Exploring Audio Diffusion Synthesis: A Novel Approach to Sound Generation Using Neural Networks

Cameron Olson

Abstract

This research paper explores Audio Diffusion Synthesis, a novel sound generation technique utilizing Neural Networks. Specifically, the paper will heavily concentrate on the Dance Diffusion architecture developed by Zack Evans, a researcher at Harmonia. The paper begins by introducing key terminology associated with Neural Networks to aid in understanding the machine learning systems used in this project. Next, the paper discusses foundational aspects of machine learning for musicians. We will then delve into the world of Neural Synthesis and cover essential systems in this field. A deep dive into Denoising Diffusion Probabilistic Models (DDPMs/Diffusion) is presented to provide a comprehensive understanding of how Dance Diffusion functions. The paper details the entire project process, including system selection, data collection, advanced augmentation techniques, hyperparameter tuning, model training, and research findings. The author collected and utilized a large percussion dataset consisting of one shot audio samples, the results demonstrate the effectiveness of DDPMs in generating audio signals that closely resemble real-world audio data. Overall, this research paper provides valuable insights into the application of DDPMs in audio synthesis and highlights the potential of this technique in generating new and unique audio signals for music production and sound design.

Keywords: Audio Diffusion Synthesis, Neural Networks, Denoising Diffusion Probabilistic Models, Machine Learning, Audio Synthesis.

1 Introduction

As a student studying electronic production and design, I have always been fascinated with sound. My fascination with Neural Synthesis was ignited when I first heard "Nuns In A Mosh Pit," a song created by the DadaBots, an AI Research team that specializes in generative audio systems. The creation of this song was entirely driven by Neural Networks, a concept that once seemed impossible to me. At first, I assumed it was using MIDI generation, but I was mistaken; to my amazement, the song was entirely generated at audio rate. After this I decided I needed to understand how this was even possible, and there was no stopping me.

During my EP-401 class I dedicated a significant amount of time to immersing myself in the world of neural networks. I completed multiple courses, striving to build a solid foundation of knowledge and gain a comprehensive understanding of this complex subject. After completing my courses, and learning PyTorch, I embarked on a research project centered around audio

diffusion. My primary objectives for this project were twofold. Firstly, I aimed to achieve high-quality sound generation with significant variation. High quality was paramount, as I had previously encountered AI-generated audio with a high amount of noise, rendering it unusable. Secondly, I sought to explore novel sound design techniques that are made possible with DDPM systems.

2 Introduction to Neural Networks

2.1 What are Neural Networks

What are Neural Networks? In this section I will dive into Neural Networks, to give you a foundational understanding of what it is, how it works, and some vocab to give you enough understanding to comprehend the content of this paper. At its core neural networks are mathematical functions, data goes in one end, a result comes out the other, it's as simple as that. To make this magical machine we only need a few things, a lot of data, a network, and a way to evaluate how good the network is doing. With these key ingredients, we get a trained model. The training process though, is what I find the most exciting.

2.2 Foundational understanding Neural Networks

To build a foundational understanding of neural networks, it's important to start with the basic building blocks: neurons. These neurons are mathematical functions that take in input data, multiply it by a weight, add a bias, and then output a result. By connecting many of these neurons together in layers, we can create a neural network that can model complex relationships in data. Each neuron is also enhanced with non-linear activation functions such as the hyperbolic tangent (tanh) or Rectified Linear Unit (ReLU). which helps the network learn even more sophisticated patterns. The layers themselves are densely connected, meaning every neuron in a given layer is connected to every neuron in the previous layer. By adjusting the weights and biases of the neurons during a process called training, the network can learn to accurately predict outputs for new inputs.

2.3 Training a Neural Network

The training process for a neural network involves several key steps. We first randomly initialize the weights and biases of the neural network before starting training, then, Input data is passed through the network in a process called forward propagation, where each neuron calculates a weighted sum of its inputs, adds a bias term, and applies an activation function to produce an output. The difference between the predicted output and the actual target is measured using a loss function. Backpropagation is then used to calculate the gradient of the loss function with respect to each weight and bias, which is then used to update the weights and biases. Optimization algorithms such as stochastic gradient descent or Adam are used to adjust the parameters in the direction of the negative gradient to minimize the loss. This process is repeated iteratively until the loss is minimized to a satisfactory level. Once the neural network is trained, it can be used for making predictions on new, unseen data. In technical terms this is all called supervised learning, we will cover other types later.

3 Machine Learning for Musicians

Before exploring the practical applications of machine learning in music, it's important to understand the two main types of machine learning in this field. The first type is symbolic machine learning, which involves representing musical data in a symbolic format such as MIDI files or sheet music. This enables the use of mathematical and computational techniques to analyze and manipulate musical data. For instance, Google's Magenta Suite of machine learning tools offers various networks that can be trained on MIDI data. The second type is audio-based machine learning, which involves representing musical data in an audio format such as waveforms or spectrograms. Machine learning algorithms can then analyze the features of the audio data and make predictions based on those features. Examples of networks that utilize this type of machine learning include JukeBox by OpenAI and Dance Diffusion, which is the main network for this paper.

4 Neural Synthesis

Neural Synthesis is a cutting-edge form of audio synthesis that utilizes neural networks to generate raw audio output. If you're interested in exploring this fascinating field, there are a number of important systems that are worth playing with.

- **WaveNet and N-Synth**. One of the first systems in Neural Synthesis, coined the term itself. Allows for combining sounds and controlling them with MIDI. It's still widely used and worth exploring.

- **SampleRNN.** A legacy network that enables the generation of raw audio.

- **JukeBox.** A powerful network developed by OpenAI, trained on over 6000 artists and song lyrics. It allows for combining styles and artists, as well as adding vocals. JukeBox is a fun option for creating unique sounds and vocal arrangements.

- **Catch-A-Waveform.** Designed for small datasets of around ten seconds to one minute. Catch-A-Waveform has the potential to generate unlimited audio from small datasets, making it an exciting option for sound design.

- **RAVE.** The first real-time generative system, allowing for the production of audio on the fly.

- **ReFusion.** A fine-tuned version of stable diffusion, which takes in text input and generates spectrograms. The sound quality may be low, but the ability to use text as input makes it an interesting tool for exploration.

- **Dance Diffusion.** A modern generative network that relies on diffusion and is focused on producing high-quality output.

5 Denoising Diffusion Probabilistic Models:

5.1 Authors note

My research for my project started with trying to understand diffusion systems. To do this I read the following papers: “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”, “Denoising Diffusion Probabilistic Models”, “Improved Denoising Diffusion Probabilistic Models”, and “Diffusion Models Beat GANs on Image Synthesis”. Through all of this reading I gained a foundational understanding of Diffusion models, here are my findings.

5.2 Introduction

Denoising Diffusion Probabilistic Models (DDPM) have emerged as a promising generative modeling approach in machine learning. They leverage the concept of diffusion, a process that describes how particles or information spread, to iteratively transform a noisy input into a clean output.

5.3 Denoising Diffusion Probabilistic Models:

DDPM or Diffusion models are a class of generative models that learn the data distribution by simulating a diffusion process. In the context of machine learning, diffusion refers to the gradual transformation of a target output (e.g., an image) into a noisy input through a series of noise-adding steps. The model learns to reverse this process by iteratively denoising the input until the target output is reconstructed.

5.4 The Mathematics of DDPM

The key idea behind DDPM is to model the data distribution P(x) using a diffusion process, where x is a data point (e.g., an image). The diffusion process is modeled as a series of Markov chain steps:

$$x_t = \sqrt{1-\alpha_t} * x_{t-1} + \sqrt{\alpha_t} * \varepsilon_t$$

Here, x_t is the data point at step t, α_t is the noise schedule (a predefined function controlling the amount of noise added), and ε_t is Gaussian noise. The model is trained to learn the denoising function that maps x_t to x_{t-1}:

$x_{t-1} = f(x_t, t; \theta) + N(0, \beta_t * I)$

$f(x_t, t; \theta)$ is a neural network parameterized by θ, which learns to denoise x_t. $N(0, \beta_t * I)$ is Gaussian noise with mean 0 and covariance matrix $\beta_t * I$, where β_t is a scalar and I is the identity matrix.

5.5 Training and Sampling

During training, the DDPM learns to denoise data points across multiple diffusion steps. The model is trained using a supervised approach, minimizing the Mean Squared Error (MSE) between the denoised outputs and the ground truth data points from the previous step:

$Loss(\theta) = E[(x_{t-1} - f(x_t, t; \theta))^2]$

To generate new samples from the learned data distribution, DDPM starts with random noise and iteratively applies the learned denoising function, reconstructing a plausible data point:

$x_0 = \varepsilon_0$
$x_t = f(x_{t-1}, t; \theta) + N(0, \beta_t * I)$

5.6 Audio Diffusion

Denoising Diffusion Probabilistic Models (DDPM) have proven to be an effective generative modeling approach in machine learning. Most diffusion methods rely on convolutional networks known as U-Nets, which reduce data dimensionality and subsequently increase it, efficiently extracting features in the process. Typically, this is achieved using two-dimensional kernels for the image domain.

In the audio domain, researchers have attempted to apply this process to spectrograms, as exemplified by the ReFusion network, which was trained to generate spectrograms. However, this method has its drawbacks, as it often results in a significant loss of audio quality. One advantage of using spectrograms is their smaller size compared to raw audio. By adopting a one-dimensional approach, it becomes possible to effectively generate audio in the sample domain, resulting in higher quality at the expense of increased training time.

6 Project Setup

6.1 System Selection

For my project my goal was to model raw audio data while maintaining a high sample rate. Through the process of this project I got in contact with the research team the Dadabots, that helped advise me through some of the more complex details of this project. They invited me into

the open source development community for Dance Diffusion. Dance Diffusion is a neural network designed by Zack Evans, optimized for generating audio rate samples. Importantly there are several pre-trained models designed for transfer learning, a process where you use the trained models weights and biases as your initialization for training. This enables faster training times meaning you save money, as well as needing significantly less data to get high audio quality. It also has preconfigured experiment tracking software linked into the project through PyTorch Lighting. This system was the driver for the rest of my project.

6.2 Data Collection

I had several goals with collecting data, I wanted a dataset that could give me the opportunity to generate a wide amount of samples at a high quality, and extra data for doing weird sound design experiments. At the end of Collecting my data I had around twenty thousand samples, mostly consisting of drum one shots. To Collect my data I used python, and I will share several tricks to help you get started.

When it comes to data collection in machine learning, there are several tools that can be used to facilitate the process.

- **BeautifulSoup**. This is a popular library in Python for web scraping. It can parse HTML and XML files, and extract useful data from them. With BeautifulSoup, you can easily navigate the structure of a webpage, locate the relevant data you want to extract, and then save it in a format that can be used for training your machine learning models.

- **Scrapy.** Scrapy is a popular web scraping framework for Python that allows you to extract data from websites quickly and efficiently. It provides powerful tools for navigating and parsing HTML, handling cookies and sessions, and storing data in a variety of formats.

- **OS.** This is a built-in library in Python that provides a way to interact with the file system in a platform-independent way. It can be used to create, move, and delete files and directories, as well as perform other operations like listing the contents of a directory.

- **Pathlib.** This is another built-in library in Python that provides an object-oriented interface to the file system. It is similar to os in many ways, but it provides a more intuitive way to work with paths and file names.

- **Glob.** This is a library in Python that provides a way to search for files and directories that match a certain pattern. For example, you could use glob to find all files in a directory that have a certain file extension, or that match a certain naming convention.

- **Shutil.** This is a library in Python that provides a set of high-level operations for working with files and directories. It can be used to copy, move, and delete files, as well as perform other operations like archiving and compressing files.

6.3 Advanced Data Augmentation Techniques

Audio data is a critical component of many machine learning applications, including speech recognition, audio classification, and sound event detection. However, acquiring large amounts of high-quality audio data can be time-consuming and costly. To address this challenge, data augmentation techniques have been developed to increase the size and diversity of audio datasets without the need for additional data collection.

CJ Carr, and Zack Zukowski, who together make up the Dadabots advised me on raw audio dataset augmentation techniques. Here is what I learned.

1. Random Phase Flipping is an easy way to essentially double your data by flipping the phase, which on the number scale flips the numbers over, making negative positive and vice versa. Both samples sound the same, but to the model, these are functionally new samples.

2. Random Pitch Shifting is another extremely effective way to double your data by applying random pitch shifting over a copy of your original dataset. It's important to note that randomness is key. If you just pitch the samples, the network might learn this pitch change and destroy your dataset. Depending on the application, using pitch as an augmentation tool can be more or less effective, so it's best to keep your pitch shift range to a low value.

3. Random Saturation can also effectively double your dataset by applying random amounts of saturation to a copy of your dataset. Make sure to use a low range for saturation amount. I found this technique to be particularly effective on drums.

By mixing and matching these techniques, you can technically multiply your dataset by eight. However, I wouldn't recommend using all of them simultaneously. While these techniques technically generate new data, it still derives from existing samples in the dataset. Collecting more data is always the better option, but when acquiring additional data is difficult or impossible, data augmentation can be a powerful tool.

7 Model Training

7.1 System Hardware Notes

To train models at scale you have to be aware of several things. The first is the importance of GPUs. When you are training a model like dance diffusion you are training over a million parameters. If we stay on the CPU, each of these computations has to happen in sequence, meaning it will take a long time. To speed up training, we do computations in parallel on GPUs, specifically nVidia GPUs that have CUDA support. To get access to a GPU I used Googles Colaboratory, It provides a Jupyter notebook-like interface, is sharable, and it provides access to

a high-performance computing environment, including powerful CPU and GPU resources, and the ability to use pre-installed libraries such as TensorFlow, Keras, PyTorch. An alternative to Colab is Kaggle, a data science community website that provides free GPU access, as well as competitions. If you want even more power than Google Colaboratory, I would recommend AWS SageMaker. Amazon's complete Machine learning Pipeline.

7.2 Hyperparameters:

Understanding Hyperparameters is key to getting the best results out of a machine learning system. Hyperparameters are tunable parameters that define the structure and behavior of a machine learning model. They can have a significant impact on the performance of the model. Here's a list of some common hyperparameters in machine learning:

1. **Learning rate.** The step size taken by the optimization algorithm (e.g., gradient descent) while updating the model's weights. A smaller learning rate leads to slower convergence, while a larger learning rate may cause instability in the learning process.

2. **Batch size.** The number of samples used to update the model's weights in each iteration of the optimization algorithm. Smaller batch sizes can lead to more frequent updates and faster convergence, but may be noisy, while larger batch sizes reduce noise but may require more computation.

3. **Number of epochs.** The number of times the entire dataset is passed through the model during training. Too few epochs can lead to underfitting, while too many epochs can lead to overfitting.

4. **Regularization.** Techniques such as L1 or L2 regularization that apply penalties to the model's weights to prevent overfitting. The strength of the regularization is controlled by a hyperparameter, often denoted as lambda or alpha.

5. **Activation function.** The function used to introduce non-linearity in the model, such as ReLU, sigmoid, or tanh. Different activation functions can affect the model's ability to learn complex patterns.

6. **Number of layers and units per layer.** In neural networks, these hyperparameters determine the architecture of the model. Deeper networks with more layers can learn more complex features but may be harder to train and more prone to overfitting.

7. **Dropout rate.** In neural networks, the probability of temporarily dropping out (i.e., not updating) a unit during training. Dropout is a regularization technique that helps prevent overfitting.

8. **Weight initialization.** The method used to set the initial values of the model's weights, such as random initialization or pre-trained weights. Proper weight initialization can help the optimization algorithm converge faster.
9. **Momentum.** In optimization algorithms like stochastic gradient descent (SGD), momentum is a hyperparameter that helps the optimizer navigate through local minima and converge faster.

10. **Optimizer.** The optimization algorithm is used to update the model's weights, such as gradient descent, SGD, Adam, or RMSprop. Different optimizers have their own hyperparameters and can impact the speed and quality of convergence.

The choice of hyperparameters can significantly impact a model's performance, and it's crucial to carefully select and tune them using techniques like grid search, random search, or Bayesian optimization. For Dance Diffusion focus on the Batch Size, tune it to your GPUs ram.

7.3 Interpreting Results.

Another important tool to have in your tool belt is being able to interpret the results of your training. You want to look at loss and accuracy, making sure that you minimize loss, and maximize accuracy. There are two main concepts to understand if you want to be able to improve the results of your model. Overfitting and Underfitting.

Overfitting occurs when a model learns the training data too well, capturing not only the underlying patterns but also the noise and specific details that are not relevant to the overall trend. As a result, the model performs exceptionally well on the training data but poorly on new, unseen data.

This usually happens when the model is too complex, has too many parameters, or is trained for too long. In overfitting, the model becomes sensitive to minor fluctuations in the training data, which leads to poor generalization.

To prevent overfitting, you can:

- Simplify the model (by reducing its complexity or the number of parameters)
- Obtain more training data
- Implement regularization techniques (like L1 or L2 regularization)
- Use techniques like cross-validation to fine-tune model hyperparameters

Underfitting occurs when a model fails to learn the underlying patterns in the training data, resulting in poor performance on both the training data and unseen data. This usually happens when the model is too simple, has too few parameters, or is not trained for long enough.

In underfitting, the model is unable to capture the complexity of the data, and its predictions tend to be inaccurate. This can lead to low accuracy and poor generalization.

To overcome underfitting, you can:

- increase the complexity of the model (by adding more layers, nodes, or features)
- Train the model for a longer time
- Add more features to the input data
- Adjust model hyperparameters

The goal in machine learning is to find a balance between overfitting and underfitting, which results in a model that performs well on both the training data and new, unseen data. This is typically referred to as achieving a good bias-variance tradeoff.

8 Research findings

8.1 High-quality audio synthesis with Dance Diffusion

In my experiments, I observed that Dance Diffusion effectively achieved high-quality audio synthesis regardless of the dataset size. This can be attributed to the transfer training capabilities available with the system. Transfer training allows the model to utilize pre-trained weights and fine-tune them for the specific task, thus enabling high-quality audio generation even with limited datasets.

8.2 Generalization and large datasets

While Dance Diffusion provided satisfactory results in terms of audio quality, I found that obtaining a model capable of generalizing and generating sounds outside of the distribution required significantly larger datasets. Larger datasets provide a broader range of sounds, allowing the model to learn more diverse patterns and consequently generate novel sounds beyond the scope of the training data.

8.3 Creative potential of undertraining diffusion models

My research also explored the creative potential of undertraining diffusion models. I discovered that by intentionally undertraining the models, you could encourage the system to make mistakes and accidentally combine sounds, leading to unique and interesting outputs. This approach can be particularly valuable for artists and sound designers looking to generate unconventional and innovative audio.

8.4 Drum sample generation quality

Through the use of drum samples, I was able to generate audio at a 44.1 kHz sampling rate, exhibiting good dynamic range. These findings demonstrate the potential of Dance Diffusion in creating high-quality drum sounds suitable for various music production applications.

8.5 Glitch samples and bass growls

Working with sounds that do not contain direct transients, I was able to generate intriguing glitch samples and bass growls. These results highlight the versatility of Dance Diffusion in creating a wide range of sounds, including unconventional samples that can be used to enrich electronic music compositions and other creative projects.

In summary, my findings reveal the capabilities of Dance Diffusion in achieving high-quality audio synthesis, the importance of large datasets for generalization, and the creative potential of undertraining diffusion models in generating unique sound designs.

9 Conclusion

In conclusion, this research paper has demonstrated the potential and versatility of Dance Diffusion as an audio synthesis system. Additionally, the exploration of undertraining diffusion models has revealed the potential for creative applications by intentionally encouraging the system to generate unconventional and innovative audio outputs. These findings not only offer valuable insights into the world of audio synthesis, but also provide artists and sound designers with new approaches for creating diverse and engaging soundscapes. As the field of neural synthesis continues to evolve, the possibilities for creative exploration and innovation in audio generation will only expand, making this an exciting frontier for future research and artistic endeavors.