Audio Engineering Society

Conference Paper 9846

Presented at the 143rd Conference on Convention
2017 October 18–21, New York, NY, USA

Improving Neural Net Autoencoders for Music Synthesis

Joseph Colonel[1], Christopher Curro[1], and Sam Keene[1]

[1]*The Cooper Union for the Advancement of Science and Art*

Correspondence should be addressed to Joseph Colonel (`colone@cooper.edu`)

ABSTRACT

We present a novel architecture for a synthesizer based on an autoencoder that compresses and reconstructs magnitude short time Fourier transform frames. This architecture outperforms previous topologies by using improved regularization, employing several activation functions, creating a focused training corpus, and implementing the Adam learning method. By multiplying gains to the hidden layer, users can alter the autoencoder's output, which opens up a palette of sounds unavailable to additive/subtractive synthesizers. Furthermore, our architecture can be quickly re-trained on any sound domain, making it flexible for music synthesis applications. Samples of the autoencoder's outputs can be found at `http://soundcloud.com/ann_synth`, and the code used to generate and train the autoencoder is open source, hosted at `http://github.com/JTColonel/ann_synth`.

Introduction

Recent developments in the field of artificial neural networks have opened the door for a wide variety of creative uses. While much research has been done in the field of computer vision and generative networks, advancements have also been made in the field of music and audio. Google's WaveNet architecture generates a piano composition one sample at a time that sounds as if a trained pianist is playing [1]. The University of Montreal MILA lab's work with Lyrebird has produced a neural network which can generate speech that mimics a human's voice with only one minute of training audio [2]. Another of Google's architectures, the NSynth, uses instrument classes to generate specific timbres and can create sounds that interpolate between classes [3].

Another example is an autoencoding neural network (autoencoder) architecture for use in musical synthesis, proposed by Andy Sarroff [4]. Compared to the work of [1], [2], and [3], this architecture has the advantage of being easy to train, tune, and alter by new users. Furthermore, this lightweight architecture allows for real time tuning, audio generation, and performance.

In [4]'s implementation, the autoencoder's encoder compresses an input magnitude short time Fourier transform (STFT) frame to a latent representation, and its decoder uses the latent representation to reconstruct the original input. By modifying the latent representation of the input, the decoder generates a new magnitude STFT frame. However, [4]'s proposed architecture suffers from poor performance, measured by mean squared error (MSE) of a given input magnitude STFT frame and its reconstruction. Thus the autoencoder has a poor range of musical applications.

Our work builds on [4]'s initial results and improves the designed autoencoder through modern techniques and frameworks. These improvements reduce MSE for each of [4]'s proposed topologies, thus widening the scope of the autoencoder's musical applications. We present comparisons of our topology's performance and [4]'s, and also develop an expansion of useable topologies. Furthermore we outline our design process and choices in the hopes of laying a sound foundation for further work.

Methods

Autoencoders

An autoencoder is comprised of an encoder and a decoder [5]. The encoder shrinks the dimension of an input into a latent space using a discrete number of values, or "neurons." We may conceive this latent space as the hidden layer (HL) representing high level, descriptive features of the input. The decoder then expands the dimension of the latent space to that of the input in a manner that reconstructs the original input. In this work, the encoder maps an input vector $x \in [0,1]^d$ to the hidden layer $y \in [0,1]^e$, where $d > e$. Then, the decoder maps y to $\hat{x} \in [0,1]^d$. In its traditional formulation, the encoder maps $x \rightarrow y$ via

$$y = s(Wx + b) \tag{1}$$

where $W \in \mathbf{R}^{(e \times d)}$, $b \in \mathbf{R}^e$, and $s(\cdot)$ is an activation function that imposes a non-linearity in the neural net. The decoder has a similar formulation:

$$\hat{x} = s(W'y + b_{\text{out}}) \tag{2}$$

with $W' \in \mathbf{R}^{(d \times e)}$, $b_{\text{out}} \in \mathbf{R}^d$.

The autoencoder trains the weights of the W's and b's in the nets to minimize some cost function. The cost function used in this work is mean squared error (MSE):

$$C(\theta_n) = \frac{1}{d}\sum_{k=1}^{d}(\hat{x} - x)^2 \tag{3}$$

where θ_n is the autoencoder's trainable weight variables.

The process of choosing activation functions $s(\cdot)$ and cost functions relies heavily on the domain of a given task.

Autoencoding Learning Task Description

The autoencoding neural network used here takes 1025 points from a 2048 point magnitude Fourier frequency transform as its input, i.e. $x \in [0,1]^{1025}$. These 1025 points represent the DC and positive frequency values of a given frame's STFT. All audio processing was handled by the LIBROSA Python library [6]. In this application, LIBROSA was used to create Python arrays from .wav files sampled at 22.05kHz, perform STFTs of length 2048 with centered Hanning window, hop length 0, and reconstruct .wav files with sampling frequency 22.05kHz from reconstructed magnitude STFT frames. The phase of each magnitude STFT frame was passed directly from input to output, circumventing the autoencoder.

The neural network framework was handled using TensorFlow [7]. All training used the Adam method for stochastic gradient descent with mini-batch size of 100 [8]. Learning rates used for training varied from 10^{-3} and 10^{-4} between runs. This is a departure from [4]'s proposed architecture, which used the momentum method of stochastic gradient descent with learning rate 5×10^{-3} and momentum 0.5 [9].

All topologies were trained using $70,000$ magnitude Fourier frequency transform frames, with 10,000 frames held out for testing and another 10,000 for validation. All audio was generated using a MicroKORG synthesizer/vocoder. By restricting the corpus to pieces generated from a MicroKORG, the autoencoder need only learn higher level features of audio made of harmonic synthesizer content, rather than that of voice or percussion. The MicroKORG has a maximum of four note polyphony for a given patch, thus the autoencoder must learn to encode and decode mixtures of at most four complex harmonic tones. These tones often have time variant timbres and effects, such as echo and overdrive.

The encoder and decoder are then trained on these magnitude STFT frames to minimize the MSE of the original and reconstructed magnitude STFT frames. Several different network topologies were used, varying the depth of the autoencoder, width of HLs, and choice of activation function. We first recreated [4]'s topology using the Adam training method, and then proceeded to design a four layer deep autoencoder that can be used for unique audio effect generation and audio synthesis. For both the one and two layer models, no additive bias

Table 1: Depth 1 Topology MSEs

HL Depth	Momentum	Adam
8	4.40×10^{-2}	5.30×10^{-3}
16	4.14×10^{-2}	5.28×10^{-3}
64	2.76×10^{-2}	7.10×10^{-4}
256	1.87×10^{-2}	1.64×10^{-4}
512	1.98×10^{-2}	9.62×10^{-4}
1024	3.52×10^{-2}	7.13×10^{-5}

Table 2: Depth 2 Topology MSEs

HL Depths	Momentum MSE	Adam MSE
256-8	1.84×10^{-2}	1.91×10^{-3}
256-16	1.84×10^{-2}	1.19×10^{-3}
256-32	1.84×10^{-2}	7.30×10^{-4}

term b was used, and all activations were the sigmoid (or logistic) function:

$$S(x) = \frac{1}{1+e^{-x}} \tag{4}$$

A more in depth look at the neural network design choices we made is in the **Discussion** section.

Optimization Method Improvements

Table 1 and Table 2 show the MSEs of the network topologies outlined by [4]. The first column of Table 2 describes the autoencoder's topology, with the first integer representing the neuron width of the first layer, and the second integer representing the neuron width of the second layer. Table 3 shows the MSEs of a 4 layer deep autoencoder, with encoder HL depths $512 \to 256 \to 128 \to 64$. Table 3 also shows the MSEs of three different topologies that were chosen for the 4 layer deep autoencoder: one with sigmoid activations throughout, one with ReLU activations throughout, and a hybrid model. This model used a sigmoid on the innermost HL and on the output layer, with all other layers using a ReLU activation. This hybrid topology performed best.

Figure 1 shows graphs of an input magnitude FFT frame (top) and corresponding reconstructed magnitude FFT (bottom), with magnitude on the y axis and frequency bin on the x axis. Contrary to [4]'s work, the signal reconstruction improves both qualitatively and quantitatively as the depth of the hidden layer is increased.

Table 3: Deep Topology MSEs and Train Times

Activations	MSE	Time to Train
All Sigmoid	1.72×10^{-3}	20 minutes
All ReLU	8.00×10^{-2}	60 minutes
Hybrid	4.91×10^{-4}	25 minutes

Discussion

There are several distinctions between the architecture originally proposed by [4] and the architectures used in this work: the choice of the autoencoder's stochastic training method, the regularization techniques used to create a robust latent space, the activation functions chosen, the use of additive bias terms b, and the corpus used for training.

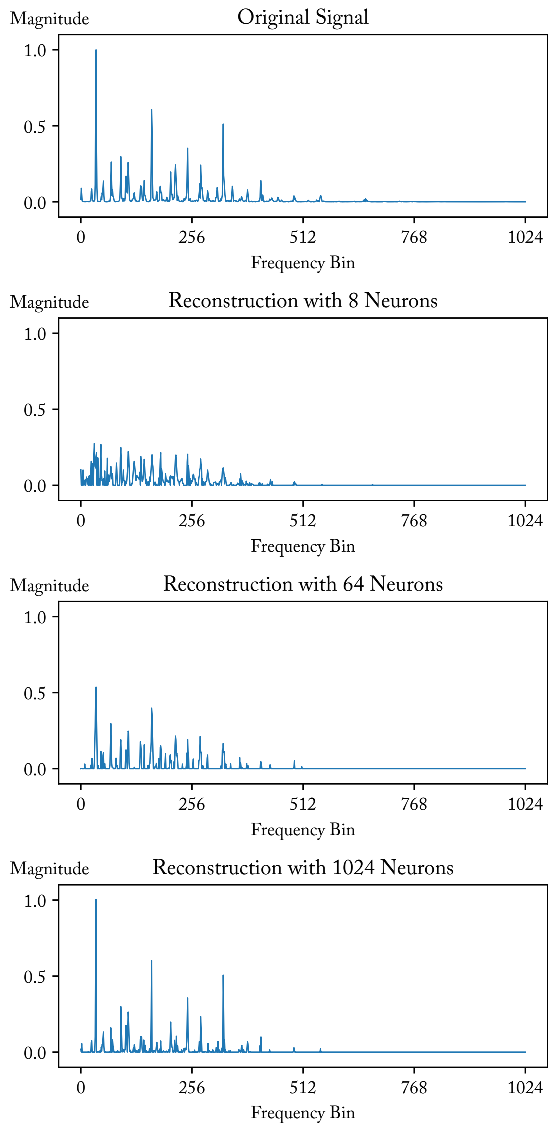
Training Methods

The improved MSEs in Table 1 and Table 2 demonstrate the ability of the Adam method to train autoencoders in this context better than the momentum method [4] proposed. The momentum method produced MSEs orders of magnitude higher than Adam, suggesting that the method found a poor local minimum and did not explore further. The result of the 8-neuron hidden layer in Figure 1 demonstrates the poor reconstructions produced by an autoencoder with MSE on the order of 10^{-3}. [4]'s performance suggests that the momentum method produced similar results. While these reconstructions are interesting to listen to, they do not accurately reconstruct an input magnitude STFT frame. The adaptive properties of the Adam technique ensure that the autoencoder searches the weight space in order to find robust minima.

Regularization

[4] suggested using denoising techniques to improve the robustness of autoencoder topologies. We found that denoising was not necessary to create robust one and two layer deep autoencoders. However, we did encounter issues when training the four layer deep autoencoder topology. Our original topology used sigmoids as activations throughout. The autoencoder did

Fig. 1: The topmost plot shows the magnitude spectrum of the original signal, while the remaining show the reconstructed magnitude spectra for various sized hidden layers.



not converge, however, potentially due to the vanishing gradient problem inherent to deep training [10]. To fix this we used sigmoid activations on only the hidden and output layers, and used rectified linear units (ReLUs) for the rest of the layers [11]. The ReLU is formulated as

$$S(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{5}$$

This has the benefit of having a gradient of either 0 or 1, thus avoiding the vanishing gradient problem.

We explored two regularization techniques: dropout and an $l2$ penalty [12] [13]. Dropout involves multiplying a Bernoulli random vector $z \in \{0,1\}^{t_i}$ to each layer in the autoencoder, with t_i equal the dimension of the i^{th} layer. Dropout encourages robustness in an autoencoder's encoder and decoder, and the autoencoder's quantitative performance did reflect this. However, the dropout regularizer hampered the expressiveness of the autoencoder by ignoring slight changes to the latent space.

The second technique, $l2$ regularization, proved to perform the best in qualitative listening comparisons. This technique imposes the following addition to the cost function:

$$C(\theta_n) = \frac{1}{d}\sum_{k=1}^{d}(\hat{x}-x)^2 + \lambda_{l2}\|\theta_n\|_2 \tag{6}$$

where λ_{l2} is a tuneable hyperparameter and $\|\theta_n\|_2$ is the Euclidean norm of the autoencoder's weights. This normalization technique encourages the autoencoder to use smaller weights in training, which we found to improve convergence.

Activation Functions

The choice of sigmoid activation for the hidden layer was motivated by the use of multiplicative gains for audio modulation. Because the range of the sigmoid is strictly greater than 0, multiplicative gains were guaranteed to have an effect on the latent representation, whereas a ReLU activation may be 0, thus invalidating multiplicative gain.

The choice of sigmoid activation for the output layer was twofold. First, a magnitude STFT frame has a min of 0 and max of 1, which neatly maps to the range of the sigmoid function. Second, experiments demonstrated that while the ReLU activation on the output would produce acceptable MSEs, the sound of the reconstructed signal was often tinny. The properties of the sigmoid activation lend themselves to fuller sounding reconstructions.

Additive Bias

Finally, we found that using additive bias terms *b* created a noise floor within the autoencoder. When present, using gain constants in the hidden layer produced noisy results. Though additive bias terms did improve the convergence of the four layer deep autoencoder, we ultimately chose to leave them out in the interest of musical applications.

Corpus

An issue with [4] is the use of several genres of music to generate a dataset. As different genres have different frequency profiles, the net's performance drops. For example, a rock song's frequency profile can be broken down as the sum of spiky low frequency content created by drums, tonal components from guitar and bass, complex vocal profiles, and high frequency activity from cymbals. Including several genres of music in a corpus trains an autoencoder to be a jack of all trades, but master of none. By focusing the corpus of our exercise on tonal sounds, we encourage the net to master representations of those sounds. Thus when it comes time for modifying an input, the latent space contains representations of similar yet distinct synthesizer frequency profiles.

Summary

We present an improved method of creating lightweight, easily trainable, deep autoencoders that can be used as a synthesizer and audio effect. Ultimately a 4 layer deep topology was chosen, using both sigmoid and ReLU activations and eschewing the traditional additve bias term between layers. The authors have uploaded audio samples produced from the final autoencoder at *soundcloud.com/ann_synth* , and the code used to generate and train the autoencoder is open source, hosted at `github.com/JTColonel/ann_synth`.

References

[1] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016. [Online]. Available: http://arxiv.org/abs/1609.03499

[2] "Lyrebird, an api for speech synthesis," https://lyrebird.ai/, 2017.

[3] 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 e-prints*, Apr. 2017.

[4] A. Sarroff, "Musical audio synthesis using autoencoding neural nets," Dec 2015. [Online]. Available: http://www.cs.dartmouth.edu/~sarroff/projects/autoencoding-synthesizers/

[5] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[6] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, 2015, pp. 18–25.

[7] M. Abadi, "Tensorflow: Learning functions at scale," *ICFP*, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=2976746

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[9] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available:

http:
//proceedings.mlr.press/v28/sutskever13.html

[10] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[13] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *NIPS*, vol. 4, 1991, pp. 950–957.