Representation Learning for Drum Loops with a Variational Autoencoder

James McDermott[1]

University College Dublin
`james.mcdermott2@ucd.ie`

Abstract. A representation is learned for MIDI drum loops, using a variational autoencoder. The aim is to create a representation which will be useful as a component in human-computer interfaces and in music generation systems. A large library of MIDI drum loops is described and used to train an autoencoder neural network in an unsupervised fashion. The result is a low-dimension representation which captures essential dimensions of variation in the data, and can be used to generate new drum loops and interpolate between pairs of loops.

Keywords: Representation learning; neural networks; variational autoencoder; music; drums; drum loops; MIDI

1 Introduction

Many digital musical instruments are controlled by a large number of numerical parameters. Choosing a timbre for a synthesizer, for example, amounts to a search problem in a space consisting of one dimension per parameter. Geometric operations such as *tweaking* a single timbre, *blending* between a pair of timbres, and following a *trajectory* are possible in this space. These operations are valuable as a means of control both for novice and expert users ("I'll try a variation of the current sound"), and as a means of communication between users ("can you push that sound a bit farther?"). They also enable alternative methods of control such as random generation of novel timbres, and interactive search methods such as interactive evolutionary computation. Moreover, similar geometric ideas are seen as a core achievement of recent work on learned representations, which become components in machine learning algorithms: a well-known example is the "king - man + woman = queen" vector algebra of Mikolov et al. [14].

In contrast to control of synthesized timbre, editing music using a *sequencer* equates to manual control of a large number of note-on/note-off, velocity, and pitch parameters, arranged into a time-grid. Although geometric operations can be defined in such a space, in practice the results fail to give a meaningful sense of tweaking, blending, or trajectory. What is needed is a new *representation*, that is an alternative space together with a mapping from that space to the concrete musical data, such that these geometric operations become meaningful for users and usable in algorithms.

In this paper, an autoencoder neural network architecture is proposed for the problem of learning representations suitable for the control of drum sequencers. The representation is learned from a corpus of drum loops in MIDI format. The neural network is a *variational autoencoder* (VAE) [12]. The VAE has several properties which may make it suitable for the task, as follows. (1) An autoencoder (AE) [9] learns from a corpus of data with no requirement for labels, because it is trained to reproduce its input at the output. (2) The middle hidden layer of the AE is usually much smaller (fewer neurons) than the input, forcing the AE to find regularities in the data and take advantage of them. Such regularities amount to an implicit sense of "what tracks in the corpus are like". (3) By discarding the input half of the AE, and working in the space of the real-valued variables of the hidden layer, we obtain a new representation. Running the AE forward from the hidden layer gives a new drum sequence. (4) The VAE allows tight control of the distribution of data in the hidden layer, when mapped from the input, in contrast to other AEs [12].

Moreover, it is hypothesized that geometric operations on the hidden layer variables tend to map to geometric operations as heard by listeners, such as well-formed variations of an object (formed by a small change from a point in the hidden layer space) or a conceptual blend or interpolation (formed by an interpolation between a pair of points in the hidden layer space). A well-behaved distribution at the hidden layer allows the user to explore and use geometric operations freely and helps to prevent poor results in areas of the new representation which are of low density with respect to the training data.

AEs and variants have been used for some similar tasks in the domain of image processing and computer vision, and to some extent for audio processing, but have been much less used in the domain of musical control and at the MIDI level. In this paper we investigate VAE performance (success in the input-output reproduction task) on a corpus of MIDI drum tracks, investigate the influence of VAE hyperparameters, and demonstrate the distribution achieved in the hidden layer. Finally, we audition the results of random generation, tweaking, blending, and trajectories in the learned representation.

2 Problem Definition

We are concerned with short drum loops represented at the symbolic level, e.g. at the MIDI level, rather than raw audio. As we will see, this task is sufficiently large to be interesting and challenging, and as discussed above, also has useful applications in novel user interfaces. It is natural to consider the symbolic level, since the computational cost will be far lower; the features present at the symbolic level are somewhat semantic, in comparison to those at the raw audio level (but still, far less semantic than those we would like to learn in a new representation); and good symbolic corpora are now available.

In our problem, we are given a space in which drum loops can exist. Under some simplifying assumptions (see Section 5), any drum loop corresponds to a point in this space, and any point in the space corresponds to a drum loop.

However, the vast majority of points in the space will correspond to "ill-formed" drum loops, effectively sounding like random drum hits, rather than sounding musical. Of course, some proportion will also exist in a grey area which may be called semi-musical, depending on context and taste. The space is also high-dimensional. Moreover, as already discussed, several operations in the space of drum loops which might be useful to musicians or algorithms turn out not to work well, such as random generation, tweaking, blending, or following trajectories.

A *representation*, in our context, will mean a space, together with a mapping from it to the original space. (As a special case, the original space is itself a representation by this definition.) The new space corresponds to a low-dimensional manifold embedded in the original space. In the context of neural representation learning, the new space is often referred to as the *latent space*, and points in the new space as *latent codes*. Introducing a second space is motivated by the possibility that it will have better properties than the original. Ideally, we would like to have a lower-dimensional representation, in which geometric operations correspond to some extent to human understanding of these operations: for example, the "blend" of two drum loops should correspond to a stylistic conceptual blend [10], rather than a mere cross-fade. Also, for practical reasons we would like to have a representation in which the mapping to the original drum loop space can run in real-time.

Such a representation would be useful as a component in human-computer interfaces: for example, we could allow a user to explore the space of drum loops by moving sliders on a synthesizer-style interface. It would also be useful in algorithms: for example, metaheuristic search could proceed more successfully when geometric operations, which are based on geometric concepts [15] give appropriate results.

3 Related Work

There are several possible approaches to creating a good representation:

- Standard dimensionality-reduction algorithms such as principal components analysis (PCA), multi-dimensional scaling (MDS), and t-distributed stochastic neighbour embedding (t-SNE);
- Manual design;
- Representations learned from data using neural networks and similar approaches.

Algorithms such as MDS and t-SNE do allow a dataset to be embedded into a new, lower-dimensional space, preserving distances between pairs of points. Mathematical results are available concerning the worst-case distortion suffered by such embeddings, e.g. [2]. However, these methods do not allow new points to be mapped to the new space, and do not allow a mapping from the new space to the original. Thus, they do not serve as representations for our purposes.

Principal components analysis (PCA) is a well-known linear method for representation learning. It works by creating a new space by rotating the original (an invertible linear transformation achieved by a matrix multiplication), such that the first of the new axes is maximally aligned with variation in the data, the second with remaining variation, and so on, and then discarding those axes which contribute the least. Thus, it achieves dimensionality reduction. A point sampled in the low-dimensional space can be mapped (by a single matrix multiplication, hence quickly) to a point in the original space. Therefore, PCA qualifies as a method of representation learning.

Manual design of good representations is a mainstay of research in metaheuristics and other fields [16]. In the context of evolutionary computation and search, the original space is referred to as the "phenotype" space, and the new representation space is referred to as the "genotype" space. Metaheuristic algorithms search in the genotype space, running their mappings forward to create points in the original space for evaluation by an objective function. Creating a suitable representation for drum loops by manual methods would be possible, and an interesting project, but would necessarily rely on human ingenuity and domain knowledge, and would be vulnerable to the creator's biases.

Therefore, we focus instead on learning a representation from data. We next consider related work in two categories: neural networks applied to representation learning, and various techniques applied to learning and representations for drum patterns.

3.1 Representation learning with neural networks

In this paper we will consider *representation learning*: that is, defining a space and using data to learn a mapping from it to the original space. The desired geometric properties may either be encouraged by the learning process, or may emerge naturally. Since the goal is just to learn a mapping, there is no requirement for labelled data, hence the process is unsupervised.

In recent years, the field of representation learning has made great progress through novel neural network architectures. In particular, *variational autoencoders* (VAEs) [12] and *generative adversarial networks* (GANs) [7] and their many variants represent the state of the art. Each uses an interesting neural architecture and method of training on unlabelled data. Each has been used extensively in learning good representations for image corpora. As an example, on the well-known MNIST dataset, consisting of images of handwritten digits, both VAEs and GANs have been shown to be capable of learning a representation in which random points in the new space correspond to well-formed digits; and interpolations from one point to another in the new space correspond to good interpolations between digits in the image space. Further developments of the two basic models have improved on training stability and on the behaviour of the learned representation. In the current paper, we consider the VAE, since it is the model on which much of the recent work has been based. Future work will investigate GANs and alternative VAE models, as discussed in Section 7.1.

3.2 Representations for drums and music

As discussed, good progress has been made using neural network approaches to representation learning in recent years. Many of the applications are in the domain of image processing and computer vision, and to a lesser extent in audio processing [21] and music at the audio (sample) level [6]. Of more interest in the current contxt is recent research in representation learning at the symbolic (MIDI) level.

The Google Magenta project has released several models for representation learning and generation of both drums and melodic material, using VAEs and recurrent neural networks (RNNs)[1]. The multitrack MusicVAE model [18] allows for multiple musical instruments playing simultaneously, together with drums. It operates at the symbolic level and is trained using MIDI data. It combines a VAE and an RNN. The authors emphasise the ability to carry out interpolations and meaningful transformations in the latent space, such as increasing the overall pitch range. They can also "condition" on chords, that is rather than asking the network to generate material from scratch, a chord sequence can be input and the network is then required to generate suitable material.

Several other authors have also used recurrent networks, including Trieu and Keller [20], who used a generative adversarial network (GAN) to improvise jazz, and Sturm [19] who used an RNN for folk music and added deep analysis of its inner workings. Meanwhile, Manzelli et al. [13] combined both audio and symbolic representations.

Kaliakatsos-Papakostas [10] has described an interesting project in which high-level features of drum loops are manually defined and used as a type of representation. The features are "one-way": they can be extracted from drum loops, but there is no immediate way to generate a drum loop given a feature vector. However, a metaheuristic search method is proposed to search for loops whose features closely match those of a desired feature vector. This allows the features to function as a representation, and allows "conceptual blending" (the focus of the paper) and other tasks such as tweaking and interpolation. However, because of the metaheuristic search, the mapping from features to drum loops cannot run in real-time, an important drawback both for interactive use and for algorithms. Kaliakatsos-Papakostas defined a space of 32 features, which is comparable in size to that developed in the current paper.

Eigenfeldt and Pasquier [5] describe a system for the creation of electronic dance music, including drum beats. Concerning drum tracks, they defined a set of "one-way" features. Again, they used a search method to find loops corresponding to desired feature vectors. It is of interest that they define exactly 3 features for drum beats – named density, syncopation, and symmetry – each applied at 3 levels – the open/closed hi-hat, the kick and snare, and both together. Since they use just 4 drum types and eight-bar phrases, the "size" of their representation (9 features) is not inconsistent with that developed in the current paper.

[1] E.g. `https://github.com/tensorflow/magenta/tree/master/magenta/models/drums_rnn`

Based on our literature review, it is clear that recurrent neural architectures are common. They are a natural choice because, like language, music is essentially sequential, and recurrent architectures have a "memory". However, the short drum loops we consider may have less need for memory. Dependencies between the information at different time-steps in the loop can be sufficiently captured by a relatively small feed-forward network. Also, drum loops have a particular "timeless" quality: rather than a sequential journey from beginning to end, a drum loop is "constant". In this work, we have chosen to focus on feed-forward networks only.

We also do not consider convolutional neural networks (CNNs) in this paper. CNNs have been central to many successes of deep learning with image data in recent years. They are particularly suitable for image data because the assumption they effectively encode is that there is a strong relationship between the information in adjacent pixels, arising from their assumed physical proximity in the scene depicted by the image.

In the drum loop case, this assumption is more problematic. There is certainly a relationship between vertically-adjacent values in a drum loop: they are different types of drum playing at the same time. However, there is no real sense that one pair of drum types is "more adjacent" than another. Thus a typical convolution kernel would be inappropriate. Similarly, horizontally-adjacent values are sequential in time, and so are in a relationship, but the relationship between lagged values (e.g. the values 4 beats apart, in a 4/4 time signature) may be much more significant. So, again, a typical image-processing kernel will be inappropriate. Possibilities arise here for future work, to be discussed in Section 7.1. In the current paper, we consider fully-connected networks only. This can be hypothesized to be sufficient, if inefficient: a fully-connected layer can learn any mapping that can be learned by a convolutional layer, at the cost of inefficiency in time and training data.

4 Proposed Model

We propose to use a standard VAE model for this representation learning task. It is a bottleneck architecture, where the input is passed through an encoder to a narrow middle layer, to be then passed through a decoder to the output, which is of the same size and shape as the input, with the goal of *reconstructing* the input at the output.

The main difference between a typical bottleneck autoencoder (AE) and a VAE is a complexity in the hidden layer. The encoder takes a drum loop as input, and outputs a pair of vectors representing the mean and variance of its latent code. The decoder takes as input samples from a Gaussian distribution with the given mean and variance, and outputs a drum loop. The purpose of this complexity is that it allows tight control over the distribution of latent codes. This is desirable because it allows us to sample from a Gaussian distribution in the latent space with confidence that the decoder has previously succesfully decoded similar points to well-formed drum loops. Without this control, the danger would be that a point sampled at the hidden layer was left “unused” by the network, and so would lead to a malformed result.

The details of the architecture are illustrated in Fig. 1.

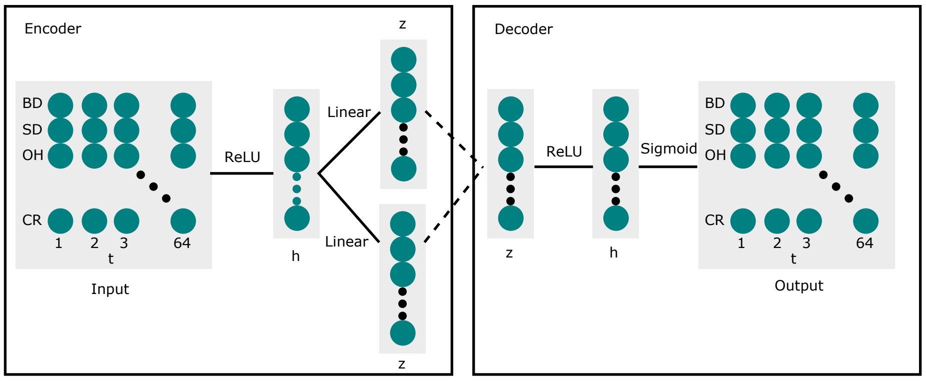

Fig. 1. VAE architecture. The input to the decoder is of size (9 x 64), a direct representation of a drum loop. In the encoder, the layers are a fully-connected layer with ReLU activation and output of size h, and then a pair of parallel fully-connected layers leading to the variational outputs μ and σ, each of size z. In the decoder, the input is a vector of size z, and the layers are a fully-connected layer with ReLU activation and output of size h, followed by a fully-connected layer with sigmoid activation with output of size (9 x 64).

In this paper, we choose $h = 400$ and $z = 20$ based on preliminary experiments. Larger values tend not to improve the final loss, as demonstrated in Section 6.3. The model is trained with the Adam optimizer [11] for 300 epochs. Again, preliminary experiments were used to check that this number was sufficient.

The loss function to be optimised in the VAE is: $L = L_R + \lambda L_{KL}$, where the reconstruction loss $L_R = \mathrm{BCE}(x, \hat{x})$, BCE is binary cross entropy, x is a datapoint, $\hat{x}$ is the reconstruction of x, L_{KL} is the Kullback-Leibler divergence from the desired distribution to the distribution observed in the latent codes, and λ is a hyperparameter controlling the strength of the Kullback-Leibler regularisation.

A mean square error loss function was also tested in preliminary experiments and did not give any improvement. It is known to result in more “blurry” results in image-processing VAE tasks.

5 Data and Preprocessing

The data used is a commercially-available library of drum loops, *Mega Pack* from Groove Monkee[2]. The library is categorized by style and tempo. For each

[2] `https://groovemonkee.com/collections/midi-loops/products/mega-pack`

"song", several loops are provided, giving (for example) an intro, a main beat, and a fill. 29,694 loops are provided. We discard any loops not in 2/4, 4/4 or 8/4 time signatures. We discard any loops which are less than 8 or more than 32 beats in length. We also ignore tempo information. For loops of 8 or 16 beats, we replicate them to bring their length to 32 beats. We quantize all hits to the nearest half-beat, giving 64 beats in all. We consider only 9 types of drum hits: BD (bass), SD (snare), OH (open hi-hat), CH (closed hi-hat), RD (ride), CR (crash), LT (low tom), MT (mid-tom), and HT (high tom). Any other drums are mapped to their nearest equivalents. As a result of this preprocessing, we have 9,468 drum loops, each of shape (9 x 64).

The library is thus comparable to MNIST, which consists of 60,000 images, each of size (28 x 28), with 10 digit labels which are thrown away when using the dataset with unsupervised approaches such as VAEs[3]. Although MNIST is seen as a small-scale and relatively easy problem for modern neural network methods, it was originally proposed as a problem of real-world significance. Similar to MNIST, the small scale of the drum loops does not imply that the problem is too small to be useful. In fact, in a sense our task is much more interesting. In MNIST there are 10 digit classes. Each digit could be represented (discarding handwriting style) in just 4 bits ($2^3 < 10 < 2^4$). No such representation would be possible for drum tracks. In fact, it seems unlikely that a very small representation (less than 10 dimensions, say) would be sufficient to capture most of the interesting and important variation in the drum loop corpus. This hypothesis will be tested in Section 6.3.

Although the library is commercially available, a sample is available for free download[4]. Our code for preprocessing the data and for learning the representation is available[5] and works also with the cut-down library. The neural network is implemented using PyTorch[6], and our code is partly based on a previous model[7].

6 Experiments and Results

We now proceed to evaluate the learned representation in several ways. In several cases we show results by plotting samples of the resulting drum loops as step sequencer-like grids. Corresponding drum loops are also available[8].

6.1 Samples

Sampling from the latent space and running forward to give new drum loops can give us an idea of the quality of the representation. Fig. 2 shows some results.

[3] `http://yann.lecun.com/exdb/mnist/`
[4] `http://groovemonkee.com/collections/midi-loops/products/drum-freebie-pak-gm`
[5] `https://github.com/jmmcd/drum-manifold`
[6] `https://pytorch.org/`
[7] `https://github.com/znxlwm/pytorch-generative-model-collections`
[8] `https://github.com/jmmcd/drum-manifold`

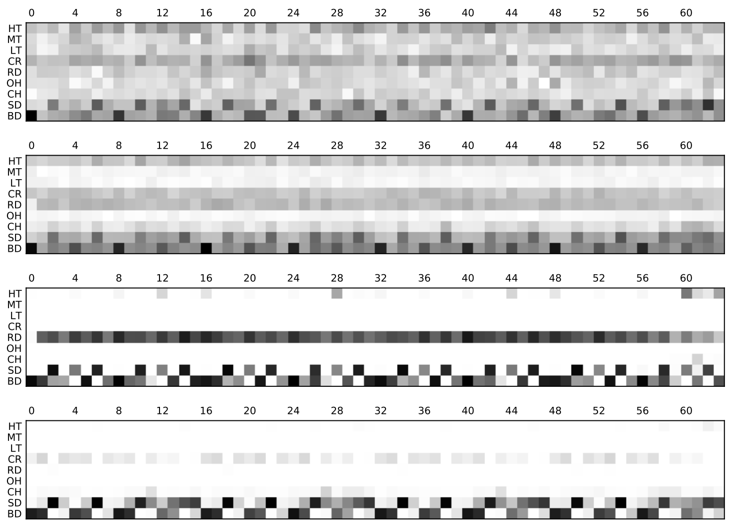

Fig. 2. Samples from the new representation after 1 epoch (top), after 5 epochs (second from top) and after 300 epochs (bottom two).

In the early epochs of learning (top) almost nothing has been learned about the drum loops except that bass drums (BD) on multiples of 4 are common. After 5 epochs (second from top), the model has learned something about bass and snare patterns and begins to de-emphasise mid- and low-toms. After 300 epochs (bottom two), the model has learned to produce "clean"-looking loops, with regularity and with the idea of embellishment at the end of the loop.

6.2 Comparison with Principal Components Analysis

As already described, Principal Components Analysis (PCA) can be used to learn a low-dimensional representation directly from the data. Here, we implement the standard PCA algorithm, retaining the 20 dimensions which capture the greatest variance in the data. Thus, the dimensionality is the same as in the VAE hidden layer. We then sample from the low-dimensional PCA space. Some results are shown in Fig. 3: they demonstrate almost none of the learning demonstrated by the neural model in the previous section. Thus, PCA fails to provide a useful representation in this case. It seems that the distribution of the drum loops in the original space cannot be well captured in by the linear transformation offered by PCA. A VAE uses non-linear transformations, and these seem to be essential. This is an expected result.

6.3 Analysing hyperparameters

Several neural network hyperparameters are worth investigating. The number of neurons per layer represents an important trade-off: more neurons gives more

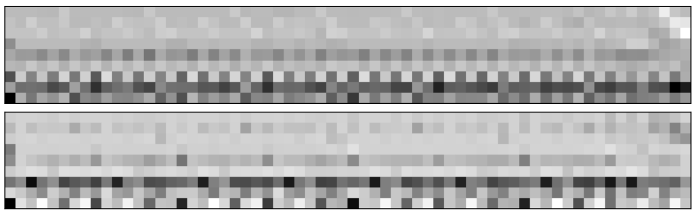

Fig. 3. Samples from a PCA representation.

representational power, i.e. ability to represent detailed differences between drum loops; but it requires more computation time, it may encourage over-fitting, and in the case of the latent code layer, it means the user of the eventual latent code would be forced to work in a higher-dimensional space. In this experiment we start with a known-good configuration $z = 20$ and $h = 400$, and vary the z and h values.

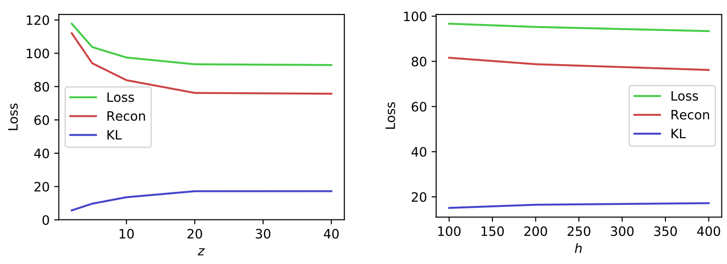

Fig. 4. The effect on network loss of the hyperparameters z, the size of the latent space (left) and h, the size of the hidden layers (right). The values shown are the two components of the loss and their weighted sum $L = L_R + \lambda L_{KL}$ at the end of training.

For $z < 20$ there is some degradation of reconstruction quality, because the small latent codes are insufficient to encode all the detail of the drum loops. For $z > 20$ there is no further benefit. For h again smaller values slightly degrade performance, but the effect is slight. Therefore, remaining experiments use $z = 20$ and $h = 400$.

6.4 Distribution of latent codes

Next, our goal is to demonstrate that the VAE approach is effective. In particular, the VAE introduces a regularisation penalty proportional to L_{KL}, which measures how dissimilar the distribution of latent codes is from the desired Gaussian distribution. The purpose of this penalty is to force the network to find a representation in which the latent codes of the training data is Gaussian-distributed. The strength of the penalty is controlled by the λ parameter.

We return to the known-good configuration $z = 20$ and $h = 400$, and test two values, $\lambda = 0$ (no regularisation) and $\lambda = 1$ (the default for VAE). Results are shown in Fig. 5. $\lambda = 0$ gives no control over the distribution (left) whereas $\lambda = 1$, the VAE default value (right), forces the distribution to be similar to a Gaussian with variance 1 centred at the origin.

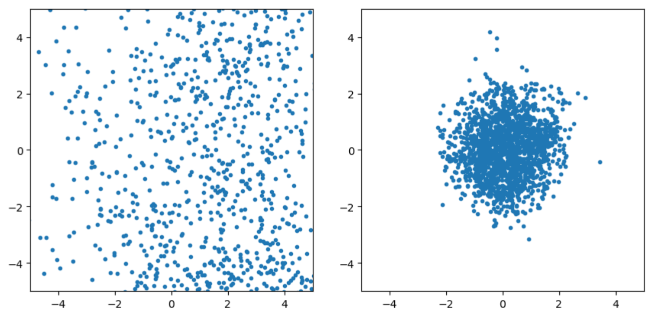

Fig. 5. The effect of the λ parameter on the distribution of latent codes. The first two dimensions of the latent codes are shown.

6.5 Reconstructions

The ability to reconstruct a drum loop, that is to take a loop, encode it to the new representation, and then decode it, is a good test of the quality of the learned representation. Ideally the reconstruction should closely match the original. Our representation succeeds in this, with some caveats. An example is shown in Fig. 6. Here the original drum loop (top) is unseen during training.

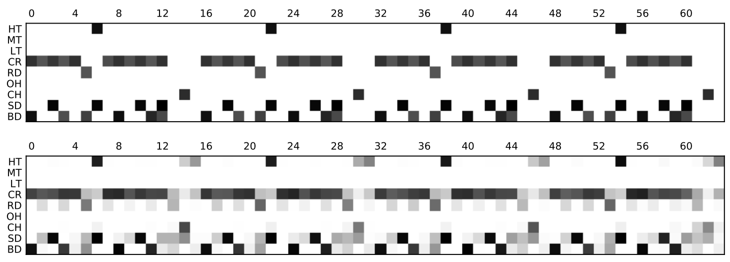

Fig. 6. Reconstruction of drum loops: original (top) and reconstruction (bottom)

The original pattern is closely matched, overall. However, a large number of "ghost notes" are introduced in the reconstruction, i.e. low-velocity drum hits. This does not occur for all original patterns, but is more common in loops sampled from the model than in the original corpus. The equivalent in an image-processing context would be a blurry image: such results are known to occur when training VAEs using mean square error loss functions, instead of binary cross entropy, for example. We have tried both and are using binary cross entropy for all reported results. In our context, the result may be due to the sigmoid mapping at the output, which would need a very large negative value as input in order to produce an output of velocity 0. Alternative activation functions will be investigated in future work to deal with this issue.

6.6 Interpolations

Interpolation or blending between a pair of items is a fundamental ingredient of some algorithms, such as evolutionary search. To test interpolation in the learned representation, we choose two fixed points in the learned representation, and take several points evenly spaced in the linear interpolation between them. For each point, we map it to a drum loop via the VAE decoder, and plot the results in Fig. 7. There is a strong sense of changing character as the interpolation progresses, especially in the BD (bass drum) and SD (snare drum) pattern, which goes from quite a complex and varied beat to a simple one. It is somewhat obscured by the presence of some ghost notes. Meanwhile, a simple pattern also gradually emerges in the CR (crash) and HT (high tom) drums.

7 Conclusions

In this short paper we have taken initial steps towards learning a good representation for drum loops. The idea of using a variational autoencoder to learn a representation from an unlabelled corpus has been motivated and investigated. A simple VAE architecture has been proposed for the task, using fully-connected layers with ReLU, Linear and Sigmoid activation functions. The effects of several hyperparameters have been investigated, and we have found that a reasonably small network is sufficient to obtain good reconstructions, samples, and interpolations in the latent space. They are far better than could be obtained with a simple baseline such as PCA. Results are presented as images in the paper and as MIDI files online.

The latent codes obtained in our representation are of size 20 (fewer are insufficient, and more give no apparent advantage). This size is quite comparable to the feature vectors developed manually by Eigenfeldt and Pasquier [5] and by Kaliakatsos-Papakostas [10]. Taken together, this can be counted as (weak) evidence that around 10-30 dimensions can represent sufficient detail of drum loops. This is an encouraging result since this number is also typical of many software synthesizers. The relevance is that human users are capable of understanding and controlling this number of dimensions in the context of a single musical instrument.

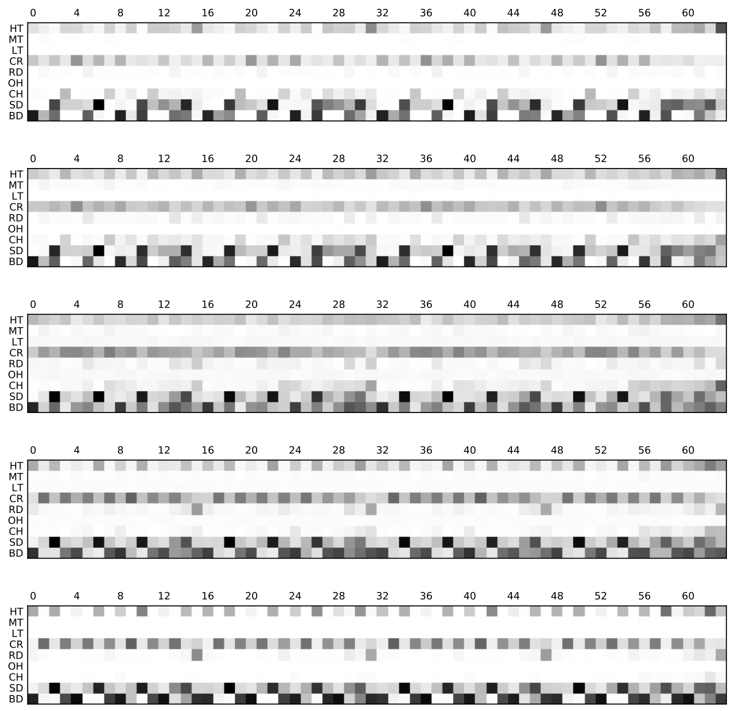

Fig. 7. Interpolation in the latent space. The top loop has the latent code $(-3, 0, 0, \ldots, 0)$, and the bottom has $(+3, 0, 0, \ldots, 0)$. The others are intermediate. Thus, the interpolation along a single dimension only. The two values (-3, 3) are quite "extreme" in a Gaussian distribution with standard deviation of 1.

7.1 Future Work

Several avenues are now open for future work.

As discussed, the VAE gives us control over the distribution of the training data when mapped into the new representation, which in turn allows us to sample, tweak, and interpolate in the new representation without the danger of sampling from an under-learned area. However, the VAE does not guarantee some other properties which would be useful in a representation, especially for human-computer interfaces.

One such representation property is sometimes referred to as being *disentangled* or having *semantic* dimensions. This means that varying just one dimension of the latent space will cause just one perceptual property of the output to vary. There may be no way to define just what a perceptual property is for any particular domain, but for example in the MNIST dataset of handwritten digits, it is possible to learn a disentangled representation in which two of the dimensions represent the *slant* of the digit (from left-slanting to right-slanting) and the *boldness* of the digit (from very thin to very bold). The properties of *slant* and *boldness* are semantic and are closely aligned to our perception.

In the representation learned by a VAE, dimensions are entangled and not semantic. The Beta VAE strongly increases the regularisation penalty, which is claimed to have the effect of encouraging disentangled representations [8]. The Shrink AE and Dirac Delta VAE enforce a very tight distribution, forcing the network to use the non-saturating parts of a tanh activation function, which may have the same effect [3]. The Wasserstein GAN encourages a disentangled distribution by using the Wasserstein or Earth-mover's metric on probability distributions, which improves training behaviour [1]. The Wasserstein AE transposes the same idea to the VAE world, where training is generally easier [17]. However, the most interesting possibility for us is the infoGAN, i.e. information-theoretic GAN, in which a semantic/disentangled property of a subset of the dimensions in the latent code is explictly maximised by the loss function [4].

Another property which could be of interest is to look for a distribution other than a Gaussian in the latent codes. Although a Gaussian is natural for statistical approaches, a human user may prefer to think of the representation as being uniform in a hypercube. VAEs do not naturally allow this, but GANs allow extra flexibility in the distribution [7].

In future work, we can also consider both CNNs and RNNs. As discussed in Section 3.2, typical image-processing convolution kernels may not be appropriate for drum loops, and so there is the possibility of creating custom kernels to encode our assumptions about the importance of horizontal and vertical relationships in a drum loop.

Finally, the real test of any representation is a set of user studies, and this is also planned.

Acknowledgments. Thanks to the anonymous reviewers for helpful suggestions.

References

1. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN. arXiv preprint arXiv:1701.07875 (2017)
2. Bourgain, J.: On Lipschitz embedding of finite metric spaces in Hilbert space. Israel Journal of Mathematics **52**(1-2), 46–52 (1985)
3. Cao, V.L., Nicolau, M., McDermott, J.: Learning neural representations for network anomaly detection. IEEE Transactions on Cybernetics (2018). https://doi.org/10.1109/TCYB.2018.2838668, in press
4. Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P.: InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In: Advances in Neural Information Processing Systems. pp. 2172–2180 (2016)
5. Eigenfeldt, A., Pasquier, P.: Evolving structures for electronic dance music. In: Proceedings of the 15th annual conference on Genetic and evolutionary computation. pp. 319–326. ACM (2013)

6. Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., Norouzi, M.: Neural audio synthesis of musical notes with wavenet autoencoders. arXiv preprint arXiv:1704.01279 (2017)
7. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
8. Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., Lerchner, A.: Beta-VAE: Learning basic visual concepts with a constrained variational framework. In: ICLR 2017 (2016), `https://openreview.net/forum?id=Sy2fzU9gl`
9. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
10. Kaliakatsos-Papakostas, M.: Generating drum rhythms through data-driven conceptual blending of features and genetic algorithms. In: International Conference on Computational Intelligence in Music, Sound, Art and Design. pp. 145–160. Springer (2018)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
12. Kingma, D.P., Welling, M.: Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114 (2013)
13. Manzelli, R., Thakkar, V., Siahkamari, A., Kulis, B.: An end to end model for automatic music generation: Combining deep raw and symbolic audio networks. In: Pasquier, P., Bown, O., Eigenfeldt, A. (eds.) 6th International Workshop on Musical Metacreation (MUME 2018). Salamanca, Spain (June 2018), held at the Ninth International Conference on Computational Creativity, ICCC 2018
14. Mikolov, T., Yih, W.t., Zweig, G.: Linguistic regularities in continuous space word representations. In: Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 746–751 (2013)
15. Moraglio, A.: Towards a geometric unification of evolutionary algorithms. Ph.D. thesis, University of Essex (November 2007), `http://eden.dei.uc.pt/~moraglio/`
16. Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Physica-Verlag, 2nd edn. (2006)
17. Rubenstein, P.K., Schoelkopf, B., Tolstikhin, I.: Learning disentangled representations with Wasserstein auto-encoders. In: ICLR 2018 Workshop (2018), `https://openreview.net/pdf?id=Hy79-UJPM`
18. Simon, I., Roberts, A., Raffel, C., Engel, J., Hawthorne, C., Eck, D.: Learning a latent space of multitrack measures. arXiv preprint arXiv:1806.00195 (2018)
19. Sturm, B.: What do these 5,599,881 parameters mean? an analysis of a specific lstm music transcription model, starting with the 70,281 parameters of the softmax layer. In: Pasquier, P., Bown, O., Eigenfeldt, A. (eds.) 6th International Workshop on Musical Metacreation (MUME 2018). Salamanca, Spain (June 2018), held at the Ninth International Conference on Computational Creativity, ICCC 2018
20. Trieu, N., Keller, R.: Jazzgan: Improvising with generative adversarial networks. In: Pasquier, P., Bown, O., Eigenfeldt, A. (eds.) 6th International Workshop on Musical Metacreation (MUME 2018). Salamanca, Spain (June 2018), held at the Ninth International Conference on Computational Creativity, ICCC 2018
21. Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499 (2016)