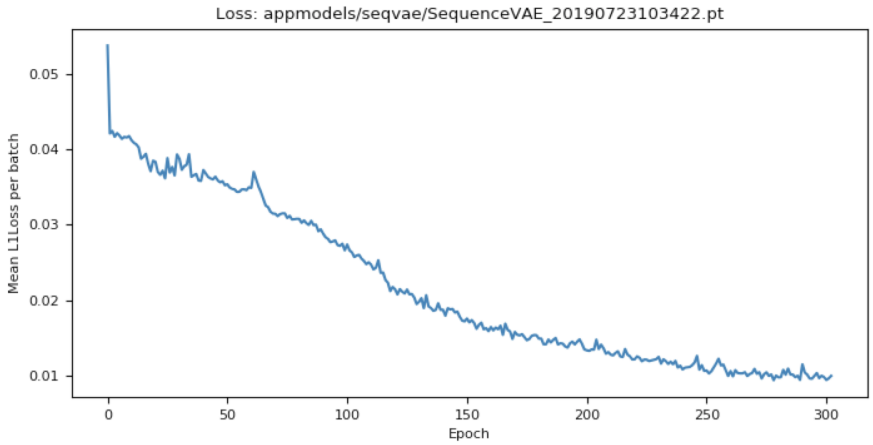


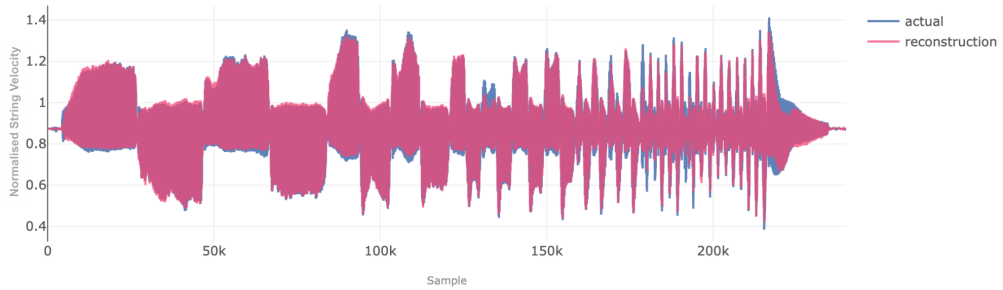
Figure 4.3: Loss curve for VRAE detailed in Table 4-A.

4.1.4 Evaluation

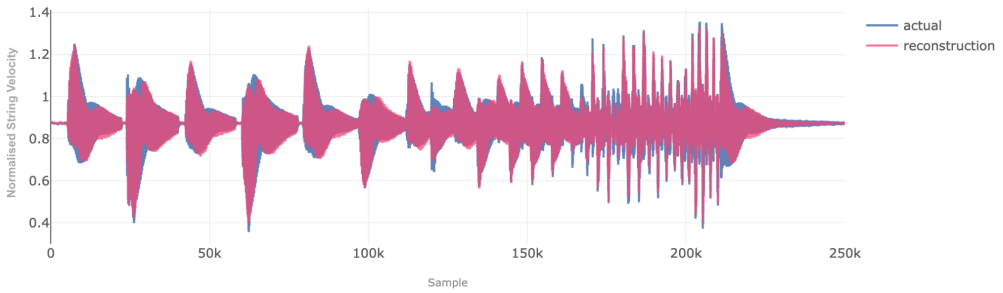
With expertise transfer in mind, in this section we evaluate whether the VRAE was able to develop a feature space sensitive to meaningful aspects of the training data. Instead of an extensive technical benchmark against varied test data, we focus on detailed aspects of the test set, as there are immediately subtle issues in signal resynthesis demanding attention. The issue of disentangling the latent space, for intuitive control in some digital instrument, is also illustrated.

4.1.4.1 Signal Resynthesis

Autoencoders often lose information upon resynthesis (reconstruction of the input signal using the decoder), as the latent bottleneck forces extreme dimensionality reduction. Despite this, when using only 3 latent dimensions, Figure 4.4 demonstrates a largely successful reconstruction at face value. Further to this, resynthesis of a 2-octave G major scale is performed in Figure 4.5. Although timbre was harsh on some out-of-sample notes, it is possible to identify all notes, despite training on only open strings.



(a) Regular



(b) Slow Bouncing

Figure 4.4: Reconstruction (using overlapp-add method) of the data presented in Figure 4.2c, bowing on open G with increasing rate.

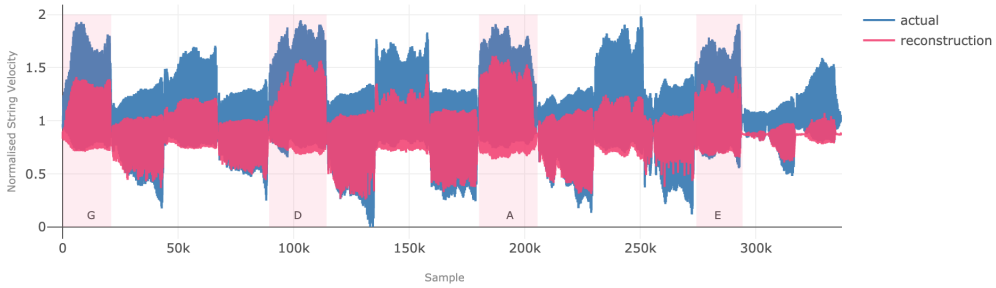


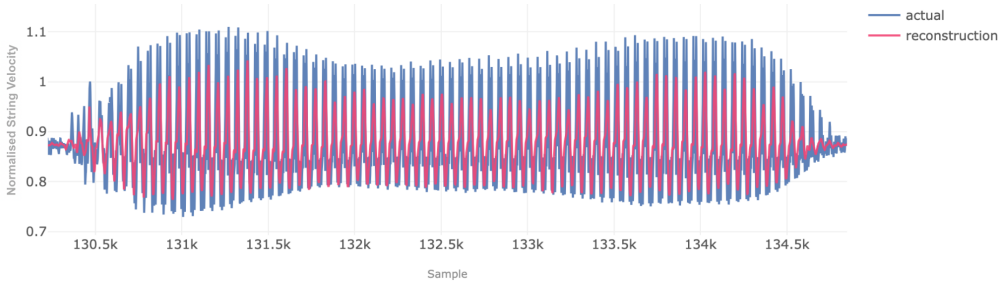
Figure 4.5: Example reconstructing a 2-octave G major scale. The in-sample notes (open strings) are highlighted.

Resynthesis Anomalies: There are two visible discrepencies in Figure 4.4a, shown in greater detail in Figure 4.6. These were both anomalies, i.e. poorly represented samples in the input distribution, with limited representation in the latent distribution. The first anomaly, Figure 4.6a, reflects a stroke in which the bow angle caused a harsh timbre. The reduced amplitude envelope of the reconstructed signal hints at an inability to meaningfully re-produce this, audibly resulting in a much quieter note. The second anomaly, Figure 4.6b, reflects where the bow plucked a string on release from the violin. This feature was not audibly present in the reconstructed signal, and only the decay of the previous bowed stroke is heard.

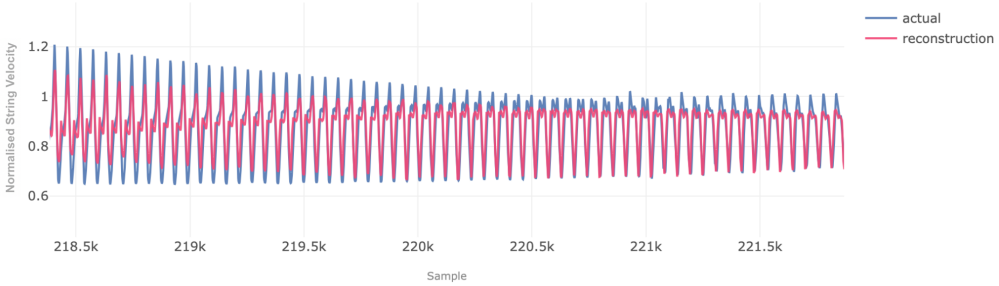
There is a recurring but less obvious issue in Figure 4.4b, demonstrated in Figure 4.7 (the 6th stroke). By bouncing the bow on the string, each stroke exhibits a unique decay in string velocitiy that isn't present in regular bowing. The reproduction suggests the model doesn't obviously differentiate this behaviour, appearing to reconstruct some warped version of down

and up strokes. This might reflect insufficient training, data, or simply a need to be overly economical with only 3 latent dimensions. It is less audible than the timbral changes resulting from the aforementioned anomalies. Together with the other anomalies, this suggests the model might fail to reconstruct more complex (and unusual) phenomena in string vibrations known in literature. For example, multiple flyback and Schelleng ripples, as detailed by Woodhouse et al. [60].

Future evaluation should examine whether this reflects the models inability to interpolate between possible vibrational properties in the signal, as this suggests a failure in expertise transfer, and a poor generative model. It is possible that the latent representation of these events is still unique and identifiable. This might also be remedied in future work using a larger dataset including more varied performance.



(a) Anomalous bow angle on the 10th stroke.



(b) Bow release clipping another string.

Figure 4.6: Examples of poor reconstruction in Figure 4.4a.

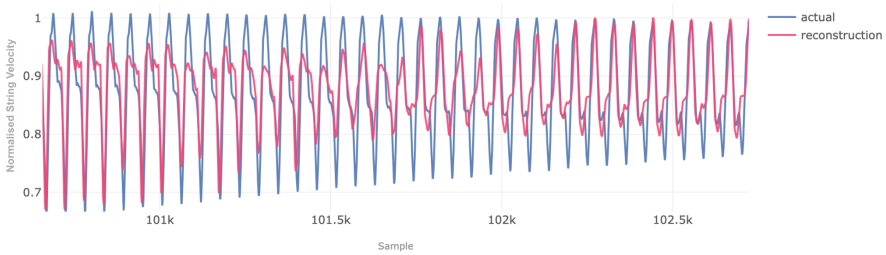


Figure 4.7: Example of poor reconstruction in Figure 4.4b.

Amplitude Variance: The most noticeable issue in the resynthesis was something resembling ring modulation. It was harsh for the G-string resynthesis, less noticeable for the D-string and minor for the other strings. In Figure 4.8 we perform a short-time Fourier transform on the original and reconstructed signals for regular bowing on the G string, demonstrating

the noise and subharmonics introduced in reconstruction.

Looking closer at an arbitrary snippet of the reconstructed Helmholtz motion in Figure 4.9, there is periodically fluctuating peak amplitude over time. Although this is not completely unusual on a larger scale, it should not be as noticeable between successive peaks. We demonstrate this for a handful of consecutive samples on Figure 4.10 - note the varying peak amplitudes in the reconstruction, alongside irregular phase misalignment. The model understandably doesn't deploy heuristics about the nature of Helmholtz motion, and largely re-produces the signal; yet the subtle misalignment causes a perceptually significant difference in resynthesis. This is also troubling in that it suggests the model isn't harnessing certain invariances over time. We explore a potential solution to this later, in 4.2.

Figure 4.10 also serves to give a view of reconstruction quality at the level of subtle vibrational effects in the string. Even after decimating the original signal by a factor of 4, there is a level of detail lost between successive peaks.

In the next section we explore the latent representation learned, in order to better understand the kind of features being extracted by the model.

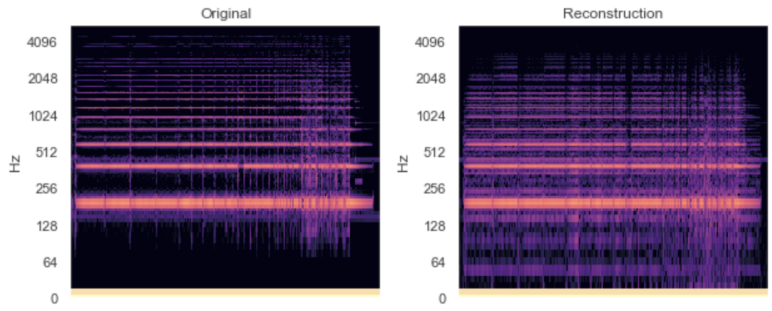


Figure 4.8: STFT of original signal and noisy reconstruction for regular open G bowing. Only the reconstruction has subharmonics of G-string frequency (∼196Hz).

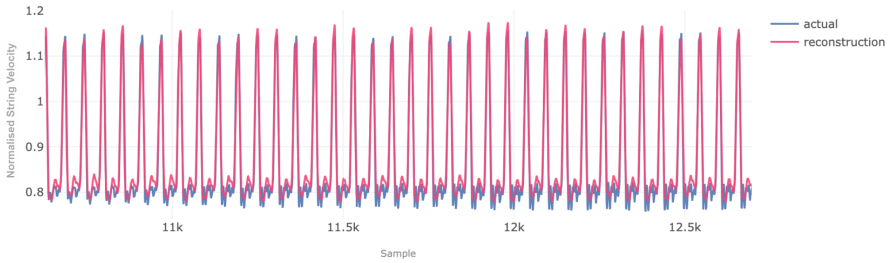
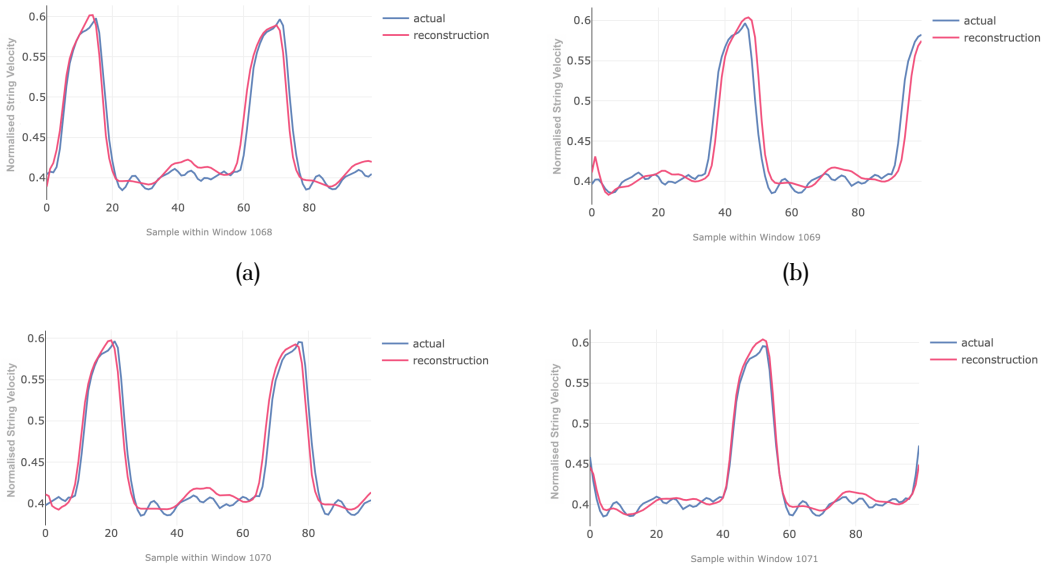


Figure 4.9: Inconsistent peak amplitudes in reconstructed Helmholtz motion.



(a) (b)

(c) (d)

Figure 4.10: Consecutive windows (25 sample step) of reconstructed signals - note the subtle variations in peak amplitude and phase across windows.

4.1.4.2 Latent Representation

Manifold Sampling: We can perform pair-wise sampling from the manifold to understand the relationship between the 3 latent dimensions. Additionally, this can suggest underlying feature representations. This is shown in Figure 4.11. It involves reconstructing (decoding) latent vectors where one dimension is held constant, i.e. $z = (z_1, z_2, 0.0), \quad (0.0, z_1, z_2)$ or $(z_1, 0.0, z_2)$ where z_1 and z_2 are linearly spaced, and mapped through the inverse CDF (percent point function, PPF) of the standard normal distribution. For example, looking at the rightmost plot. As z_2 in $(0.0, 0.95, z_2)$ is altered linearly, there is an increase in frequency and then amplitude. In all cases for z_2, altering z_1 linearly in $(0.0, z_1, z_2)$ appeared to cause a change in bowing direction. More granular sampling might elucidate these relationships, but it is evident already that simple features such as amplitude might be encoded through non-linear relationships between more than 2 dimensions. This is not surprising, given the complexity of the signal in question, and limited 3D space.

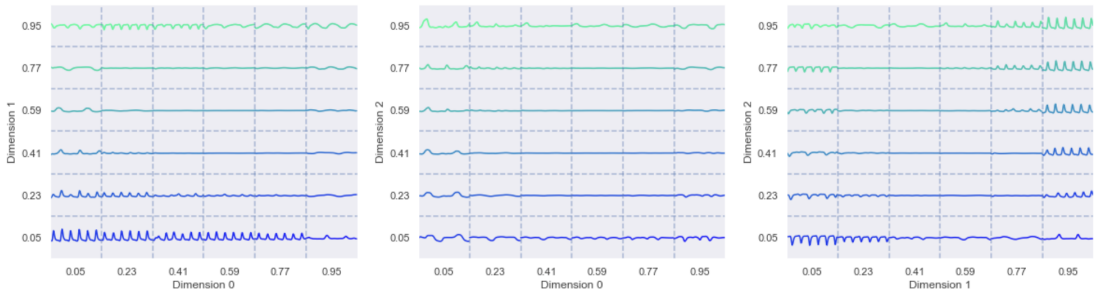
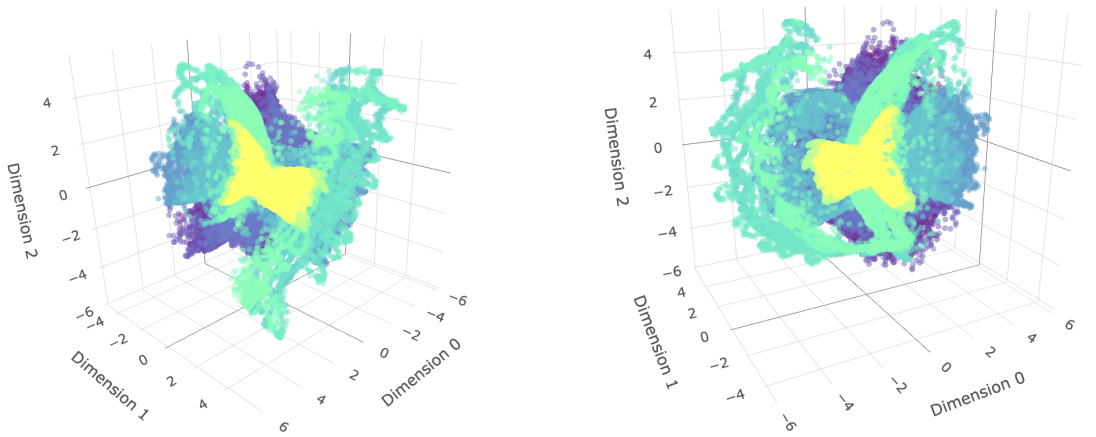


Figure 4.11: Sampling from VRAE manifold. Linearly-spaced pairs of dimension values were used to fill empty 3D latent vectors, before decoding (reconstruction).

Latent Embeddings: In Figure 4.12 we plot all latent embeddings of the dataset (across all strings) in this 3D space. Note that the colour gradient reflects time (sample) in the original dataset, which sequentially moved across violin strings. The embedding is composed of four overlapping pairs of spherical caps, where caps oppose each other in pairs like an hourglass. The quirky geometries are more obvious in Figure 2.2 where the embeddings are separated by string. Circles are native figures in polar coordinates, and often characterise a phase portrait, making it unsurprising that they might emerge in the latent embedding of our windowed signal. It follows intuitively that the pairs of cup relate to opposing directions of bowing, per string. Beyond phase and bowing direction, it is less intuitive how other features may be absorbed into the latent geometry. For example, certain timbres may be achieved by making unique concentric patterns of the respective string's hourglass. It is unclear why each string has formed contrasting shapes of hourglasses in their embeddings.

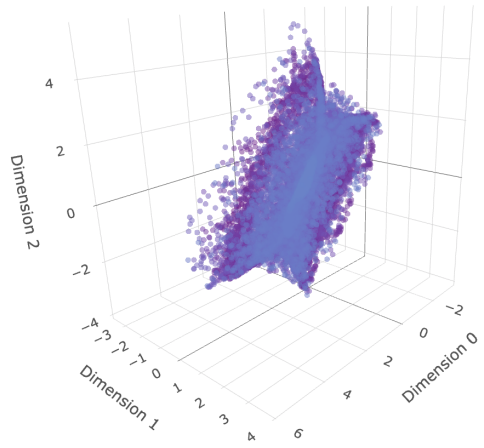


(a) (b)

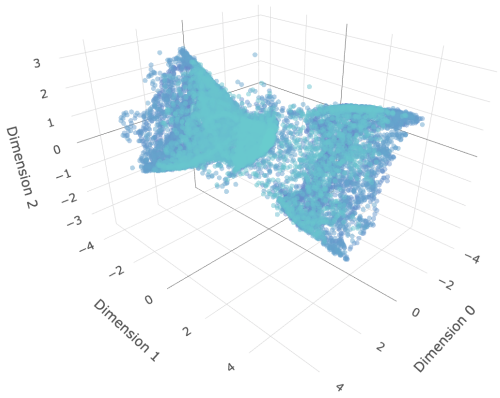
Figure 4.12: Latent embedding of full dataset, with colour gradient over sample/time (bowing was sequentially performed on the G, D, A then E strings).

Latent Traversal: In Figure 4.14 we confirm our intuitions about the latent geometry. We plot the trajectory of latent vectors over a bow direction change, with the colour gradient reflecting time (sample). Latent embeddings form the vertices of star polygons, as opposed to circles. This discretisation is unsurprising given that windows encompass at least one cycle

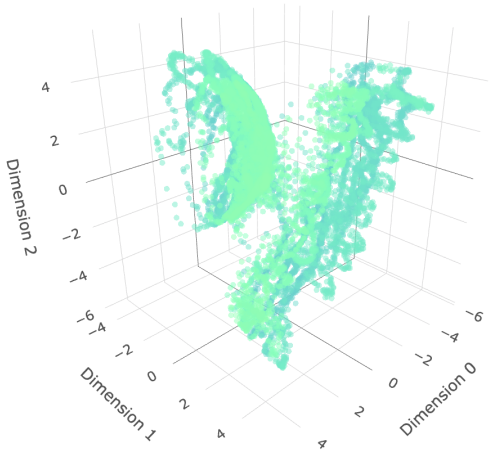
and we use large windowing steps (25%). As bow direction changes, latent vectors transition between spherical caps of the hourglass, as shown in Figure 4.14c. It remains unclear how the geometry of the star polygons incorporate many features of the signal over time. However, the consistent latent trajectories are promising, and suggest some approach introduced introduced in 2.5.5 might be used to harness the model in a generative capacity. We explore this later in 4.3.



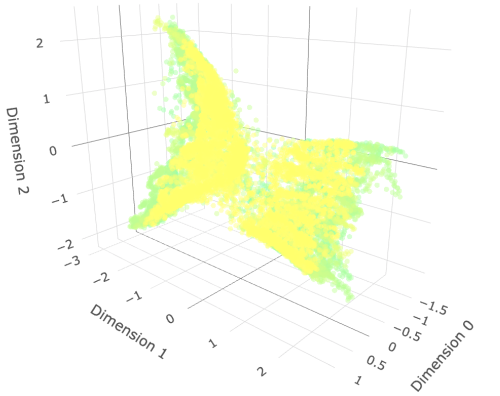
(a) G String



(b) D String

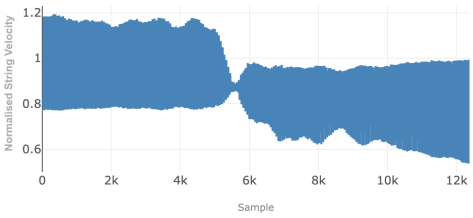


(c) A String

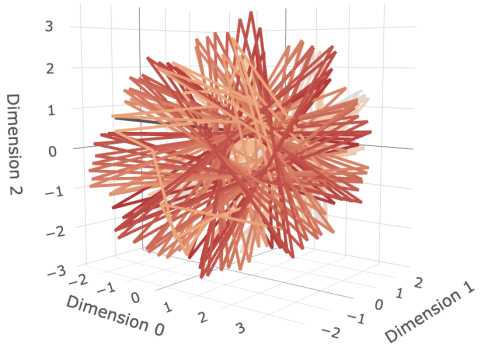


(d) E String

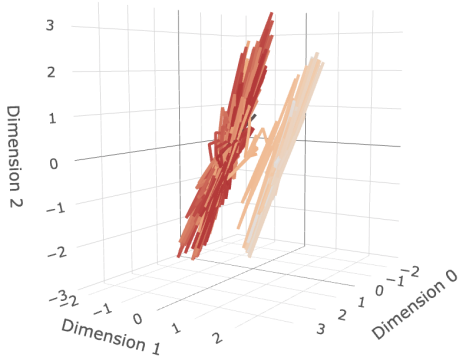
Figure 4.13: Latent embedding from Figure 4.12, with the same colouring, separated according to the string of input samples.



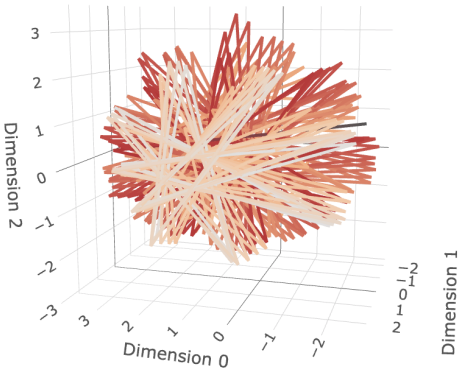
(a) Bow direction change on G string



(b) Facing down-bow embeddings



(c) Facing down-up transition



(d) Facing up-bow embeddings

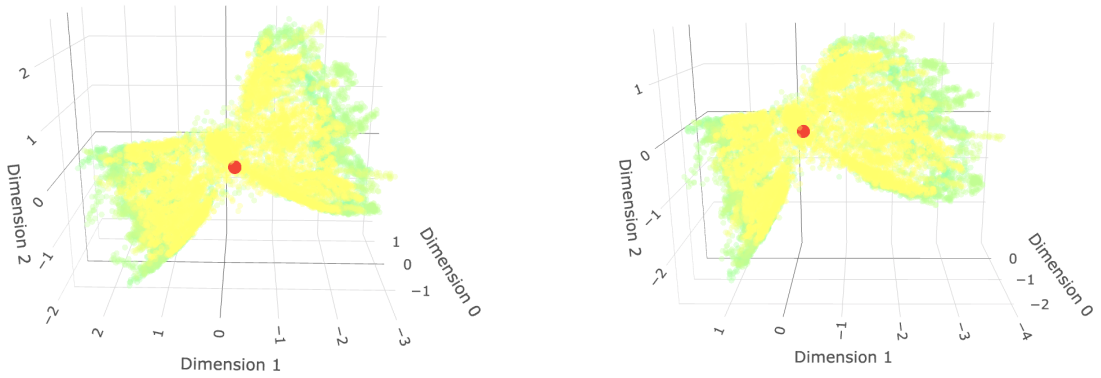
Figure 4.14: Trajectory of latent embeddings over a bow reversal - forming separate star polygons. The colour gradient corresponds to time/sample (later samples are brighter).

Latent Warping: Despite all suspicions over latent oddity, we performed a crude experiment to confirm whether simple transformations of latent embeddings would disrupt resynthesis (or maybe produce interesting effects). We found the K-Means centroid[2] of all latent embeddings for bowing on the open E-string, and applied an attractive force from this position. Force was applied according to Hooke's law of a spring's restoring force, $F_s = -kx$, with x as the distance between an embedding and the centroid, and k was empirically set to 0.3. The warped latent space is shown in Figure 4.15.

This arbitrary manipulation of latent space largely distorted the resynthesised signal (Figure 4.16), making it more harsh and unpleasant. Alone, this suggests more thoughtful interpretation and use of the latent space is necessary. However, with this manipulation timbre was consistently altered. Most regular up strokes had a different and prominent tone, around ~220Hz (A3), as per the spectra in Figure 4.17. This reflects periodically prominent amplitudes, shown in Figure 4.16b. This pitch is novel - it doesn't reflect the open A string bowing in the original dataset (which is A4, ~440Hz). Where Dimension $1 > 0$, the notably expanded spherical cap of the warped E-string (Figure 4.15b) begins to resemble the original A-string

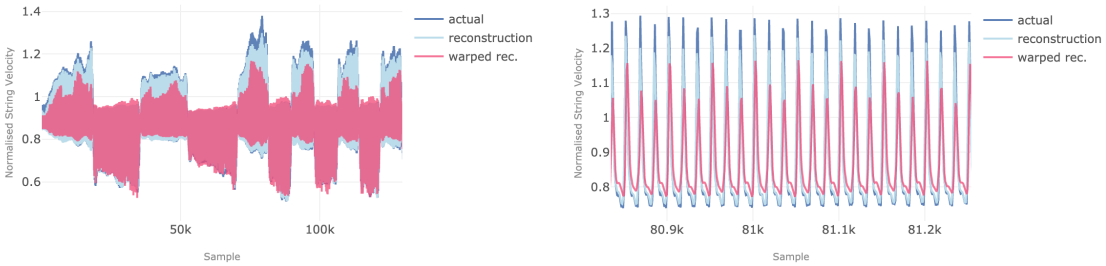
[2]We performed K-Means clustering using the `KMeans` class of scikit learn with default parameters: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

embeddings (Figure 2.2c). Arguably this might explain how an A-note was produced on a lower octave, alongside the noticeable change on only one stroke direction. It might simply be coincidental that using $k = 0.3$ resulted in a new frequency component which is almost $3\times$ smaller than the original, A3 (220Hz) compared to E5 (659Hz). Although this observation is not particularly explainable in terms of latent features, it suggests this model might not be far from use in a generative capacity. Future work might create an interactive latent space, such that simple mutations can be explored and potentially disambiguated.



(a) Pre-Warp (b) Post-Warp

Figure 4.15: Pre-and-post warping the latent space of open E string embeddings, using a centroid attractor (red sphere).



(a) Contrasting Up/Down-Bow Warped Rec. (b) Periodic Amplitude in Warped Rec.

Figure 4.16: Resynthesis of regular open E bowing, after latent warping in Figure 4.15.

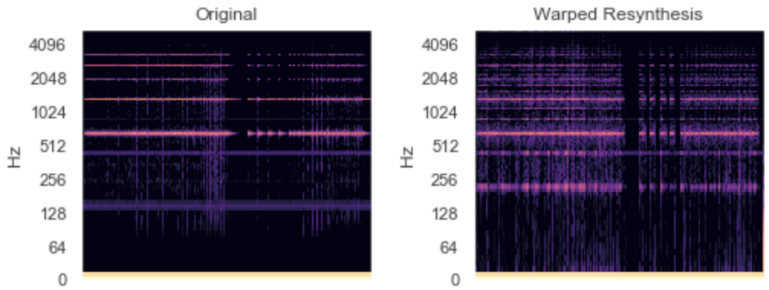


Figure 4.17: STFT of signal reconstruction after latent warping, for regular open E (∼659Hz) bowing. Note the prominent ∼220Hz component (A3) introduced.

4.1.4.3 Phase Invariance & Posterior Collapse

The previous section illustrated that a key obstacle in harnessing latent space might be disentangling phase, or developing a technique to learn meaningful sequences of latent vectors. Key to this issue is phase-variant input data, which naturally means any representation learned includes phase. Given the signal in question is periodic, it feels inefficient to have the complication of learning phase from the irregular windows presented. We believe that ultimately avoiding this overhead might be instrumental in developing more meaningful feature representations. The challenge of achieving phase-invariance is similar to harnessing invariances (e.g. peak amplitude) across consecutive windows, as proposed in 4.1.4.1 and later addressed.

Notably we did attempt a basic solution to this, using a pitch-synchronous overlap-add methodology (PSOLA). This involved: segmenting the input signal at pitch markers (peaks/local maxima) according to empirically derived thresholds; extracting a window of samples around each marker that encompasses neighbouring peaks; and applying the Hann function. This is a common technique used to modify the pitch and duration of signals, which we suspected might encourage learning local features in a less phase and frequency sensitive manner.

We immediately ran into a well-known issue called posterior collapse (or latent variable collapse), whereby the model fails to learn meaningful latent representation of data points. This doesn't rule out good reconstruction of the input signal, as the decoder might be powerful enough alone. In our case, this manifested itself as the decoder consistently learning to re-producing the Hann function, shown in Figure 4.18. The result of this is the resynthesis of pure(-ish) tones. Without applying the Hann function to input windows, the model learning did not converge with reasonable loss. There are several remedies to posterior collapse presented in literature (e.g. [61, 62]), such as weakening the decoder, but we have left this for future work.

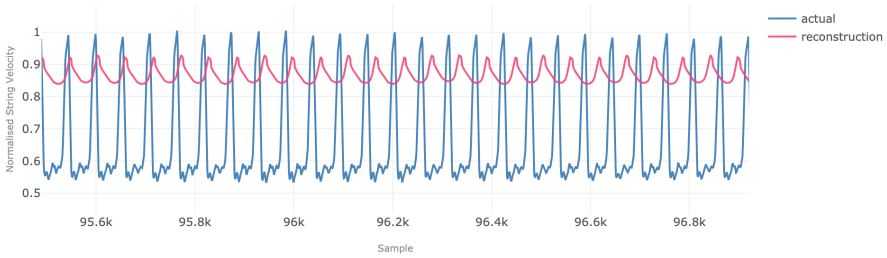


Figure 4.18: Example of failed reconstruction due to posterior collapse, during PSOLA-esque input processing. The reconstructed signal reflects application of a Hann window to input samples.

4.2 Sequence Transformer

It became evident in 4.1.4.1 that quality of resynthesis by the VRAE wasn't satisfactory. It was hindered by not harnessing invariances between consecutive signal windows. In this section, we look to tackle that.

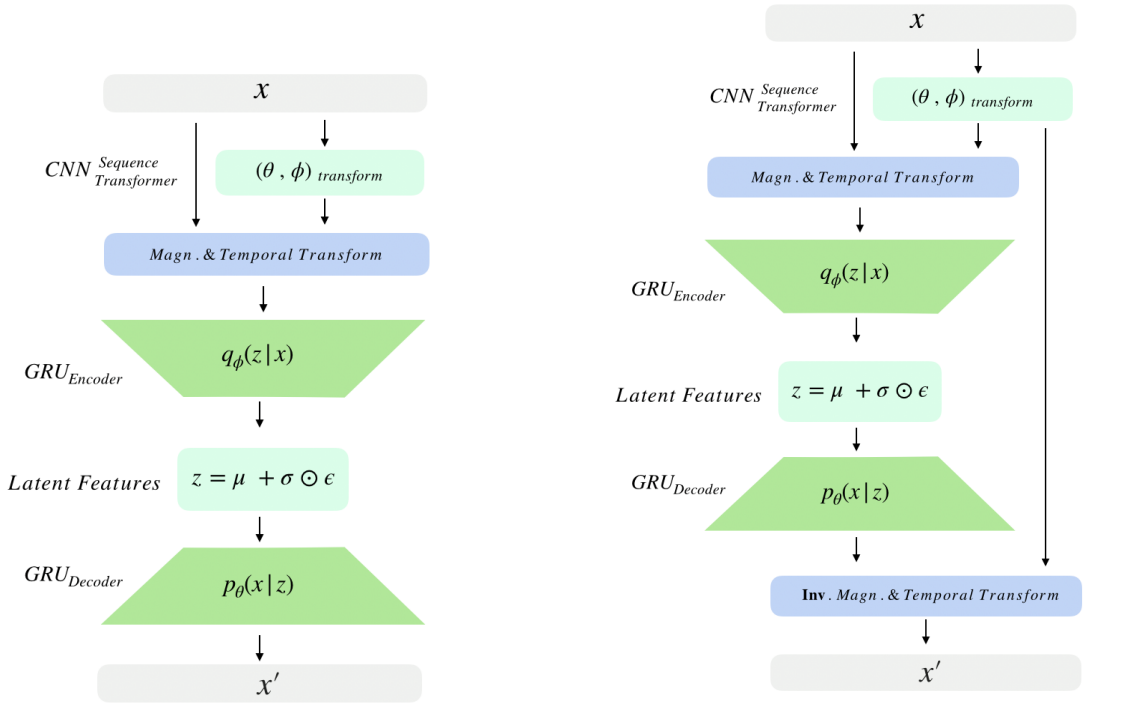
4.2.1 Learning Invariance

Generally, in order to improve reconstruction quality of a VAE, the encoder and decoder hidden state may be increased in size. More obviously, the number of latent dimensions used to describe the input signal could simply be increased. However, an alternative approach is to remove invariances from the learning task using a reversible transformation, and then to re-introduce them later.

This was introduced by Jaderberg et al. [63] to remove spatial invariances in images, through translation, scaling, rotation and warping. This involved creation of a 'Spatial Transformer' CNN sub-network, that might learn to ignore spatial invariances as part of a global learning task. CNNs by design harness phase/spatial invariance, in order to detect underlying semantic features. This makes CNNs suitable in adjusting away task-irrelevant features from input samples. Oh et al. [64] adapted this as a 'Sequence Transformer' (ST), for transformations on clinical time series, to improve classification performance. In a similar vein to Bidart et al. [65] —which introduces a reversible affine transform to a VAE that classifies handwritten digits— an ST can be added to a VRAE. As per Oh et al. [64], the CNN sub-network explicitly learns the parameters for a reversible temporal transformation ($\theta^{(i)} \in \mathbb{R}^{n\times 2}$) and magnitude transformation ($\phi^{(i)} \in \mathbb{R}^{n\times 2}$) of a time series. As part of efficiently constructing the latent space and improving reconstruction quality (refer back to Equation 2.2), this might naturally tweak the input signals. For example, amplitude variances might be stretched out or phase variability might be shifted away, vastly reducing the scale of the VRAE's learning problem.

There are two forms of VRAE with Sequence Trasnsformer (VRAE-ST) demonstrated in Figure 4.19. We will test both of these models. In terms of the functional differences:

1. Figure 4.19a is a VRAE-ST which *only* transforms input sequences prior to encoding. We will refer to this variant as a VRAE-STe. This model assumes the decoder is already powerful enough to reasonably reconstruct any input data, but that it might be provided more meaningful latent encodings if the encoder is able to ignore task-irrelevant invariances in the input.
2. Figure 4.19b is a VRAE-ST with both input transformation (prior to encoding) and inverted output transformation (following decoding). This model also allows the encoder to ignore task-irrelevant invariances, before eventually reversing input transformations to simplify the decoder's learning task.



(a) Encoder transformer (VRAE-STe).

(b) Reversible encoder/decoder transformers (VRAE-ST).

Figure 4.19: VRAE-ST variants - as part of the VRAE, a CNN which extracts transformation parameters θ and ϕ for static transformation layers.

4.2.2 Implementation

Generally, the network follows the same shape as a typical VRAE. The first addition is a 'localization' layer, a small CNN, which extracts the four parameters for transformation. The CNN implementation is demonstrated in Figure 4.20 - featuring two simple 1D convolution layers with pooling and output mapped through a FC layer for the four transformation parameters. The final restricted activation layer follows the implementation of STs by Oh et al. [64]; where gradient clipping of the transformation parameters prevents the sub-network from getting stuck in unrecoverable states during training.

The four acquired parameters are directly applied via static transformation functions. Temporal transformations are implemented as an affine transformation, using the `affine_grid` and `grid_sample` utilities provided in PyTorch. These associate an input signal with a grid of coordinates, and perform bilinear sampling to apply a given transformation; the same approach as Bidart et al. [65]. The affine matrices have 2×3 dimensions, where $\theta_{1,2}$ populate the first two parameters of an otherwise restricted matrix: $((\theta_1, \theta_2, 0),\ (0, 1, 0))$. This allows shifts, shear and scaling along the temporal axis. As window sizes are retained after temporal transformation, contiguous value padding is performed where necessary. Magnitude transformations are much simpler, performing multiplication of (ϕ_1) and addition (ϕ_2) to the temporally adjusted signal. This allows shifting and scaling along the magnitude axis. For both sets of parameters ($\theta_{1,2}$ and $\phi_{1,2}$), we introduce an initial network bias of $(1, 0)$ reflecting identity transformation.

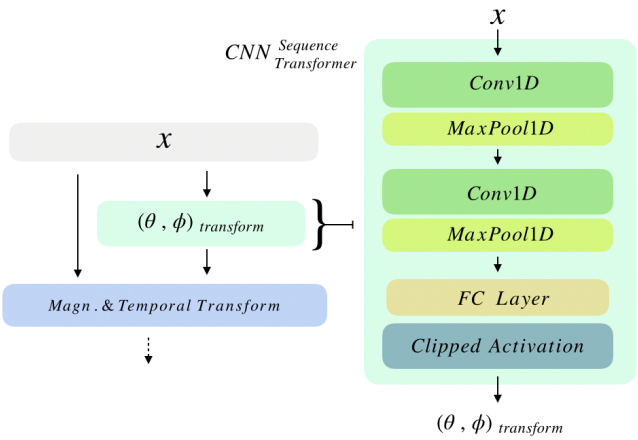


Figure 4.20: Sequence Transformer (ST) sub-network, a simple 1D CNN.

The final configuration for the VRAE-ST and VRAE-STe can be found in Table 4-B and Table 4-C respectively. For brevity we have only detailed the ST parameters, as other parameters follow the original VRAE detailed in Table 4-A (other than the increased learning rate). We were surprised to find that the VRAE-STe performed well under a kernel size of 1, such that the Conv1D layers behave as FC layers.

Hyperparameter	**Value**	**Note**
Conv1D_1	KernelSize=7, Channels=8	-
Conv1D_2	KernelSize=5, Channels=10	-
$\text{MaxPool1D}_{1,2}$	KernelSize=2, Stride=2	-
FC Dims.	24	-
Dropout	0.3	Applied after Conv1D_2
Activation	ReLU	Applied after $\text{MaxPool1D}_{1,2}$
Clipped Activation	ValueRange=$\pm$1.75, MaxDelta=$\pm$0.01	-
Learning Rate	$1e^{-3}$	-

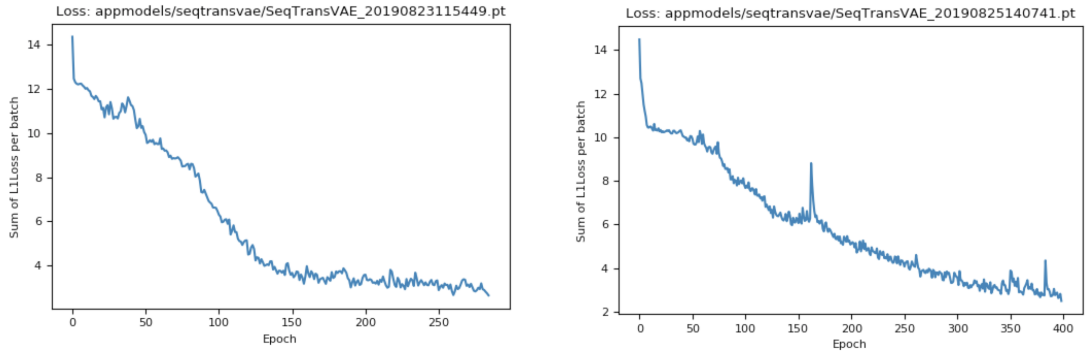
Table 4-B: Configuration for ST of the VRAE-ST.

Hyperparameter	**Value**	**Note**
Conv1D_1	KernelSize=1, Channels=8	-
Conv1D_2	KernelSize=1, Channels=10	-
$\text{MaxPool1D}_{1,2}$	KernelSize=2, Stride=2	-
FC Dims.	24	-
Dropout	0.3	Applied after Conv1D_2
Activation	ReLU	Applied after $\text{MaxPool1D}_{1,2}$
Clipped Activation	ValueRange=$\pm$1.75, MaxDelta=$\pm$0.01	-
Learning Rate	$1e^{-3}$	-

Table 4-C: Configuration for ST of the VRAE-STe.

4.2.3 Training

Training the VRAE-STs was no different to a typical VRAE, as the ST is optimised during VRAE training to minimise the VRAE's loss. The loss curves are shown in Figure 4.21. VRAE-STe training terminated after 309 of 500 epochs (∼15.45 hours) due to the early stopping criteria, and the VRAE-ST terminated after 423 epochs (∼21.15 hours). Given the increased network size versus the vanilla VRAE (more parameters introduced by the ST), we increased the learning rate to prevent excessive training times, at the potential sacrifice of model accuracy. Unlike Oh et al. [64] we introduced dropout within the ST and reduced the FC dimensions in order to avoid overfitting; which was important in preventing the ST converging to a single set of transformation parameters across all input samples.



(a) VRAE-STe (b) VRAE-ST

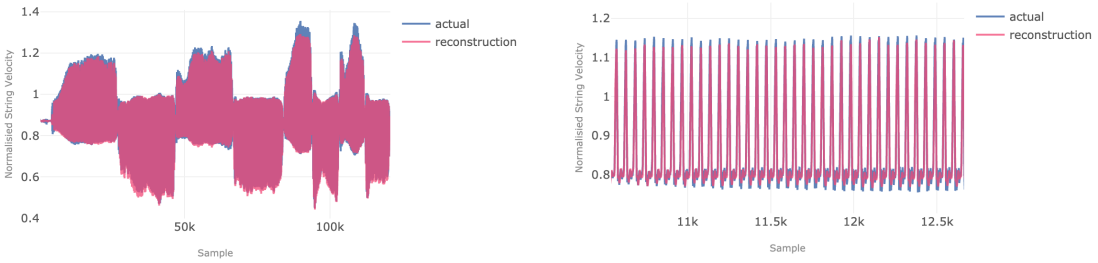
Figure 4.21: Loss curves for VRAE-ST variants detailed in Tables 4-B & 4-C.

4.2.4 Evaluation

Similarly to 4.1.4, in this section we evaluate the resynthesis quality of the VRAE-STs before examining the corresponding latent representations. Additionally, we compare the transformations that the models learn to perform.

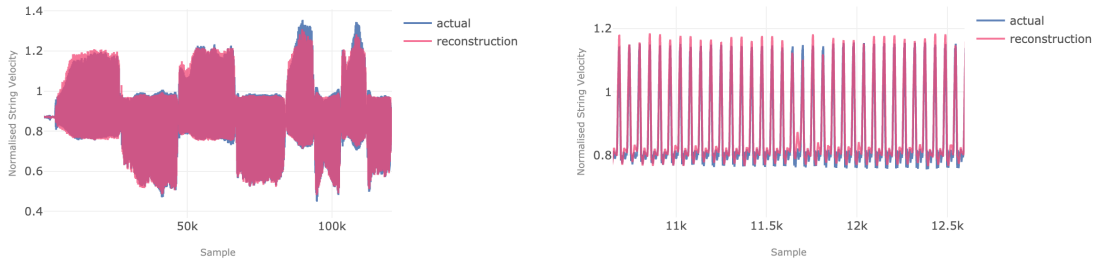
4.2.4.1 Signal Resynthesis

Both models were able to produce a perceptually clearer but still imperfect signal. In the original VRAE we noticed periodic changes in peak amplitude over successive cycles of the resynthesised signal (Figure 4.9), which caused an unpleasant effect resembling ring modulation. In both Figure 4.22b & 4.23b there is visibly less amplitude variation - the peaks are smoothed out. Examining the spectra in Figure 4.24, the VRAE-STe and VRAE-ST seem to blur any spurious frequencies below G3 (196Hz). Despite the visible noise, the VRAE-ST produces a much clearer signal than the other models, with greater depth in timbres produced. The original VRAE, in comparison, sounds dull and incomplete. The VRAE-STe sits somewhere in the middle.



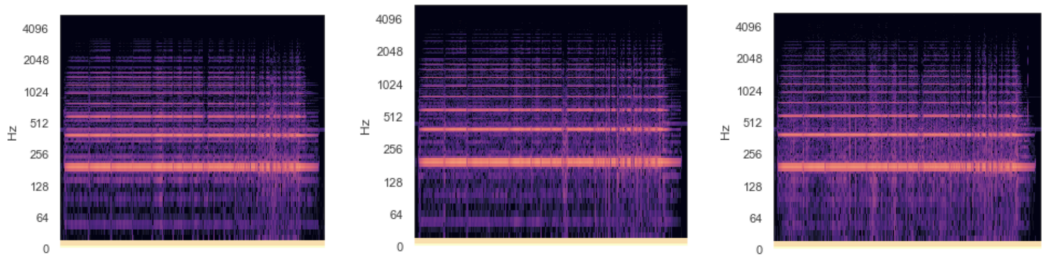
(a) (b)

Figure 4.22: VRAE-STe reconstruction for a segment of regular bowing on open G.



(a) (b)

Figure 4.23: VRAE-ST reconstruction for a segment of regular bowing on open G.



(a) VRAE (b) VRAE-STe (c) VRAE-ST

Figure 4.24: Comparing STFT of regular open G (196Hz) bowing for all VRAE(-ST) models.

4.2.4.2 Sequence Transformations

Unsurprisingly, the VRAE-ST variants developed contrasting tendencies in transforming input samples. We will detail these in turn.

VRAE-STe Transformations: Figure 4.26 shows that there was highly consistent transformation across the entire dataset, especially temporally. Every sample was compressed temporally, with minor shifts; and every sample was stretched in magnitude, with some degree of shift downwards. It is unclear exactly why this was advantageous in feature learning, but one explanation is that this resembles downsampling, where there is less complex information processed by the encoder/decoder. Examples are shown in Figure 4.26. Given that samples were only transformed on input (prior to encoding), the decoder may have relied on

certain variances to be represented in latent space. This might have handicapped the range of transformations performed.

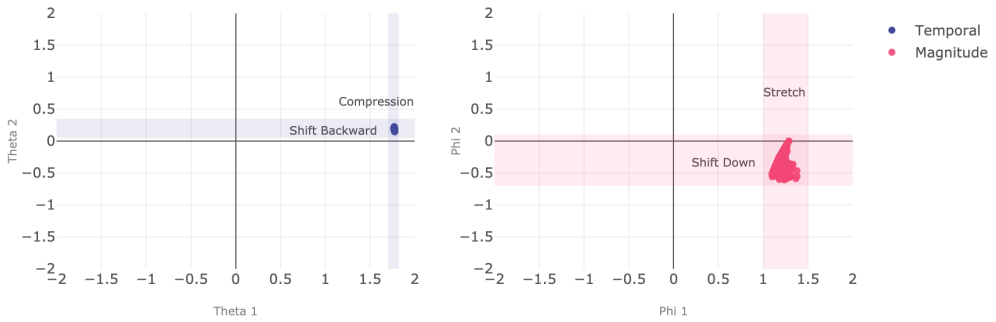
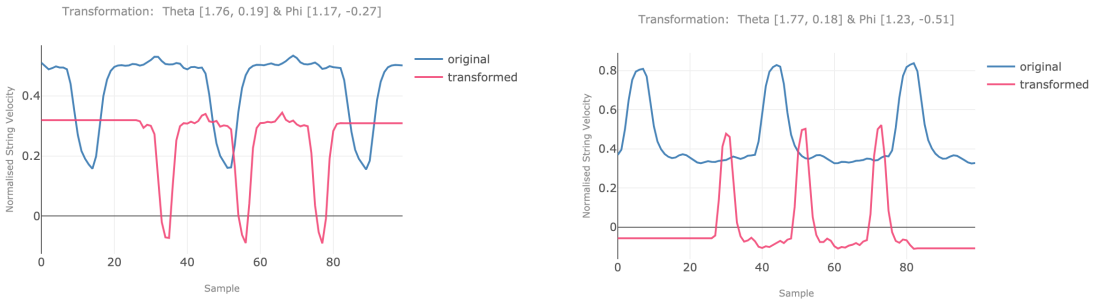


Figure 4.25: Transformations performed by the VRAE-STe across full dataset.



(a) (b)

Figure 4.26: Example transformations performed by the VRAE-STe.

VRAE-ST Transformations: Figure 4.28 demonstrates a more varied range of transformations in this model. There was a highly uniform relationship in magnitude transformations, where samples were scaled and shifted proportionally - a simple linear scheme that could be harnessed by the reversible transformation. Bigger stretches required shifting the sample more, seemingly corresponding to the desired amplitude normalisation. Examples are shown in Figure 4.28. Similarly to the VRAE-STe, the model compressed all samples temporally (resembling downsampling), but also introduced varying shifts in time.

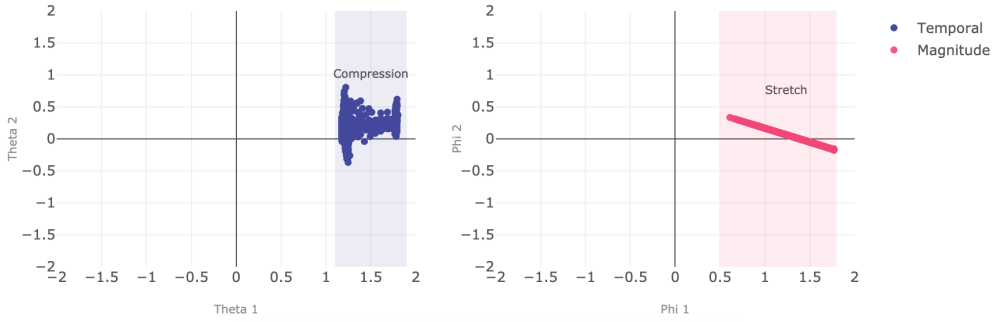
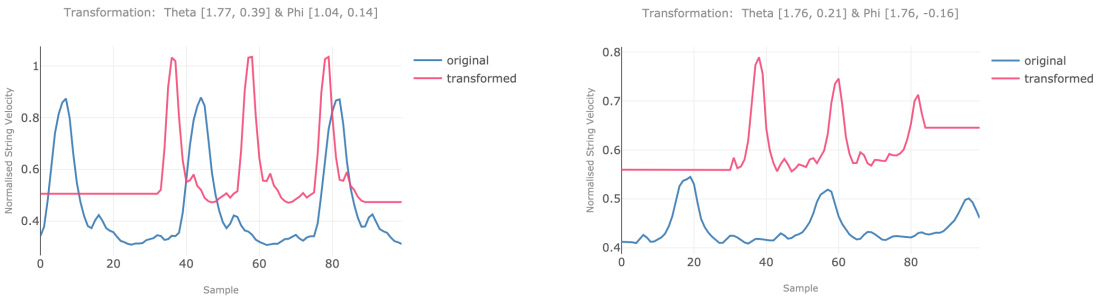


Figure 4.27: Transformations performed by the VRAE-ST across full dataset.



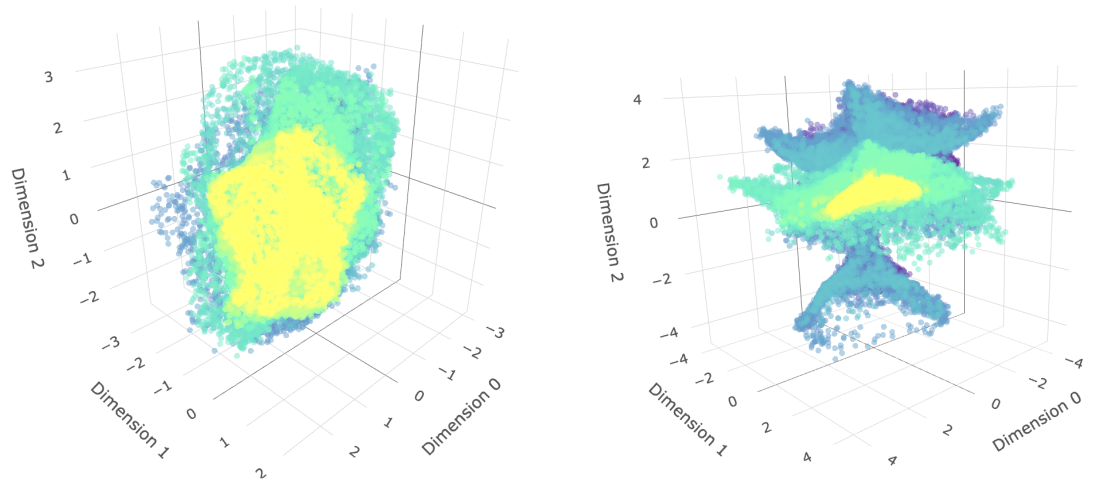
(a) (b)

Figure 4.28: Example transformations performed by the VRAE-ST.

4.2.4.3 Latent Representation

In Figure 4.29 we observe the latent embeddings of the VRAE-ST variants from the same angle as the original VRAE embeddings in Figure 4.12. Both are notably different, reflecting the alternate manifolds developed. Both variants continue to take the shape of spherical caps (seperated plots can be found in Appendix B). Unlike before, embeddings for each string don't always resemble an hourglass .

Spherical embeddings suggests that introducing an ST did not avoid the complication of phase variance across windows. There was clearly no drastic change in the latent geometry such that it becomes more explainable and malleable, suggesting further methods are needed to harness the latent space. It is also possible that this improved quality of resynthesis might hinder generative applications, if it is achieved by normalising latent samples according to important perceptual features (such as amplitude). This trade-off will ultimately depend on the target application of the model.



(a) VRAE-STe (b) VRAE-ST

Figure 4.29: Latent embedding of full dataset by the VRAE-ST variants, with colour gradient over sample/time.

4.3 LatentRNN

We observed that the VRAEs embedded the dataset in hourglass shapes, where latent vectors formed trajectories resembling star polygons. Despite being unintuitive in terms of feature representation, in this section we *briefly* propose a method of harnessing these latent sequences, which might be a key focus of future work.

4.3.1 Learning Latent Sequences

In order to disentangle representations learned by a VAE, various rigorous approaches to regularisation have been deployed in existing work (refer back to 2.5.5). Unfortunately regularisation can be challenging where the input space is poorly defined. As an alternative approach, Ha et al. [56] and Pati et al. [57] both demonstrated that an additional RNN within a VAE could learn how to traverse latent space ('LatentRNN'). Given the latent space may be highly nonlinear and complex, RNNs seem well suited to learn patterns of latent vectors. We propose that a LatentRNN could be introduced to a VRAE, learning meaningful latent sequences for a given task. This is demonstrated in Figure 4.30. At any point, the LatentRNN can use the history of latent vectors to forecast the next. This can then be used recursively to forecast a sequence (multi-output forecast), for generative uses.

One performance application of a LatentRNN might be to hold a given note (with respective timbre), freeing up the performer to do something else. For example, a violinist may trigger the LatentRNN via some pedal before releasing the bow, causing the LatentRNN to continue the stroke generatively. After learning to better harness the latent space, the performer may additionally be able to interact with the LatentRNN in order to introduce effects (subtely altering the sequence of latent variables generated).

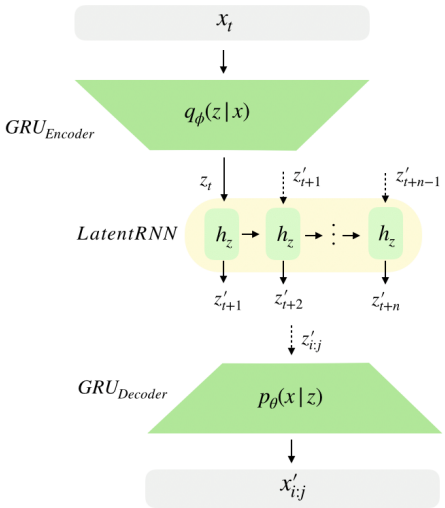


Figure 4.30: LatentRNN layer of a VRAE, demonstrating recursive forecasting of latent variables at future time steps.

4.3.2 Implementation

Unlike the previous models we explored, the LatentRNN combines multiple models in a single architecture. We leverage a regular VRAE (i.e. not an ST variant) similar to that implemented in 4.1.2 (but instead with 5 latent dimensions), using it as a *pre-trained* submodel of the LatentRNN. This technique is often referred to as 'transfer learning'. The RNN component of the LatentRNN is a basic GRU, and does not harness the multi-output forecasting approach developed in Chapter 3. The final configuration for the LatentRNN is shown in Table 4-D.

Hyperparameter	Value	Note
Hidden Cells	512	-
No. Layers	2	-
In/Out Dims.	5	Window size of 25, steps of 1.
Dropout	0.1	-
Activation	Tanh	-
Batches	25	Stateless training (GRU state resets).
Epochs	500	Early stop w. patience 15, min. delta $1e^{-6}$.
Learning Rate	$5e^{-4}$	-
Loss	MSELoss	-
Optimiser	Adam	Default parameters.

Table 4-D: Configuration for LatentRNN.

4.3.3 Training

During training, mini-batches of the windowed input signal were first passed through the VRAE encoder, and then the respective latent encodings were windowed further. The LatentRNN was trained on these latent windows, learning to forecast the following latent encoding. Training took ~6.4 hours, terminating at 195 of 500 epochs due to the early stopping criteria. The loss curve is shown in Figure 4.31.

The VRAE submodel was 'frozen' throughout training, with parameters being explicitly excluded from gradient computation - i.e. it was not further optimised as part of the LatentRNN training process.

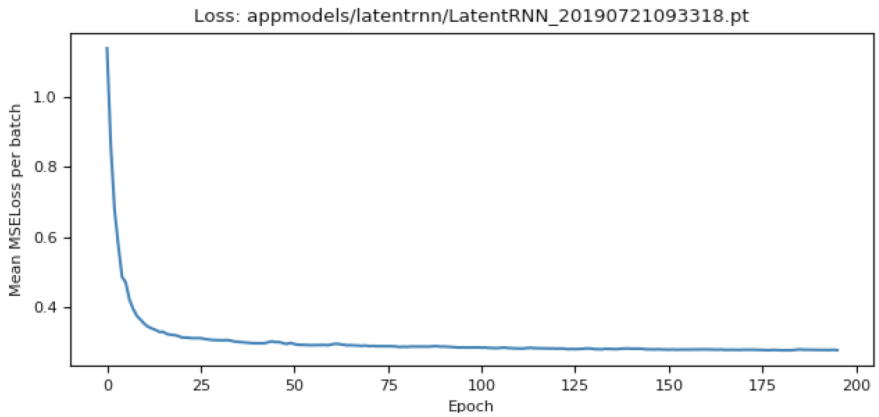


Figure 4.31: Loss curve for LatentRNN detailed in Table 4-D.

4.3.4 Evaluation

In Figure 4.32 we demonstrate the LatentRNN recursive forecast, where at sample 18775 the LatentRNN is used generatively instead of input samples being directly reconstructed. Evidently it is unable to accurately forecast a single cycle accurately using the recursive strategy, however the first ∼50 samples resemble a peak. This behaviour was consistent across trials. The loss curve (Figure 4.31) might illustrate this inaccuracy, with convergence to a non-trivial amount of mean-squared error (∼0.2) for a single latent vector forecast.

Although underwhelming, this is promising, as there are two key ways that forecasting performance may be drastically improved. First, we might harness the multi-output approaches developed in Chapter 3 or by Fox et al. [58], improving forecast accuracy by avoiding the recursive approach. Recursive forecasting causes errors in prediction to rapidly compound. Second, we may construct the LatentRNN using a Mixture Density Network (MDN) which outputs the parameters of a mixture of Gaussian distributions for prediction. This was used for latent forecasting by Ha et al. [56], and for prediction of continuous signal values by Martin et al. [37] in interactive music systems. Both authors highlighted the importance of using MDNs given complex input data demanding some amount of stochasticity.

Dijan [66] found that when implementing the LatentRNN proposed by Ha et al. [56], a *large* MDN-LSTM was required with: 8 gaussians, 1024 hidden units, a 1024 unit FC layer, and sequences of 500 latent vectors. This suggests we might also need to increase the LatentRNN model size by a reasonable factor.

In addition to developing the model further, the training procedure adopted should be better informed by the desired performance application. For example, instead of framing the training task as simple prediction, it might involve decoding latent vectors and evaluating generated timbres with a perceptual criteria. This amounts to a rich opportunity for future work.

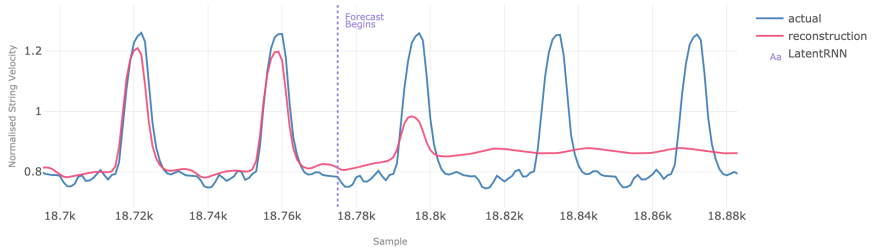


Figure 4.32: Example of a LatentRNN generative forecast, beginning at the purple line (sample 18775). Recursive forecasting accuracy rapidly deteriorates.

4.4 Summary

In this chapter VRAEs were developed to perform dimensionality reduction on bowed string signals, exposing a latent space that can support analysis and generation of signals. In a 3D latent space, sequences of input signals were encoded as polygon stars, where the geometry described properties such as pitch and timbre.

Achieving reasonable (re)synthesis quality was vital in evaluating VRAE suitability for performance applications. Generally, resynthesis was accurate for the dataset, and further for a largely out-of-sample scale. However, introduction of the VRAE-ST was necessary to remove an unpleasant effect resembling ring modulation. The ST supported the VRAE by learning a combination of reversible amplitude normalisation and downsampling, simplifying the learning task.

We noticed that anomalies (e.g. anomalous bow angle) were perceptually masked in resynthesis, with low amplitudes. In terms of expertise transfer, this suggests a potential deficiency in the feature representation learned - the model cannot interpolate between all plausible vibrational properties of the signal. This should be closely examined in future work.

Despite a reduced feature representation compared to RNN hidden state, harnessing the VRAE's latent space poses a reasonable challenge. In terms of analysis, it is hard to infer properties of the signal from latent trajectories. Through basic manipulation of the latent space, we altered the tone of isolated bow strokes, but this was not a clean signal transformation and is hard to explain. This doesn't proclude generative use of the latent space. Preliminary assessment of traversing latent space using RNNs (a LatentRNN) showed promise in learning to generate meaningful signals.

Chapter 5

Conclusions

This work has explored unsupervised learning techniques that are able to model high-resolution time-domain signals from digital musical instruments. Fixed feature engineering poses challenges in flexibly extracting salient features, so the modelling problem was re-framed as a task in expertise transfer. The key achievements of this work were as follows:

1. In Chapter 3 it was shown that RNNs could accurately learn a model an oscillating drumhead signal. Through a multi-feature representation of time-domain windows, basic LSTMs were able to harness harmonic properties for long-term multi-output forecasts (∼18ms). This amounted to multiple cycles, and far exceeds the single-sample prediction demonstrated by Martin et al. [37] as part of their interactive music system, which harnessed MDRNNs. Forecast accuracy was demonstrated to be useful in characterising events such as drum strikes. This provides a more continuous and flexible signal than typically used by systems which detect discrete onsets, such as Sensory Percussion by Sunhouse [3, 4]. However, LSTM hidden state alone was shown to be inaccessible as a tool to explain features observed in the signal.

2. In Chapter 4 it was shown that a VRAE could learn a model of high-resolution time-domain bowed string velocity signals. Existing work applying VAEs in musical tasks has focused on lower-resolution frequency [5] or symbolic [6, 7, 8] representations. The VRAE harnessed RNNs and performed substantial dimensionality reduction on input samples, constructing a 3D latent space that could be used for analysis and generation of signals. Properties of the signal (e.g. timbre) were defined by the geometry of a sequence of latent vectors. We were able to messily manipulate a signal's pitch for bowing in one direction through an arbitrary linear manipulation of latent space, but this was not an intuitive outcome. This characterises the enigmatic and nonlinear feature representation adopted by the VRAE. Two enhancements were also presented:

   (a) In Section 4.2 we introduced a novel model, the Variational Recurrent Autoencoder with Sequence-Transformer (VRAE-ST). The reversible ST learned to perform amplitude normalisation, audibly improving resynthesis, which originally had an unpleasant effect resembling ring modulation. Bidart et al. [65] recently also harnessed a transformer alongside a VAE, but in the domain of image processing.

   (b) In Section 4.3 we demonstrated that an RNN could be used to traverse the rather unintuitive VRAE latent space; where timbre was represented in the geometry of latent vector sequences. The basic LatentRNN developed was almost able to recursively forecast the next cycle of the vibrating string. The only existing musical application of a LatentRNN is by Pati et al. [8], for inpainting of symbolic musical scores.

5.1 Future Work

There are several ways in which this work could be developed. Notably the models developed weren't deployed in a generative or performance application, suggesting there are various outstanding challenges.

- *Improving latent interpretability:* Harnessing the unintuitive and nonlinear latent geometry seemingly requires some combination of regularisation and latent sequence learning. We proposed that a LatentRNN might be useful way to circumvent this problem, as it is unclear how regularisation may be performed. Simple adjustments to the model might also help disentangle features, as investigated by Dubois [67]; including scaling KL-loss with a new hyperparameter (e.g. β-VAE variants) or performing annealing. Refer back to 4.3.4 for future work relating to improving the LatentRNN.
- *Phase-invariant feature learning:* We believe there is more room to develop the VRAE-ST model introduced in this work, to better account for phase-invariant features in the input signal. Ultimately we believe this would drastically improve control of the feature representation. Potentially this requires re-thinking the input data, moving away from naively processing overlapped segments time-series data. When attempting a simple PSOLA-esque strategy involving windowing around pitch markers, models experienced posterior collapse, as the decoder learned to simply re-produce the Hann window applied to each segment. However, this approach might be re-considered.
- *Gesture-synthesis mappings:* Performance applications of these models will require a meaningful mapping between gestural control and synthesis. Fasciani et al. [55] made a valuable distinction between performance postures and gestures, and used this to develop SOMs linking input (frequency-domain) features to synthesis timbres. This is an appealing route to follow, given Fortuin et al. [53] proposed a SOM-VAE variant which also models time-series data.
- *Process improvement (incl. hyperparameter optimisation):* There is an abundance of low hanging fruit in terms of improving the models, which might be trivial for someone with greater deep learning experience. For example, hyperparameter optimisation, evaluating different neural networks (LSTMs versus GRUs), improved data preprocessing and more complex training schemes.
- *Larger training & test data sets:* It goes without saying that the datasets explored in this work were minimal, and this poses concerns for the transferability of the models developed. Future work should begin by carefully constructing a wider and more varied dataset, which might be informed by the aforementioned gesture-posture distinction.
- *Real-time architecture:* Performance applications would demand real-time use of these models, which they are currently not suited for in both architecture and performance. There might be several challenges in deploying these deep learning models on an embedded device, including memory requirements, CPU requirements and software compatibility. Instead, an approach involving OSC communication between an embedded device (part of the digital musical instrument) and a remote host might be necessary. Martin et al. [37] demonstrated this architecture for an interactive musical system using neural networks. The buffering and time-series windowing (or alternative) schemes will need careful consideration alongside the input dimensions expected by any model.

Bibliography

[1] K. Buys and A. Mcpherson, "Real-time bowed string feature extraction for performance applications," *Sound and Music Computing*, 2018.

[2] L. S. Pardue, K. Buys, M. Edinger, D. Overholt, and A. P. McPherson, "Separating sound from source: sonic transformation of the violin through electrodynamic pickups and acoustic actuation," *New Interfaces for Musical Expression*, p. 6.

[3] "Sunhouse | Home." Available at http://sunhou.se/, Accessed on 2019-07-17.

[4] "US20160093278a1 - Systems and methods for capturing and interpreting audio - Google Patents." Available at https://patents.google.com/patent/US20160093278A1/en, Accessed on 2019-07-17.

[5] P. Esling, A. Chemla, and A. Bitton, "Bridging Audio Analysis, Perception and Synthesis with Perceptually-Regularized Variational Timbre Spaces," in *ISMIR*, p. 7, 2018.

[6] O. Fabius and J. R. van Amersfoort, "Variational Recurrent Auto-Encoders," *arXiv:1412.6581 [cs, stat]*, Dec. 2014. arXiv: 1412.6581.

[7] G. Hadjeres, F. Nielsen, and F. Pachet, "GLSR-VAE: Geodesic latent space regularization for variational autoencoder architectures," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–7, Nov. 2017.

[8] A. Pati and A. Lerch, "Latent space regularization for explicit control of musical attributes," *ICML Workshop on Machine Learning for Music Discovery Workshop (ML4MD), Extended Abstract*, 2019.

[9] A. McPherson, "The Magnetic Resonator Piano: Electronic Augmentation of an Acoustic Grand Piano," *Journal of New Music Research*, vol. 39, pp. 189–202, Sept. 2010.

[10] F. Morreale, A. Guidi, and A. Mcpherson, "Magpick: an Augmented Guitar Pick for Nuanced Control," *New Interfaces for Musical Expression*, 2019.

[11] R. H. Jack, T. Stockman, and A. McPherson, "Effect of Latency on Performer Interaction and Subjective Quality Assessment of a Digital Musical Instrument," in *Proceedings of the Audio Mostly 2016*, AM '16, (New York, NY, USA), pp. 116–123, ACM, 2016. event-place: Norrköping, Sweden.

[12] A. P. McPherson, R. H. Jack, and G. Moro, "Action-Sound Latency: Are Our Tools Fast Enough?," *Proceedings of the International Conference on New Interfaces for Musical Expression*, July 2016.

[13] A. P. McPherson and V. Zappi, "An Environment for Submillisecond-Latency Audio and Sensor Processing on BeagleBone Black," p. 7, 2015.

[14] L. Yang, K. Z. Rajab, and E. Chew, "The filter diagonalisation method for music signal analysis: frame-wise vibrato detection and estimation," *Journal of Mathematics and Music*, vol. 11, pp. 42–60, Jan. 2017.

[15] V. Välimäki, J. Pakarinen, C. Erkut, and M. Karjalainen, "Discrete-time modelling of musical instruments," *Reports on Progress in Physics*, vol. 69, pp. 1–78, Oct. 2005.

[16] H. L. F. Helmholtz, *On the sensations of tone as a physiological basis for the theory of music, 2nd English ed.* On the sensations of tone as a physiological basis for the theory of music, 2nd English ed, Oxford, England: Dover Publications, 1954.

[17] A. Torin, B. Hamilton, and S. Bilbao, "An Energy Conserving Finite Difference Scheme for the Simulation of Collisions in Snare Drums," in *DAFx*, 2014.

[18] A. Torin, "Percussion instrument modelling In 3d: sound synthesis through time domain numerical simulation," June 2016.

[19] J. S. Hsu and T. Smyth, "Percussion Synthesis using Loopback Frequency Modulation Oscillators," in *SMC*, p. 8, 2019.

[20] H. F. Nweke, Y. W. Teh, M. A. Al-garadi, and U. R. Alo, "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges," *Expert Systems with Applications*, vol. 105, pp. 233–261, Sept. 2018.

[21] J. Françoise, "Gesture-Sound Mapping by Demonstration in Interactive Music Systems," in *Proceedings of the 21st ACM international conference on Multimedia (MM'13)*, (Barcelona, Spain, France), pp. 1051–1054, Oct. 2013.

[22] J. Françoise, N. Schnell, R. Borghesi, and F. Bevilacqua, "Probabilistic Models for Designing Motion and Sound Relationships," in *NIME*, 2014.

[23] J. Françoise, N. Schnell, and F. Bevilacqua, "A multimodal probabilistic model for gesture-based control of sound synthesis," in *ACM Multimedia*, 2013.

[24] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, Feb. 1989.

[25] S. Fine, Y. Singer, and N. Tishby, "The Hierarchical Hidden Markov Model: Analysis and Applications," *Machine Learning*, vol. 32, pp. 41–62, July 1998.

[26] S. Manitsaris, A. Glushkova, E. Katsouli, A. Manitsaris, and C. Volioti, "Modelling Gestural Know-how in Pottery Based on State-space Estimation and System Dynamic Simulation," *Procedia Manufacturing*, vol. 3, pp. 3804–3811, Jan. 2015.

[27] C. Volioti, S. Manitsaris, E. Katsouli, and A. Manitsaris, "x2gesture: how machines could learn expressive gesture variations of expert musicians," *Proceedings of the International Conference on New Interfaces for Musical Expression*, p. 6.

[28] "XMM — Probabilistic Models for Motion Recognition and Mapping: Ircam-RnD/xmm," May 2019. Available at https://github.com/Ircam-RnD/xmm, Accessed on 2019-07-17.

[29] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 3104–3112, Curran Associates, Inc., 2014.

[30] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, Apr. 1990.

[31] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, Mar. 1994.

[32] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[33] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," *arXiv:1409.1259 [cs, stat]*, Sept. 2014. arXiv: 1409.1259.

[34] "Illustrated Guide to LSTM's and GRU's: A step by step explanation." Available at https://www.kurious.pub, Accessed on 2019-07-26.

[35] D. Eck and J. Schmidhuber, "Finding temporal structure in music: blues improvisation with LSTM recurrent networks," in *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pp. 747–756, Sept. 2002.

[36] L. Hantrakul and Z. Kondak, "GestureRNN : A neural gesture system for the Roli Lightpad Block," in *NIME*, 2018.

[37] C. P. Martin and J. Torresen, "An Interactive Musical Prediction System with Mixture Density Recurrent Neural Networks," *arXiv:1904.05009 [cs, eess]*, Apr. 2019. arXiv: 1904.05009.

[38] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," *arXiv:1609.03499 [cs]*, Sept. 2016. arXiv: 1609.03499.

[39] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset," *arXiv:1810.12247 [cs, eess, stat]*, Oct. 2018. arXiv: 1810.12247.

[40] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush, "LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks," *arXiv:1606.07461 [cs]*, June 2016. arXiv: 1606.07461.

[41] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and Understanding Recurrent Networks," *arXiv:1506.02078 [cs]*, June 2015. arXiv: 1506.02078.

[42] C. M. Yeh, N. Kavantzas, and E. Keogh, "Matrix Profile VI: Meaningful Multidimensional Motif Discovery," in *2017 IEEE International Conference on Data Mining (ICDM)*, pp. 565–574, Nov. 2017.

[43] P. Malhotra, L. Vig, G. Shro, and P. Agarwal, "Long Short Term Memory Networks for Anomaly Detection in Time Series," *Computational Intelligence*, p. 6, 2015.

[44] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection," *arXiv:1607.00148 [cs, stat]*, July 2016. arXiv: 1607.00148.

[45] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding," *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '18*, pp. 387–395, 2018. arXiv: 1802.04431.

[46] J. Cowton, I. Kyriazakis, T. Plötz, and J. Bacardit, "A Combined Deep Learning GRU-Autoencoder for the Early Detection of Respiratory Disease in Pigs Using Multiple Environmental Sensors," *Sensors (Basel, Switzerland)*, vol. 18, Aug. 2018.

[47] G. Arvanitidis, L. K. Hansen, and S. Hauberg, "Latent Space Oddity: on the Curvature of Deep Generative Models," *arXiv:1710.11379 [stat]*, Oct. 2017. arXiv: 1710.11379.

[48] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv:1312.6114 [cs, stat]*, Dec. 2013. arXiv: 1312.6114.

[49] C. Doersch, "Tutorial on Variational Autoencoders," *arXiv:1606.05908 [cs, stat]*, June 2016. arXiv: 1606.05908.

[50] T. Spinner, J. Körner, J. Görtler, and O. Deussen, "Towards an Interpretable Latent Space – An Intuitive Comparison of Autoencoders with Variational Autoencoders," p. 1, Oct. 2018.

[51] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," *arXiv:1704.01279 [cs]*, Apr. 2017. arXiv: 1704.01279.

[52] C. X. Hernández, H. K. Wayment-Steele, M. M. Sultan, B. E. Husic, and V. S. Pande, "Variational encoding of complex dynamics," *Physical Review E*, vol. 97, p. 062412, June 2018.

[53] V. Fortuin, M. Hüser, F. Locatello, H. Strathmann, and G. Rätsch, "SOM-VAE: Interpretable Discrete Representation Learning on Time Series," *arXiv:1806.02199 [cs, stat]*, June 2018. arXiv: 1806.02199.

[54] A. van den Oord, O. Vinyals, and k. kavukcuoglu, "Neural Discrete Representation Learning," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 6306–6315, Curran Associates, Inc., 2017.

[55] S. Fasciani and L. Wyse, "Vocal Control of Sound Synthesis Personalized by Unsupervised Machine Listening and Learning," *Computer Music Journal*, vol. 42, pp. 37–59, Apr. 2018.

[56] D. Ha and J. Schmidhuber, "World Models," *arXiv:1803.10122 [cs, stat]*, Mar. 2018. arXiv: 1803.10122.

[57] A. Pati, A. Lerch, and G. Hadjeres, "Learning to Traverse Latent Spaces for Musical Score Inpainting," *arXiv:1907.01164 [cs, eess, stat]*, July 2019. arXiv: 1907.01164.

[58] I. Fox, L. Ang, M. Jaiswal, R. Pop-Busui, and J. Wiens, "Deep Multi-Output Forecasting: Learning to Accurately Predict Blood Glucose Trajectories," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, (New York, NY, USA), pp. 1387–1395, ACM, 2018. event-place: London, United Kingdom.

[59] "PyTorch." Available at https://www.pytorch.org, Accessed on 2019-07-17.

[60] J. Woodhouse and P. Galluzzo, "The Bowed String As We Know It Today," *Acta Acustica united with Acustica*, vol. 90, pp. 579–589, July 2004.

[61] A. A. Alemi, B. Poole, I. Fischer, J. V. Dillon, R. A. Saurous, and K. Murphy, "Fixing a Broken ELBO," *arXiv:1711.00464 [cs, stat]*, Nov. 2017. arXiv: 1711.00464.

[62] A. Razavi, A. v. d. Oord, B. Poole, and O. Vinyals, "Preventing Posterior Collapse with delta-VAEs," *arXiv:1901.03416 [cs, stat]*, Jan. 2019. arXiv: 1901.03416.

[63] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, "Spatial Transformer Networks," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2017–2025, Curran Associates, Inc., 2015.

[64] J. Oh, J. Wang, and J. Wiens, "Learning to Exploit Invariances in Clinical Time-Series Data using Sequence Transformer Networks," *arXiv:1808.06725 [cs, stat]*, Aug. 2018. arXiv: 1808.06725.

[65] R. Bidart and A. Wong, "Affine Variational Autoencoders: An Efficient Approach for Improving Generalization and Robustness to Distribution Shift," *arXiv:1905.05300 [cs]*, May 2019. arXiv: 1905.05300.

[66] "World Models applied to Sonic | Dylan's Blog." Available at https://dylandjian.github.io/world-models/, Accessed on 2019-08-24.

[67] Y. Dubois, "Experiments for understanding disentanglement in VAE latent representations: YannDubs/disentangling-vae," Aug. 2019. original-date: 2019-01-30T22:21:55Z.

[68] "numpy.lib.format — NumPy v1.18.dev0 Manual." Available at https://numpy.org/devdocs/reference/generated/numpy.lib.format.html, Accessed on 2019-08-07.

Appendix A

Custom Electronic Drumhead

Typical electronic drum kits have proprietary and inaccessible handling for sensor data, providing simply a MIDI-based interface for note onsets. In order to model the actual vibrations of a muted drumhead using continuous sensors, we created a simple test rig using electromagnetic induction.

A.1 Design

The rig consisted of a 12 inch basswood snare drum, where the batter head was 3-ply mesh (typical of electronic/hybrid drumkits), and the resonant head and rim were removed. We used an existing adjustable crossbar to create a frame across the drum diameter, and mounted laser-cut platforms at desired sensor locations (see Figure 1.1). For preliminary testing, these locations were in the centre of the drumhead and also to one side. Thin copper wires were run along the drumhead surface above these platforms, and 12mm $\times$ 2mm (diameter $\times$ depth) N42 Neodymium magnets were sellotaped to the platforms. Only a single loop of copper wire was used on the drumhead to minimise weight and hence interference with the natural modes of vibration. Through Faraday's Law, changes in magnetic flux along the surface wires (from interaction with the magnets) creates voltage depending on rate of motion. Therefore when the drumhead vibrates, for example due to a strike, the wires also oscillate relative to the stationary magnets. This requires the platforms to be raised close to the wires, without obstructing the drumhead vibrations following forceful drumstick impacts.

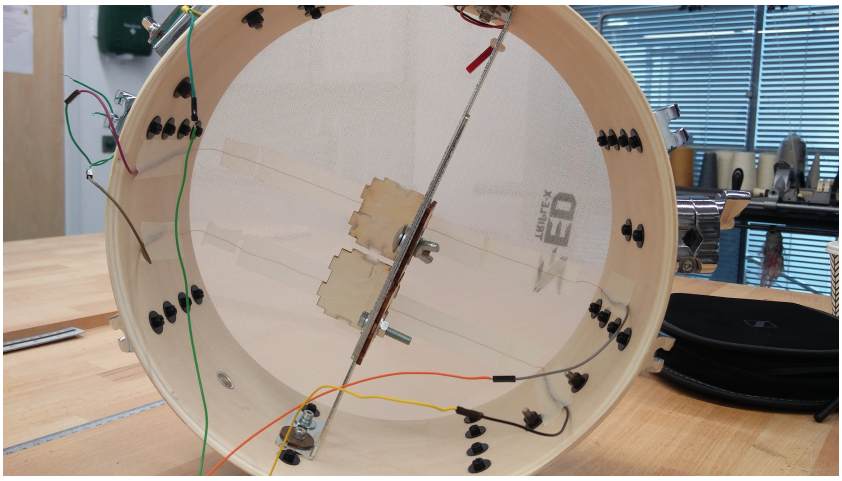


Figure 1.1: Drum rig viewed from the base, where the electromagnetic pickups can be seen along the crossbar.

A.2 Sensor Technology

The combination of only a single loop of wire and subtle vibrations of the *muted* drumhead result in a very small induced signal (on the order of μV). This requires amplification to ensure a reasonable signal-to-noise ratio. The pre-amplifier used was custom-built for previous work involving string pickups on electric violins (see Figure 1.2 for the design). It features OPA1612 op-amps, which have very low voltage noise density - and are described in further detail in [2]. The amplified signals are then directly connected to analog inputs of the Bela board.

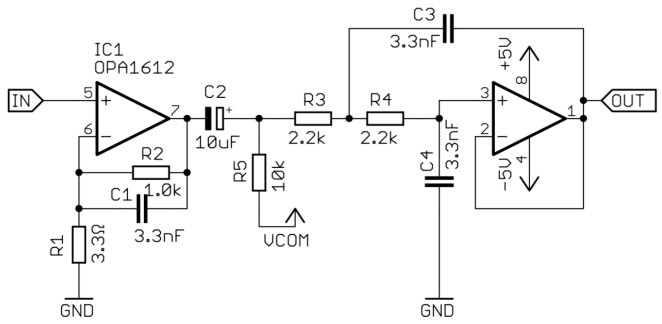
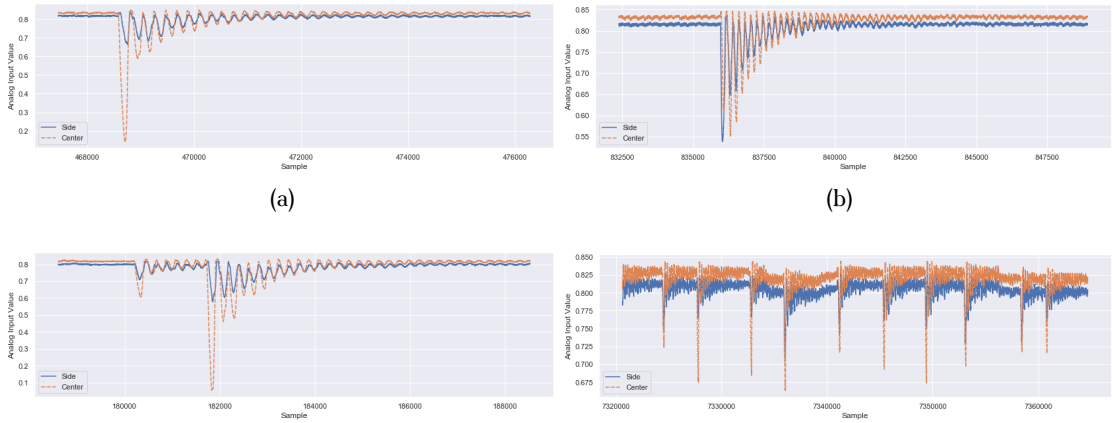


Figure 1.2: Pre-amplifier schematic for each input electrodynamic pickup on the drumhead (wire along surface), as per [2].

A.3 Example Output

Sensor data was exported over OSC from the Bela board using the tools detailed in Appendix C (see `oscsend.cpp` and `oscrec.py`). Some isolated examples of different drum strokes are demonstrated in Figure 1.3 - each are distinct with a high level of detail.



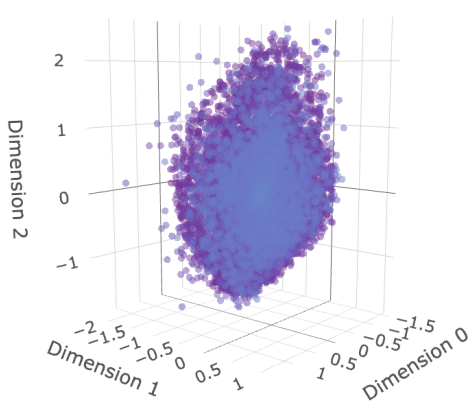
(a) (b)

(c) (d)

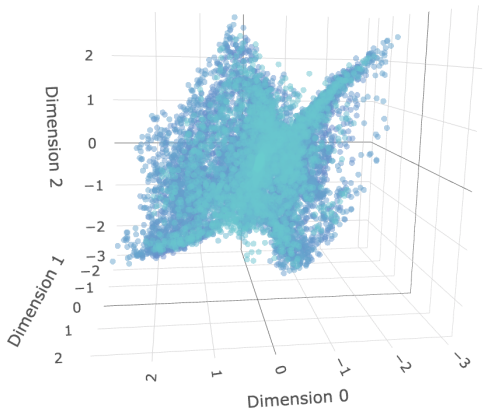
Figure 1.3: Example gestures recorded from the custom drumhead. Top row: full stroke at the center (left) and full stroke at the side (right). Bottom row: a flam (left) and rolls (right).

Appendix B

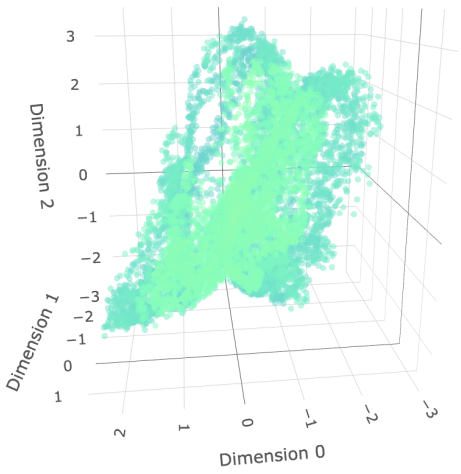
VRAE-ST Latent Embeddings



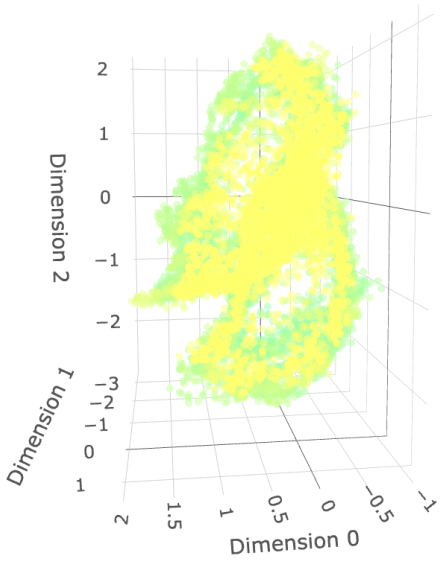
(a) G String



(b) D String

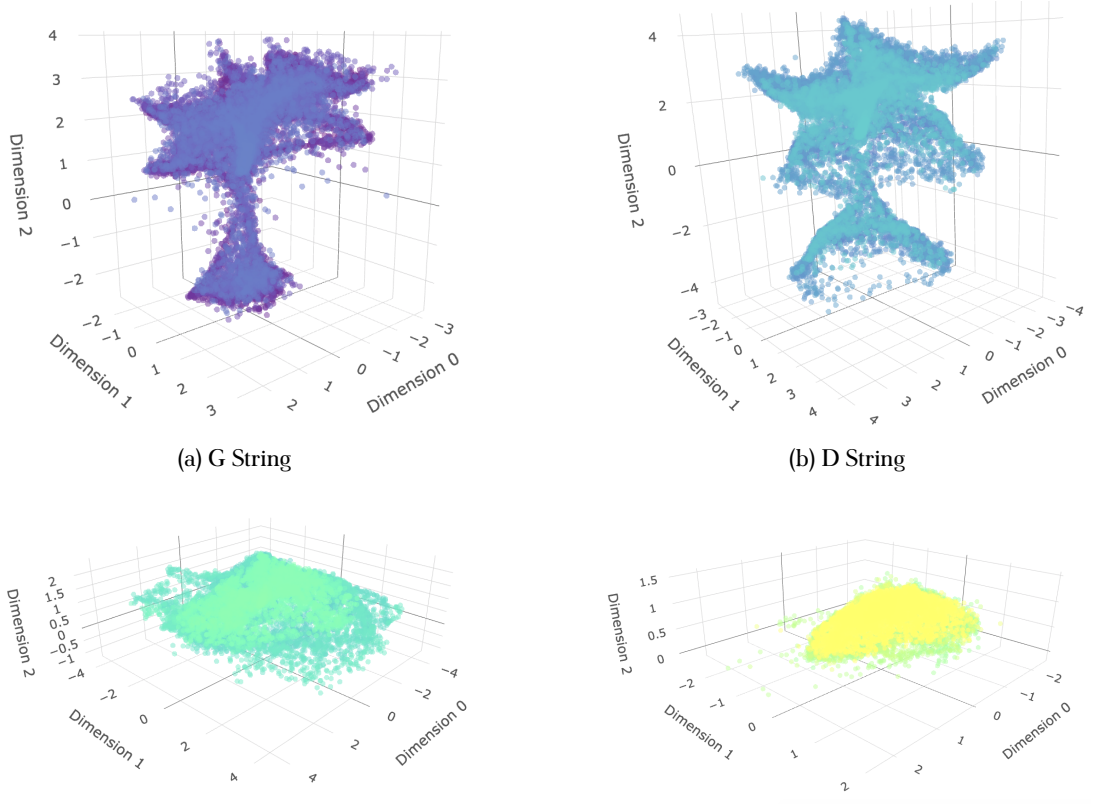


(c) A String



(d) E String

Figure 2.1: VRAE-STe latent embeddings from Figure 4.29a, with the same colouring, separated according to the string of input samples.



(a) G String

(b) D String

(c) A String

(d) E String

Figure 2.2: VRAE-ST latent embeddings from Figure 4.29b, with the same colouring, separated according to the string of input samples.

Appendix C

Model Training Framework

The software framework produced is an analytics-oriented bundle of tools and code notebooks, supporting the training and evaluation of PyTorch [59] models for time series. The various components of this are detailed in the following sections: (1) data collection (2) model definition (3) configuration (4) evaluation. The project used Python v3.6.5, and package versions can be found in `requirements.txt` from Listing 1. The full repository is made available at https://github.com/Kappers/APP.

C.1 Module Organisation

In Listing 1 the various scripts and utilities used throughout this work are described. Note that `torch_rnns/` reflects early investigations of RNNs with PyTorch, for Chapter 3, and is not discussed here. The top-level training framework was developed for Chapter 4, as a generalised and flexible environment that can be used in later work - this is detailed in the following section.

C.2 Data Collection

There are utility scripts for collecting data from sensors on a Bela board, and equally for simple conversion of existing WAV files - in both cases, a serialised binary format for the `NumPy` environment [68] is produced (`.npz`) which can be loaded via helpers in `utils/misc.py`.

- **WAV conversion:** This can be simply performed using the `utils/npz_to_wav.py` script with the `rev` argument.

- **Collection from Bela:** This requires coordinating `oscsend.cpp` on a Bela board, and `oscrec.py` on a host. Firstly, the OSC endpoints should be defined in global scope for both files, alongside the number of sensor data channels and packet size. Following that:

    1. Define a data-buffer size (total volume to be collected) in `oscrec.py` as a multiple of the OSC packet size. Consider the sampling frequency to inform this.
    2. Begin running `oscrec.py`, it will wait for OSC packets.
    3. Begin running `oscsend.cpp`, and immediately it will start buffering and transmitting OSC packets. This is where you might meaningfully interact with sensors.
    4. When `oscrec.py` is finished, it will terminate. Before termination, a progress bar will be logged as the buffer is filled. You can use a key-interrupt to finish data collection early.

```
requirements.txt # Python package versions, exported using '> pip freeze'
oscsend.cpp # OSC sender from Bela to a receiver, for use as render.cpp
models.py # PyTorch model classes, inheriting from SequenceModule
train.py # Training tool, requiring model config in appmodels/
appmodels/
- seqvae/ # Example models of SequenceVAE class.
    - config_violin.v3_100w_big.yaml # Config for a trained model
    - SequenceVAE_20190723103422.pt # PyTorch weights for above model
    - train_20190723103422.yaml # Training meta-information for above model.
- latentrnn/
    ...
- seqtransvae/
    ...
utils/
- gen_testdata.py # Dumps noisy sin function dummy data
- misc.py # Various helpers, e.g. file management and windowing
- npz_to_wav.py # Converts wav to and from numpy time series files
- oscrec.py # Python-based OSC receiver for data from oscsend.cpp
- plotpickle.py # Plots a pickled matplotlib graph file/object
- scope.py # Oscilloscope for playing back time series
notebooks/
- HHMM_Constructor.ipynb # HHMM experimentation tool
- LSTM_Hidden_Sounds.ipynb # LSTM hidden state exploration and sonification
- VAE_Explorer.ipynb # VAE analysis
- VAE_LatentOdyssey.ipynb # LatentRNN analysis
- VAE_ReconstructionQuality.ipynb # VAE reconstructed audio analysis
torch_rnns/ # DEPRECATED training, for LSTM anomaly detection
- lstm_train.py # Training script, model parameters defined globally
- logs/
    - train_20190531093809_lstm_2l128h_100h50p_100ep.log
    ...
- lstm_2l100h_100h50p_100ep/ # Output training results
    - model # PyTorch weights for given model
    - test.pdf # Graphing of loss and performance
  ...
```

Listing 1: Module structure and overview.

C.3 Model Training

C.3.1 Model Definition

Every model trained is defined in `models.py`, according to Listing 2. Several example models are provided in `models.py` (that are not all evaluated in this thesis), but the official PyTorch [59] documentation presents valuable tutorials on how to produce custom neural network modules.

Models are interacted with in a generic way from `train.py`: they are loaded from config using the `from_conf()` factory; the processing of a batch of data is assumed to be imple-

mented via the PyTorch `forward()` method; and loss per training epoch is calculated using `compute_loss()`. Notably, config files are expected by `train.py` which specify the model class, parameters and existing learned weights if they exist. This is detailed in the next section.

```
...
class SequenceModule(nn.Module):
    ...
    def init_submodels(self, submodels, submodel_confs):
        # If the model wraps another model (e.g. LatentRNN and VAE).
        raise NotImplementedError()

    @classmethod
    def from_conf(cls, conf):
        raise NotImplementedError("Implement class factory for init'able from config dict")

    def compute_loss(self, x_batch, y_batch):
        raise NotImplementedError("Evaluate batch and return loss")
...
class SequenceVAE(SequenceModule):

    def __init__(self, ...):
        super(SequenceVAE, self).__init__()
        # Model-specific initialisation here.
        ...
    ...
    def forward(self, x):
        # Typical PyTorch method to map some input data, x.
        ...
        return out
...
```

Listing 2: `SequenceModule` inheritance example in `models.py`, for definition of a new model (e.g. `SequenceVAE`). The `NotImplementedError` lines can be interpreted as abstract methods requiring subclass implementation.

C.3.2 Configuration

Training begins with a config file, that is provided to `train.py` as an argument. This is a `.yaml` file declaring the various models to be built, as a collection of parameters. These are used by `train.py` to construct a model instance, prepare training data and limit the training process. During training, learned network weights and training meta-information are generated which can be referred to by path in this config file. An example config file is annotated in Listing 3.

C.3.3 Training and Logging

If training —successive epochs with batch sizing determined by config— is performed without the `stdout` command, three files will be updated after each epoch: model weights, training

```yaml
#
# Free-text about this given config file.
#
name: seqvae # Acronym for the respective network.
version: 0.1 # Free-text version label that can be tracked in version control.
submodels: # List of models which form the given network.
    - name: 'SequenceVAE' # The class of a given submodel.
      order: 1 # Order of the submodel in network pipeline.
      pretrained: # Weights and metadata, if they exist (submodel trained).
          model: 'appmodels/seqvae/SequenceVAE_20190723103422.pt'
          meta: 'appmodels/seqvae/train_20190723103422.yaml'
      freeze : true # Whether submodel shouldn't undergo any training.
      params: # Model parameters, used by the respective class from_conf().
          layers: 2
          dropout: 0.3
          hidden: 128
          latent: 3
          input: 1
          stateful: true
      train:
          data_path: # List of data files (.npz) to be used in training.
              - '~/APP/data/sample_allnote_rate_150s.npz'
          preprocess: # Pre-processing for data
              fs: 44100.0 # This is static information for data
              resample_factor: 4 # Decimation
              highpass: 100.0 # Lowpass is also available
              scale: true # Transform data to range 0-1
              # Additional options are available, see train.py
              ...
          epochs: 500 # Number of maximum training epochs
          batch: 256 # Batch size
          learning_rate: 0.0005
          window:
              step: 25 # Sliding window steps
              x: 100 # Input vector in samples
              y: 100 # Output vector in samples
              mirror: true # Forces input vector to equal output vector
          early_stopping: # Criteria to early-terminate training
              min_loss: 0.000001
              patience: 25
              after_epoch: 200
          split: # Data split. Anything not in train is used for validation.
              train: 0.8
```

Listing 3: Example configuration file for model `SequenceVAE`. Note that it has been trained, so existing weights and training metadata are found as paths under the `pretrained` key.

metadata and logs. These are shown in Listing 4 & 5. This allows training on remote hosts, with full visibility of training progress, that can be recorded in version control. Note that the respective training optimisation procedure is declared within a given model.

```
appmodels/seqvae »  cat train_20190723103422.yaml
conf:
  name: SequenceVAE
  ...
  params: {dropout: 0.3, hidden: 128, input: 1, latent: 3, layers: 2, stateful: true}
  pretrained: {meta: '', model: ''}
  train:
    batch: 256
    data_path: [/homes/tmk31/app/data/sample_allnote_rate_150s.npz]
    early_stopping: {after_epoch: 200, min_loss: 1.0e-06, patience: 25}
    ...
- batch_size: [256, 100, 1]
  epoch: 302
  total_loss: 0.009938601404428482
  total_time: 224.83093523979187
...
```

Listing 4: Example training meta-information (`.yaml`) output from `train.py` for a given config file. This details model and training parameters as well as per-batch training information.

```
appmodels/seqvae » cat train_20190723103422.log
Loading model config...
> Training SequenceVAE
Loading data...
Preparing model...
Training SequenceVAE submodel...
Path used for weights: appmodels/seqvae/SequenceVAE_20190723103422.pt
Path used for train. meta: appmodels/seqvae/train_20190723103422.yaml
Will early stop with: 25 patience, 1e-06 min loss
Starting epoch 0/500...
...
Starting epoch 302/500...
> 292 batches of [256, 100, 1]
> Total Loss: 0.009938601404428482, Val Loss: nan
> Finished in 224.8309s
> Writing updated weights...
> Writing updated training meta...
Starting epoch 303/500...
Early stop: 26 waited, 0.009368088096380234 best loss (1e-06 min loss)
Add model weights path to config for future use:
    appmodels/seqvae/SequenceVAE_20190723103422.pt
```

Listing 5: Example training logs (`.log`) output from `train.py` for a given config file. This will also aggregate any warnings or errors raised during training.

C.4 Evaluation Tools

After training, or when `train.py` is provided a config file referencing pre-trained weights, loss curves are plotted and the reconstructions of the configured data are displayed. Most of the visualisations produced in this report have been produced using notebooks (see `notebooks/`), where code snippets for loading models can be found. For an example, see Figure 3.1 - this notebook is equipped with widgets that can be used to selective and load a config file.

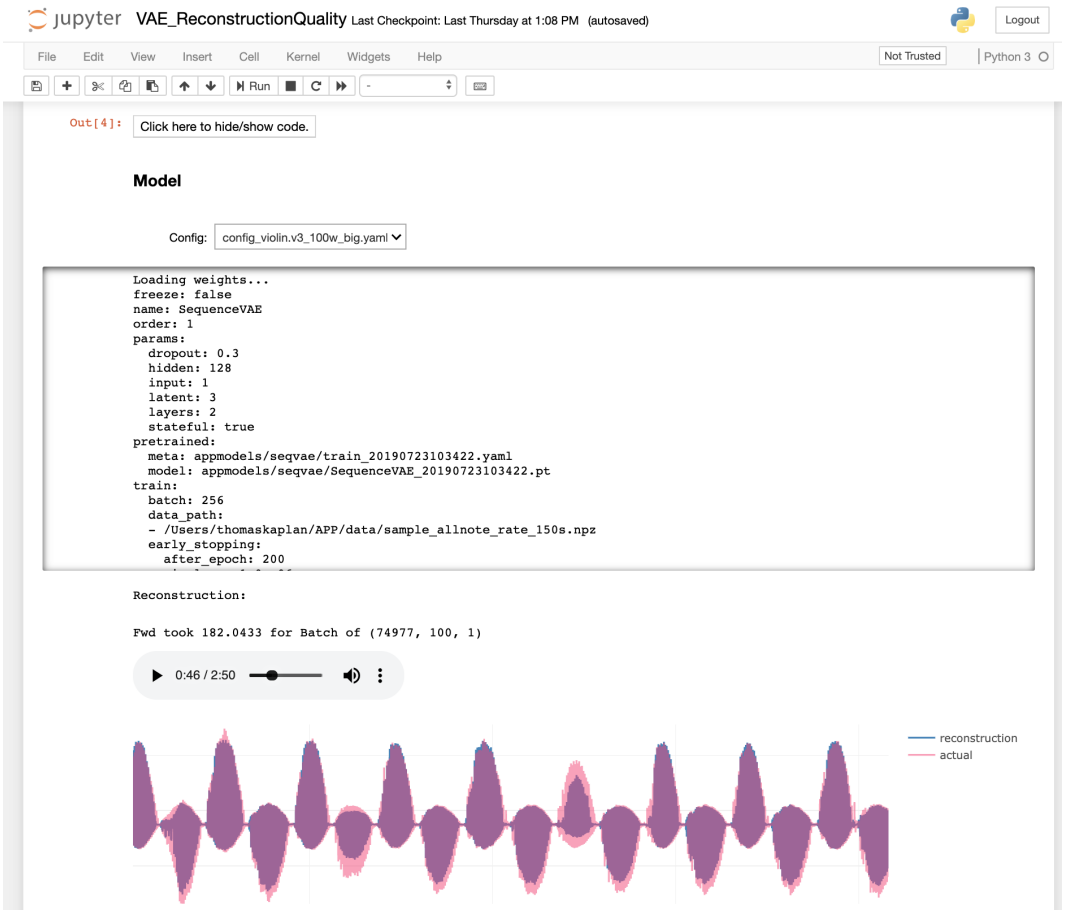


Figure 3.1: Demonstration of the `VAE_ReconstructionQuality.ipynb` notebook, which loads a pre-trained model before: (1) plotting reconstructions and (2) creating an audio widget for evaluation.