Department of Computer Science
George Mason University Technical Reports

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/ 703-993-1530

Using Variational Autoencoders to Improve FM Synthesizer Patch Discovery

Bryan Hoyle
bhoyle@gmu.edu

Technical Report

Abstract

Variational Autoencoders (VAEs) have been used to automatically discover a useful compression of a complex domain into smooth spaces in which simple euclidean operations can be performed in order to discover and create useful new examples within a problem space. This paper applies this technique to the domain of patch discovery in Frequency Modulation (FM) synthesis in order to turn a highly non-linear mixed metric parameter space into a smooth space in which a simple hill climber with human feedback can be used. This is then integrated into Edisyn, a synthesizer patch editor with automated patch discovery mechanisms, in order to provide a new algorithm with which it can operate. This paper then discusses qualitative results as well as future work to gather quantitative data with which it can compare against other prior results.

1 Introduction

When dealing with synthesizers, a patch is the collection of parameters describing a specific set of values on a particular synthesizer in order to create a specific timbre. The goal of creating patches is to create sounds that fulfill some goal in the context of a creative audio work. These goals can be as varied as creating a sound similar to a bowed string instrument to creating a sound effect for a game such as an alarm or a laser. The particularly important part of this process that the artist generally has to do manually is find a way to map the idea they have in their head to the values of the patch that can accomplish this. However, this is a difficult task for multiple reasons: synthesizers can have poorly designed interfaces, limited ways to modify patches, or be fundamentally difficult to understand. Of the common types of synthesizers, the one most widely regarded as difficult to program is frequency modulation (FM) synthesis [8]. FM synthesis is highly non-linear and non-intuitive and it is difficult to predict how a change in one parameter will affect the overall sound. Because of this, creating useful patches often takes a long time and lots of trial and error. Therefore, it is of interest to reduce the barrier of entry to this task.

Common efforts in reducing this difficulty come in many forms. The most common is the creation of patch editing software that provides a way to edit parameters in a UI that is more uniform across synthesizers or in a medium that is more convenient. A menu on a hardware synthesizer that takes multiple button presses and scrolling on a wheel to open and navigate through can become a nested dropdown on a computer, for example. While this helps with some of the UX issues, there are still the problems with parameter mapping and understanding at play which need to be addressed. Less commonly, there have been attempts to create automated patch discovery tools [9] which use various techniques to attempt to assist the artist in creating patches by either reducing complexity or searching the space and presenting possibly interesting ones to the user directly. Edisyn [9], for example, is an open source patch editor that attempts to address both of these issues simultaneously by providing both a convenient consistent UI to multiple synthesizers as well as ways to assist in patch discovery using evolutionary computation (EC) methods.

EC methods are a set of algorithms which attempt to search for optimal examples within some space. They do this by using a concept called a *fitness function* which is nothing more than a way to score an example with a metric of "goodness." They then use various techniques to narrow down what parts of the space they are working within. One of the most common (and simple) EC techniques used is called a hill-climber: it samples multiple points in a space, scores them with the fitness function, then attempts to figure out which direction is "uphill" and will attempt to continue in that direction.

One of the major problems with applying hill-climbing to synthesizer patches is the specifics of the patch structure itself: patches often have non-metric parameters (such as square-wave vs sawtooth-wave) which create large audible non-linearities in the search space. Another issue is the size of the space in which it is searching as high dimensional spaces can often be difficult to find a direction in. Lastly, there is very little way to automatically rank candidates, as how good a patch is depends entirely on its purpose and the ear of the artist creating it. Thus, EC methods in this area usually use the artist themselves as the fitness function, but this introduces its own host of issues such as human attention span and speed of ranking. Most solutions in this space, therefore, attempt to limit the number of poor patches presented to the artist and attempt to converge quickly to the desired patch [9].

In order to support these goals, this paper attempts to add to the growing collection of patch discovery methods by using a machine learning model called a Variational Autoencoder (VAE) to simplify the space within which the EC needs to search. In reducing the space to hopefully more useful regions as well as making it smooth, this technique ideally mitigates some of the issues that other attempts have run into. We then integrate the VAE into Edisyn and use it in conjunction with a hill-climber to generate patches and provide qualitative comparisons against prior techniques.

2 Previous Work

Prior work on assistive patch exploration has been varied, but many use evolutionary techniques [10, 1, 9], with [1] being particularly well known as it became used in a commercial product at one point. Interestingly, some modern approaches to timbre generation sidestep patches entirely, with techniques like Wavenet Autoencoders [2] being used in a Google Magenta project called the NSynth that generate audio directly from the neural net architecture.

2.1 Edisyn

Of note to this research is Edisyn. Edisyn is a patch editor for many synthesizers which is open source and written in Java. Edisyn comes with a set of algorithms for patch discovery, namely a hill-climber and a constrictor, which uses interactive-EC in order to facilitate human-computer interaction [9]. Edisyn comes with multiple EC techniques, such as a hill climber which attempts to climb in the patch space directly using techniques to attempt to avoid steep non-linearities, and a constrictor which attempts to bound search within a section of timbral space in order to reduce the area within which it has to look. The techniques used in Edisyn performed statistically significantly better than direct patch creation. An example of Edisyn's hill climbing interface can be seen in Figure 1.

2.2 Variational Autoencoders

An autoencoder is a machine learning technique which uses statistical methods to compress information by discovering commonalities within a set of data. Specifically, they attempt to take high dimensional data and learn a smaller space, known as a latent space, that hopefully can still represent the useful qualities of the original data. They achieve this by taking samples, encoding them into the latent space, then decoding them back into the original space and penalizing the autoencoder based on how different the decoded version is from the encoded version. Variational Autoencoders attempt to extend this technique to not only learn a compressed space, but a smooth space, in which two samples that are close to each other in this latent space should share similar characteristics in their original space.

The way they work is that they use posterior distribution approximations in combination with standard autoencoders in order to, ideally, create smooth latent spaces [6]. Specifically, for this paper, we use β-Variational Autoencoders [4] in order to be able to control the amount of smoothness of the discovered latent space. These latent spaces can then be used for many tasks, such as semantic distance between texts [5], image generation [11], and other tasks which require finding some sort of euclidean vector space for a complex domain. The information compressive nature of autoencoders also generally mean that the space that is created is generally less semantically sparse: it is harder to find a point in the compressed space (within some bounds) that decompresses to something out-of-task than it is to find an out-of-task sample in the original domain. For the purposes of this paper, an out-of-task sample would be a non-useful patch.

The way Variational Autoencoders achieve the smooth latent space is by learning a set of distributions during training rather than just directly compressing. For each latent dimension, two parameters are learned: a mean (μ) and a variance (σ). During training, the VAE compresses down to these two parameter vectors then, instead of directly decompressing from μ, it samples from a normal distribution described by those parameters. That sampled value is then fed through the decompressor and treated like the output from a standard autoencoder. Intuitively, this makes sense: you are teaching the network that anything in a small radius around each point in the latent space should decompress to the same/similar value, creating smoothness. Due to the fact that Gaussians will sample less often further away, the requirement for similarity should also fall off over distance. In order to get a gradient to backpropagate through this sampling step, a reparameterization trick is used: $\mathcal{N}(\mu, \sigma) = \mu + \sigma \cdot \mathcal{N}(0, 1)$, thus the derivative

Figure 1: Edisyn's hill climber interface. Note that the user can rank up to 3 patches which they like which will be used to generate the new patches. There is also an area to store patches that were particularly liked so one can reuse them later. The amount of randomness in the algorithm is controlled by the "Mutation Rate" knob.

in respect to μ and σ can be easily found by just treating $\mathcal{N}(0,1)$ as a constant. In order to stop the network from learning a distribution with a σ of 0, thus reducing the network to a standard autoencoder, a Kullback–Leibler (KL) divergence term between the learned distribution and a standard normal is added, thus penalizing the network the further from standard normal it becomes. This KL term is scaled by a parameter called β, which allows one to tune the amount of smoothness required for the task [4].

After the model is trained, one can discard the σ parameter and only use μ. This means that it has an identical interface to standard autoencoders for compression and decompression tasks after training, but ideally with a latent space in which euclidean operations make sense.

3 Implementation

For this project, we focused specifically on the Yamaha DX7 as the synthesizer to build the autoencoder for. This is because it is a hard to program FM synthesizer as well as being one of the most popular synthesizers ever created. This means that there is potential interest in any solution that improves patch discovery as well as a large corpus of human curated patches to use to train a Variational Autoencoder. For this project we used approximately 26,000 patches, 5,000 of which were used as a test set to monitor reproduction loss during training.

The patches were preprocessed into vectors. For metric parameters, we scaled the value between zero and one based on the range of the parameter. For categorical parameters, we used one-hot encoding. We then concatenated these values together to create a single vector of length 225. This was used as the input and target of the VAE. For loss, we initially used a mixed loss function with mean squared error (MSE) for the metric parameters and cross-entropy for the categorical parameters, but that had loss scaling issues and had trouble converging. Thus, we settled on MSE for the loss across the entire output, which gave good results. This loss was combined with the KL-divergence loss term with a β value of 0.1, giving a space that is a little less smooth but with better reconstruction results. The inner dimension of the final model was 64, which seemed to give a good tradeoff between reconstruction loss and space constriction. The nonlinearity used between layers was SELU [7] due to its excellent gradient and self-normalizing characteristics. The full architecture is summarized in Table 1.

After the model was trained, it was integrated into the hill climber. The hill climber would populate the initial list by feeding the initial patch into the encoder section of the network, sample a Gaussian centered at the μ vector given by the encoder and a variance controlled by the *Mutation Rate* parameter as seen in Figure 1. These vectors would then become the next generation of children with which hill-climbing can be performed.

If only one child is selected to mutate on, then the next

Layer Name	Input Size	Output Size	Activation
Encoder 1	225	128	SELU
Encoder 2	128	76	SELU
Encoder 3	76	64	SELU
μ Layer	64	64	None
σ Layer	64	64	None
Decoder 1	64	76	SELU
Decoder 2	76	128	SELU
Decoder 3	128	225	SELU

Table 1: Network architecture for the final VAE model. Note that each layer listed is a fully connected linear layer.

generation is created by the technique used to create the initial samples, but centered around the chosen child. If two children are selected, half of the children are generated around the highest ranked point, a quarter around the second highest ranked point, and a quarter are sampled from around their latent vector averages. If three children are selected, a similar process is run as for two children, but the children are also divided up among all of the possible combinations of vector averages between the three selected children, including the overall average point. At any point, a user can select a child to store in the bank of six patches at the top to use as a reference for later.

After the user runs this loop for a while, they can select the patch they like the most and the process is complete.

4 Qualitative Results

Due to time constraints, a quantitative comparison against the results given in [9] could not be given, however, qualitative results can be discussed. The author was able to use the hybrid VAE-Hill-Climbing technique to generate useful, interesting, or musical patches in a few minutes each. This is in comparison to the half-hour or even longer it often takes to do the same process manually. The patches generated by this technique were generally mostly usable directly out of the algorithm, often with no tweaking necessary. On the occasions that a patch did require tweaking, it only required about a minute and was pretty close to finished. The timbral quality of patches that were generated during the hill climbing exploration steps were not always great, but it did feel like it was better than the previously implemented versions in Edisyn in several ways: the first being that it generates much more diverse useful sounds with less time spent exploring, and, secondly, the differences between specific children were often different in more tangible ways rather than being the same patch but with more or less vibrato or tremolo. There were also fewer unexpected abrupt changes between children. This lends itself to finding useful patches quickly and allowing the user to get a wider range of sounds rather than being stuck in large swaths of similar sounding space. However, this diversity in the patches sometimes creates a good bit of unusable children, which can require multiple iterations of generation before a useful child is discovered. The technique felt better to use in general and seemed immediately useful.

5 Conclusion and Future Work

In order to confirm or deny the author's intuitions, however, an experiment setup similar to the one given in [9] needs to be conducted. If there is a statistically significant improvement over the technique given, it can be assumed that the new version is better at finding patches that are considered musical or useful. Doing a larger hyperparameter search over architectures for embedding could also give better results, as the combined encoder-decoder loss leaves a lot to be desired over a standard autoencoder (which, given the same architecture, had half the loss of the VAE), meaning that a good bit of the patch information is being destroyed. Additionally, other generative techniques may be considered, such as Generative Adversarial Networks [3], which may give better curated patches, limiting the amount of time required to listen to obvious rejects. Other future work includes creating a system in which a collection of patches along with some metadata can be automatically fed into a program which generates and trains a VAE such that it can be rapidly integrated with Edisyn. Overall, this technique is promising, but needs proving out before any definite conclusions can be made.

References

[1] P. Dahlstedt. A mutasynth in parameter space: Interactive composition through evolution. *Organised Sound*, 6, 07 2001.

[2] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders. 2017.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[4] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[5] N. Jung and H. I. Choi. Continuous semantic topic embedding model using variational autoencoder, 2017.

[6] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2014.

[7] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks, 2017.

[8] S. Luke. *Computational Music Synthesis*. Zeroth edition, 2019. Available for free at http://cs.gmu.edu/∼sean/book/synthesis/.

[9] S. Luke. Stochastic synthesizer patch exploration in Edisyn. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (EvoMUSART)*, 2019.

[10] J. McDermott, M. O'Neill, and N. J. L. Griffith. Interactive EC Control of Synthesized Timbre. *Evolutionary Computation*, 18(2):277–303, 06 2010.

[11] A. Sagar. Generate high resolution images with generative variational autoencoder, 2020.