TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

Music Generation Using Deep Learning

By:
Arju Bindukar (073/BEX/408)
Bikash Timsina (073/BEX/410)
Keeran Dhakal (073/BEX/415)
Pradeep BC (073/BEX/424)

A PROJECT WAS SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE BACHELOR'S DEGREE IN ELECTRONICS AND COMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
LALITPUR, NEPAL

April, 2021

APPROVAL LETTER

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled **"Music Generation Using Deep Learning"** submitted by Arju Bindukar, Bikash Timsina, Keeran Dhakal and Pradeep BC in partial fulfillment of the requirements for the Bachelor's Degree in Electronics and Communication Engineering.

Supervisor, Suman Sharma, Lecturer
Department of Electronics and Computer Engineering
Institute of Engineering, Pulchowk Campus

Internal Examiner, Anand Kumar Sah, Lecturer
Deputy Head of Department, Electronics and Computer Engineering
Institute of Engineering, Pulchowk Campus

External Examiner, Krishna Ram Dhunju, Computer Engineer
Nepal Rastra Bank

Head of Department, Ram Krishna Maharjan, Professor
Institute of Engineering, Pulchowk Campus

DATE OF APPROVAL:

COPYRIGHT

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to the Department of Electronics and Computer Engineering, IOE Pulchowk Campus and our supervisor Suman Sharma who opted to supervise us in this wonderful project on the topic Music Generation using Deep Learning, which also helped us in doing a lot of study and we came to know about so many new things.

Secondly, we would also like to thank our friends who helped us a lot in finalizing this project within the time frame.

Arju Bindukar(073/BEX/408)
Bikash Timsina(073/BEX/410)
Keeran Dhakal(073/BEX/415)
Pradeep BC(073/BEX/424)

ABSTRACT

Music generation is a very creative problem that tests creative capacity of a composer, whether it a human or a computer. Almost all of music is some alteration of a sonic idea created before. Music contains melodies and harmonies that come in some fixed patterns. With enough data and the correct algorithm, deep learning algorithms should be able to learn these patterns and make music that would sound similar to that composed by human beings. There are various architetures that can be used to solve this problem. LSTM(Long Short-Term Memory) can be used to learn more complex and long-term dependencies. Autoencoders and its variants can be used to extract essential features of music and generate new music by decoding the feature set and adding variations to it. A GAN(Generative Adverserial Network) uses an adversarial feedback loop to learn to generate some information that seems real. This project experiments with these various architectures for generating music.

Keywords: Music, deep learning, LSTM, Autoencoder, GAN

TABLE OF CONTENTS

List of Figures

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
AIVA	Artifical Intelligence Virtual Artist
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network
AE	Auteencoder
DAE	Deep Autoencoder
VAE	Variational Autoencoder
GAN	Generative Adversarial Network
PCA	Principal Component Analysis
MIDI	Musical Instrument Digital Interface
AMT	Automatic Music Transcription
PPQN	Pulses Per Quarter Note
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
RMS	Root Mean Square
GUI	Graphical User Interface
ORM	Object Reational Mapper
MVC	Model View Controller
MVT	Model View Template
MVP	Minimum Viable Product

1. INTRODUCTION

1.1. Background

Music is a sequence of musical notes that is pleasant to human ears. Music is one of the necessities of human life. Composition of a musical piece is one of the highest demonstrations of human creativity. This poses an interesting problem for artificial intelligence. There has been a great advance in the field of generating musical compositions thanks to the use of Deep Learning technique.

Deep learning is machine learning techniques based on Artificial Neural Network (ANN). Multiple layers are processing multiple hierarchical levels of abstraction, which are automatically extracted from data. It has become the fastest-growing domain. It is widely used nowadays for classification and prediction tasks in fields like image recognition, voice recognition or translation. A traditional system is severely limited in the sense that it does not generalize very well and can't handle unstructured data.

The success of deep learning is due to the availability of massive data, and the availability of efficient and easily affordable computational power. Growing application of deep learning is the generation of content like art, text, and field with which we are concerned, the music.

1.2. Motivation

The motivation is in using the capacity of deep learning architectures and training techniques to automatically learn musical styles from arbitrary musical corpus and then to generate samples from the estimated distribution. Deep learning is the choice for music generation because it is general. As opposed to handcrafted models, such as grammar-based or rule-based music generation systems, a machine learning-based generation system can be agnostic, as it learns a model from an arbitrary corpus of music. As a result, the same system may be used for various musical genres. Therefore, as more large scale musical data sets are made available, a machine learning-based generation system will be able to automatically learn a musical style from a corpus and to generate new musical content. Deep learning techniques can make creation feasible when the desired application is too complex to be described by analytical formulations or manual brute force design, and learning algorithms are often less brittle than manually designed rule sets and learned rules are more likely to generalize accurately to new contexts in which inputs may change. Moreover, as opposed to structured rep-

resentations like rules and grammars, deep learning is good at processing raw unstructured data, from which its hierarchy of layers will extract higher level representations adapted to the task.

1.3. Objectives

Our primary objective is to develop a system that is capable to learn different kinds of music styles from a data set and generate new music. To study some of the deep learning models, experiment with them, compare them and choose the best among them.

1.3.1. Project Objectives

1. To compose polyphonic music in two modes:
    - Autonomous mode: mode without human intervention.
    - Interactive or assisted mode: Mode with some control interface for the human user to have some interactive control over the process of generation.
2. To train the model with popular styles of music.

1.3.2. Academic Objectives

1. To be familiar with Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL).
2. To learn about different tools, technologies in the field of AI, ML and, DL.

1.4. Problem statement

There are various kinds of music circulating, yet composers are still trying to create new music and listeners are searching to listen to new music constantly. This also applies to the content creators who are overusing the same music for their contents and videos. Some limited numbers of royalty free music are available online and they are the ones that are mostly used by the content creators. One of the ways to solve this is to create an automated music composer using a computer algorithm. The problem is to create a model and learn musical styles, and then generate new musical content. This is challenging to model because it requires the function to be able to recall past information to project in the future.

1.5. Scope of Project

Music composition using deep learning is an interesting way to understand human creativity and the way it works. Composers are constantly creating new music and listeners are trying to listen to new music. This problem can be solved by using a computer algorithm to generate new music.

1.6. Software Requirements

Python

Python is the widely used high-level programming language that is stable, flexible, and provides various tools to developers, and is one of those languages that is both powerful and is quite simple to use. It has a large and comprehensive standard library. All these properties of Python make it the first choice for doing any project related to AI.

NumPy

NumPy is a scientific linear algebra library for the Python programming language, with support for large, multi-dimensional arrays and matrices, along with an enormous collection of high-level mathematical functions to run on these arrays. This library is used in Keras, and the training and testing data have to be NumPy arrays before feeding them to any network. Moreover, we use NumPy in the encoding and decoding part to deal with matrices (e.g. adding a vector to multiple rows in a matrix).

Tensorflow

Tensorflow is an open source library for machine learning. It is a symbolic math library that uses dataflow and differentiable programming to perform various tasks focused on training and inference of deep neural networks. It allows developers to use its tools and resources to develop various machine learning applications. It was developed by the Google Brain team.

Tensorflow enables you to build dataflow graphs and structures to define how data moves through a graph by taking inputs as a multi-dimensional array called tensor. It allows to construct a flowchart of operations that can be performed on these inputs, which enters the network at one end and comes at the other end as output.

Keras

Keras is a high-level library for machine learning with Neural Network that allows us to prototype quickly and change parameters without rewriting a lot of code. It runs on top of tensorflow with tensorflow being the backend. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout and batch normalization.

Mido

Mido stands for Midi Objects for python. Mido is a library in python language that deals with the midi files. It is used to extract the details that is contained in a midi file. Mido makes it easy and straight forward to read from midi files and also write to the midi files. This library is used in preprocessing of datasets and postprocessing of the generated midi data to convert it into the midi file.

Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows us to create and share documents that contain live code, equations, visualizations and narrative text. We use Jupyter Notebook in experimentation to be able to execute blocks of code in any sequence we choose. This helps to rapidly prototype experiments and Neural Network models.

Django

Django is a high-level,fully featured server side Python Web framework that encourages rapid development and clean, pragmatic design. It consists of an object-relational mapper (ORM) that facilitates between relational database (”Model”), http requests handler (”View”) that also does the work of dispatching regular-expression based urls and web templating system(”Template”). MVT is a special case of Model-View-Controller (MVC) where Djnago itself takes care of the controller part leaving us with the templates. Model here indicates for relational database model and not AI model but we do not have database(model) in this project. Template is a presentation layer for handling user interfaces. View executes the

business logic to render the template.

JavaScript

JavaScript, the programming language of the Web, is used to add a variety of dynamic functions to web pages. It allows us to implement complex features on frontend of web pages such as creating dynamic and interactive experience for the user.

HTML/CSS

HTML(Hypertext Markup Language) and CSS(Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the structure of the page, CSS, the visual and aural layout, for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications.

JSON

JSON stands for JavaScript Object Notation.It is a light-weight text data interchange fomat which is completely language independent.It is based on two structures;a collection of name-value pairs and ordered list of items

AJAX

AJAX stands for Asynchronous JavaScript And XML. It allows web pages to be updated asynchronously by exchanging data with a server behind the scenes. It signifies that it is possible to update parts of a web page, without reloading the whole page. In this project, jQuery Ajax is implemented as it is compatible with several browsers.

Tone

Tone.js is a web audio framework for creating interactive music in the browser. It provides high-performance building blocks to create our own synthesizers, effects, and complex control signals.

1.7. Target Market

Audiences for our system are the music hobbyist, artists, content creators. Anyone can quickly come up with unique, interesting and beautiful songs in cost effective way. We also want to target our system to the youtuber. We have provided a deep learning-based solution to quickly generate music from different genre.

1.8. Competitors

AI has hailed as necessay technology even for music industry. They are used in everything from the production of songs and albums to how we listen to our tunes on streaming platforms. Some of them are:

- AIVA(Artificial Intelligence Virtual Artist): It is an electronic composer specialized in classical and symphonic music composition. It became the world's first virtual composer to be recognized by music society. AIVA uses deep learning and reinforcement architectures. Rather than replacing musicians, AIVA is working to enhance the collaboration between AI and human.
- DeepBach: It aims to generate Bach like chorale pieces music by employing pseudo-Gibbs sampling.
- Magenta research project: It is an opensource research project exploring the role of machine learning as a tool in the creative process. It is powered by tensorflow. One of the features project is musicVAE which uses generative capabilities of variational autoencoder to generate music.
- WaveNet: It is a DeepMind's a deep generative model of raw audio waveforms. They are able to generate speech which mimics human voice and which sound more natural than the best existing Text-to-Speech system. Same network are used to synthesize music.

  There are many other companies like Amper Music, Ercett Music, Humtap, computoser which are commercially generative music at a pretty high cost.

2. LITERATURE REVIEW

This section presents the research work of some prominent authors in the same fields. The researches conducted previously are:

Briot et al. [1] has done a detailed survey and an analysis of different ways of using deep learning (deep artificial neural networks) to generate musical content. It proposes a conceptual framework and typology aimed at a better understanding of the design decisions for current as well as future systems.

Goodfellow et al. [5] The Deep Learning textbook is a resource intended to help students and practitioners enter the field of machine learning in general and deep learning in particular.

Kang et al. [8] explores various ways from simple Naive Bayes algorithm to neural network models like LSTM RNN and encoder decoder LSTM. They found that Naive Bayes and encoder decoder LSTM is prone to over-fitting.

Guo et al. [7] presents a controllable music generation system by the use of variational autoencoder. Many of the music generation system is fully autonomous and do not offer control over generation process. This paper tries to solve that problem.

Roche et al. [10] investigates the use of non-linear unsupervised dimensionality reduction techniques to compress a music data set into a low-dimensional representation which can be used in turn for the synthesis of new sounds. They systematically compare (shallow) autoencoders(AEs), deep autoencoders(DAEs), recurrent autoencoders (with Long Short-Term Memory cells – LSTM-AEs) and variational autoencoders(VAEs) with principal component analysis(PCA) for representing the music data sets in lower dimensional vector.

Weel et al. [11] 's thesis used a Long Short-Term Memory(LSTM) network and, through prediction of what musical pitches are most probable to follow a segment of input music, generated new music. The network trained on a data set of 74 Led Zeppelin songs in MIDI format. All MIDI files were converted into 2-dimensional arrays which mapped to musical pitch and MIDI tick.

Ycart et al. [12] investigates the predictive power of simple LSTM networks for polyphonic MIDI sequences, using an empirical approach. Such systems can then be used as music language model which, combined with an acoustic model, can improve automatic music transcription(AMT) performance.

Goodfellow et al. [6] presents the concept of Generative Adversarial Network in his paper in 2014 which generates new data based on the real data distribution.

Yu and Canales [13] investigates the conditional LSTM GAN which generates melody from lyrics. The midi data is preprocessed with the triplets of music attributes which are midi number, note duration and rest duration.

Dataset [2] is the collection of classical music dataset in piano.

Lee et al. [9] has worked on polyphony music generation using GAN architecture. In this architecture, generator has been composed of RNN while discriminator of CNN. Chords are accumulated for preprocessing of data.

Dong et al. [4] presents a new convolutional GAN model for generating binary-valued multi-track sequences and categorize all MIDI tracks into five instrument families: bass, drum, guitar, piano and strings and merge the tracks within each instrument family.

de Almeida Mousaco Pinho [3] presents the idea of generating music with frequency analyis with the use of DCGAN.

3. THEORETICAL BACKGROUND

3.1. Music Theory

Music is essentially composed of Notes and Chords. We choose to understand this with perspective of the piano instrument because we deal with piano instrument during music generation. Moreover same theory applies for almost all instruments:

- Note: The sound produced by a single key is called a note. There are seven notes which are A, B, C, D, E, F, G.
- Chords: The sound produced by 2 or more keys simultaneously is called a chord. Generally, most chords contain at least 3 key sounds. For example, the A Major chord is composed of an A, a C sharp and an E note.
- Octave: A repeated pattern is called an octave. Each octave contains 7 white and 5 black keys. Each set of notes (A to G) belongs to an octave. Octaves are used to designate which pitch range the notes are in. For example, A1 (A in octave 1) will be much deeper and lower than A7 (A in octave 7) which is higher pitch and sharper sounding.
- Measure: Measure is a segment of time within a piece of music defined by a given number of beats. In our data sets 4 beats make one measure.
- Motif: A short musical idea, melodic, harmonic, rhythmic, or any combination of these three. A motif may be of any size, and is most commonly regarded as the shortest subdivision of a theme or phrase that still maintains its identity as an idea.
- Transposition: It is the process of shifting a melody, a harmonic progression or an entire musical piece to another key, while maintaining the same tone structure.

3.1.1. MIDI Standard

MIDI (Musical Instrument Digital Interface) is a technical standard that describes a communications protocol, digital interface, and electrical connectors that connect a wide variety of electronic musical instruments, computers, and related audio devices for playing, editing and recording music. A single MIDI link through a MIDI cable can carry up to sixteen channels of information, each of which can be routed to a separate device or instrument. This could

be sixteen different digital instruments, for example. MIDI carries event messages; data that specify the instructions for music, including a note's notation, pitch, velocity. In this project we are concerned with MIDI file format as we have to extract data from midi format songs. As an example, let us consider a digital piano, where a musician pushes down a key of the piano keyboard to start a sound. The intensity of the sound is controlled by the velocity of the keystroke. Releasing the key stops the sound. Instead of physically pushing and releasing the piano key, the musician may also trigger the instrument to produce the same sound by transmitting suitable MIDI messages, which encode the note-on, the velocity, the note-off, and other information. These MIDI messages may be automatically generated by some other electronic instrument or may be provided by a computer. It is an important fact that MIDI does not represent musical sound directly, but only represents performance information encoding the instructions about how an instrument has been played or how music is to be produced. A file format that stores and exchanges the data is also defined.

Advantages of MIDI include small file size, ease of modification and manipulation and a wide choice of electronic instruments and synthesizer or digitally-sampled sounds. A MIDI recording of a performance on a keyboard could sound like a piano or other keyboard instrument; however, since MIDI records the messages and information about their notes and not the specific sounds, this recording could be changed to many other sounds, ranging from synthesized or sampled guitar or flute to full orchestra. A MIDI recording is not an audio signal, as with a sound recording made with a microphone.

3.2. Deep learning Architectures

There are many deep learning architectures that can be implemented for music generation purposes. However, we only dealt with these deep learning architectures for this project.

3.2.1. Dense Neural Networks

Dense Neural Network commonly known as Fully Connected Networks, is a variant of the traditional neural networks generally with more hidden layers in it. These are relatively simpler compared to other deep learning architectures. Each layers are connected through the activation functions. Apart from their independent uses in deep networks, these can also be stacked through other architectures such as CNN, LSTM, etc.

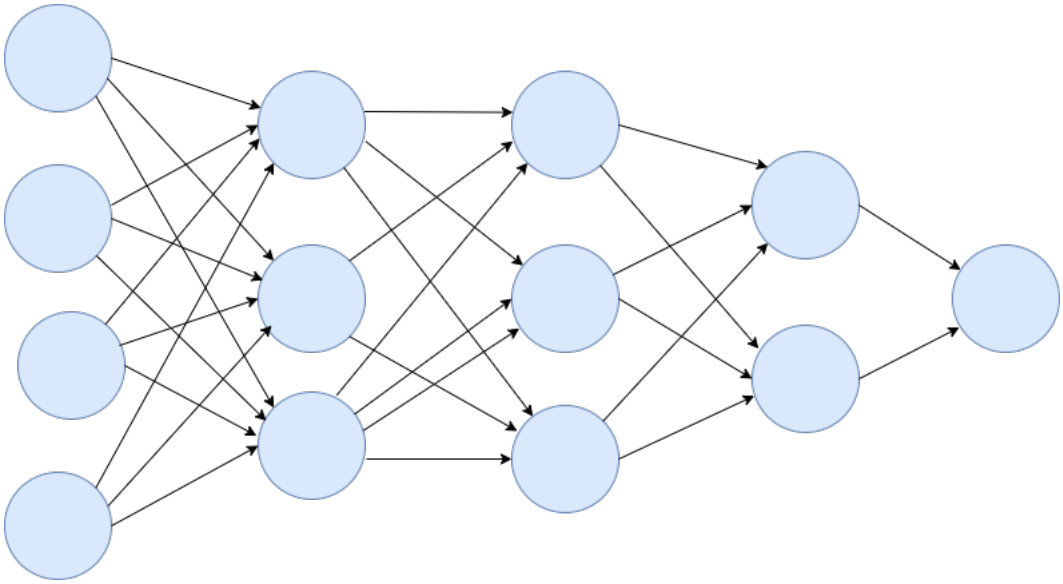


Figure 3.1: Dense Neural Network

3.2.2. LSTM (Long-Short Term Memory)

LSTM is a variant of Recurrent Neural Network to handle a sequence problem. Its node has a feedback connection unlike feed-forward neural network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Vanishing gradient problem causes earlier layers in deep layer insensitive to loss. That is a hindrance for learning.

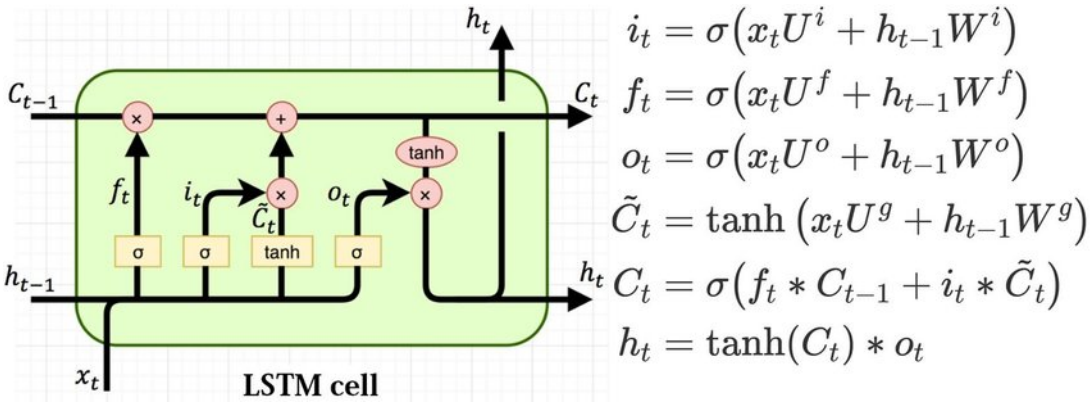


Figure 3.2: LSTM cell

3.2.3. Seq2Seq LSTM

Seq2seq is a family of machine learning approaches used for language processing. Applications include language translation, image captioning, conversational models and text summarizing. It is also used to generate music, as music generation is also a sequence to sequence problem. Seq2seq turns one sequence into another sequence. It does so by use of a recurrent neural network (RNN) or more often LSTM to avoid the problem of vanishing gradient. The context for each item is the output from the previous step. The primary components are one encoder and one decoder network. The encoder turns each item into a corresponding hidden vector containing the item and its context. The decoder reverses the process, turning the vector into an output item, using the previous output as the input context.

3.2.4. Autoencoder

Autoencoder takes a high dimensional input vector through encoder to give lower representation latent vector and then passes through decoder to reconstruct an input from that lower representation. Autoencoder encoder/decoder combination of architecture may have some generative characteristics. They are a self-supervised technique for representation learning, where our network learns about its input so that it may generate new data just as input. Autoencoder is non linear technique of dimensionality reduction. The desirable properties of a latent space can be summarized as follows:

- Expression: Any real example can be mapped to some point in the latent space and reconstructed from it.
- Realism: Any point in this space represents some realistic example, including ones not in the training set.
- Smoothness: Examples from nearby points in latent space have similar qualities to one another.

Although autoencoder itself doesn't fulfill all these properties, it's variant variational autoencoder can learn above properties.

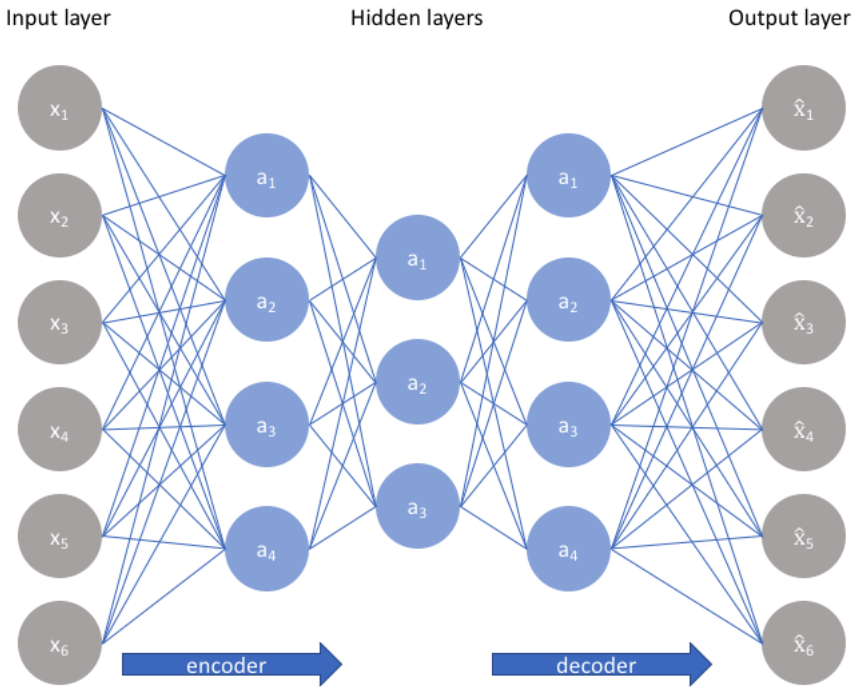


Figure 3.3: Autoencoder architecture

Mathematical representation:

Function p is encoder and function q is decoder. Let h denote latent space representation. Objective of autoencoder is to minimize the mean squared error between reconstructed output (x') and input (x). Mathematically,

$h = p(x)$,

$x' = q(h)$,

Therefore, $x' = q(p(x))$

Objective: $minimize(mean(|x - x'|^2))$

3.2.5. Variational Autoencoder

It's a type of autoencoder with added constraints on the encoded representations being learned. Variational autoencoder is different from autoencoder in a way such that it provides a statistic manner for describing the samples of the data set in latent space. Therefore, in variational autoencoder, the encoder outputs a probability distribution in the bottleneck layer instead of a single output value. VAE is a generative model because sampling a point

from a distribution, we can generate new input data samples.

Variational autoencoder uses KL-divergence as its loss function, the goal of this is to minimize the difference between a supposed distribution i.e. Gaussian distribution and original distribution of data set.

The encoder outputs parameters to $q(z|x)$, which is a Gaussian probability density. We can sample from this distribution to get noisy values of the representations z. Because it is a Gaussian Distribution it contains two parameters in a latent space which is μ and σ. We randomly sample similar points z from latent normal distribution that is assumed to generate data.

$z = \mu + \sigma * \varepsilon,$

where ε is a random standard normal tensor used for reparameterization trick. The decoder is another neural network. Its input is the representation z, it outputs the parameters to the probability distribution of the data, and has weights and biases. The decoder is denoted by $p(z|x)$. This simply maps these latent space points back to original input data. By constructing our encoder model to output a range of possible values (a statistical distribution) from which we'll randomly sample to feed into our decoder model, we're essentially enforcing a continuous, smooth latent space representation. For any sampling of the latent distributions, we're expecting our decoder model to be able to accurately reconstruct the input. Thus, values which are nearby to one another in latent space should correspond with very similar reconstructions. Unlike autoencoder, variational autoencoder doesn't output a single value to describe each latent space attribute rather formulates a probability distribution for each latent attribute.

Mathematical representation:

Variational autoencoder uses KL-divergence as its loss function, the goal of this is to minimize the difference between a supposed distribution and original distribution of data set.

Suppose we have a distribution z and we want to generate the observation x from it. In other words, we want to calculate

$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$

But, the calculation of p(x) can be quite difficult

$p(x) = \int p(x|z)\, p(z)\, dz$

This usually makes it an intractable distribution. Hence, we need to approximate $p(z|x)$ to $q(z|x)$ to make it a tractable distribution. For this we minimize the KL-divergence loss which calculates how similar two distributions are:

$$\min KL(q(z|x)\,||p(z|x))$$

By simplifying, the above minimization problem is equivalent to the following maximization problem :

$$E_{q(z|x)} \log p(x|z) - KL(q(z|x)\,||p(z))$$

The first term represents the reconstruction likelihood and the other term ensures that our learned distribution q is similar to the true prior distribution p.

Thus our total loss consists of two terms, one is reconstruction error and other is KL-divergence loss:

$$Loss = L(x,\hat{x}) + \sum_j KL\left(q_j(z|x)\,||p(z)\right)$$

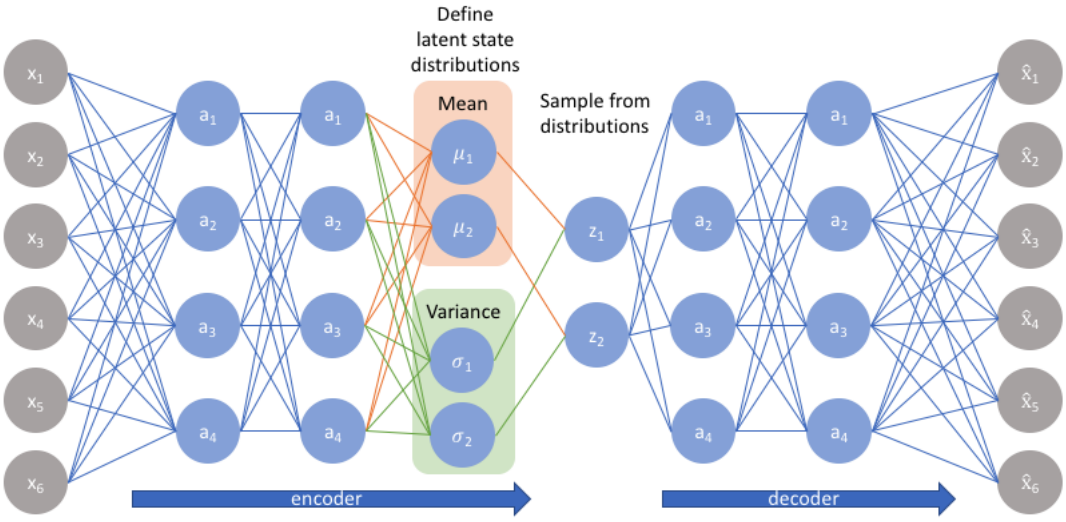


Figure 3.4: Variational autoencoder architecture

3.2.6. Generative Adversarial Network

Generative Adversarial Network commonly known as GAN, is a deep learning framework developed by Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, et al. in 2014. GANs use an adversarial process of learning between two models for generative model estimation. It employs the use of two models namely Generative model which is commonly called as Generator and Discriminative model commonly called as Discriminator. The generator network is used for the generation of the samples based on latent space vectors while the discriminator network discriminates between the samples of original data or generated data. These two networks are trained with an adversarial process which means, one's loss while training is another's gain. The training process of the generator network includes the maximization of the probability of the discriminator misclassifying the data, while the training process of discriminator includes the maximization of its ability to classify the generated data from the original or training data. This can be commonly interpreted as a minimax game between two players.

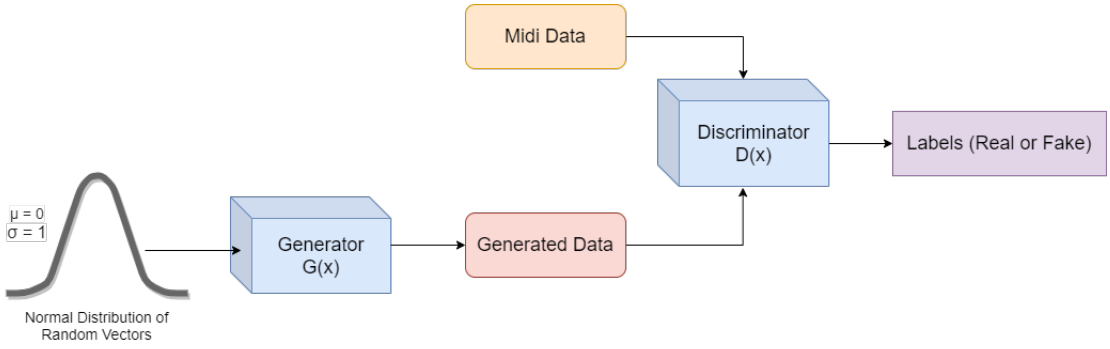


Figure 3.5: Block Diagram of GAN

The training process of the GAN can be illustrated by an example of a practical problem, the interaction between a counterfeiter and a bank. The bank classifies the counterfeit money from the authentic money by examining various features on them. Meanwhile, the counterfeiter gets feedback on the classification of the features based on which the bank is identifying the counterfeit money and hence tries to lessen the differences in the features of the authentic and counterfeit money so that the bank has hard time figuring out the counterfeit money from the authentic ones. This can be seen as a competition between the counterfeiter and the bank. If the counterfeiter is competitive enough, he will learn all the features of the authentic money and use them in his counterfeit money to fool the banking system and thus will end up making an indistinguishable counterfeit. This competition between the counterfeiter and the bank can be seen as the competition of generator and discriminator networks.

To explain about the GAN in detail, we use the work of Goodfellow, Pouget-Abadie, Mirza, et al., the notations of the GAN will be used as it was used in the original paper.

- x represents the real data from P_{data} distribution
- z represents the random vector sampled from P_z distribution
- G represents the generator network
- D represents the discriminator network

To learn the generator's distribution P_g over data x, input noise variables $P_z(z)$ is defined and a mapping of the noise variables to data space is represented as $G(z)$ mapped with network parameters. In correspondence to the above mentioned notations, we can safely assume that $G(z)$ is the generated data output from the generator network where $D(x)$ is the probability that x came from the original data distribution P_{data} rather than the generated data distribution P_g.

To train the network, a loss function for both generator and discriminator has to be defined. Intuitively, the discriminator's loss function is defined on the basis of how often it misclassifies the real data structure as fake and how often it gets fooled by the generator. This is done by comparing the real data distribution output i.e. $D(x)$ to 1 in addition to the generated data distribution output i.e. $D(G(z))$ to 0. The generator's loss is then evaluated on the basis of how often does it generate the data that is not realistic and can't fool the discriminator. This is done by comparing the generated distribution output i.e. $D(G(z))$ to 1. Here, D and G play the two-player minimax game with the value function $V(G,D)$:

$$\min G \max D \quad V(D,\ G) = E_{x \sim Pdata}\ [\log D(x)] + E_{z\ \sim Pz}\ [\log(1 - D(G(z)))]$$

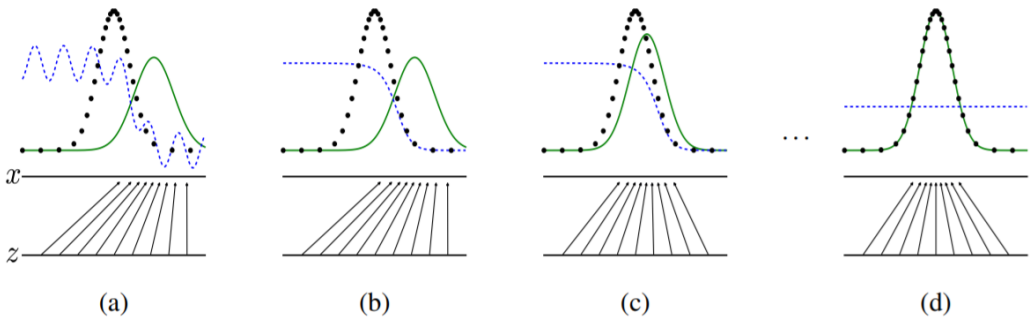


Figure 3.6: GAN with minimax training algorithm

The generated data distribution $(green,\ solidline)$ and true data distribution $(black,\ dottedline)$ forms a pair that the GANs are trained to distinguish between by simultaneously updating the discriminative distribution $(blue,\ dashedline)$. The lower horizontal line is the domain from which the z is sampled, uniformly in this case and the upper horizontal line is the domain of x in which the z is to be mapped. The upward directing arrows shows the mapping of the z in the x domain by $x = G(z)$ which is a non-uniform distribution P_g on transformed samples. G contracts in the region of the high density and expands in the region of low density of P_g.

In figure a, the adversarial pair are in near convergence which means P_g is similar to P_{data} and D is a partially accurate classifier. In figure b, the discriminator is trained to discriminate samples from data, converging to $D*(x) = \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}$. In figure c, it is shown that after an update to G, a gradient of G has guided $G(z)$ to flow to regions that are more likely to be classified as data. In figure d, it can be seen that after certain training steps, if G and D have enough capacity, they will reach an equilibrium point from which they can't improve as $P_g = P_{data}$ and hence the discriminate cannot differentiate between the two. $i.e\ D(x) = \frac{1}{2}$.

3.3. Activation Functions

3.3.1. Linear Activation Function

It has a linear graph, which means it just maps the coordinates into the same value as they are. It is basically a line with slope 1, which means it doesn't confine the output of the functions between any range. It does not help with the complexity or various parameters of usual data that is fed to the neural networks.

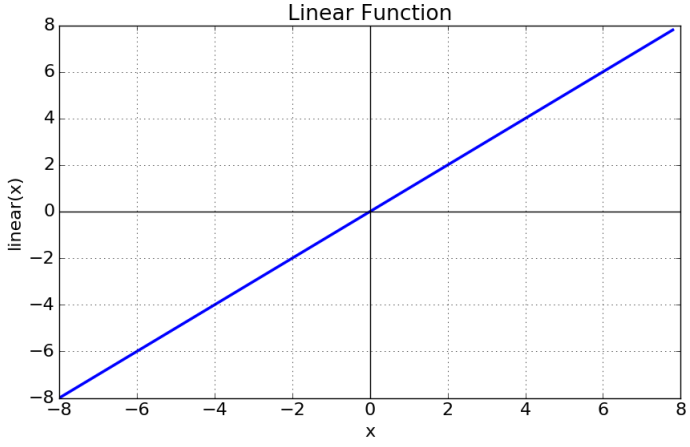


Figure 3.7: Linear Activation Function

3.3.2. Sigmoid Activation Function

Sigmoid activation function is the function which maps the output of the functions between the range of 0 and 1. The sigmoid function curve is an S-shaped curve. As the function maps values between the range of 0 and 1, it is mostly used for models where probability is calculated as an output. The function is differentiable and the slope can be found at every point of the curve. The activation function is monotonic but its derivative is not. Softmax function is the more generalized logistic activation function which is used for multiclass classification.

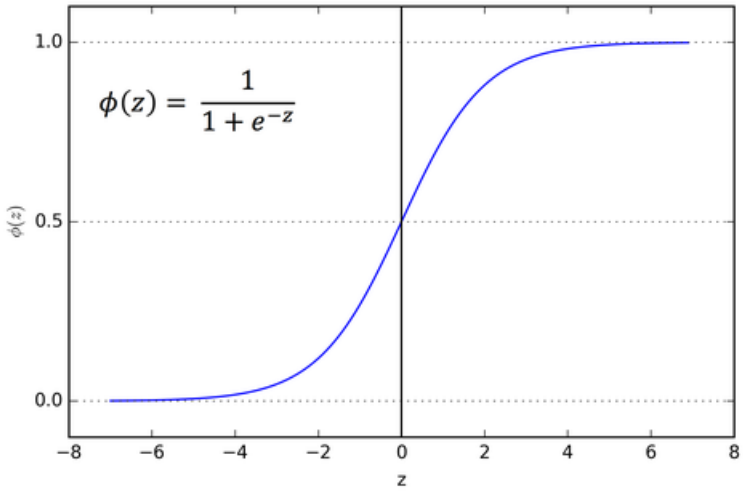


Figure 3.8: Sigmoid Activation Function

3.3.3. Rectified Linear Unit (ReLU)

The ReLU function is probably the most common activation function used in neural networks. It is used in almost all deep neural networks. ReLU functions acts as a linear function for the values greater than zero which means the value greater than zero are mapped into the same value, while it acts as a barrier for the negative values mapping it to zero.

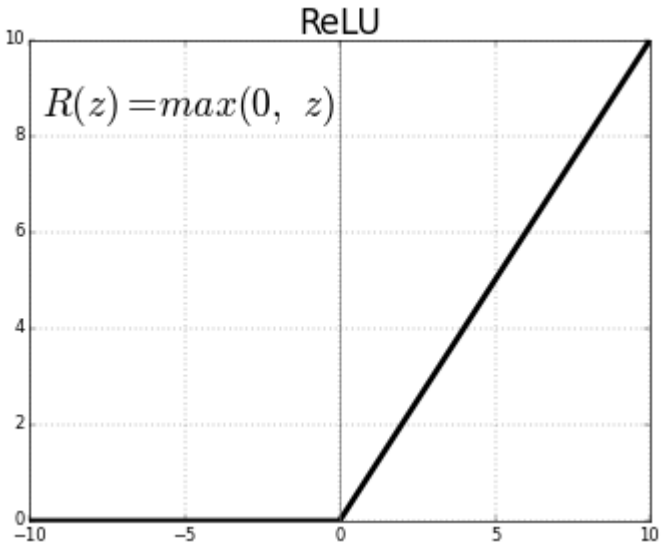


Figure 3.9: ReLU Activation Function

The issue with ReLU activation function is that all the negative values become zero immediately which decreases the ability of the network to fit or train properly on available training sets. The attempt to solve this issue is done with Leaky ReLU which is basically an extended version of ReLU, which helps to increase the range of ReLU function. The Leaky ReLU function maps the negative numbers into a linear range with a very small slope.

3.4. Optimization Algorithms

The relationship between optimization and deep learning is vividly clear as optimization is the way to get to the optimum weights of the parameters. For a deep learning problem, a loss function is usually defined first. Once the loss function has been defined, an optimization algorithm is used in attempt to minimize the loss. In optimization, a loss function is often referred to as the objective function of the optimization problem. By tradition and convention, most optimization algorithms are concerned with minimization. If it is ever required to maximize an objective function, a simple solution can be implemented which is flipping the sign on the objective function.

3.4.1. Gradient Descent

Gradient Descent is an optimization algorithm that is used to find the parameters of an objective function that minimizes the cost function. It is an iterative process that constantly approaches the optimum values of parameters using gradient of cost function.

$$w_{new} = w_{old} - \alpha * \frac{\partial J}{\partial w}$$

$$b_{new} = b_{old} - \alpha * \frac{\partial J}{\partial b}$$

where, w is the weights of the network, b is the biases of the network, α is the learning rate and J is the cost function.

A cost function, also known as loss function, evaluates the performance of an algorithm. The loss function computes the error for a single training example while the cost function is the average of the loss functions for all the training examples. Thus, these terms are normally used interchangeably.

$$J = \frac{1}{N} \sum_{i=1}^{N} (Y' - Y)$$

where J is the cost function, N is the total number of training examples, Y' is the predicted value of the output and Y is the actual value of the output.

For intuition, a cost function can be thought as a large bowl. A random position on it can be stated as the current value of cost. The bottom of the bowl is minimum of the function. So, for the attainment of minimum value of cost function, a derivative of a cost function is calculated and which is basically a slope of the cost function at the current position. These slopes multiplied by the learning rate are then subtracted from the parameters to get new value of the parameters. Learning rate is the size of steps taken to reach minima. This process will go on until the optimum value of parameters are reached where cost function is minimum and the slope of cost function is zero.

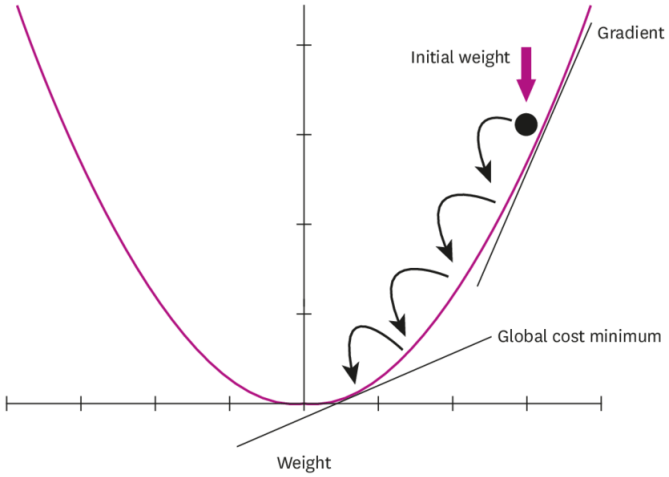


Figure 3.10: Gradient Descent

Batch Gradient Descent allows to train on batches and update parameters after training on the batch. Stochastic Gradient Descent updates the parameters after every iteration of training data.

3.4.2. Adaptive Moment Estimation (Adam)

A good choice of optimization algorithm can train a network effectively and a lot faster. So, a wise choice of optimization algorithm can save training time by a wide margin.

The Adam optimization algorithm is an extension to the stochastic gradient descent algorithm and has recently seen its broader use in deep learning applications. Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper titled “Adam: A Method for Stochastic Optimization”.

Adam is different to stochastic gradient descent in the sense that stochastic gradient descent maintains a single learning rate for all weight updates and the learning remains same throughout the training. Adam combines the advantages that is brought by the two other extensions of stochastic gradient descent which are Adaptive Gradient algorithm and Root Mean Square Propagation (RMSProp). Adaptive Gradient Algorithm maintains a per-parameter learning rate that improves performance on problems with sparse gradients. Root Mean Square Propagation (RMSProp) also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight.

4. METHODOLOGY

4.1. System Block Diagram

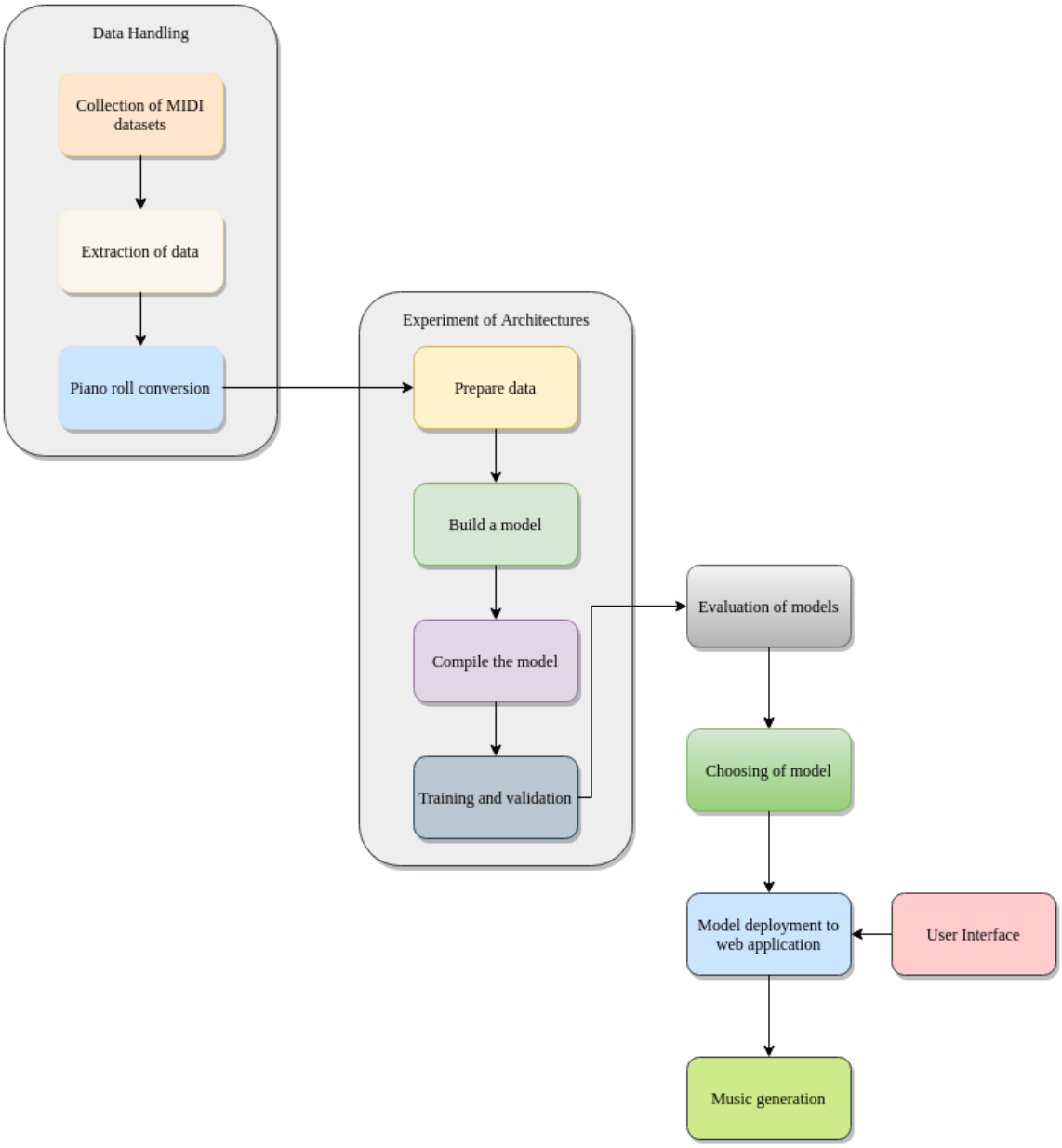


Figure 4.1: System Block Diagram

4.2. Description of working principle

Collection of MIDI Datasets

MIDI datasets were collected from the internet. We have used free folk music dataset. It consists of about 500 MIDI music.

Similarly, we have collected dataset of classical music available on the internet which contains more than 300 classical MIDI music played on piano.

Data Format

One of the key challenges while modeling music is selection of data representation. Possible representation is signal, transformed signal, MIDI, text, etc. Generally musical content for computer is first represented as an audio signal. It can be raw audio (waveform), or an audio spectrum processed as a Fourier transform. As the end destination of generated music content is computer, MIDI format can be one of the best ways of representation.

Extraction

Music is a very complex art form and includes dimensions of pitch, rhythm, tempo, dynamics, articulation, and others. To simplify music for the purpose of this project, only pitch and duration will be considered. We need to extract all the notes of the music from an instrument. Many MIDI files have multiple instruments in their music. But we will be concerned particularly with piano notes.

A MIDI file contains a list of MIDI messages together with timestamps, which are required to determine the timing of the messages. Further information called meta messages is relevant to software that processes MIDI files. The most important MIDI messages are the note-on and the note-off commands, which correspond to the start and the end of a note respectively. Each note-on and note-off message is, among others, equipped with a MIDI note number, a value for the key velocity, a channel specification, as well as a timestamp. The MIDI note number is an integer between 0 and 127 and encodes a note's pitch, where MIDI pitches are based on the equal-tempered scale. For example, the note A4 has the MIDI note number 69. The key velocity is again an integer between 0 and 127, which controls the intensity of the sound. It is determined by how softly key of piano is pushed. The MIDI channel is an integer between 0 and 15. That means 16 instruments can be included in MIDI message. Finally, the

time stamp is an integer value that represents how many clock pulses or ticks to wait before the respective note-on or note-off command is executed. MIDI subdivides a quarter note into basic time units referred to as ticks.

```
Note(start=0.000000, end=0.500000, pitch=69, velocity=107)
Note(start=0.500000, end=0.750000, pitch=68, velocity=107)
Note(start=0.750000, end=1.000000, pitch=69, velocity=107)
Note(start=1.000000, end=1.500000, pitch=71, velocity=107)
Note(start=1.500000, end=2.000000, pitch=69, velocity=107)
Note(start=2.000000, end=2.500000, pitch=74, velocity=107)
Note(start=2.500000, end=3.000000, pitch=74, velocity=107)
```

Figure 4.2: MIDI data extraction from MIDI file

Piano roll representation

Encoding MIDI messages directly is not effective as it can not effectively preserve the notion of multiple notes being played at once. That is problematic if we want to generate a polyphonic music. One of the alternative is piano roll representation.

The piano roll gets its direct inspiration from automated pianos. It is a two dimensional representation. The vertical axis is a digital representation of different notes: for instance, each row in the vertical axis can be associated with its own piano key corresponding to its own note. The horizontal axis is a continuous representation of time. We can represent piano roll in computer using two dimensional matrix with values. It is formed by sampling MIDI data at equal time interval.

Number of notes in a sample over entire dataset can be seen from the figure below.

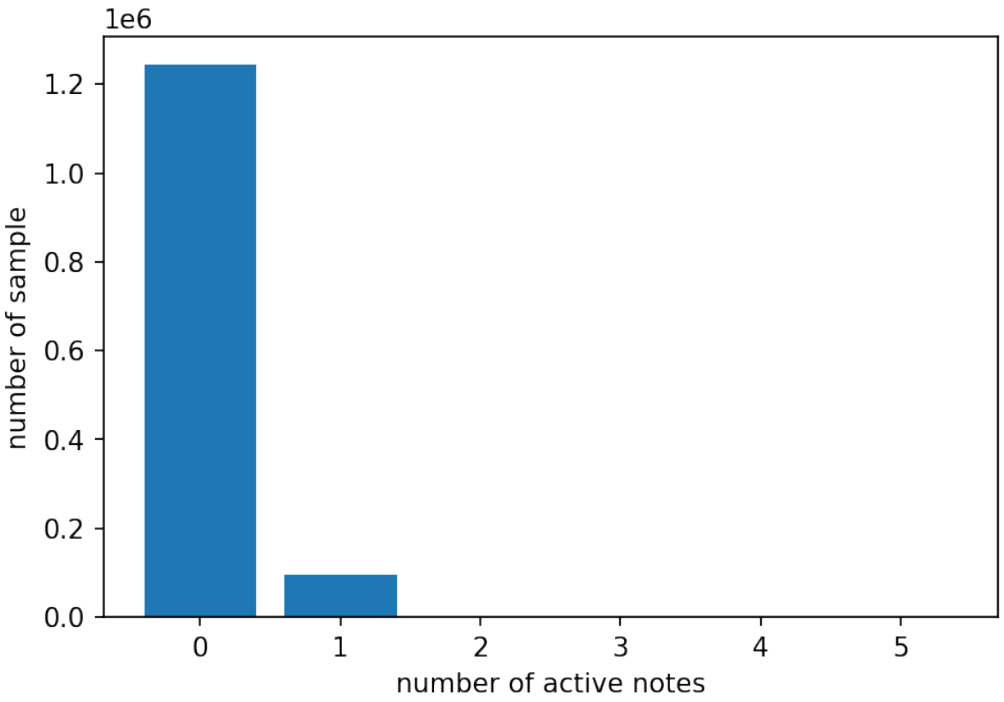


Figure 4.3: Number of notes playing simultaneously across samples

Number of sample with 0,1,2,3,4,5 active notes in a sample across entire dataset is 1245156, 96479, 157, 0, 0, 0 respectively. 0 active notes represents the samples where no notes are being played.

Preparation of data

It involves converting sequences of piano roll to a format suitable according to a model. This process is different for different deep learning models.

Building a deep learning model

Defining the model can be broken down into a few characteristics:

- Number of Layers
- Types of these Layers
- Number of units (neurons) in each Layer
- Activation Functions of each Layer

- Input and output size

Compiling the Model

After defining our model, the next step is to compile it. Compiling a Keras model means configuring it for training.

To compile the model, we need to choose:

- The Loss Function

  The lower the error, the closer the model is to the goal. Different problems require different loss functions to keep track of progress. Loss function like Mean Squared error, crossentropy loss are common. We have even used KL divergence as a loss function in variational autoencoder.

- The Optimizer

  The optimizing algorithm that helps us achieve better results for the loss function. Optimizer is gradient descent algorithm and its improvements like Mini batch gradient descent, Stochastic Gradient Descent (SGD), Adam.

- Metrics

  Metrics used to evaluate the model. For example, if we have a Mean Squared Error loss function, it would make sense to use the Mean Absolute Error as the metric used for evaluation.

Training

Deep learning neural networks learn a mapping function from inputs to outputs. This is achieved by updating the weights of the network in response to the errors the model made on the training dataset. Updates are made to continually reduce this error until either a good enough model is found or the learning process gets stuck and stops. The process of training neural networks is the most challenging part of using the technique in general and is by far the most time consuming, both in terms of effort required to configure the process and computational complexity required to execute the process.

Validation

Validation is carried out simultaneously with the training. As the model gets trained, the loss function on the training set decreases. However, training for many epochs may lead to overfitting of the model, so the model is also time and again validated against the validation set. In the validation set the gradient descent and dropout layers are turned off and the value of the loss function is observed. The increase in loss is often set as a stopping criterion for stopping training. Validation phase also helps in determining the hyperparameters of the network.

Inference and generation

Inference model should be built to use a trained network. Often inference model is extracted from trained model.

Evaluation and choosing of model

It is very hard compare the results of these algorithms that work in the area of the art generation. Music are intrinsically subjective. So it is difficult to approach them with the metrics we use for other model. The majority of approaches is usually based on peer review systems where many people listen to the music and evaluate performance.

4.3. Experiments with some architectures

We studied number of deep learning architecture like autoencoder, Seq2seq LSTM and Variational autoencoder. These experiments involve quick development of models. We choose one model from them and then make improvements on that model. Experiments carried out with these architectures are as follows:

4.3.1. Seq2seq LSTM

The encoder takes an encoded sequence of data, and outputs a vector. This vector is then decoded into another sequence by the decoder.

Prepare input

Piano roll samples is converted into group of input sequences. The output for each input sequence will be the first sample that comes after the sequence of notes in the input sequence in our piano rolls. We have put the length of each sequence to be 100 samples. This means that to predict the next sample in the sequence the network has the previous 100 samples to help make the prediction. We need to one hot encode the extracted sequence of note.

- Number of sequences = 15850
- size of each sequence = 100
- One hot encoding size = 138
- Input tensor shape = (15850, 100, 138)

Build a training model

- **Used Layers**
  - LSTM layers:
    Two stacked LSTM is used in both encoder and decoder part of network.
  - Dropout layers:
    Dropout layer is used in between two LSTM stack. Dropout randomly nullify node. Dropout of 0.2 is used. This has regularization effect on network.

– Dense layer:

Output sequence has to softmax output so dense layer is used.

- **Activation function**

Each LSTM layer is followed by ReLU activation. Output layer which is dense layer is followed by softmax activation. This is because we have one hot encoded our prepared sequences.

- **Model**

Encoder is a two stack LSTM layer which takes the input sequences from prepared input and randomly initialized initial state (cell state and hidden state). It is then followed by a dense layer with softmax activation. The softmax generates a probability for different possible category as output and we can select the maximum probability as the predicted output. For decoder, input sequence is shifted one step right with first element 0. Output sequence expected is the same as that of the input sequence.

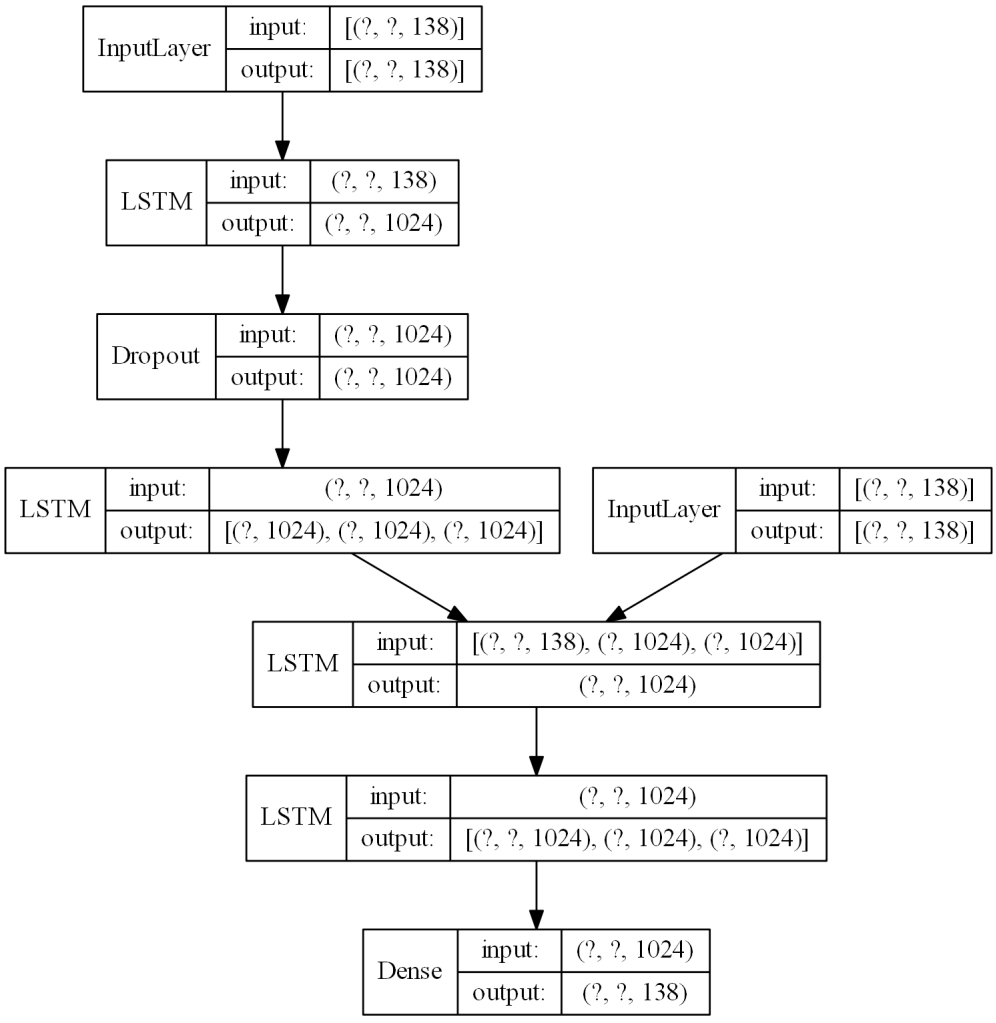


Figure 4.4: Seq2seq architecture

Compiling the model

- **Loss Function:**

  Loss function is categorical cross entropy. Categorical cross entropy is used for softmax outputs.

- **Optimizer:**

  Optimizer used is Adam optimizer with learning rate 0.001

- **Accuracy**:

  It is the ratio of number of correct predictions to the total number of input sequences.

Training and validation

- Batch size = 256
- Validation split = 0.2
- Number of epochs = 500

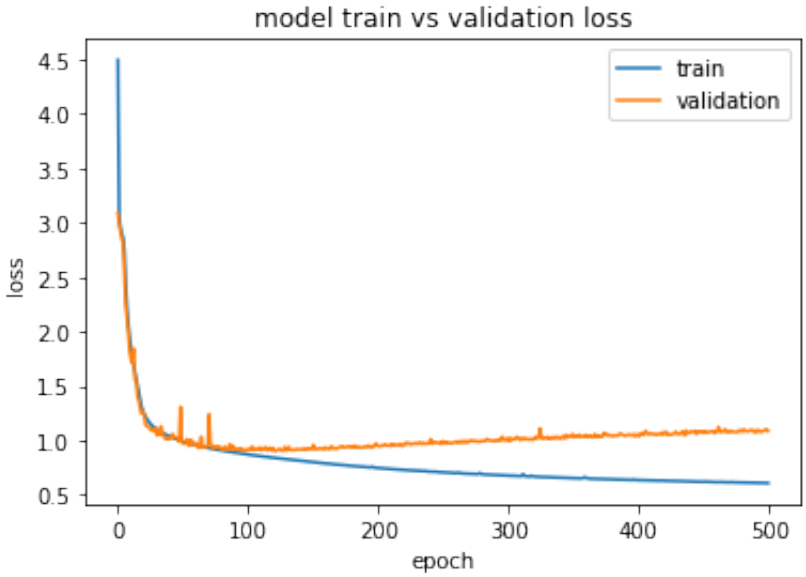


Figure 4.5: Loss vs Epochs graph for Seq2Seq LSTM

From above graph we can see there is no increase in performance after around 100 epochs. It is clear that network is overfitting to training data.

Inference and generation

Inference model encoder is the same as that of the training model. However, we need to reconfigure the decoder model to generate one output and then feed that output into the next timestamp decoder cell to generate the next output and so on. Output from the above inference model is a series of outputs. These output is converted back to piano roll representation and then into MIDI format.

4.3.2. Autoencoder

The sequence is processed by the encoder of an AE that embeds it into latent space. Modifying the values of the 100-dimensional latent vector creating a new latent vector that is decoded by the AE decoder to form a new musical sequence. This is advantageous because user are able to control the music generation process.

Prepare inputs

Piano roll samples of each song is converted into 16 measures. Each measure consists of 96 samples. Each sample size is 35 representing notes played. Here number of notes in MIDI format is 128 but only notes valued 30 to 65 are used in these dataset, because we transpose notes to a central key. We also use augmentation technique by transposing notes. This increases our available dataset to 7860 total data. This is sufficient for training the model.

- Number of data after augementation = 15560
- Number of measure in each data = 16
- Number of piano roll samples in each measure = 96
- Size of each sample = 35
- Input tensor shape = (15560, 16, 96, 35)

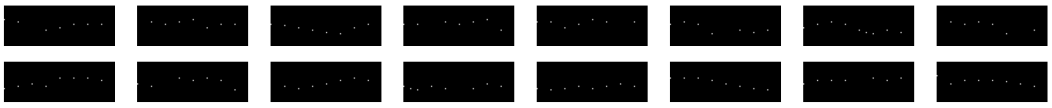


Figure 4.6: Visual representation of single data of shape (16, 96, 35)

Build a model

- **Used Layers**
  - Dense layer

    Dense Layer for creating a deeply connected layer in the neural network where each of the neurons of the dense layers receives input from all neurons of the

previous layer. At its core, it performs dot product of all the input values along with the weights for obtaining the output. Each dense layer is activated by ReLU activation accept last one.

- Time Distributed layer

  Time Distributed layer applies the same layer to several inputs. This layer allows to apply a layer to every temporal slice of an input. Each training data consists of 16 measures. All these measures goes through similar Dense layer.

- Dropout layer

  Dropout layer randomly turns off neurons with some fixed probability during the processing of each batch. This reduces model overfitting and leads to better results in the validation set. The values from the turned off neurons are set to 0. Dropout rate used is 0.2. That means 20% of nodes in a layer will be turned off or set to zero.

- Batch Normalization layer: The activation of different neurons in the same layer can have great variation. This can lead to slower learning and difficulty in hyperparameter tuning as the space in which we try to minimize the loss becomes skewed. A batch normalization layer learns parameters β and λ which normalize the activation of previous layer before feeding them to the next layer.

- Reshape layer: This layer has the responsibility of changing the shape of tensors.

- **Activation:**

  Each dense layer is followed by Relu activation accept last layer of decoder. Last layer of decoder is followed by sigmoid activation. This is because we expect output of last layer to be either 0 or 1 corresponding to note on or note off.

- **Model**

  Model is combination of encoder and decoder part as shown in block diagrams.

  - **Encoder:**

    It reduces the large sized tensor of shape (16, 96, 35) to vector of size 100.

  - **Decoder:**

    Decoder is supposed to reconstructs the input from encoded representation. Shape of output is same as shape of input to network

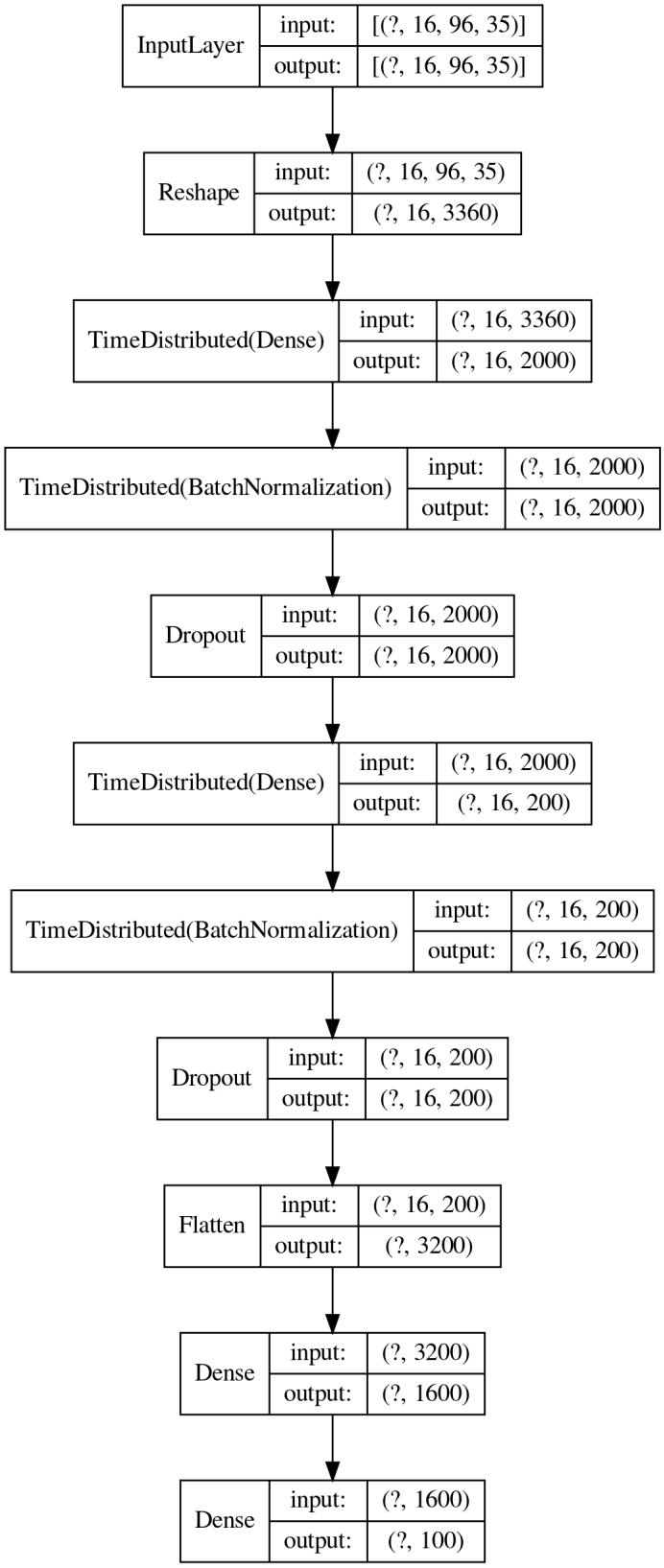


Figure 4.7: Autoencoder: Encoder

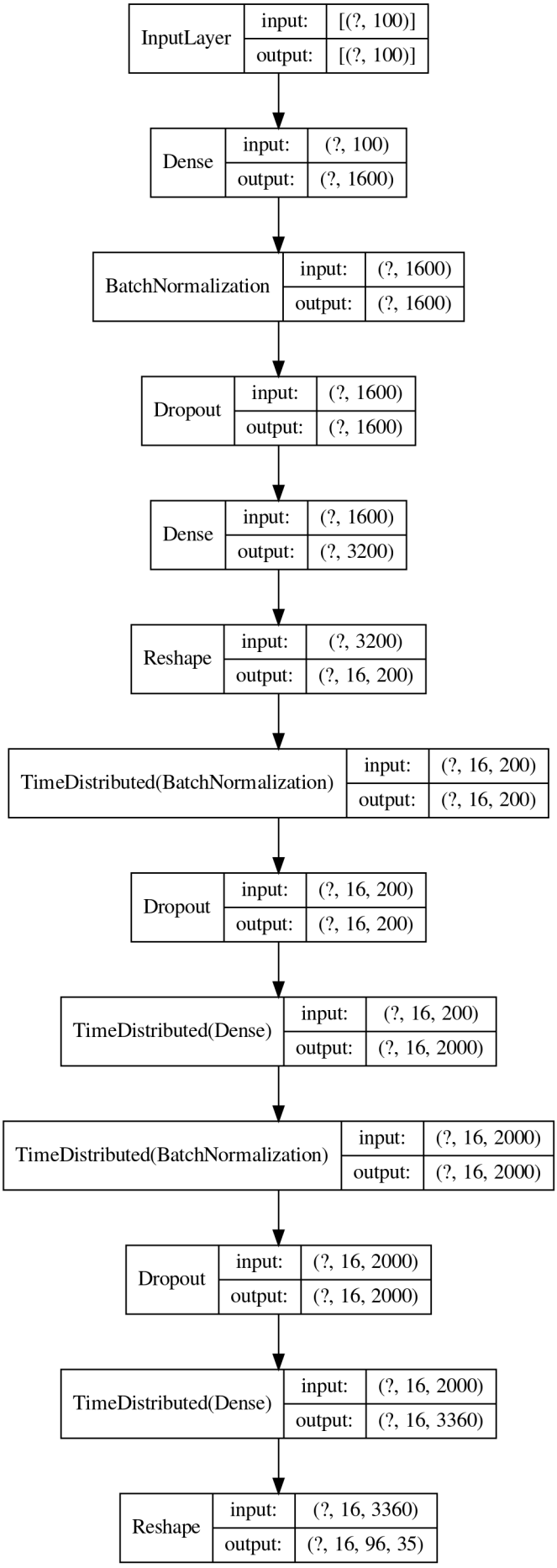


Figure 4.8: Autoencoder: Decoder

Compiling the model

- **Loss Function:**

  Both Mean squared error and Binary cross entropy were experimented with. We observed faster learning in case of binary cross entropy. This is because output from each node on output layer as activated by sigmoid activation.

- **Optimizer:**

  Adam optimizer with learning rate 0.001 is used.

Training and validation

- Batch size = 512
- Validation split = 0.05
- Number of epochs = 1000

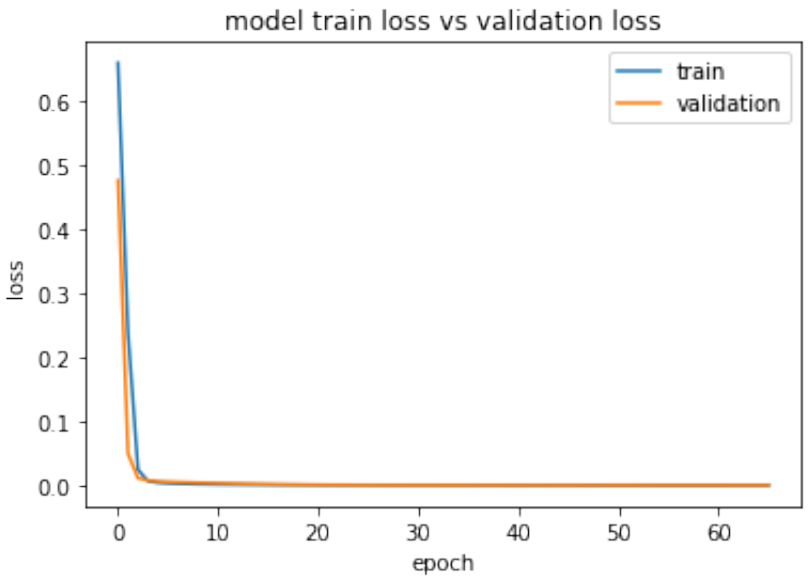


Figure 4.9: Loss vs Epochs graph for Autoencoder

Performance is measured using loss metric defined above. Weights updated is saved in every 50 epochs in order to observe the improvement.

Inference model and Generation of music

After training decoder part of the model can be used for inference. It accepts vector of size 100 and produces a output which can be changed to midi format.

4.3.3. Variational Autoencoder

Variational autoencoder is improvement to simple autoencoder for improved generalization. The difference is, an encoder network turns the input samples into two parameters in a latent space, which we will note μ and σ which are both the vector of size 100. Then, we randomly sample similar points z from the latent normal distribution that is assumed to generate the data, via $z = \mu + \sigma * \varepsilon$, where epsilon is a random normal tensor. Finally, a decoder network maps these latent space points back to the original input data.

Preparation of input

Input preparation is similarly produced as in the case of Autoencoder. Input data is tensor of shape (7840, 16, 96, 35)

Building a model

- **Used Layers**

  All layers used in autoencoder is used in VAE. Only difference is addition of lambda layer. Lambda Layer is used for transforming the input data with the help of an expression or function.

- **Activation**

  Similar to autoencoder, only last layer is followed by sigmoid activation. All other dense layer is followed by Relu activation.

- **Model**

  Model is combination of encoder and decoder part as shown in block diagrams.

  - Encoder:

    Encoder uses layers explained above to convert the inputs prepared of shape (Batch size, 16 , 96 , 96) to bottleneck layer of shape (Batch size , 100) which is a latent space representation. It simply reduces the dimension of the data. Last layer which is a lambda layer uses the expression by carrying out sampling from previous two layers μ and σ. Expression is as follows:

    $z = \mu + \sigma * \varepsilon$

    This layer is latent space representation. Every training data has to pass through this bottleneck.

– Decoder:

  Decoder is supposed to convert the output from lambda layer to original input. This decoder is same as that of simple autoencoder. Same decoder is used also as inference model.

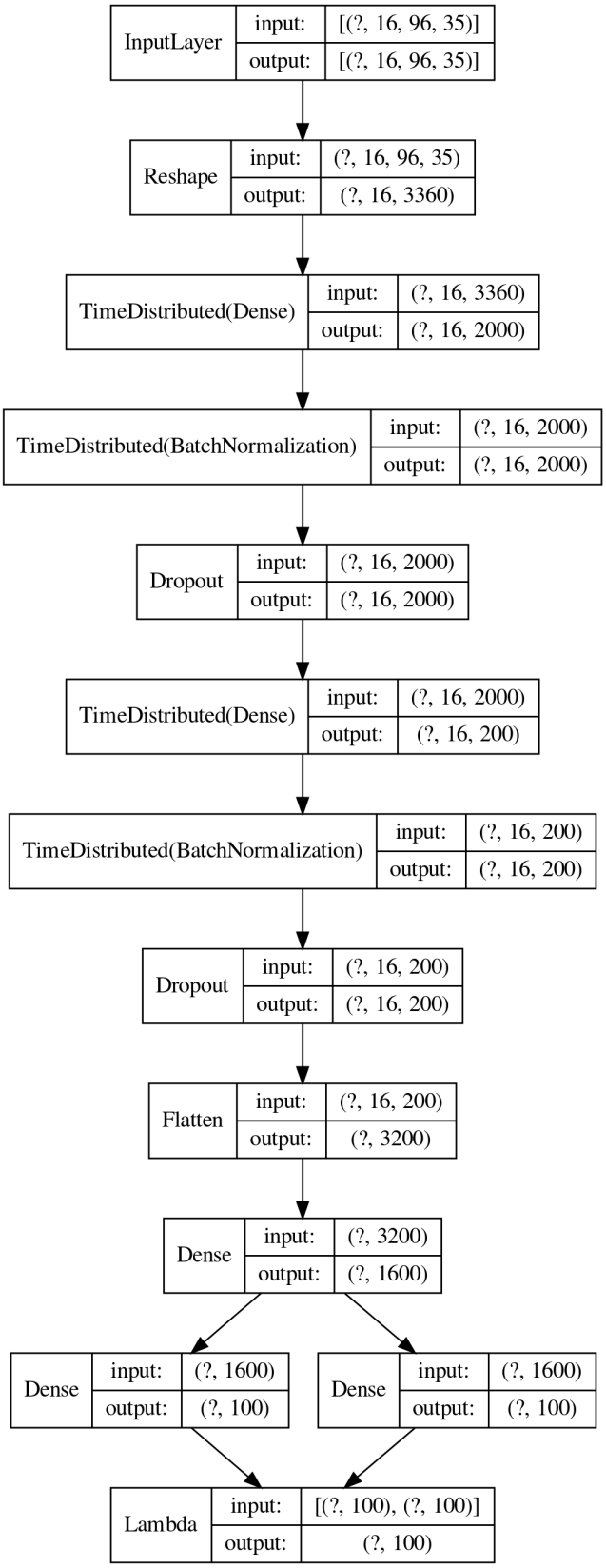


Figure 4.10: Variational autoencoder : Encoder

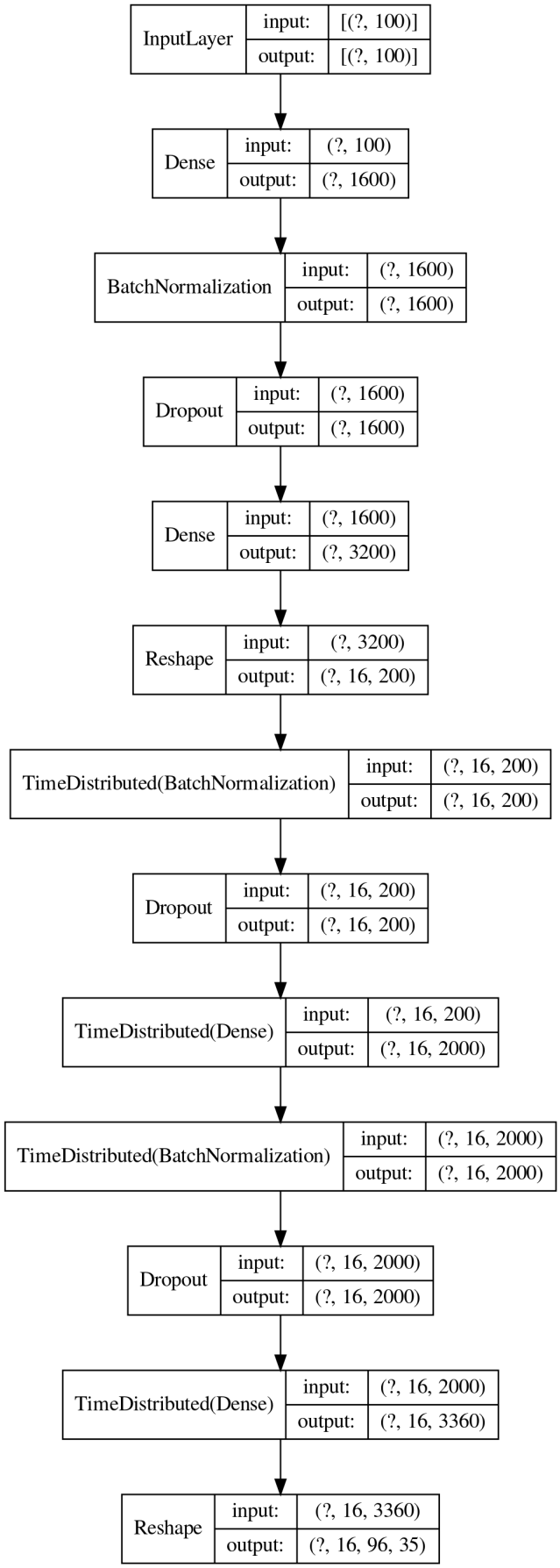


Figure 4.11: Variational autoencoder : Decoder

Compiling the model

- **Loss Function:**

  VAE loss is combination of two kinds of loss:

  - Generative or reconstruction loss:

    This is the loss used in normal autoencoder. In this experiment, binary cross-entropy is used as generative loss.

  - Latent loss:

    This loss compares the latent vector with a zero mean, unit variance Gaussian distribution. The loss we use here will be the KL divergence loss. This loss term penalizes the VAE if it starts to produce latent vectors that are not from the desired distribution.

- **Optimizer:**

  Adam optimizer with learning rate 0.001 is used.

Training and validation

- Batch size = 512
- Validation split = 0.05
- Number of epochs = 2000

Figure 4.12: Loss vs Epochs graph for Variational Autoencoder

Early stopping is also used because it allowed us to specify an arbitrarily large number of training epochs and stop training once the model performance stops improving on the validation dataset. Performance is measured using loss metric defined above. Weights updated is saved in every 50 epochs in order to observe the improvement.

Inference and generation of music

After training decoder part of the model can be used for inference. It accepts vector of size 100 and produces a output which can be changed to midi format.

Experimenting with Batch Normalization Momentum and Drop out rate

Figure below shows the nature of training with different Dropout rate and batch normalization rate. Dropout and batch normalization significantly increase training time.However batch normalization causes faster convergence and, dropout rate is regularization technique which increases the generalization by avoiding overfitting. Batch Normalization also has unintended regularization effect.

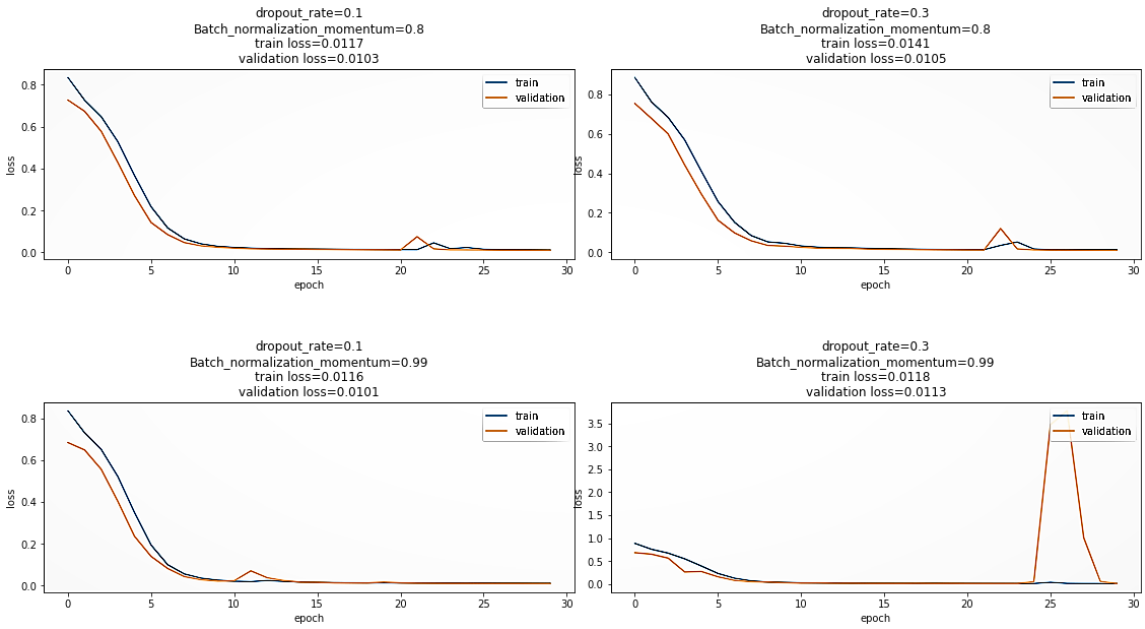


Figure 4.13: Loss vs Epoch graph for first 30 epochs at different dropout rate and batch normalization momentum

Latent space representation

Mean and standard deviation of the bottleneck layer can be seen on figure below.

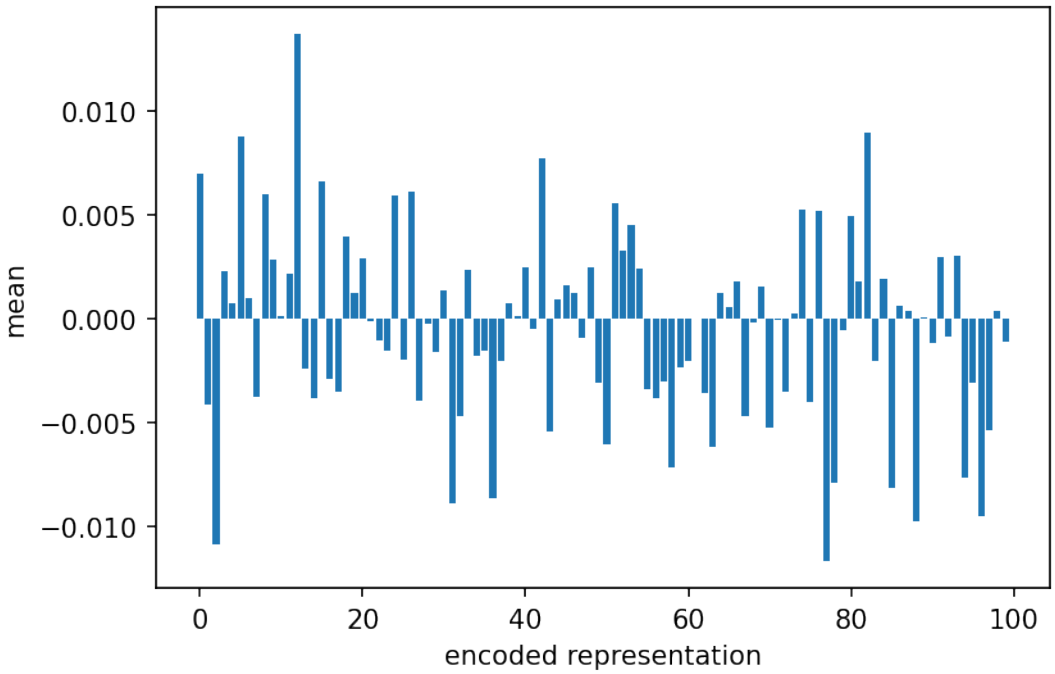


Figure 4.14: Latent space representation mean

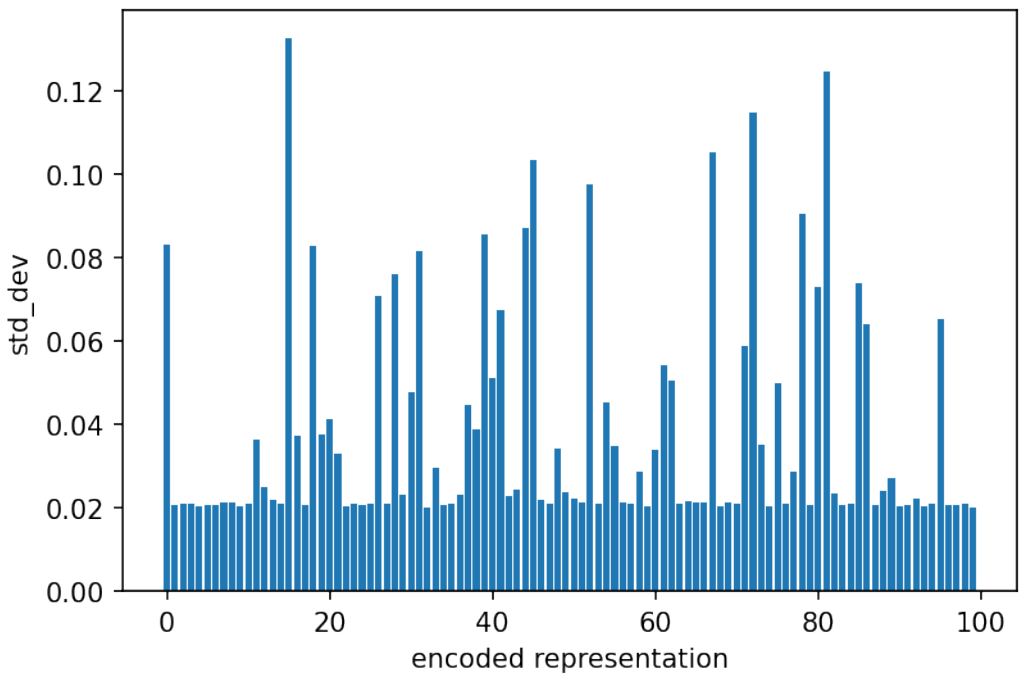


Figure 4.15: Latent space representation standard deviation

Standard deviation is measure of variability in a data. So high standard deviation or variance accounts for most variability. Higher the variability more information is contained in it. So priority is set on basis of standard deviation. Figure below shows encoded representation after sorting with respect to standard deviation.

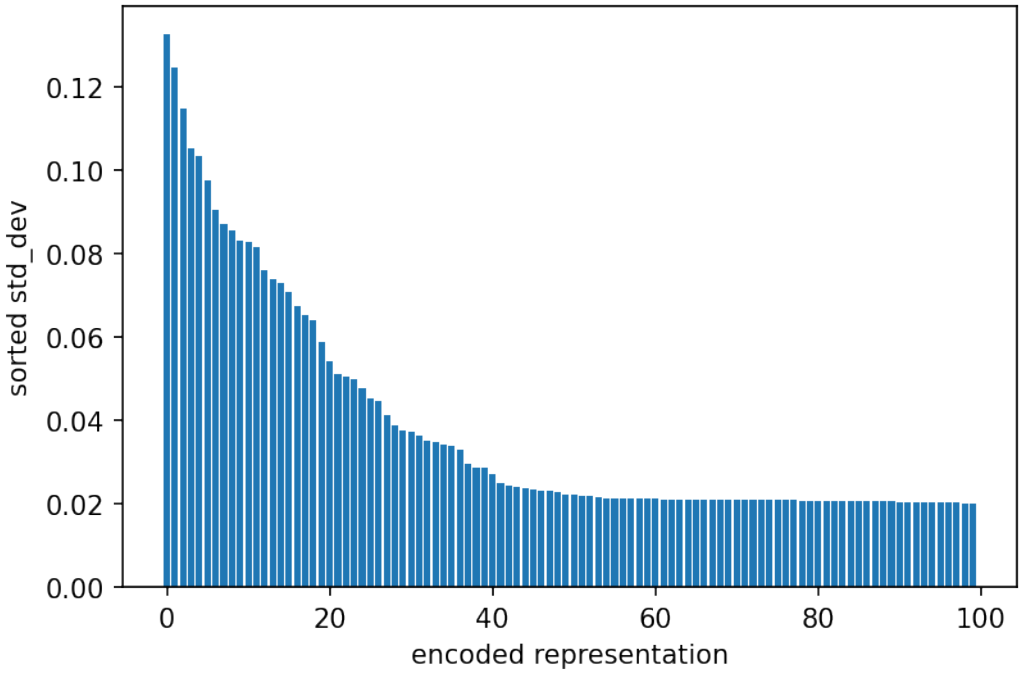


Figure 4.16: Latent space representation sorted with respect to standard deviation

4.3.4. Generative Adverserial Network

GANs are the first choice networks when it comes to creating a new and creative data out of blue with some real data distribution to depend on. It employs two different architectures that compete with each other, namely generator and discriminator. As the name suggests, the generator generates new data that resembles the original data distribution while the discriminator distinguishes between the original data and generated data. This process of conflict between generator and discriminator is capitalized by minimax objective function to seek the balance between the generator and the discriminator.

Preprocessing of Input Data

For the training of GAN, we used the classical MIDI dataset which contains the classical piano music. The dataset consists of 318 classical piano pieces in a MIDI file. MIDI file can be understood as the digital representation of sheet music. MIDI file consists of messages in which instructions are mentioned such as note_on, note_off, set_tempo, etc. The note_on message instructs to play a note and consists of note, velocity, channel and time. For each note_on message, there is always a corresponding note_off message. Velocity is the loudness of the note to be played. Channels are independent information streams. A channel is like a virtual path which streams MIDI events and the time in each midi message denotes how long to wait until another message. Time in midi message is the length of time after which another message is read.

The dataset is preprocessed into three midi attributes as a tuple which are midi note number, note duration and rest duration *i.e.* $\{midi\ number,\ \ note\ duration,\ \ rest\ duration\}$ and a sequence of 100 such tuples are prepared as a single training data sequence. Every note of piano is represented by a distinct midi number in a midi file which ranges from 21 to 108. Note duration is the length of time that a note is played and rest duration is the duration that denotes how long the silence in a piece of melody will last. Note duration and rest duration attributes are calculated from the note_on and the note_off messages using the following formula.

$$note\ duration = (note\_off_k - note\_on_k) * \frac{BPM}{60}$$

$$rest\ duration = (note\_on_{k+1} - note\_on_k) * \frac{BPM}{60}$$

where $(note\_off_k - note\_on_k)$ and $(note\_on_{k+1} - note\_on_k)$ are time lengths in seconds and BPM is the beats-per-minute value extracted from the corresponding midi file.

Music attributes are constrained to their closest discrete values $\{0.125,\ 0.25,\ 0.5,\ 0.75,\ 1,\ 1.5,\ 2,\ 3,\ 4,\ 6,\ 8,\ 16,\ 32\}$. The quantized music attributes are estimated to see if each generated sequence has a perfect scale consistency of melody. The remaining out-of-tune notes are mapped to their closest in-tune music attributes.

Finally, the total of around 7600 training sequences were grouped into batches of size 16. Hence, a batch data has a shape of (16, 100, 3) with 475 number of batches.

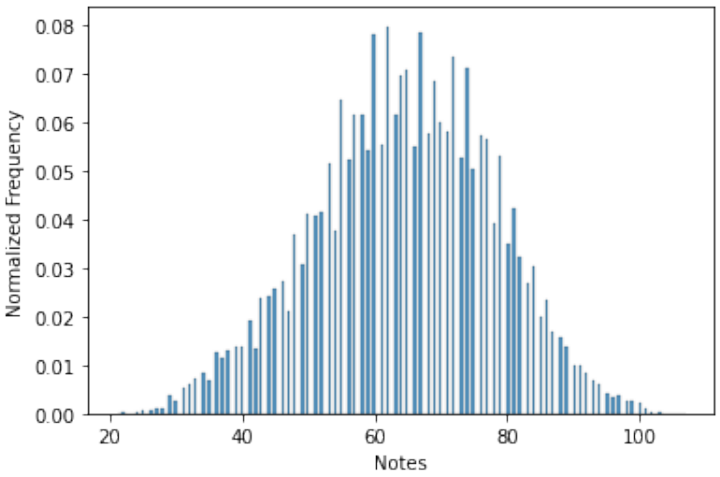


Figure 4.17: Frequency of Note Occurence

Used Network Layers in GAN

- Input layer

  Input layer expects the input data which in case of generator is noise vector while it is midi attributes in sequence in case of discriminator.

- Fully Connected Layer

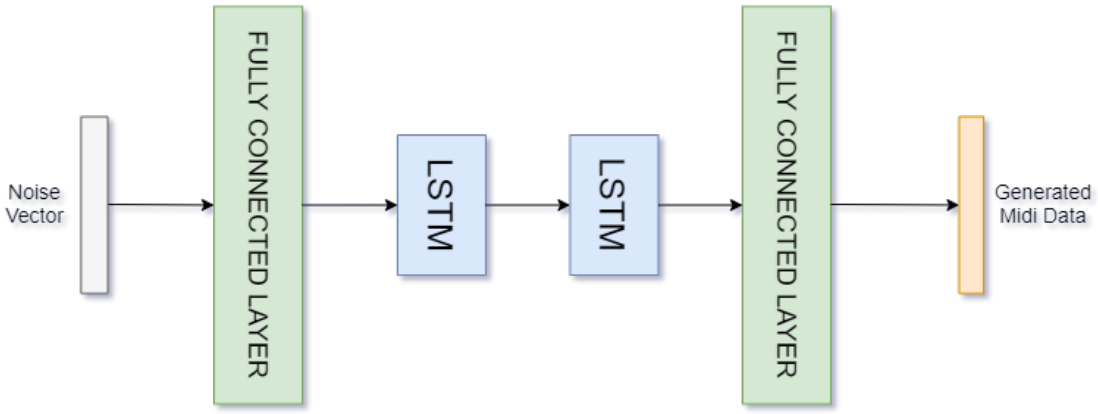
  Fully Connected Layer is also known as dense layer. It accepts the input from input layer and outputs the values according to the number of neurons defined in the layer. Activation function of linear, ReLU and sigmoid activations have been used for dense layer activation in this architecture.

- Long Short-Term Memory (LSTM)

  LSTM is a variant of RNN which helps in the memorization of sequence passed through the network. LSTM consists of LSTM cells which remembers the values over arbitrary time intervals and the gates which regulate the flow of information into and out of the cell.

Generator

Generator is responsible for the generation of midi numbers and durations that are essential in creating MIDI files. The generator is to learn the distribution of real samples, which is trained to increase the error rate of the discriminator. Generator is the building block of GAN.



.

Figure 4.18: Block Diagram of Generator

We have used 4 layers in Generator. At first, the noise is passed through the dense layer which has 400 units. For the first dense layer, ReLU activation function is used. The output of the first layer is then connected to the LSTM layer which consists of 400 LSTM cells and is activated by the tanh activation function. The third layer is also LSTM layer consisting of 400 LSTM cells followed by the tanh activation function. The final layer is composed of dense network of 3 units to output the three music attributes of midi number, note duration and rest duration. The final dense layer has linear activation function.

For the generation purpose, a noise vector of shape $(100,\ 100)$ is passed through the generator network, which then transforms the noise vector into the three above-mentioned midi attributes with a sequence of 100.

Generator Loss

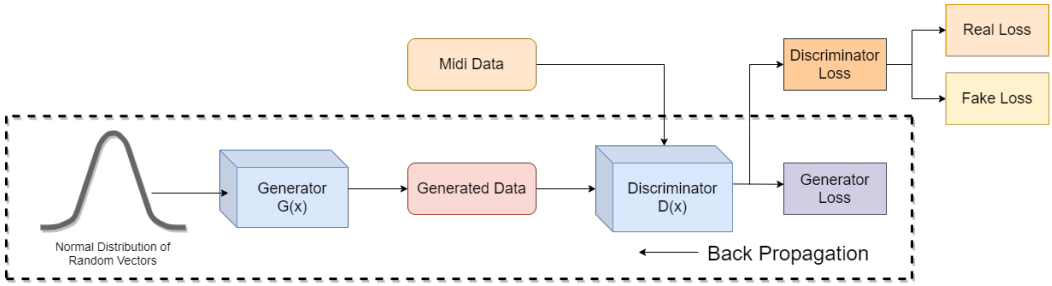


Figure 4.19: Generator Loss and Back Propagation

While the generator is trained, it samples random noise and produces an output from that noise. The output then goes through the discriminator and gets classified as either "Real" or "Fake" based on the ability of the discriminator to tell one from the other. The generator loss is then calculated from the discriminator's classification. It gets rewarded if it successfully fools the discriminator or gets penalized otherwise.

The following equation is minimized in training the generator:

$$\frac{1}{m}\sum_{i=1}^{m} log(1 - D(G(z^{(i)})$$

Discriminator

Discriminator is responsible for the classification of real and fake data distribution. Discriminator keeps learning to distinguish between the real data and fake data as it trains on more iterations of real data and generated data. Discriminator is the classifying block of GAN.

The generated midi attributes and the preprocessed real midi attributes are passed through the discriminator which classifies each sequence as either real or fake.

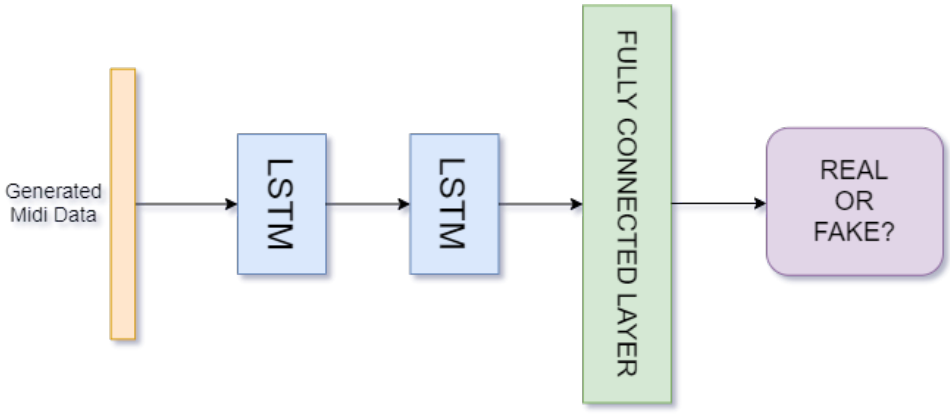


Figure 4.20: Block Diagram of Discriminator

We have used 3 layers in Discriminator. The three attributes of music in a sequence of 100 is passed through the LSTM layer which consists of 400 LSTM cells and is activated by the tanh activation function. The second layer is also same as the first one with LSTM layer consisting of 400 LSTM cells followed by the tanh activation function. The final layer however is composed of dense network of of single unit node to output a single value which is then passed through sigmoid activation function which maps the output of the dense layer to only two possible values 0 or 1 $i.e$ 0 $for\ fake$ and 1 $for\ real$.

Discriminator Loss

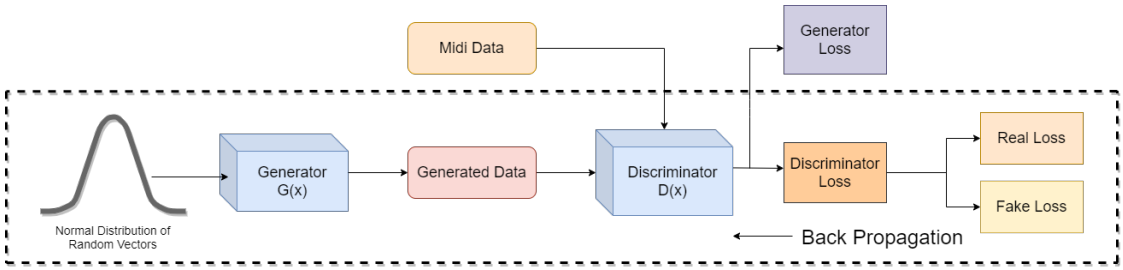


Figure 4.21: Discriminator Loss and Back Propagation

While the discriminator is trained, it classifies both the real data and the fake data from the generator. So, the discriminator loss is the sum of real loss and fake loss. It penalizes itself for misclassifying a real instance as fake, or a fake instance created by the generator as real, by maximizing the below function:

$$\frac{1}{m}\sum_{i=1}^{m}[\ log(D(x^{(i)}))\ +\ log(1-D(G(z^{(i)}))]$$

GAN Configuration

Layers	Generator Network	Discriminator Network
Input Layer	Noise vector of shape *(batch_size, 100, 100)*	3 midi attributes in sequence of 100
Layer 1	Fully Connected layer, *400 units*, relu activaion	LSTM, *400 lstm cells*, tanh activation
Layer 2	LSTM, *400 lstm cells*, tanh activation	LSTM, *400 lstm cells*, tanh activation
Layer 3	LSTM, *400 lstm cells*, tanh activation	Fully Connected layer, *1 unit*, sigmoid activation
Layer 4	Fully Connected layer, *3 units*, linear activation	N/A

Figure 4.22: Configuration of Generator and Discriminator Networks

Training of GAN and its challenges

For the training of GAN, we used Google Colab's GPU. Google Colab provides a single 12GB NVIDIA Tesla K80 GPU that can be used for 12 hours continuously. The training of the GAN was done for more than 200 epochs on Colab and it took around 7 hours for this particular model. The optimizer used for the training was Adam optimizer with parameters value as learning rate = 0.0002, $\beta_1 = 0.5$ and $\beta_2 = 0.9999$.

Similarly, other variants of GAN with different network layers were also trained previously which didn't yield any good results. Some GAN models suffered from the problems that commonly arise in GANs such as mode collapse, non-convergence, vanishing gradients, etc.

The losses of the final model for each batch and epoch are listed for 160 epochs as shown in the figure below:

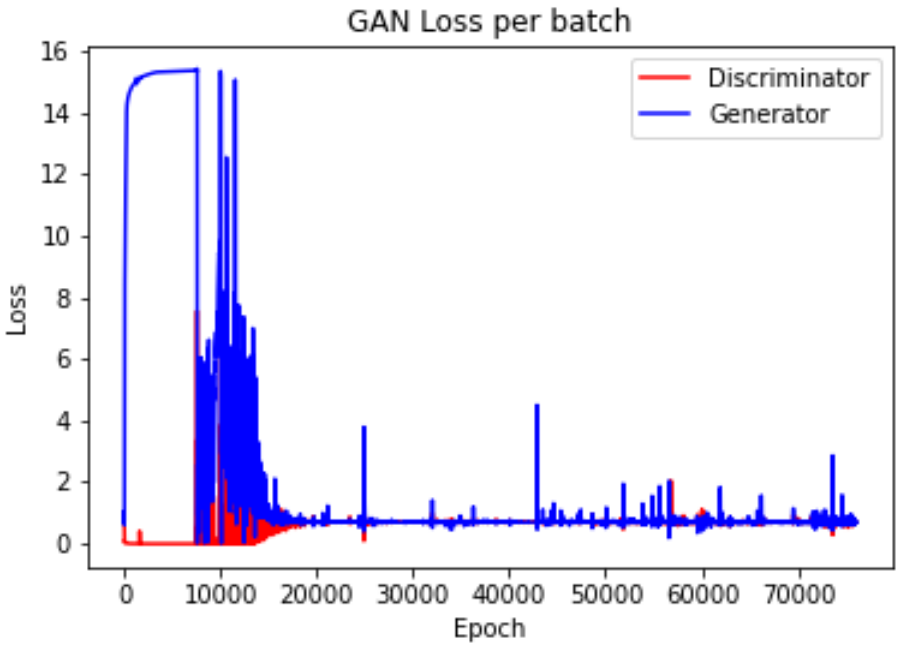


Figure 4.23: GAN Loss Per Batch

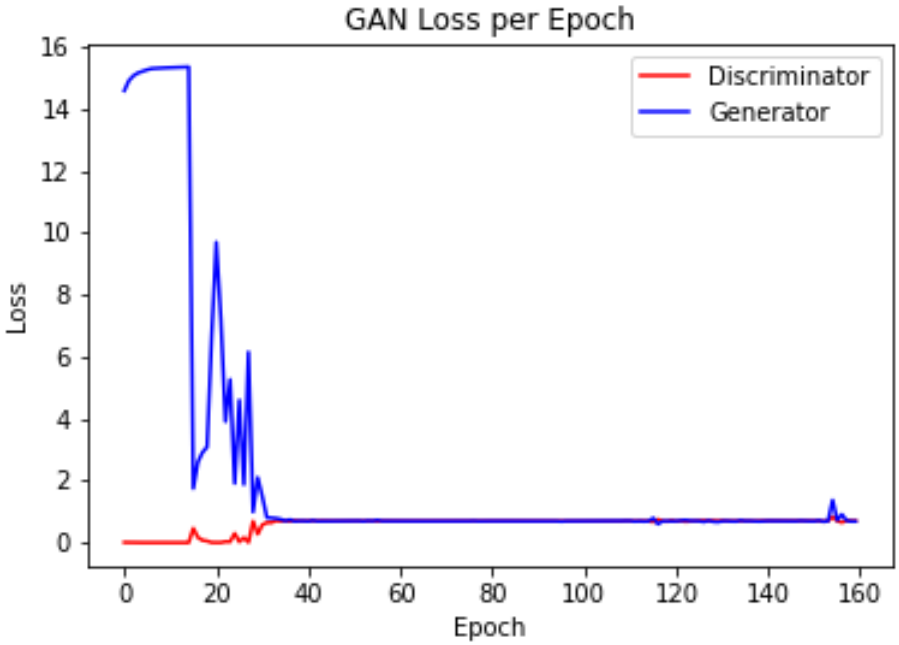


Figure 4.24: GAN Loss Per Epoch

At first, the losses diverges as the generator haven't learnt anything and produces random results so discriminator can easily classify them as fake. So the generator loss goes up while the discriminator loss remains at zero. After few epochs of training, generator seems to be learning, so the generator loss decreases while the discriminator loss slightly increases

and hence the loss values start to converge. After around 35 epochs, both generator and discriminator losses converges and stabilizes on the same range of loss and continues to remain so. As the loss stabilizes, the accuracy of the discriminator goes to around 50 %.

As it is already mentioned, a GAN model comprises of both generator and discriminator but there is a certain rule to be followed while training them. A discriminator is trained on both real data and generated data on each iteration and the loss are averaged for the discriminator loss. Meanwhile when we train the generator, the discriminator should not be made trainable. While training the generator, discriminator should only classify the real and fake objects based on what it has already learned previously but should not learn from the classification during generator's training phase. The simultaneous training of both generator and discriminator will cause instability to the network and the GAN will not learn anything and be trained as it is required to.

More often than not, GANs tend to show some inconsistencies in performance. Most of these problems are associated with their training and are an active area of research.

- Mode Collapse

  Usually we want GAN to produce a wide variety of outputs. However, if a generator produces an especially plausible output, the generator may learn to produce only that output. In fact, the generator is always trying to find the one output that seems most plausible to the discriminator.

  If the generator starts producing the same output over and over again, the discriminator's best strategy is always to reject the output of the generator. But if the next generation of discriminator gets stuck in a local minimum and doesn't find its way out by getting its weights even more optimized, it'd get easy for the next generator iteration to find the most plausible output for the current discriminator. This way, it will keep on repeating the same output and refrain from any further training.

  Wasserstein GAN (WGAN) and modified minimax loss can be implemented to avoid the problem of Mode Collapse.

- Vanishing Gradients

  Sometimes it can happen that if your discriminator is too good, then generator training can fail due to vanishing gradients. In effect, an optimal discriminator doesn't provide enough information for the generator to make progress.

  WGAN may come in handy for the riddance of vanishing gradients.

- Convergence

Since there are two networks being trained at the same time, the problem of GAN convergence may arise more often. The situation where both networks stabilize and produce a consistent result is hard to achieve in most cases. One explanation for this problem is that as the generator gets better with next epochs, the discriminator performs worse because the discriminator can't easily tell the difference between a real and a fake one. If the generator succeeds all the time, the discriminator has a 50% accuracy, similar to that of flipping a coin. This poses a threat to the convergence of the GAN as a whole. As the discriminator's feedback loses its meaning over subsequent epochs by giving outputs with equal probability, the generator may deteriorate its own quality if it continues to train on these junk training signals.

4.4. Evaluation and choosing of model

All of the proposed architectures work were successful in generating music although all models didn't produce music that is diverse and sufficiently interesting.

Seq2seq model is capable of dealing with sequence as input, regular autoencoders are not. In addition, Seq2seq LSTM can obviously take variable length inputs while regular ones take only fixed size inputs. However, observation showed that seq2seq model in certain cases outputs a few good notes and then all zeros. It also sometimes produces short melodies which repeat infinitely. The generated pieces lacked the musical clarity. Quality of the melody and rhythm within the pieces were low.

Autoencoder models, both simple autoencoder and variational autoencoder reduces the notes repeating problem. The generated pieces had far more rhythm and coherence because the model was capable of identifying motifs. Song generated by the autoencoder was more or less same as songs used for training. That indicated autoencoder was prone to overfitting. However, VAE models the network using probability distribution. Latent space representation is the sampling of Gaussian distribution, so variation is continuous and smooth.

GANs try to replicate the training data distribution. Generator tries to output similar distribution of generated data to not get caught up with discriminator. Hence, with enough training data and sufficient learning iterations and epochs, GAN was able to generate notes and melody from random noise input.

From subjective evaluation, we find that variational autoencoder and generative adversarial network are good at generating better music than the rest.

Evaluation of models is subjective. Four different models are trained and their generated

outputs are compared on the basis of how melodic the music sounds. All the models generate some kind of music, but, one generates repetitive notes which do not sound good to ear while other suffers from the problem of overfitting making the model unable to generate new music.

4.5. System design

After the extensive training and evaluation of model, it is necessary that the normal user be able to realize the model through front-end and back-end connections, user interactions and graphical visualization of assets such as piano and sliders.

So,our web app is basically a Django app with some static files(CSS, JS) and HTML templates rendered by view logic.

4.5.1. Use Case Diagram

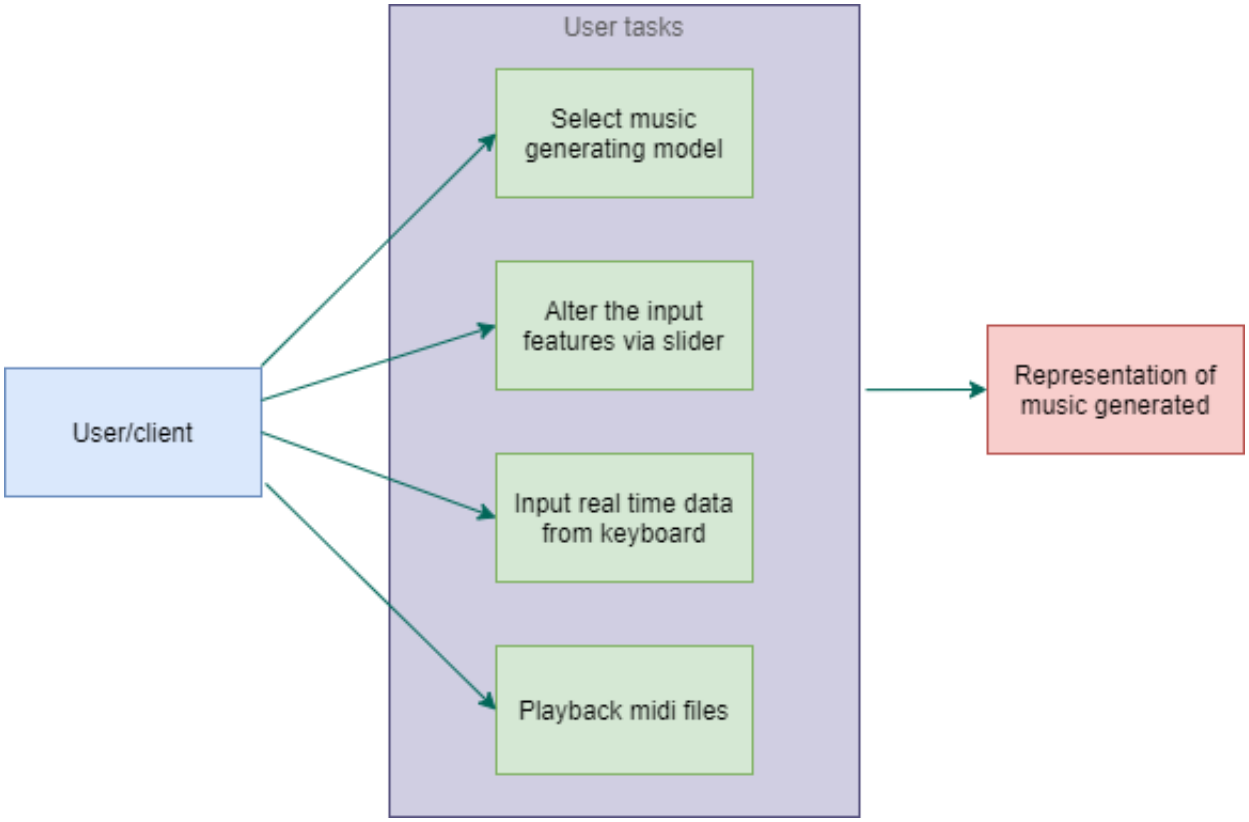


Figure 4.25: Use case diagram

Our system is currently focused solely on the end user.The user can select between the VAE and GAN model.He/she can vary the tempo of music being generated through the sliders and piano keys.The midi file can also be saved for playback

4.5.2. Frontend

The user can vary the control over the music being generated through the help of sliders. A total of 20 sliders of HTML element input of type range are all initialized with value 0. As the position of pointer in each slider is changed, the corresponding change(data) is sent as Ajax request to the server, particularly to the music composing view function where the get request is handled. The change in position is detected by the mouse events.

Control buttons are play/stop, pause/resume buttons bound with events on click.

Piano is constructed with series of buttons with proper mapping of each buttons to the notes and styled to give the resemblance of actual piano.In general piano, there are white and black keys with white key resembling for note and black for corresponding flat/sharp note. Each button pressed and released is handled with callback functions attached with event listeners.

Tone.js is one of the in-built component used as part of the project.A custom Tone synthesizer is created .It facilitates the attack and release of notes of piano upon pressing the control buttons.

4.5.3. Backend

In backend, the logic for model load and rendering of context is specified. View and template sections of Django MVT architecture have been implemented as there is no database. As the URL hits the 'localhost:8000/application', the view function is rendered. When we press the play button, initially the predictor is initialized with some random notes(music) which is fetched and played. The midi played is then extracted as python objects with the help of Mido. The real time python objects are returned as JSON objects in the browser as shown in figure 4.18. These fetched JSON response objects are mapped to the piano keyboards for the visualization of which note is currently being played.

The value and position received by the view function upon triggering of event in browser, the slider at which the event is being triggered is detected and then the corresponding node's variance is varied to generate a whole new set of note series.

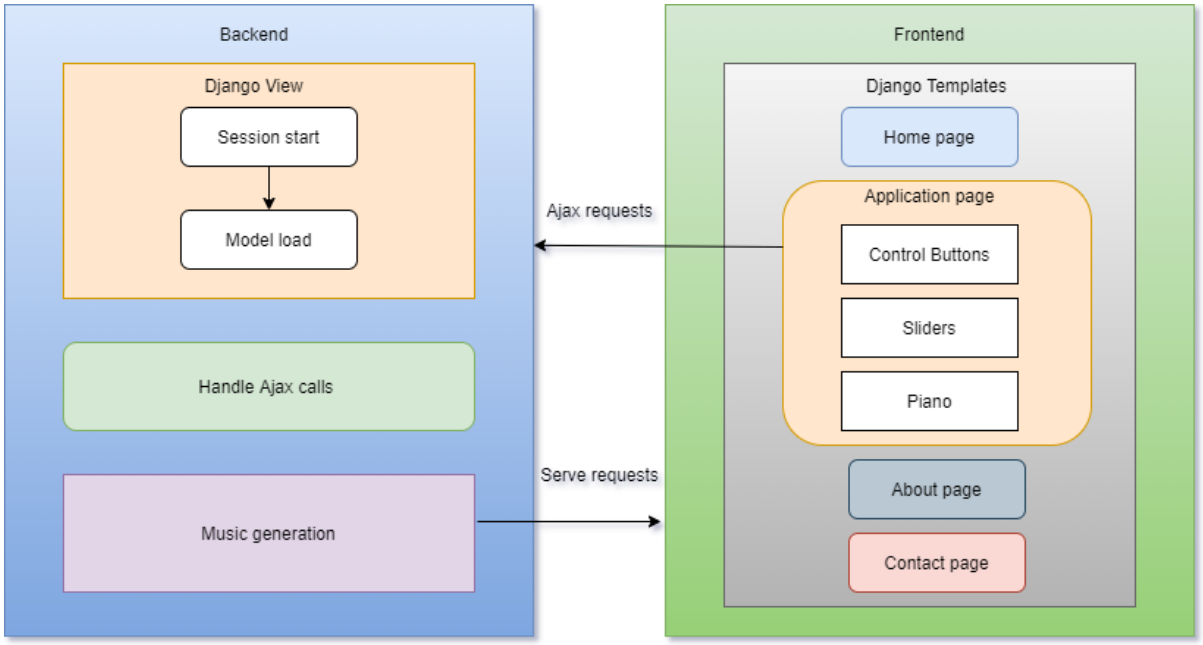


Figure 4.26: Web app architecture

All the information in the web app is transmitted in the form of request/response objects. Contents like images, HTML, CSS, JavaScript are all response objects. Request is usually lighter in size(bandwidth) than response.Any url searched is a request. The request object passed has the data such as positon of sliders,the value of sliders, pressing of keyboard buttons, request for '/application','/contact' etc.

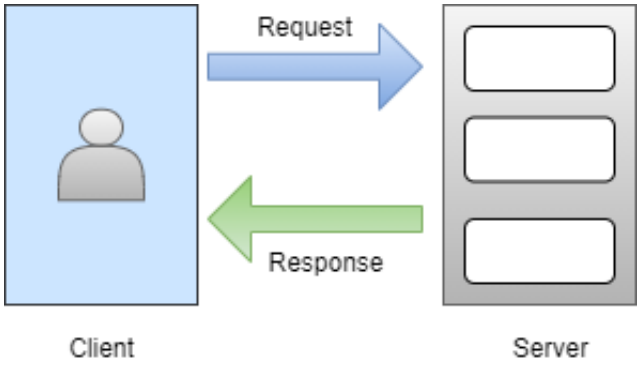


Figure 4.27: Request response cycle

Whenever a request comes to Django, it is handled by middlewares. When the Django server starts, the first thing it loads after settings.py is middlewares.The Request is processed by various middlewares one at a time firstly through security middleware. If it seems to be unhealthy, then it does not let the request go any further.Otherwise, the authentication requests are handled by authentication middleware.

Once, our request is processed by the middlewares it is passed to the URL Router. The URL router simply extracts the URL from the request and will try to match it to defined urls. Once, we get a matching URL, the corresponding view function is called. These requests are considered to be HttpRequest class objects.

Once, the view function is been executed, it's time to give a response. It has built-in support to provide responses of various types; one among them being JSON response. When the response is rendered, it will look for the HTML. The HTML page to be the server is processed by django templating engine. Once that completes, the Django simply sends the files as any server would. That response will contain the HTML and other static files.

5. RESULTS

RNNs are known for having memories. For making a decision, it considers the current input and the output that it has learned from the previous input. LSTM, a modified version of RNNs, makes it easier to remember past data in memory. LSTMs, that were implemented inside generator and discriminator, were able to find the correlation between the sequence of notes and melody that were played by the midi file and hence generated output learning from them.

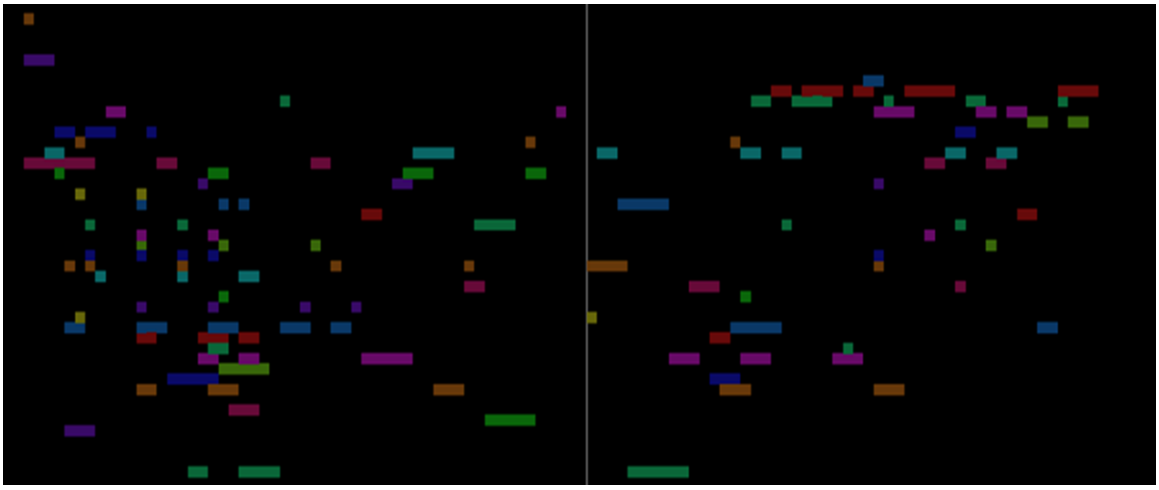


Figure 5.1: Visualization of generated midi notes and durations at 200 epochs with midi number in the vertical axis and duration in the horizontal axis

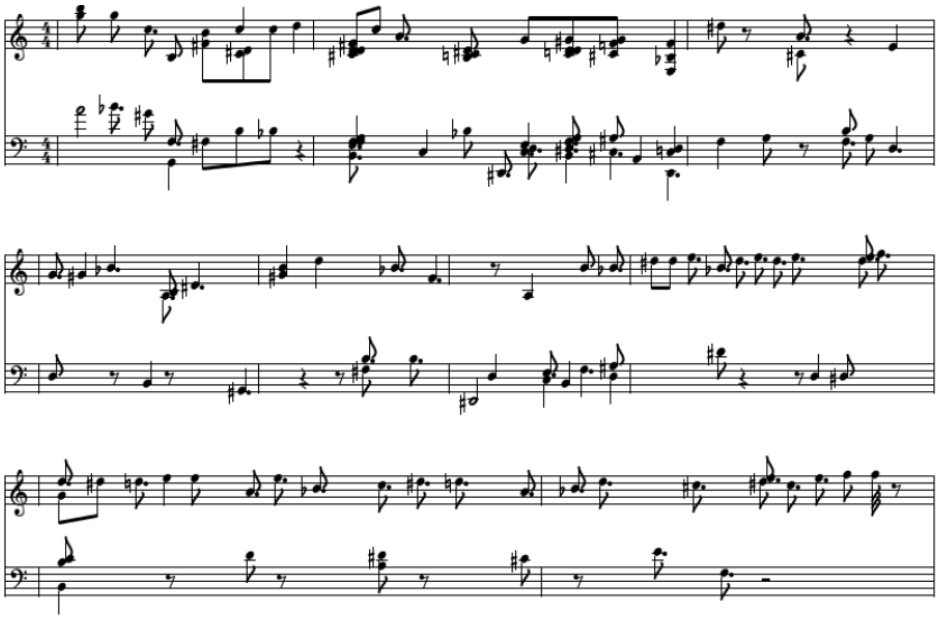


Figure 5.2: Sheet Music of generated midi file at 200 epochs

The above sheet music is the converted version of the generated midi file outputed by GAN trained for 200 epochs.

For subjective evaluation of the generated output, we asked 10 different individuals who are not from musical background and asked them following questions:

- Can it be classified as music or noise? (labelling 1 for music and 0 for noise)
- Do the notes and the melody blend well in this piece?
- How do you rate this? (0 being the minimum and 5 being the maximum)

All of the individuals classified the piece as music and said, it was pleasant enough to call it a noise. Almost all called for the music to blend well with notes and melody while being discontinuous at some point. Some even called out its resemblance to a person playing the piece and certainly not a machine. Most gave it the rating from 3 to 4, which is quite good considering it is generated by a machine.

6. CONCLUSION

Our project uses some of the deep learning techniques to generate musical content. We have experimented with various architectures. We have proposed a model which is able to generate music with some control. We have observed promising result from both generative model VAE and GAN. VAE allows the user to control the output generation. We have trained VAE model using various styles of music. We have also developed Web application for deploying VAE model. This allows user to generate music using GUI control.

6.1. Limitations

This project is highly planned and acted upon from the beginning. Nevertheless, the project had to face some of the limitations due to various factors. Different aspects of the projects such as nature of data, algorithms have their own limitations. Limitation of each architectures are already discussed previously. Some of the limitations faced by the project are:-

1. MIDI is easy to handle and operate upon computationally. However choosing a discrete representation with MIDI file as data format leads to an inevitable loss of information from the original continuous audio.
2. A structural limitation is that the music produced has a fixed size. One cannot produce longer or shorter length of music.
3. Western music is more readily available across web than Nepali music. Since deep learning architectures require large data for good generalization, we could not use Nepali music for training. So trained model cannot generate Nepali music.

6.2. Future Enhancements

It is the nature of projects in the field of computer science and information technology to require changes and modifications as demand changes and technology advances.

1. Research about the generation of music for Nepali instruments and its possible implementation.
2. Add more visualization/comparison tools for the generated music.

3. Deployment of app on the cloud.
4. Use of database model for storing the user credentials on the app for personalized experience.

References

[1] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation–a survey. *arXiv preprint arXiv:1709.01620*, 2017.

[2] Classical Music MIDI Dataset. "classical music midi dataset". URL `http://www.piano-midi.de/`.

[3] Diogo de Almeida Mousaco Pinho. Music generation using generative adversarial networks, 2018.

[4] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment, 2017.

[5] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[7] Rui Guo, Ivor Simpson, Thor Magnusson, Chris Kiefer, and Dorien Herremans. A variational autoencoder for music generation controlled by tonal tension. *arXiv preprint arXiv:2010.06230*, 2020.

[8] David Kang, Jung Youn Kim, and Simen Ringdahl. Project milestone: Generating music with machine learning. *Stanford University, CA, USA*, 2018.

[9] Sang-gil Lee, Uiwon Hwang, Seonwoo Min, and Sungroh Yoon. Polyphonic music generation with sequence generative adversarial networks. *arXiv preprint arXiv:1710.11418*, 2017.

[10] Fanny Roche, Thomas Hueber, Samuel Limier, and Laurent Girin. Autoencoders for music sound modeling: a comparison of linear, shallow, deep, recurrent and variational models. *arXiv preprint arXiv:1806.04096*, 2018.

[11] Joseph Weel, Bachelor Opleiding Kunstmatige Intelligentie, and E Gavves. Robomozart: Generating music using lstm networks trained per-tick on a midi collection with short music segments as input. 2016.

[12] Adrien Ycart, Emmanouil Benetos, et al. A study on lstm networks for polyphonic music sequence modelling. 2017.

[13] Yi Yu and Simon Canales. Conditional lstm-gan for melody generation from lyrics. *arXiv preprint arXiv:1908.05551*, 2019.

Appendix

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 6.1: GAN Algorithm as mentioned in GAN paper by Ian Goodfellow, et. al. (2014)

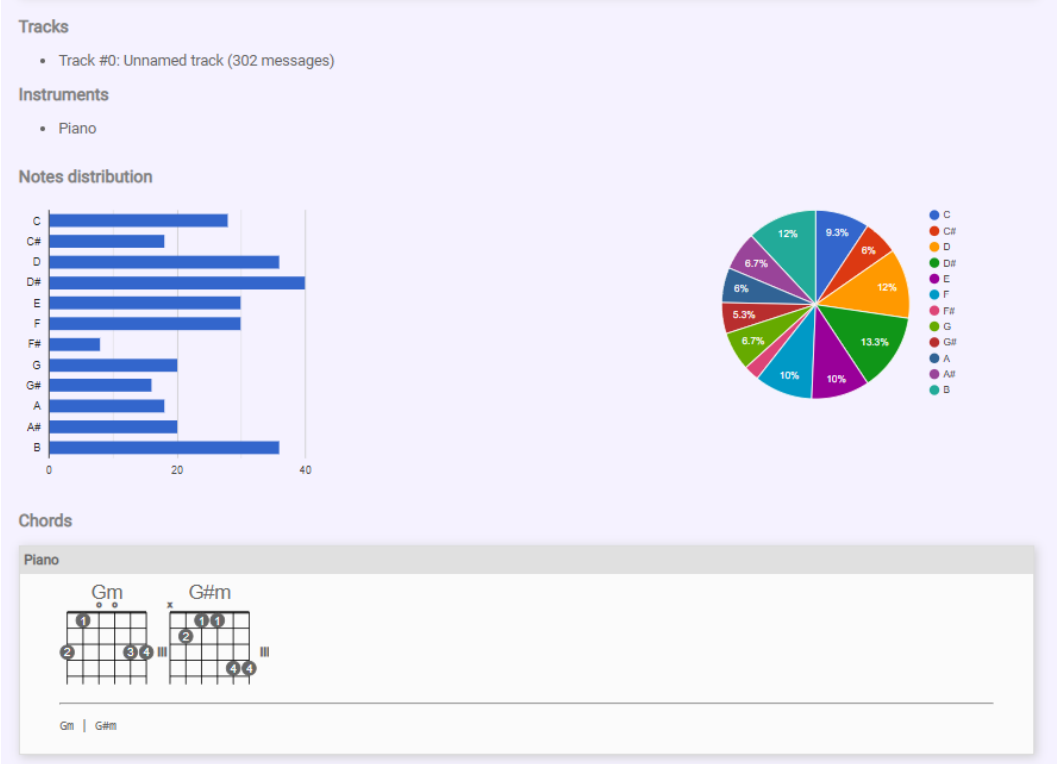


Figure 6.2: Notes Distribution in Generated MIDI file trained for 200 epochs

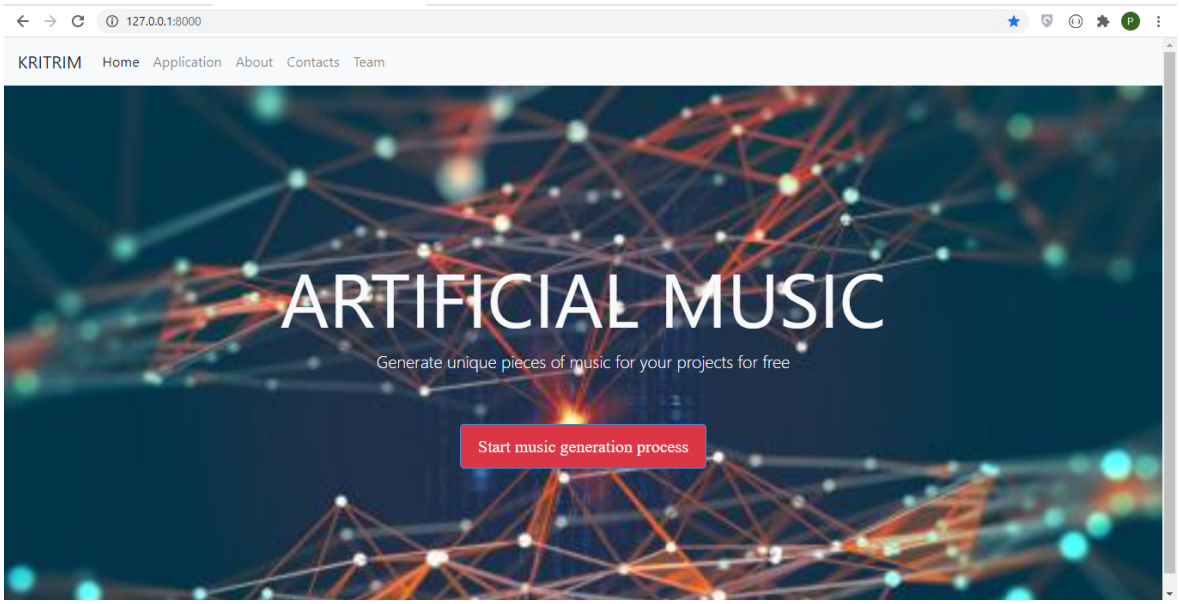


Figure 6.3: Home Page

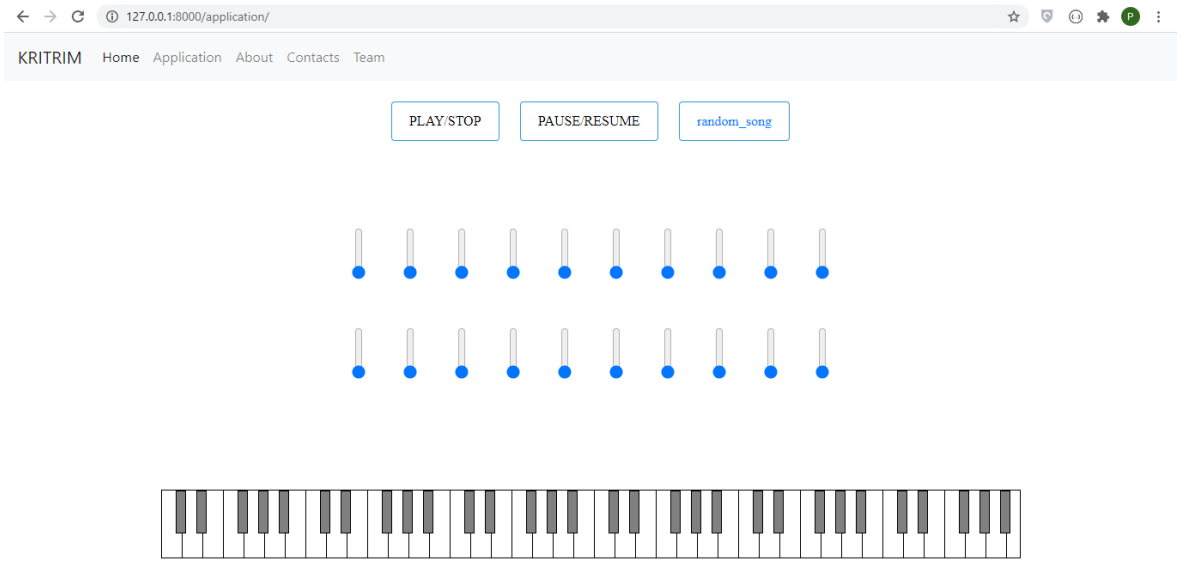


Figure 6.4: Application GUI

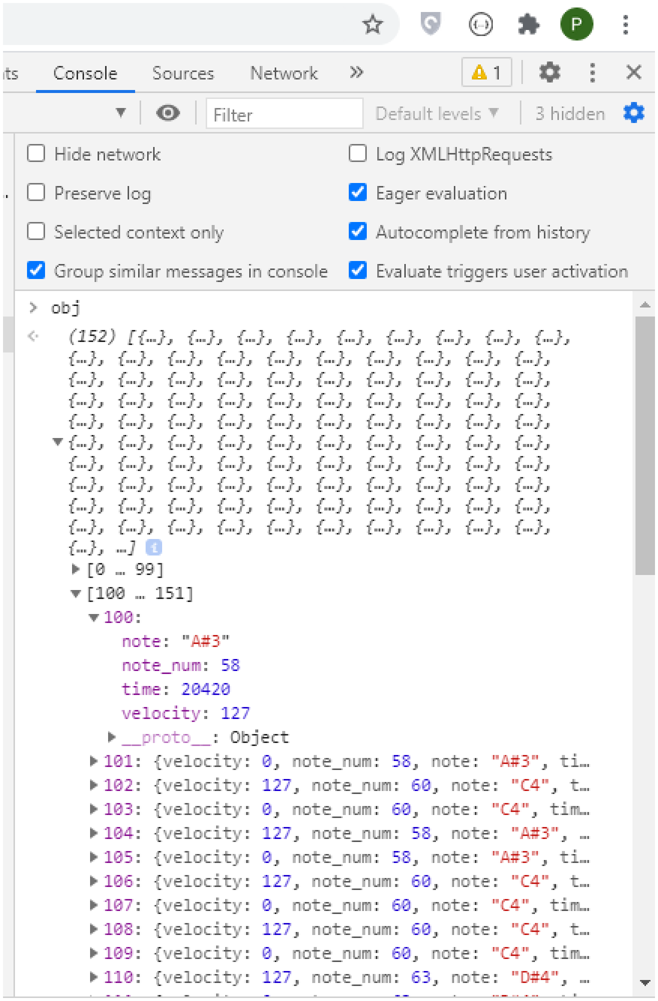


Figure 6.5: JSON response on browser console