University of Nevada, Reno

Rewind: A Music Transcription Method

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Science and Engineering

by

Chase Dwayne Carthen

Dr. Frederick C. Harris, Jr., Thesis Advisor
Dr. Richard Kelley, Thesis Co-Advisor

May, 2016

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

CHASE DWAYNE CARTHEN

Entitled

Rewind: A Music Transcription Method

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Dr. Frederick C. Harris, Jr., Advisor

Dr. Richard Kelley, Committee Member

Dr. Tomasz J. Kozubowski , Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May, 2016

Abstract

Music is commonly recorded, played, and shared through digital audio formats such as wav, mp3, and various others. These formats are easy to use, but they lack the symbolic information that musicians, bands, and other artists need to retrieve important information out of a given piece. There have been recent advances in the Music Information Retrieval (MIR) field for converting from a digital audio format to a symbolic format. This problem is called Music Transcription and the systems built to solve this problem are called Automatic Music Transcription (AMT) systems. The recent advances in the MIR field have yielded more accurate algorithms using different types of neural networks from deep learning and iterative approaches. Rewind's approach is similar but boasts a new method using an encoder-decoder network where the encoder and decoder both consist of a gated recurrent unit and a linear layer. The encoder layer of Rewind is a single layer autoencoder that captures the temporal dependencies of a song and produces a temporal encoding. In other words, Rewind is a web app that utilizes a deep learning method to allow users to transcribe, listen to, and see their music.

Dedication

I dedicate this thesis to my family and friends who have supported me.

Acknowledgments

I would like to thank my Adviser, Dr. Frederick C. Harris, Jr., and my Co-Advisor Dr. Richard Kelley, and committee member Dr. Tomasz Kozubowski for their time and suggestions. I would like to thank Vinh Le for his help in creating the front end of Rewind. I would also like to thank Zachery Newell for keeping the cubix machine running and for providing me a web node for hosting the Rewind web service and website. I would like to thank all members of the call, HPCVIS lab, and the CIL lab. Lastly, I would like to thank my family for their support.

This material is based in part upon work supported by: The National Science Foundation under grant number(s) IIA-1329469, and by Cubix Corporation through use of their PCIe slot expansion hardware solutions and HostEngine. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or Cubix Corporation.

Contents

Abstract **i**

Dedication **ii**

Acknowledgments **iii**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

2 Background **3**

2.1 Automatic Music Transcription 3

2.1.1 Overview 3

2.1.2 Data Representations of Music 3

2.1.3 AMT Evaluation Metrics 7

2.1.4 AMT Approaches 8

2.2 Deep Learning 8

2.2.1 Overview 8

2.2.2 Long Short Term Memory: LSTM 9

2.2.3 Gated Recurrent United: GRU 10

2.2.4 Encoder-Decoder Networks 11

2.2.5 Libraries and Frameworks 13

2.2.6 Past Work in AMT 14

2.3 Web 14

2.3.1 Web Frameworks 14

2.3.2 Audio in the Web 15

2.3.3 JQuery and Other Javascript Libraries 15

3 Rewind **16**

3.1 Overview 16

3.2 Functional and Non-Functional Requirements 16

3.3 Use Case Modeling 17

3.4 Architecture 21

4 Theory and Implementation **24**
4.1 Overview . . . 24
4.2 Rewind's Models . . . 24
4.2.1 Overview . . . 24
4.2.2 Data Sets and Representation . . . 24
4.2.3 Models . . . 26
4.2.4 Training Rewind's Encoder and Decoder Networks . . . 27
4.2.5 Implementation . . . 28
4.2.6 Auto-Correlation Method . . . 29
4.2.7 Difficulties . . . 29
4.3 Web Service . . . 30
4.4 Website . . . 30
4.4.1 Overview . . . 30
4.4.2 Implementation . . . 30
4.4.3 Web Synthesizer and Piano Roll . . . 32

5 Results **34**
5.1 Overview . . . 34
5.2 Results and Discussion . . . 34

6 Conclusions and Future Work **38**
6.1 Conclusions . . . 38
6.2 Future Work . . . 39

Bibliography **41**

List of Tables

3.1 Rewind's Functional Requirements 17
3.2 Rewind's Non-Functional Requirements 17

5.1 Rewind's results at 10 ms stride for the spectrogram where 1 is the proposed model and 2 is the rnn-nade [7]. 36
5.2 Rewind's performance on the Maps dataset compared to [40] at 10 ms. 36
5.3 Rewinds results at a 50 millisecond stride for the spectrogram where 2 is the proposed model and 1 is the Simple Auto-Correlation model. . 36

List of Figures

2.1 An example of a raw audio file. 4
2.2 An example of a spectrogram. 5
2.3 An example of sheet music. 6
2.4 An example of a piano roll. 6
2.5 An picture of a LSTM that consists of input gate, hidden gate, a cell, and a forget gate [29]. 10
2.6 A picture of a Gated Recurrent Unit (GRU) and its layout consisting of a reset switch, update gate, activation, and a candidate activation [29]. 11
2.7 A picture of an autoencoder [37]. 12
2.8 A picture of a encoder-decoder network with a context C demonstrated between the encode-decoder network [13]. 13

3.1 A Use Case Diagram of Rewind. 19
3.2 The Architecture of Rewind. 22

4.1 Encoding generated by the encoder network. 27
4.2 A diagram of Rewind's web service. 31
4.3 A screenshot of the website with a piano roll. 32
4.4 A screenshot of piano roll notes lighting up. 33

5.1 A comparison between two spectrograms: output of encoder model(top) vs actual(bottom). 37

Chapter 1

Introduction

Many musicians, bands, and other artists make use of MIDI, a symbolic music instruction set, in popular software to compose music for live performances, portability across other formats, and recording. However, most music is often recorded into raw formats such as Wav, MP3, OGG, and other digital audio formats. These formats do not often contain symbolic information, but may contain some form of metadata that does not typically include symbolic information. Symbolic formats, such as sheet music have been used by bands, choirs, and artists to recreate or perform songs. These symbolic formats are effectively the spoken language of music that can be retranslated back into sound. Communities such as Mirex are actively working many different problems on retrieving information from music so that creating, categorizing, and extracting information is easier. The Symbolic format is not only portable, but can be leveraged for doing different types of analysis such as genre classification, artist classification, mood detection, and etc. One key thing is that symbolic formats can be used in applications such as FL Studio [23] and others to generate new songs by assigning new sounds to the symbols of the symbolic format. There are existing software out there that can convert a digital audio format into a symbolic format what is known as music transcription. A more accurate tool can be constructed that will allow musicians, bands, and other artists to transcribe their music into symbolic format and allow them to visualize their results in a application.

There are a few music transcription applications that have been built mostly for Windows, Linux, and Mac [3, 24, 27, 49]. There is currently only one existing

website that can actively convert digital audio formats to MIDI at a decent level [32]. Some of these applications offer a way to visualize the converted files in the form of a piano roll. A piano roll is an intuitive visualization of music that does not require a user to learn sheet music, a symbolic format often used by bands and choirs. This visualization can be handy for a user to see if their music came out correctly. These applications allow a user to get a symbolic format of their music that can be used for many different reasons such as changing a song, portability to other applications, live performances, and for generating sheet music. However most of these applications do not seem as accurate as the state of art algorithms from advances in Deep Learning that have contributed to the MIR field.

Thanks to the recent advances in the Deep Learning, the Music Information Retrieval (MIR) and other fields have progressed. Recent advances such as [7, 9, 39] in the MIR field make it possible to create applications that are more accurate than their older counter-parts. With these advances one can create an application that captures the notes accurately and allows one to visualize what the notes would be for a given recording. This can reduce the amount of transcription time for music and for extracting melodic information from music. Rewind is a tool and method that will make use of a new Deep Learning method, visualize the results of the transcribed file, and allow the user to edit transcribed results.

The following is structured as follows. Chapter 2 covers background related to the MIR, Deep Learning, and Chapter 3 discusses the implementation and design of Rewind tool. Chapter 4 explains the theory and implementation behind rewinds method. Chapter 5 gives the results of the Rewind method. Finally Chapter 6 concludes and details future direction that Rewind can take.

Chapter 2

Background

2.1 Automatic Music Transcription

2.1.1 Overview

Automatic Music Transcription (AMT) is the process of converting an acoustic musical signal into some form of music notation [18]. This is a sub-problem of the Music Information Retrieval field. The overarching goal of this field is to create an AMT system that can produce complete scores [18]. With a complete system, it will be easy to extra information of music for other studies. Many have tackled this problem in different ways with different representations of acoustic signals and there has been a great deal of research using Deep Learning to transcribe music.

2.1.2 Data Representations of Music

There are many different representations of acoustic signals that are often used in AMT systems. The three levels that are commonly used in AMT systems are at a stream, frame, and note. The stream level is simply a raw acoustic signal, an example of which can be seen in Figure 2.1. Many systems often use a magnitude spectrogram generated from a fast Fourier transform (fft) representation for audio. This is often called the frame level, because a spectrogram is comprised of frequency information in multiple frames. A spectrogram is demonstrated in Figure 2.2. At the note level, the representation is mainly comprised of notes, see example in Figure 2.3. Most models are done at the frame level because it contains frequency information that is vital for

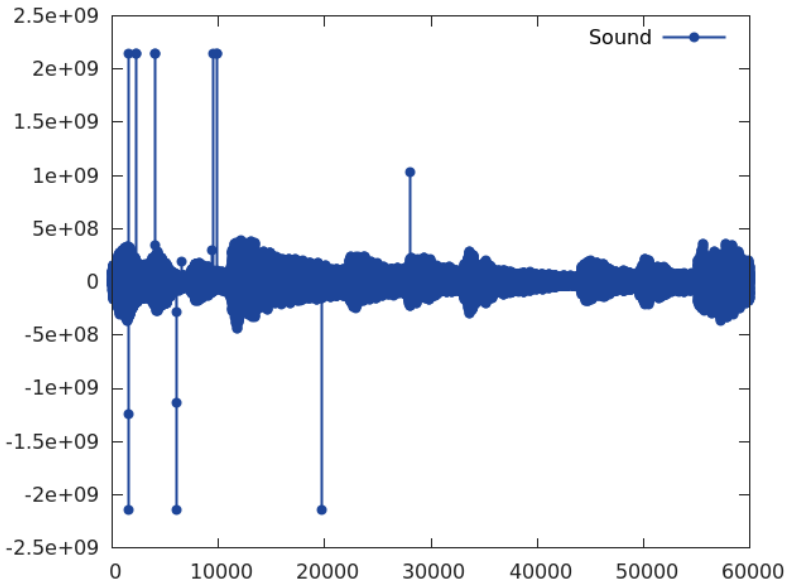

Figure 2.1: An example of a raw audio file.

predicting the notes of an acoustic signal. It is even possible to create a simple auto correlation model, to be discussed later. These audio formats are used as input into AMT systems, with the spectrogram being used most often.

Spectrograms have been used for other problems, such as speech recognition, genre classification, emotion detection, and other problems. The spectrogram is constructed from the magnitude of the short fast Fourier transform (stft) placed on the log scale. The spectrogram is in the frequency domain as opposed to the time domain of the original audio signal. One key issue in using spectrograms is the trade-off between frequency and time resolution [42]. Most projects choose a frequency resolution, which covers all piano notes and has an adequate time resolution for most type of notes. The frequency and time resolution is determined by the window size and stride of the spectrogram. In [9] they experiment with many different window sizes and the found that the best window size is 100 ms given a song at sample rate 44.1 khz. Other papers have chosen a sample rate around this window size or greater. This sample rate or greater covers most piano notes at 10 Hz per bin of the spectrogram. An adequate sample rate is required to get a decent frequency resolution for

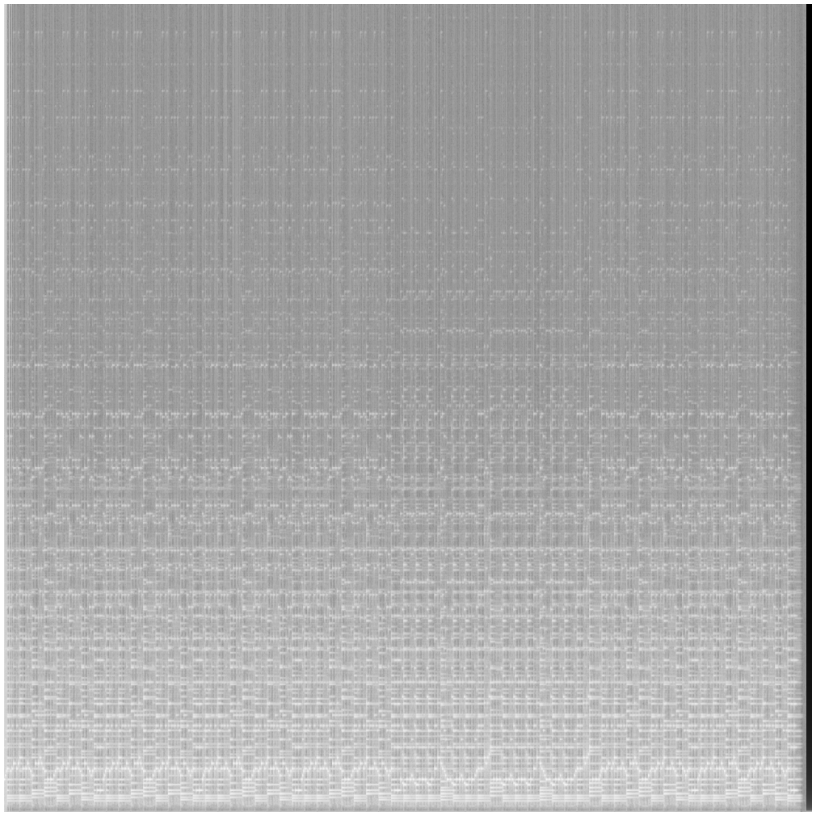

Figure 2.2: An example of a spectrogram.

converting to symbolic formats.

Symbolic formats are piano rolls, sheet music, or midis. These symbolic formats typically represent frequencies and silence with symbols. The intensity or loudness may be represented with words, velocity values of a midi, or even the color of a piano roll. Examples of a piano roll, sheet music, and midi are shown in Figure 2.4.

Most AMT systems generate piano rolls or midi due to ease of generating these formats. However, sheet music is more tricky due to having more rules and the requirement of finding the accurate notation for the audio signal. It is easy to create an AMT system due to many midis being existent and those midis can be synthesized into audio digital formats.

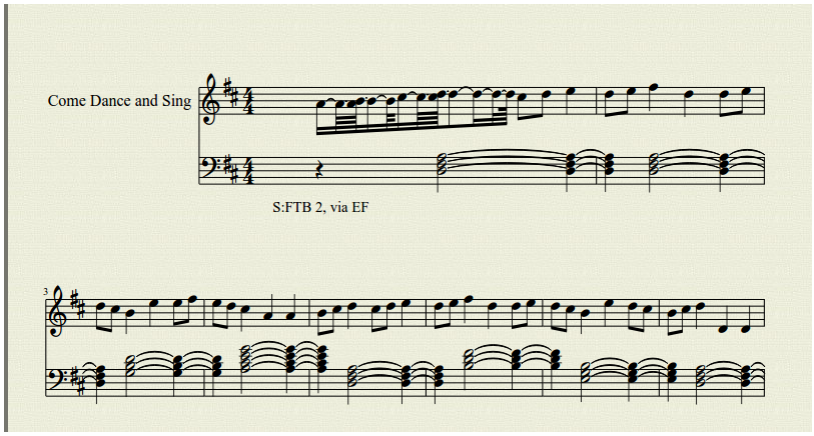

Figure 2.3: An example of sheet music.

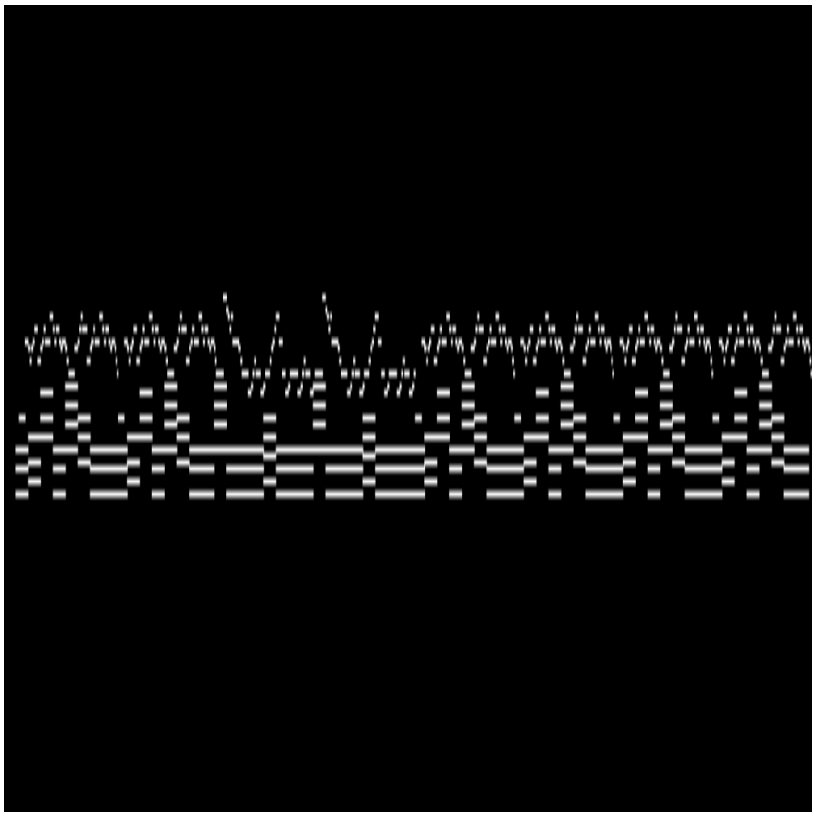

Figure 2.4: An example of a piano roll.

2.1.3 AMT Evaluation Metrics

Many AMT systems evaluate their effectiveness by means of various metrics, which include recall, accuracy, precision, and f-measure from [5]. These important metrics are commonly used in language transcription to determine how well a system translates a given language. In these systems and in music transcription the true positive, false positives, true negatives, and false negatives are used to compute the previously stated metrics. True positives are classifications that are detected as correct positive, while true negatives are classifications that are correct and negative. False negatives and false positives are the exact opposite of true positives and true negatives. Unlike language transcription, which requires classifying the correct word at a given time, music transcription requires classification of the correct set of fundamental frequencies at a given time. Classifying fundamental frequencies is difficult due to the requirement of classifying multiple notes. A midi representation has at most 128^2 possible combinations due to all 128 notes that can be on or off. All these metrics are important for determining how good a AMT system is.

Precision determines how relevant a transcription is given irrelevant transcriptions in the frame. It is defined as follows:

$$Precision = \frac{\sum_{t=1}^{T} TP(t)}{\sum_{t=1}^{T} TP(t) + FP(t)} \tag{2.1}$$

Recall is the percentage of relevant music transcribed, and is given by Equation 2.2.

$$Recall = \frac{\sum_{t=1}^{T} TP(t)}{\sum_{t=1}^{T} TP(t) + FN(t)} \tag{2.2}$$

The accuracy determines the correctness of a transcription, and is given by equation 2.3.

$$Accuracy = \frac{\sum_{t=1}^{T} TP(t)}{\sum_{t=1}^{T} TP(t) + FP(t) + FN(t)} \tag{2.3}$$

While the F-measure determines the overall quality between the precision and recall.

$$F - measure = \frac{2 * precision * recall}{precision + recall} \tag{2.4}$$

2.1.4 AMT Approaches

Many approaches have been taken for AMT at the frame level, slightly less at the note, and even fewer at the stream level. From [5] we learn that there are two major ways that this problem is currently being solved which are taking a classification approach where all notes are taken into consideration at once and the other type of approach iteratively determines the best set of frequencies by canceling out the source signal with a certain frequency. These two approaches can be expanded from [18], where it breaks down into further approaches such as time domain methods, frequency domain methods, iterative spectral subtraction, spectrogram decomposition, full spectrum modeling, spectral peak modeling, and classification-based methods. Many of these methods are used to create an AMT system and some are even combined to produce a more effective system [39]. Even more recently one system has start to use a hybrid method, that consists of an acoustic model and music language model in [41] and makes a full use of deep learning. Deep learning has become popular in solving problems such as latent semantic analysis, speech recognition, computer vision, and other problems consisting of large datasets. Both approaches aforementioned are classification-based methods that makes use of the frequency domain with the addition of predicting the probabilities of notes. The next section cover deep learning and how it is used for classification and for predicting the probabilities of notes.

2.2 Deep Learning

2.2.1 Overview

Deep Learning is a field of machine learning that makes use of deep neural network architectures in order to solve problems such as speech recognition, AMT, latent semantic analysis, and other important problems. One common issue that prevented the Deep Learning field from progressing is the exploding and vanishing gradient problem, where neural network architectures such as recurrent neural networks (rnn). There have been advances in the Deep Learning field due to increased performance

in hardware and improvement algorithms. These improvements include the creation of the LSTM [21, 22], GRU, and optimizers such as adam[25], rmsprop [46], adagrad, and adadelta [50]. These advances have allowed for many different fields to advance with the availability of different datasets.

2.2.2 Long Short Term Memory: LSTM

LSTMs were created in response to the vanishing and exploding gradient problem that was a common problem with rnns. LSTMs do not have these issues because they can remember sequences due to their memory cells. A LSTM consists of an output gate, forget gate, cell, input gate, and a hidden layer. A diagram of the LSTM can be seen in Figure 2.5. It is capable of remembering long term sequences and capturing temporal dependencies between temporal events. It makes use of a memory cell to keep track of previous values, a forget gate to determine whether to forget previous values, and a hidden gate to create a hidden state. LSTMs have had major success in problems such as AMT, translation, image caption generation, and for modeling context. LSTMs have also been used in encoding and decoding sequences. LSTMs work relatively well but it has been found that a GRU is comparable to an LSTM in audio tasks.

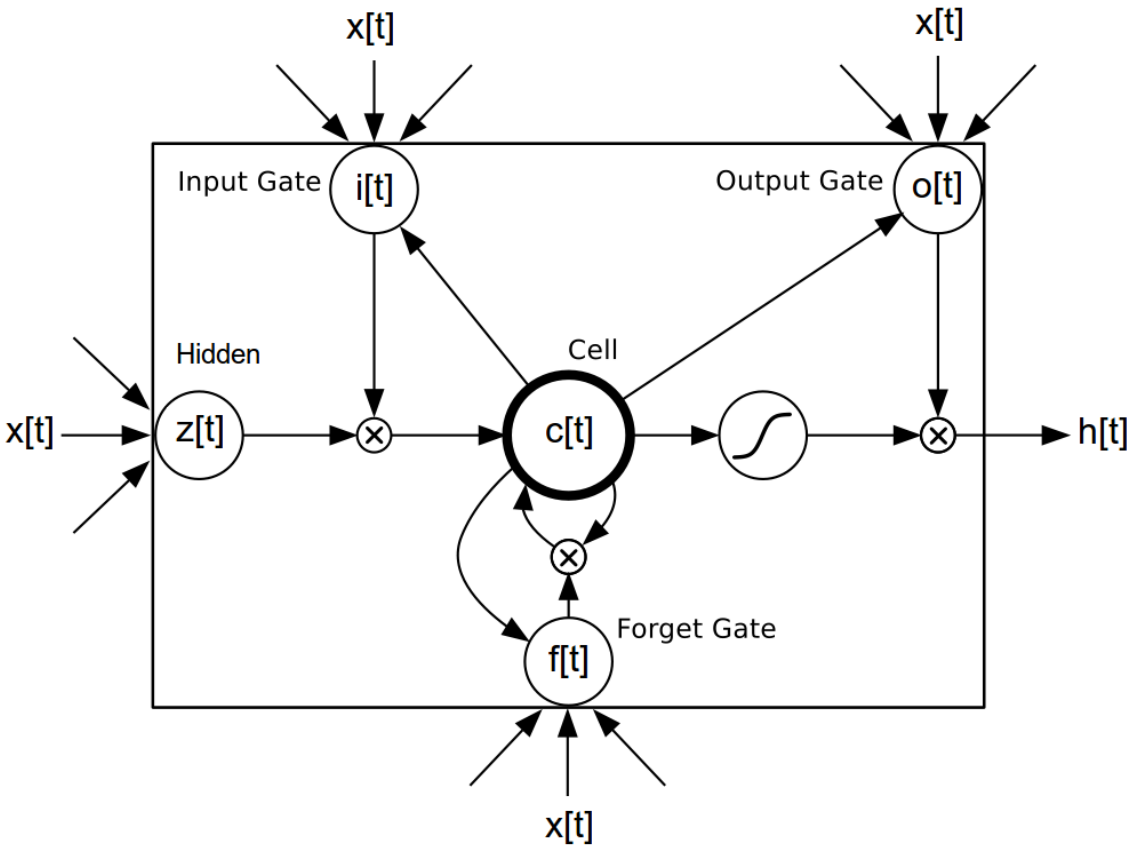

Figure 2.5: An picture of a LSTM that consists of input gate, hidden gate, a cell, and a forget gate [29].

2.2.3 Gated Recurrent United: GRU

The Gated Recurrent United created and proposed in [15], unlike LSTMs does not have a memory cell to contain old information, but it still takes into consideration the previous information. The GRU was created as a simplification of the LSTM unit. A layout of the GRU is demonstrated in Figure 2.6 and consists of a reset gate, update gate, activation, and a candidate activation. The GRU is able to look into temporal considerations and is able to remember previous activations as well. The GRU has been found to have similar performance to an LSTM in [14] and is comparable to an LSTM. It controls the amount of updates it recieves with its reset and update gates. Even though it does not contain a memory, it has been proven that it can do just as well as the LSTM. The LSTM and GRU are both good at modeling temporal

dependencies and remembering sequences. They can be used in autoencoder like [15, 43] and be used to extract information from data.

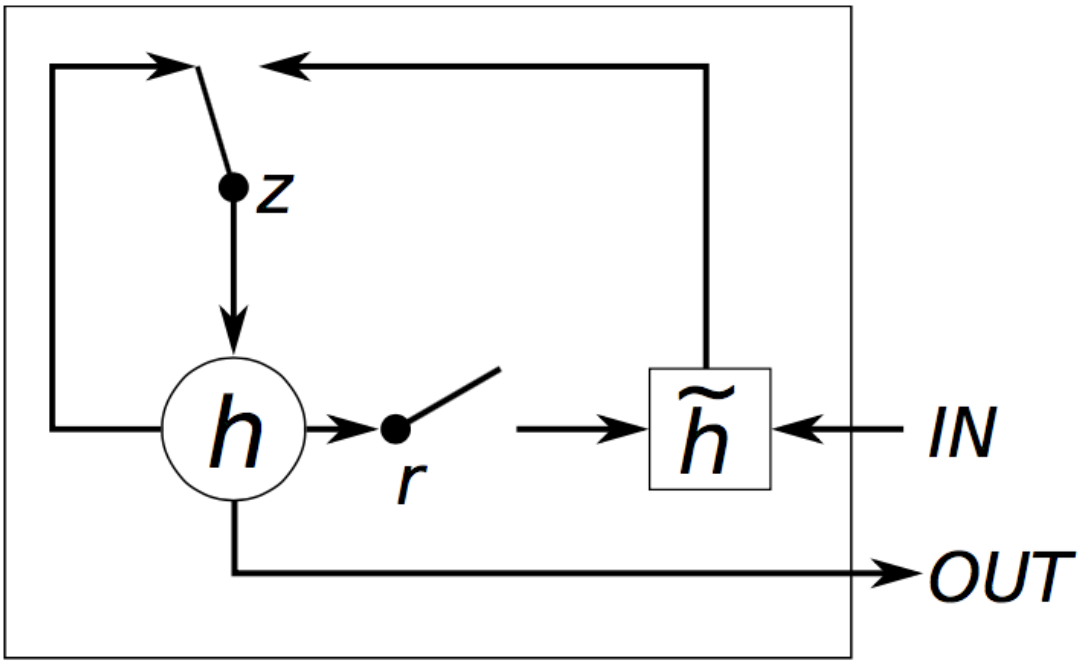

Figure 2.6: A picture of a Gated Recurrent Unit (GRU) and its layout consisting of a reset switch, update gate, activation, and a candidate activation [29].

2.2.4 Encoder-Decoder Networks

Encoder-Decoder networks consist of a encoder network and decoder network that have been used for unsupervised learning in terms of autoencoders [43, 37, 48, 4], translation [14], or captioning generation for images, video clip description, speech recognition [13, 17] or video generation. A encoder-decoder can be as simple as an autoencoder or more complex. Autoencoders are commonly used for unsupervised learning by learning the identity of the data and an encoding is produced by the encoder of the autoencoder that contains learned features. An example of an autoencoder is shown in Figure 2.7. An autoencoder is powerful for learning features contained within a dataset, and extract features if they are stacked. However there

are more complex encoder-decoder networks in [14, 13, 17], where they learn a context and map English to French.

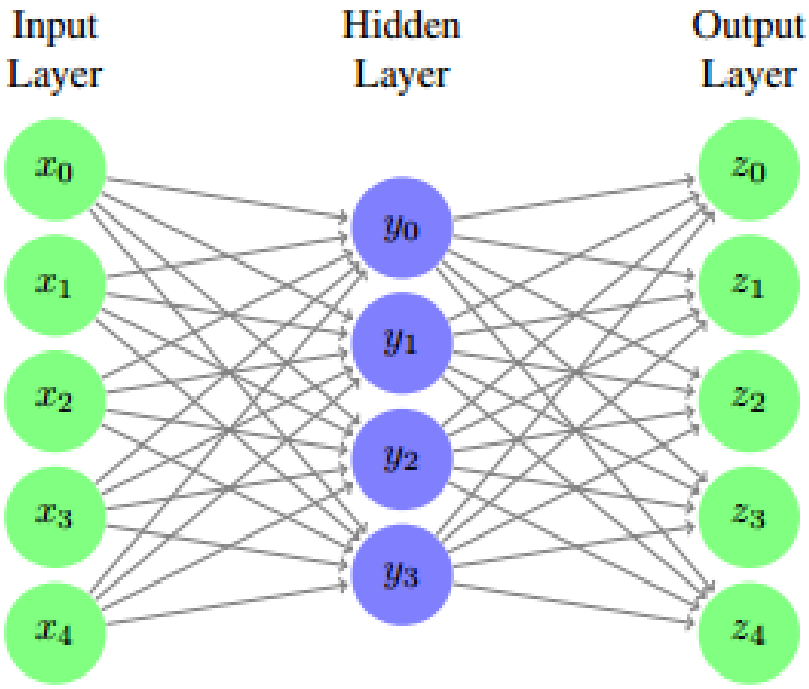

Figure 2.7: A picture of an autoencoder [37].

Another type of encoder-decoder network is a network that is least concerned about learning the identity but for mapping the input to a specific output like image caption generation, video clip description, or translation. These type of networks are commonly used to learn the input and a context associated to it with an encoder. The decoder's job based a context generated by the encoder is to produce an output like a image caption, video clip description, or etc. An example layout of this network is demonstrated in Figure 2.8. These networks have proven to be beneficial, and are state of the art.

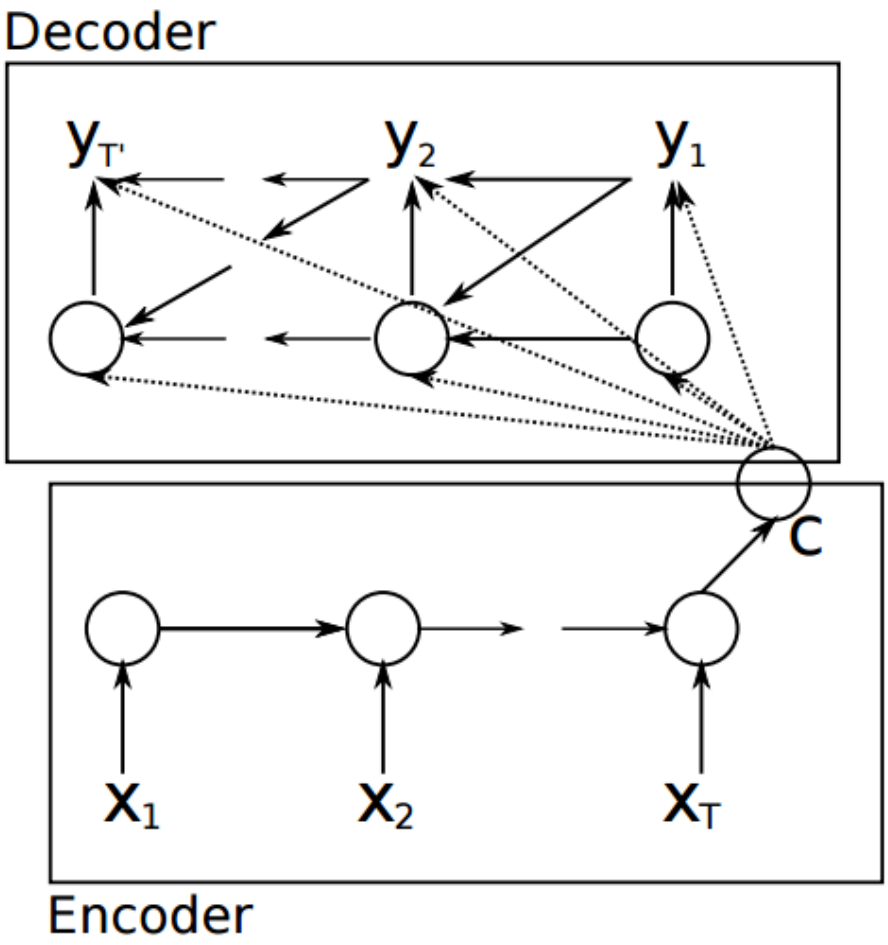

Figure 2.8: A picture of a encoder-decoder network with a context C demonstrated between the encode-decoder network [13].

2.2.5 Libraries and Frameworks

Many frameworks and libraries have been built to make deep learning possible, such as Torch [36], Theano, Keras, and cuDNN. These libraries have included CUDA integration, allowing for neural networks to be trained on GPUs at a much faster speed. The incorporation of GPUs have made it possible to train networks faster and create larger networks without the need of a large cluster. Many more people have stepped in the deep learning field. Rewind utilizes Torch for its implementation. Libraries that are specific to Torch shall be discussed later in this thesis.

2.2.6 Past Work in AMT

There has been some recent work in AMT within the deep learning field to do recognition at the frame level and very few at the note level. Most of these papers have utilized a spectrogram representation in order to transcribe music to a piano roll or midi. There has been some other work that does everything at the stream level that is covered in this tutorial [18], but in this thesis we are mainly covering the frame level. There has been some work using LSTMs and semitone filterbanks to transcribe music [9]. In [39] the idea of an an acoustic model, that converts an audio signal to a transcription, and a music language model, that improves a transcription overall, is introduced. This paper introduces using a music language model to improve the accuracy of a transcription of a acoustic model likes [9] and others as well. Boulanger-Lewandowski in [7] uses a deep belief network to extract features from a spectrogram and utilizes a RNN to create a transcription along with a innovative beam search to transcribe music. Boulanger-Lewandowski's beam search is possible thanks to the generative properties of the deep belief netowrk that is merely a bunch of staked restricted Boltzman machines (RBM). This beam search is also utilized in combination with rnn-nade as a music language model and an acoustic model that uses a deep neural network and rnn for recognizing frames [41]. The acoustic and music language model were both effective on training on the maps dataset. A follow-up paper produces a hash beam search that finds a more probable transcription in a fewer epochs [40]. The idea behind an acoustic model and music language will be explained later in Chapter 4.

2.3 Web

2.3.1 Web Frameworks

There exist several web frameworks in many languages, and some web frameworks are simple development platforms. An example of web frameworks that are easy development platforms and get running are Django [34] and Flask [35]. Both Django

and Flask use Python as the language to write modules for frameworks. Python has many libraries and a community that is actively adding new libraries everyday. Flask works well writing small microservice applications that are used for serving simple web services. Django works well for writing a scalable web application including easy database integration and support for adding security. These web frameworks are useful for designing web services and web sites.

2.3.2 Audio in the Web

Recently Chrome and FireFox have been adding or creating audio frameworks into web browsers such as WebAudio [31] or WebMidi [47]. These frameworks are starting to be used by various parties to create proof of concept applications or actual applications on the web, such as the online sequencer [30]. There are even libraries being built around web apis, such as WebMidi and midi.js [16]. These libraries and frameworks make it possible to create interesting media, such as Rewind.

2.3.3 JQuery and Other Javascript Libraries

Several Javascript libraries have been written to make it easier to display content, do web requests, and other functionality within the web browser. Remodal, [10] a Javascript library that makes it easier to do CSS animations and create modal windows. Another useful library is jQuery [45], which has a lot of useful functionality and can do most web requests or other functionality with less lines of code. An offshot from the jQuery is a library called jQueryUI [44], which is used to easily create user interface elements in the web browser. All of these libraries are utilized by many web applications to create web sites.

Chapter 3

Rewind

3.1 Overview

Rewind is both, a method and tool, meant to be used for transcribing digital audio music in a web interface. This web interface is meant to display the results of a transcription and to allow the user to download the result. What is unique about this web interface and transcription is that the user can play the resulting transcription. In this chapter the requirement, use cases, and architecture for Rewind, will be discussed.

3.2 Functional and Non-Functional Requirements

The functional requirements for Rewind are detailed in this Section, and the requirements are demonstrated in Table 3.1. The non-functional requirements for Rewind are detailed in this Section as well and the requirements are demonstrated in Table 3.2.

Table 3.1: Rewind's Functional Requirements

Functional Requirements	
Name	Description
FR1	The system shall transform transcriptions into a piano roll.
FR2	The system shall be able transcribe an audio digital format into a symbolic format as midi.
FR3	The system shall be able to resample audio for input into the models.
FR4	The system shall allow the user to hear the results of their transcription.
FR5	The system shall be able to preprocess datasets for training the models of the system.
FR6	The system shall be able to pause and play a transcribed result.
FR7	The system shall upload audio files.
FR8	The system shall download midi files.
FR9	The will allow the use to edit a transcription.

Table 3.2: Rewind's Non-Functional Requirements

Non-Functional Requirements	
Name	Description
NFR1	The models of the system shall be implemented in a machine learning library like torch.
NFR2	The system shall be well documented.
NFR3	The system shall have limits on music size that is reasonable.
NFR4	The system shall support most audio formats.
NFR5	The system shall use Django as web server framework and for future expandability.
NFR6	The system will utilize parallel computing such as CUDA to expedite the process of transcription.
NFR7	The system shall be viewable in web browsers such as Chrome or Firefox.
NFR8	The system will be expandable for other problems and areas.
NFR9	The system shall play transcriptions as a piano.

3.3 Use Case Modeling

This section details the use cases of Rewind and covers the different scenarios of Rewind. The use cases were created in the need of generating transcriptions of digital

audio content and to make it easier for users to view these transcriptions. Both the back end of Rewind, being the trained models, and the front end of Rewind, being the Graphical User Interface (GUI) of Rewind, are covered by these use cases.

In the full use case diagram shown in Figure 3.1, there are four actors being the: User, Developer, Web Service, and the Rewind Server. The User are those who are interest in creating a transcription of a digital audio song. The Developer is one whom that is expanding and or improving the accuracy of Rewind. The Web Service is a service that allows the Rewind client to convert a digital audio format into transcription. The Rewind Server serves a website to the Rewind client. The following sections detail the use cases of Figure 3.1.

Play/Pause Playback

The user has the option to pause or playback a given transcription in the Rewind client.

Download Transcription

When a transcription has been received from the server, the user may download a transcription that one had requested.

Inspect Piano Roll

The user may look around the piano roll within the Rewind client.

Get Information About Project

The Rewind client will provide the user the option to get information about the Rewind project and how the project works.

Upload Audio File

The user in this use case will upload a file that they wish to transcribe.

Receive Transcription

When the server has received a transcription from the Web Service, the Rewind client will receive the transcription for playback and visualization.

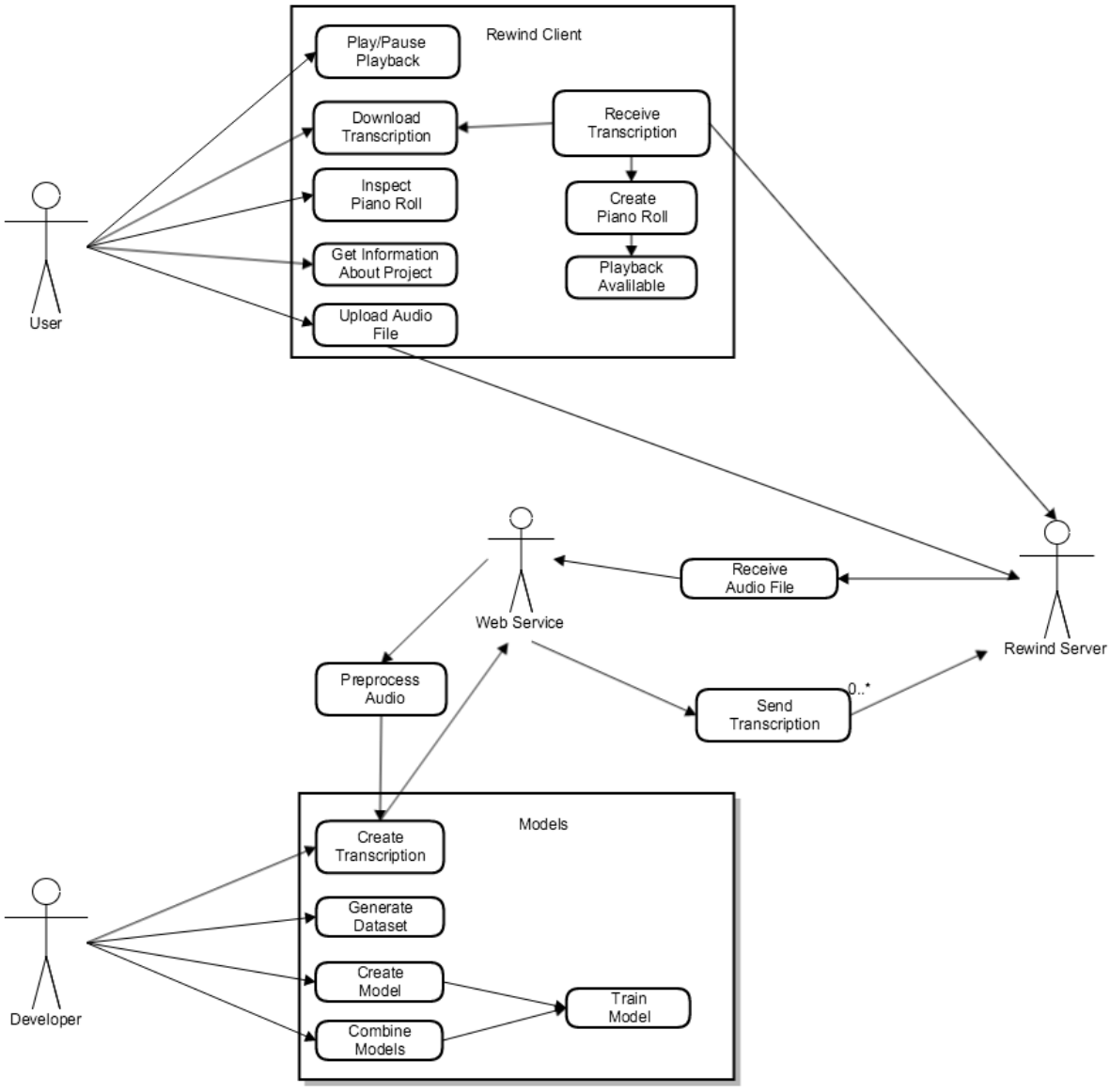

Figure 3.1: A Use Case Diagram of Rewind.

Create Piano Roll

After receiving the transcription the rewind client will build a piano roll transcription for the user to see.

Playback Available

After the piano roll has been inside of the Rewind client, then the client will allow the user to playback the transcription and will let the user know that playback is available.

Receive Audio File

In this use case, the web service receives an audio from the server and is now ready to preprocess the audio file for transcription by the models.

Create Transcription

The create transcription use case can occur in two different ways: one is that the web service sends an audio file to the models for transcription or a developer wishes to test the capabilities of the models and invokes the service.

Send Transcription

When the models have finished transcribing, then the transcription will be sent to the web service where the Rewind server will then the data to client.

Preprocess Audio

The models before they can transcribe any audio have to make sure that the files themselves are the proper format. If they are not the proper, then by default the models will transform the music into the proper format.

Generate Dataset

The developer may wish to generate a new dataset for training the models, which is possible. This is so the developer may tweak Rewind and make its overall transcription accuracy better.

Create Model

The developer is also able to create new models that can be utilized for transcription or research.

Combine Models

The developer may wish to combine multiple models together in order to improve transcription.

Train Models

The developer has the option of training the models in order to determine if the new model is better than current model utilized by the web service.

3.4 Architecture

The architecture consists of multiple parts, which are the client, models and web service, and the server. Each part is unique and has been designed to handle different parts of Rewind's functionality. The models are used for producing transcription, and the web service is used to interface with the model and send outputs to the client through the server. All visualization, downloads, and uploads are handled by the client. The server pushes all content needed to run the website to the client. An overall diagram of the architecture is demonstrated in Figure 3.2.

Models and Web Service: The models and web service component of the architecture are used to process data for training a model, generating transcriptions with a preexisting model to be sent through the web service, and training models. This component contains Rewind's method or AMT algorithm for creating transcriptions of digital audio formats. The web service was created as a way for Rewind's models to send transcriptions to the client. The web service for Rewind was written in Flask [35], as it requires small amount of code to get a web service written.

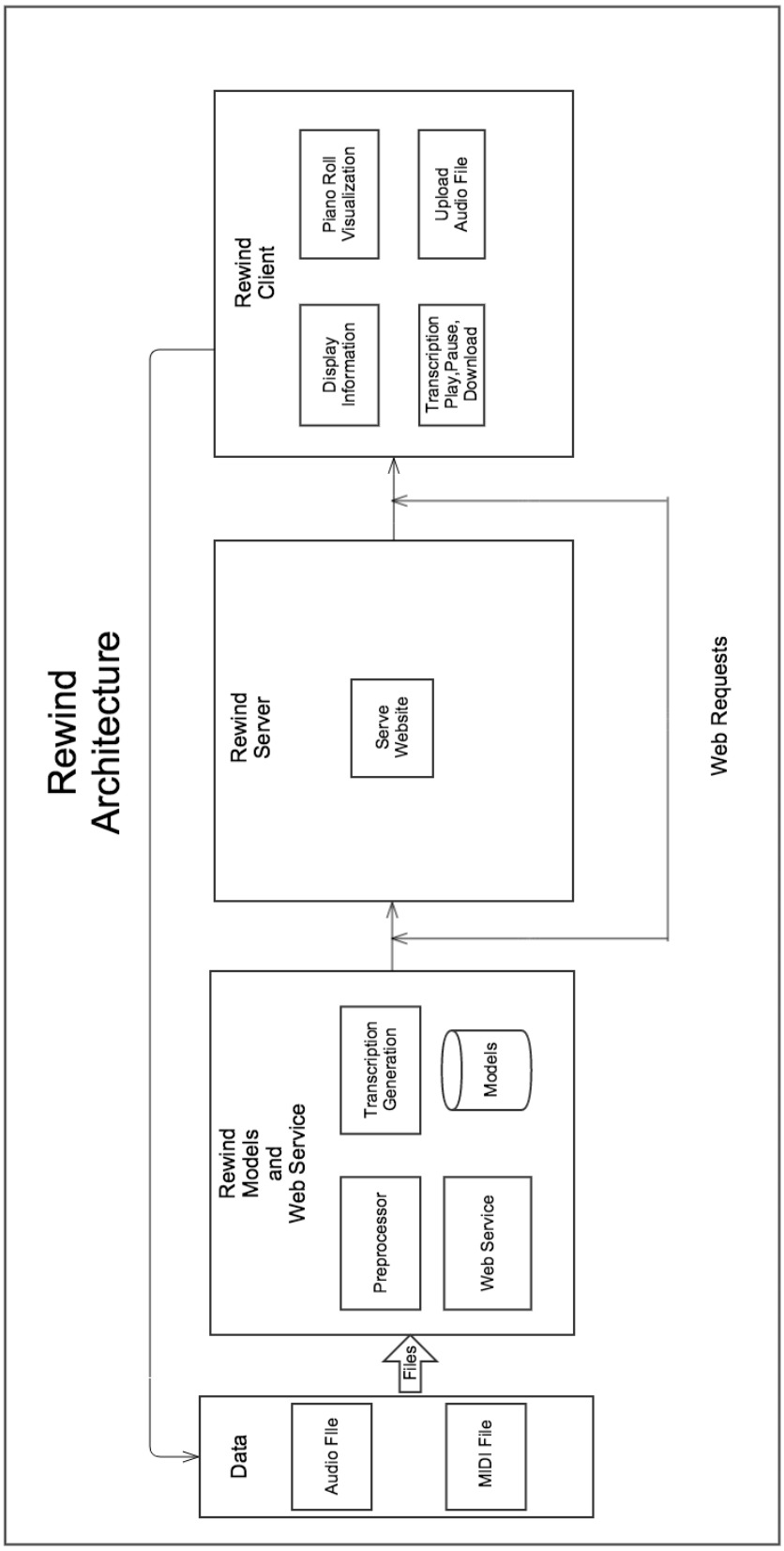

Figure 3.2: The Architecture of Rewind.

Server: Rewind's server was created with Django's web framework [34]. The rewind server serves up the website to the client, which includes all of the html, Javascript, and css files. It also handles sending uploaded audio files to the web service and forwarding the content back too the client.

Client: The client handles creating a piano roll for visualization, uploading audio files to the web service, and giving the ability to download a transcription. The client is a web browser such as Google Chrome, Firefox that is utilized by a user. All sound playback is handled by the client and allows the user to pause and play sounds. The client's job is to light up the notes in the piano roll as the note on hits.

Chapter 4

Theory and Implementation

4.1 Overview

The following chapter covers the theory and implementation behind Rewind. The first section covers the encoder-decoder network of Rewind. The second section covers the web service created for forwarding transcriptions Rewind's server. The third section covers Rewind's server.

4.2 Rewind's Models

4.2.1 Overview

Rewind is very much like other AMT in that it determines the fundamental frequencies of the notes and what notes are on at the frame level. Rewind utilizes a classification based method to determine whether a note is on or off, but with a threshold probability. The following sections layout Rewind's data representation, models, and the difficulties in constructing the method behind Rewind.

4.2.2 Data Sets and Representation

Like most other frame based systems, Rewind utilizes the spectrogram as its main input and a ground truth midi as the target. A multitude of datasets were utilized for training Rewind's models which are: Nottingham [1], JSB Chorales [2], Poliner and Ellis [33], Maps [19], MuseData [11], and Piano.midi.de [26]. All of these datasets were split into 70% for training, 20% for testing, and 10% for validation. These

datasets consisted of midi only or midi with aligned audio that were processed and made into datasets with timidity[20], Torch's audio library [12], and a midi library [6].

Choosing a good sample rate is important for the frequencies exist in the audio. According to the nyquist theorem [42] there are frequencies up to half of the sample rate. This means that the sample rate must be chosen such that it covers all fundamental frequencies in all or most music pieces. For Rewind, a sample rate of 22 kHz was chosen, as it covers most of the fundemental frequencies support by MIDI. In generating the spectrogram's as input a window size of 116 ms with a 10 ms stride or 50 ms stride was chosen for the input representation. The 116 ms was chosen as it has a high frequency resolution and is most likely to give a high accuracy, especially when considering the finding of [9]. In [9] it was found that a window size at 100 ms or higher would produce high accuracy. The 116 ms allows for a frequency resolution of 8 Hz which will cover all of the notes on a piano. The chosen window function was set to be the hann window function [42]. The spectrograms were normalized using the mean and the standard deviation following the literature. All spectrograms were generated with the torch audio library.

Midi's from the aforementioned datasets were used to generate ground truth transcriptions. These transcriptions were aligned based on their actual play times, due to the fact that all audio generated by midi with timidity or aligned audio were close to the MIDI. MIDI's for note on were ascribed as 1 value and note off values were ascribed as 0. The loudness of notes in MIDI were not considered, as the note on and note off were the most important factors. Another consideration that was made in generating the ground truth for spectrograms was that notes could actually exist in frames before there actual playtime due to the overlap caused by the window size of spectrogram. This issue was ignored as the primary problem is to determine the actual note time.

4.2.3 Models

Rewind has two types of models: encoder and decoder model. The encoder and decoder is very similar to the encoder-decoder network in Figure 2.8, referenced in [13, 14, 17]. The encoder model of Rewind utilizes an autoencoder that uses a GRU for its encoder, whose output is squashed by a rectified linear unit and a linear layer for its decoding layer. The decoder model of Rewind utilizes a GRU for the first layer and then a linear layer whose output is squashed by a sigmoid activation function. Both the encoder and decoder networks are trained with different error functions. The rest of this section explains the encoder and decoder utilization by Rewind for training.

The encoder network utilizes an autoencoder to create an encoding for spectrograms, such as the encoding demonstrated in Figure 4.1. These encoding are meant to be a generalization of the spectrogram and to make it easier for the decoder network to learn a transcription. An autoencoder was chosen because a deep neural network (stacked auto encoders) has been used for extracting features from spectrograms in the case of speech recognition [8] and other similar works that utilize deep belief networks (stacked restricted Boltzman machines) have been used to extract features in [28]. In [17], a deep belief network, along with an autoencoder, are used to produce a generative model for spectrograms. All of these papers utilized either a deep belief network or deep neural network or autoencoder, which are used for extracting information and dimensionality reduction. These autoencoders representation can be expanded even further when using networks that have recurrences, such as a GRU or LSTM in the case of [43], where the encoder and decoder of the autoencoder are both LSTMs for learning over video sequences and generating video sequences. Unlike [43], Rewind's encoder model utilizes a linear neural network for the decoder and a GRU for the encoder with a rectified linear unit (ReLU) to squash the output of the GRU. A GRU is utilized as long term memory. Also as mentioned in Section 2.2.3, GRUs are comparable to LSTMs to a degree.

Figure 4.1: Encoding generated by the encoder network.

The decoder network consisted of two types of networks being a GRU with a linear layer and two GRUs stacked onto of each in parallel with a linear layer. Both types of networks are squashed with a sigmoid function. The GRU in both networks was chosen because it produces the lowest error rate. This network's objective function is binary cross entropy, so that this decoder network will learn a distribution of notes where a probability of one indicates a note on and a probability of zero indicates a note off. In [38] binary cross entropy was used for unsupervised learning and clustering using nodes that use the sigmoid function to minimize entropy. In [7, 39, 41] binary cross entropy is used for minimizing the log probability, which also utilizes a sigmoid function to create a binary probabilities, like [38]. The binary cross entropy function is demonstrated in Equation 4.1, where the sum is taken over all distributions [39]:

$$\sum_{i} t_i \log p_i + (1 - t_i) \log (1 - p_i) \tag{4.1}$$

The probabilities constructed from the sigmoid function can be used to construct a MIDI, and are utilized in previously mentioned papers. The decoder network's job is to spit these probabilities for each encoding passed by the encoder network.

4.2.4 Training Rewind's Encoder and Decoder Networks

As mentioned in the previous section, Rewind has encoder-decoder network. This network can be split up into a encoder and decoder. Both the encoder and decoder are trained separately from each other, with two different optimizers being rmsprop [46] and adadelta [50]. These networks are trained separately to allow for future expansion, where other problems can be explored using the encoder part of the encoder-decoder network by simply building another decoder. These optimizers were chosen over sgd because they find a solution much quicker and are less prone to producing wrong

results to local maxima. Both of these networks were trained separately, since the encoding needed from the encoder had to be learned first. After learning the encoding for the encoder, the resulting encoding can be passed into the decoder for generating transcriptions. Both, the encoder model and decoder, were trained differently.

As mentioned in Section 4.2.3, the encoder network is an autoencoder that creates a temporal encoding. This temporal encoding is utilized by the encoder network to be passed to the decoder. This encoder is optimized with rmsprop in order to ensure a quicker convergence. The encoder model is trained using the mean squared error objective function. The idea behind the encoder is to get a temporal linear regression of the spectrogram that is passed in and to capture the most relevant frequencies with the relu activation function.

As mentioned in Section 4.2.3, the decoder network creates binary probabilities based on the encoding passed into it from the encoder network. All binary probabilites that come out of the model are rounded to produce a one or zero to the generated output of the model. This network utilizes binary cross entropy as a loss function and utilizes adadelta to ensure a quick convergence. In training this network the following metrics discussed in Section 2.1.3 are reported: precision, accuracy, recall, and f-measure. These metrics are reported at the frame level and are used as bench marks to determine whether or not a model is good.

4.2.5 Implementation

All of the models implemented for Rewind were implemented in torch and utilize several different libraries from torch. These libraries are rnn [29], torch-audio, midi, nn, cunn, optim, and cutorch. The rnn library was used for its LSTM and GRU implementations. The cunn and cutorch libarires make it possible for Rewind to uses a Nvidia GPU to accelerate computation time. The torch-audio and midi libraries were utilized for converting audio to spectrogram and for generating the ground truth midis. The optim package allowed for the use of state of the art optimizers adadelta and adagrad. Torch was chosen for implementation for Rewind, as it is a relatively

simple machine learning library, and its underlying implementation can be written in C.

4.2.6 Auto-Correlation Method

An auto-correlation method was constructed as a way to implement the web service faster without the need of a fully trained model. This model is very noisy at best, but does manage to extract most of the notes. The process simply creates a spectrogram of the required audio file and then each bin of the spectrogram is normalized with the standard deviation and mean. After these transformations have been made, a threshold is applied, where anything greater than the threshold is a 1 and anything less is a 0. Subsequently, one simply only needs to go to each frequency bin that matches a midi note and extract the frequencies that are on. This auto correlation method is only meant as a test model for a web service. However, in Chapter 5, results are reported for its accuracy in comparison to Rewind's Network.

4.2.7 Difficulties

There are several key issues in designing these models and determining the best data representation for audio. One key issue in choosing the data representation where audio can be represented as a raw signal or spectrogram. A spectrogram has issues of frequency resolution vs temporal resolution. If a spectrogram has a low temporal resolution, then most likely important phrases in the transcription will be missed at the sacrifice of greater frequency resolution. If the audio representation has a low frequency resolution, then it will be difficult to determine what notes are actually being played especially at the lower frequencies. The spectrogram generated by torch-audio uses a short-time fourier transform (stft), that must have the stride rate and window size to be chosen for the desired temporal resolution and frequency resolution. The overlapping frames of the stft, makes it difficult to extract notes due to the decreased time resolution. Rewind only considers things at the frame level and does not worry about extracting the exact note. One key issue in designing the

models is determining a model that will keep track of long term dependencies, which LSTMs and GRUs have done quite well. One last key issue in designing a model for AMT is the fact that most standard midi datasets are western classical music and do not incorporate other genres such as jazz, rock, or other cultural music.

4.3 Web Service

Rewind's web service was implemented in Flask [35] as a small web service that could be utilized by Rewind's server for making transcriptions of uploaded audio files. Flask is good for creating a small microservice web applications or web services. All audio files and transcriptions are sent through post requests. Figure 4.2 demonstrates a diagram of the communication of audio files and transcriptions going in and out of the web service. This web service communicates with the models of Rewind and creates a midi file from the passed in audio file. All transcriptions generated by the web service are piano only. This is meant to make Rewind scalable for other web apps and servers.

4.4 Website

4.4.1 Overview

The following section covers Rewind's website implementation, visualization, and synthesizer. The overall goal of Rewind's website is to provide a front end for a user to visually see the transcription and hear the results of Rewind's models. This front end is meant to provide a way for a user to analyze transcriptions. This website was built as a prototype.

4.4.2 Implementation

Rewind's website was implemented in the Django web framework and utilized the following javascript libraries: remodal, jQuery, jQuery UI, and midi.js and its various dependencies. Django was chosen for Rewind because it allows Rewind to be scalable

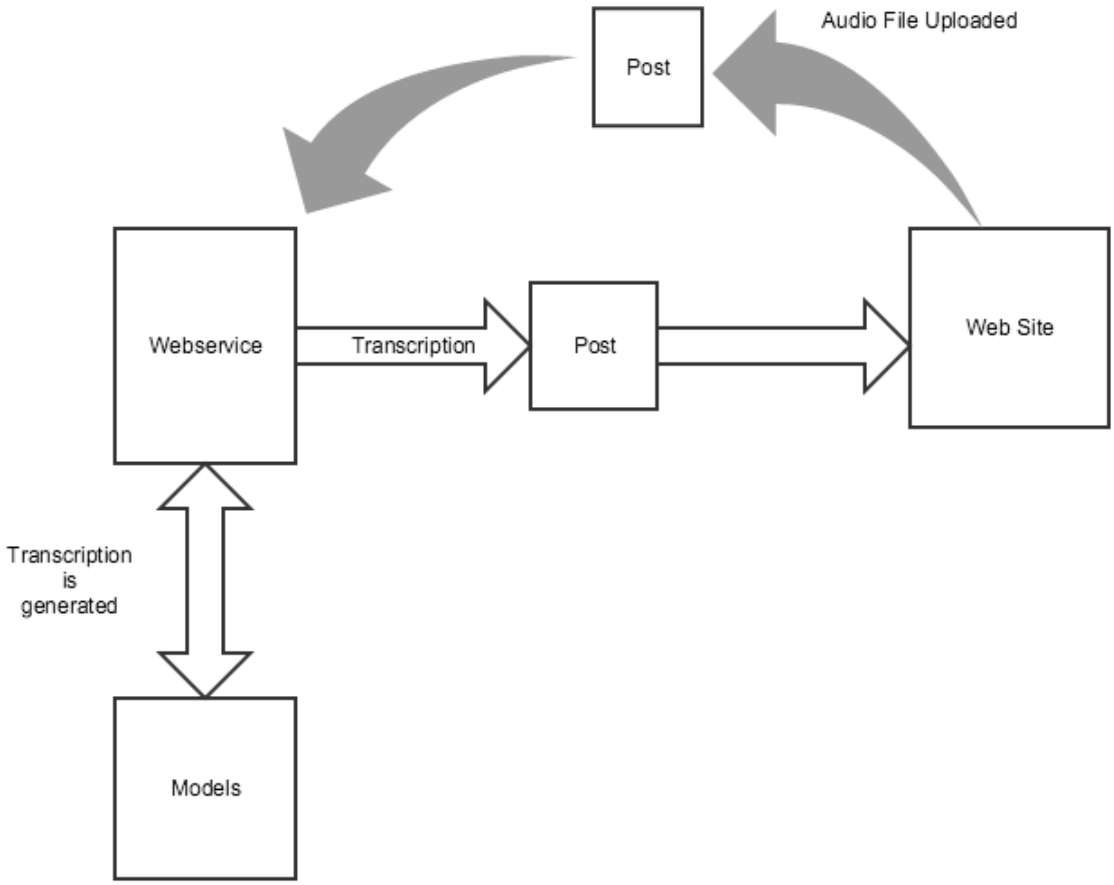

Figure 4.2: A diagram of Rewind's web service.

for future web apps to be developed, easy database integration, and easy incorporation of security. Django requires the use of Python to implement the server. Python has a large amount of libraries that can be used. Midi.js is utilized for its ability to parse MIDI files and generate sounds for those MIDI files. The jQuery and jQuery UI libraries has many useful features for designing interfaces, doing different web requests, and other functionality. The remodal library allow for modal windows to be displayed on the website. These libraries have made it possible to make a website for Rewind.

When the user first opens the website, the user is presented with a view shown in

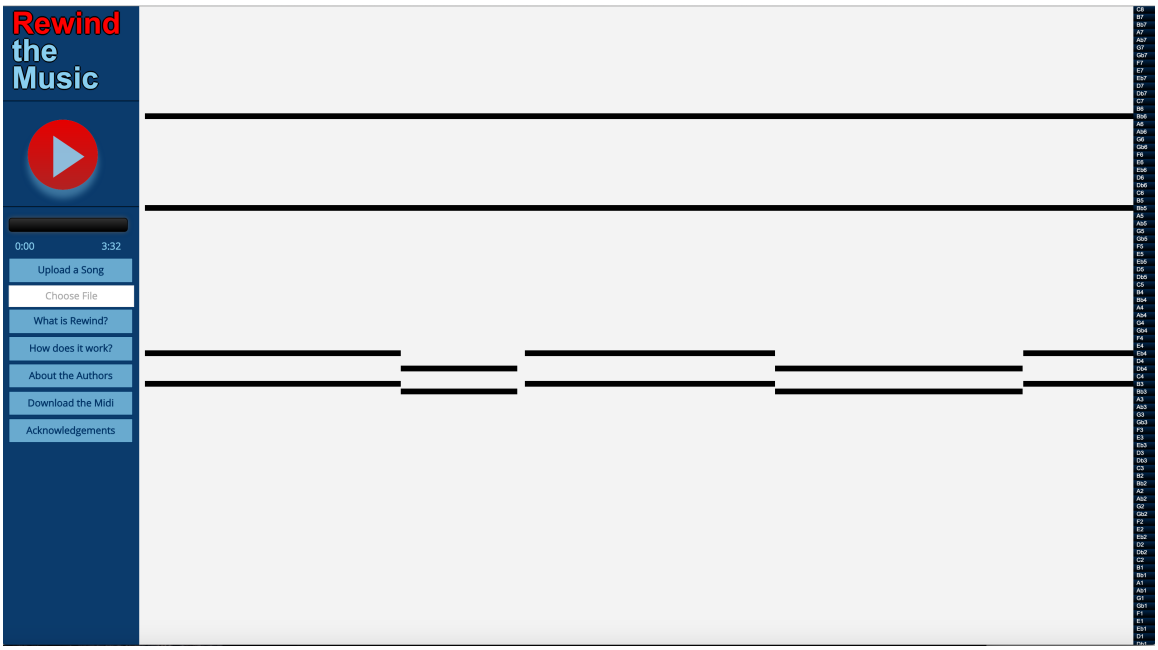

Figure 4.3: A screenshot of the website with a piano roll.

Figure 4.3. The user has several options available to them, such as loading an audio file to be converted, get information about the project, and the authors, and to play the currently converted transcription or default. When the user uploads a file, it is sent through a post request to the web service and converted to a midi and sent back to the website through another post request. The web page will then be populated with a piano roll, as is demonstrated in Figure 4.3. The user can pause or play a song in the website, or even set the position of the song using the time bar. This simple interface website allows user to interact with transcriptions generated by the models of Rewind.

4.4.3 Web Synthesizer and Piano Roll

Rewind has a built in web synthesizer thanks to midi.js, which is used to playback transcriptions generated by Rewind's models. Midi.js has several dependencies, which are used to playback sounds and can handle different platform setups. It can parse midi events and make it possible to extract time delta for constructing piano rolls and note information. Midi.js can load many different sound fonts to load different

sounds such as piano, flute, drums, and other sounds. This allows for Rewind to be scalable for more complex models in the future.

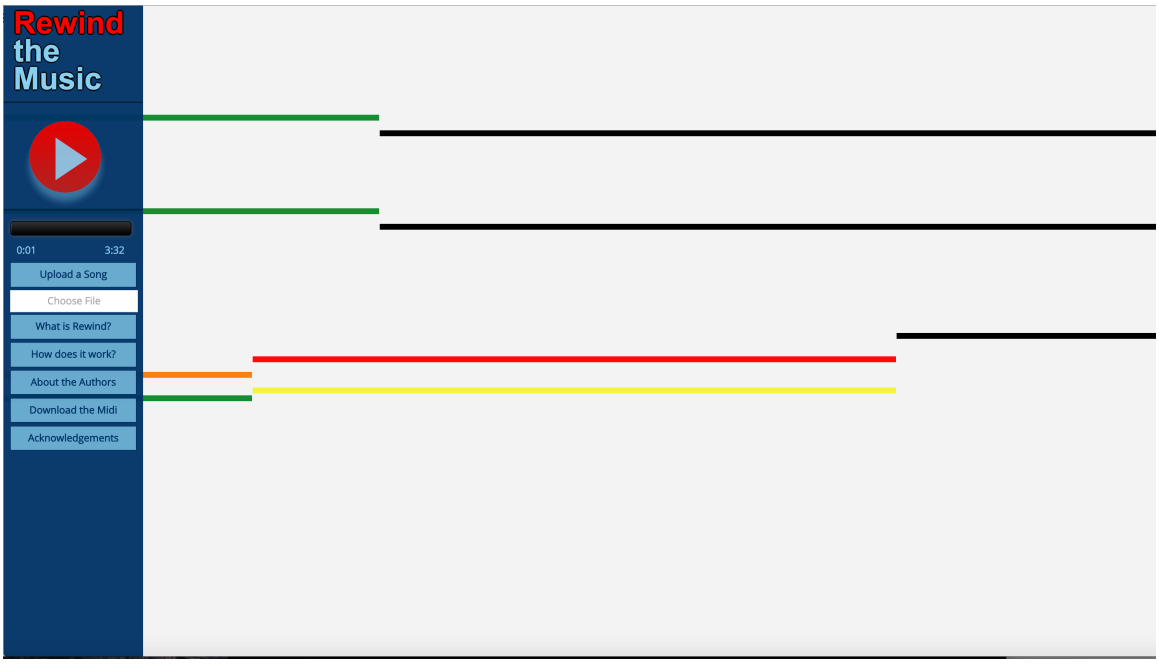

Figure 4.4: A screenshot of piano roll notes lighting up.

The piano roll constructed for visualization in Rewind are based on the time duration and time position information collected from midi.js. The user has the ability to scroll through the piano roll using the time bar. As a song plays the piano roll will light up as demonstrated in Figure 4.4, and the screen will transition to another part of the piano every second. This piano roll allows the user to see what their transcriptions are and to see if there are any erroneous notes. With the piano roll having the ability to have different colors, it is possible to represent different instruments with different colors, but currently the colors are represented based on the note itself. There is some future work to be developed, regarding the ability of adding or removing certain notes from the transcription using the piano roll.

Chapter 5

Results

5.1 Overview

In this section we present the precision, recall, f-measure, and accuracy of Rewind's transcriptions on the following datasets: Nottingham consisting of 1000 or mores songs, JSB Chorales consisting of 200 or more songs, Poliner-Ellis consisting of 30 songs, MuseData consiting of 700 songs, the Maps dataset consisting of 169 songs, and a custom dataset that consists of 160 songs split evenly from country, rock, jazz and classical. The custom dataset was added since all of the benchmark datasets currently used in the AMT are currently only classical piano music and orchestralcmusic. The results are compared to other existing works at the frame level.

5.2 Results and Discussion

In Table 5.1 and Table 5.2, the overall results of Rewind at a 10 ms stride, a standard for AMT systems, at the frame level are demonstrated and compared to [7, 40]. The 10 ms stride results were trained with two parallel GRUs with a linear layer. All tests were also ran at 50 ms with an exception of the Maps dataset demonstrated in Table 5.3. The 50 ms results are included to demonstrate that a higher stride leads to better results due to the higher temporal resolution. The 50 ms results were trained with a single GRU and linear layer. Ideally a higher stride rate would allow one to capture shorter notes, but the overlapping windows of the spectrogram makes it difficult to capture shorter notes. However, it does help raise the accuracy to a greater

degree when comparing results between the tables. The simple auto-correlation results are reported in order to give a comparison of simple manual versus a deep learning algorithm in Table 5.3. A visual comparison of the encoder networks output to another spectrogram is demonstrated in Figure 5.1 to demonstrate the quality of the learned spectrogram versus the actual spectrogram. It effectively demonstrates that the decoder can effectively extract the note information fro the encoding.

As demonstrated in Table 5.1, Rewind's model leads to relatively good results on the JSB and Nottingham datasets, while with the custom dataset, and MuseData, the results are only acceptable or need some improvement. However further inspection of the model, shows that the model is learning the songs due to the high precision within all of the results. An important detail about the MuseData dataset and custom dataset is that these datasets utilize more than one instrument in their midis. Meaning that Rewind is sensitive to the harmonics of multiple of instruments. The reason these datasets are so low, is due to rounding the probabilities to one as was discussed in Chapter 4. Despite being low, the MuseData dataset has a relatively high precision, meaning that it handles false positives relatively well, but does not handle false negatives according to the recall. Despite these two datasets, the results for the Nottingham and JSB dataset are comparable to [7], and have a relatively high f-measure.

The results demonstrated in Table 5.2 are compared against ConvNet acoustic model at the frame level [40]. Upon examining the table, the Convnet is better overall in accuracy, recall, and f-measure, but Rewind has the higher precision. The ConvNet [40] utilizes a hash beam search to find the most probable sequence. If Rewind was to utilize the same hash beam search, it may have been able to achieve an even better accuracy, recall, and f-measure.

Table 5.1: Rewind's results at 10 ms stride for the spectrogram where 1 is the proposed model and 2 is the rnn-nade [7].

	Accuracy		Precision		Recall		F-Measure	
Models	1	2	1	2	1	2	1	2
Nottingham	95.1%	97.4%	98.0%		96.9%		97.5%	
JSB	82.8%	91.7%	34.4%		88.8%		82.8%	
Poliner-Ellis	34.4%	79.1%	66.9%		41.5%		34%	
MuseData	34%	66.6%	56.8%		45.9%		16.2%	
Custom	16.2%		51.1%		19.2%		27.9%	

Table 5.2: Rewind's performance on the Maps dataset compared to [40] at 10 ms.

	Proposed	Simple Auto-Correlation	ConvNet[40]
Accuracy	51.6%	6.4%	**58.87%**
Precision	**76.5%**	21.8%	72.40%
Recall	61.4%	8.2%	**76.50%**
F-Measure	68.1%	11.2%	**74.45%**

Table 5.3: Rewinds results at a 50 millisecond stride for the spectrogram where 2 is the proposed model and 1 is the Simple Auto-Correlation model.

	Accuracy		Precision		Recall		F-Measure	
Models	1	2	1	2	1	2	1	2
Nottingham	21.5%	94.0%	29.2%	97.9%	44.7%	95.9%	35.3%	96.9%
JSB	20.8%	81.6%	32.9%	92.1%	36.2%	87.7%	34.5%	89.9%
MuseData	11.8%	23.0%	15.8%	60.2%	31.9%	27.2%	21.1%	37.4%
Poliner-Ellis	6.6%	42.6%	17.7%	70.5%	9.7%	51.8%	12.5%	55.8%
Custom	8.5%	20.4%	12.2%	44.5%	21.8%	27.3%	15.6%	33.9%

.

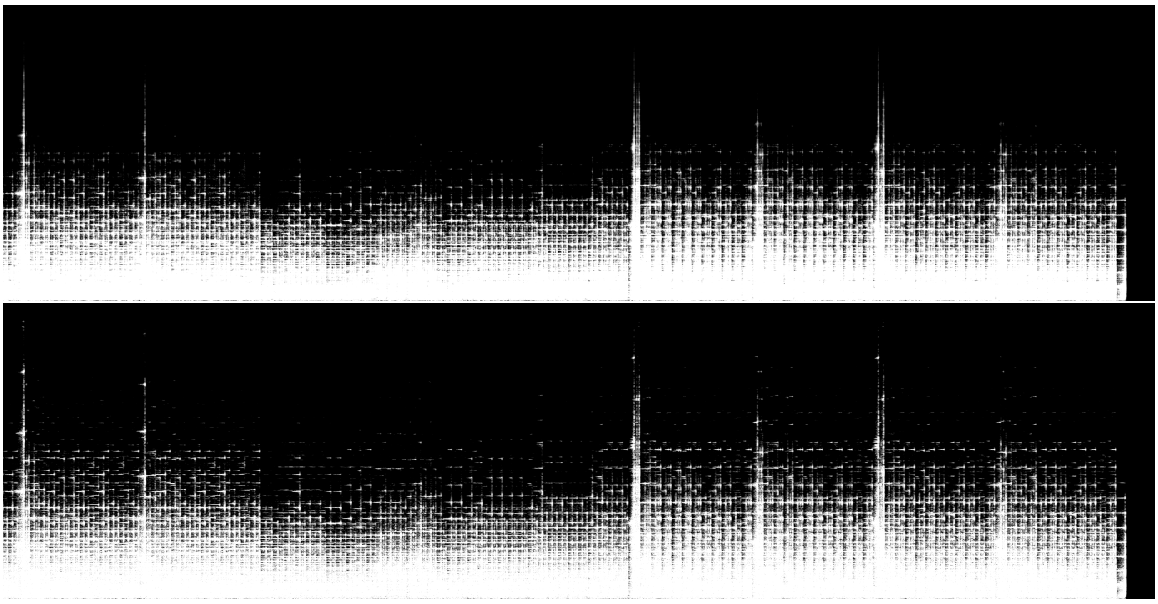

Figure 5.1: A comparison between two spectrograms: output of encoder model(top) vs actual(bottom).

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Rewind demonstrated a encoder-decoder network that is comparable to the results to [7] in terms of the Nottingham and JSB dataset. It also achieved a higher precision than [40] on the Maps dataset. However, it suffered from issues in connection with choosing a threshold to generate an on value in the transcription on datasets such as MuseData and the custom dataset built by Rewind. The custom dataset demonstrated that AMT systems can work with multiple genres, but there may be other factors that cause transcription metrics to go down, such as multiple instruments being existent in the song or an improper threshold. Despite the threshold issue, Rewind does manage to follow the underlying frame distribution in the lower classified datasets. The simple auto-correlation model results demonstrated in Chapter 5 was to be a comparison between Rewind's encoder-decoder network and other models. While this is an interesting approach, it was only added as a way for Rewind's service to be quickly developed and is only reported to demonstrate its accuracy. Rewind's encoder-decoder has demonstrated a model that has a high precision and comparable results. It also has demonstrated a web app that can be utilized for analyzing transcription generated by Rewind's encoder-decoder or simple auto-correlation model.

Rewind's web site provides users with a way to hear and see their transcriptions. This web site can be used as a way to validate the transcription generated by different models used in Rewind. It provides users with a way to transcribe their favorite music

in an automatic way. By utilizing Django, Rewind is scalable for other projects in the future. The overall goal of Rewind's web site is to provide the users, such as bands, musicians, composers, and others a way to transcribe their favorite music.

6.2 Future Work

Rewind has demonstrated a model that works at the frame level. Previous work, such as [40], have used a frame level model in conjunction with a note level model to get a note level transcription. One key thing for the encoder-decoder network would be to add another layer, which can do note level transcription and utilize other algorithms from [7] to produce a more probable transcription. This is possible due to the separation of the encoder and decoder in the encoder-decoder network. This thesis thoroughly explores AMT as its problem, but the encoder part of the network could be used for other problems such as genre classification, audio generation like [43], or audio transformation where a sound is transform into another sound. A deeper architecture could be considered for experimentation for the encoder network, using possibly more GRUs or LSTMs for larger datasets. One thing this thesis did not consider is how an autoencoder trained on one dataset could potentially be used on other datasets. One other issue that Rewind would like to solve is being able to produce a transcription for each instrument in a song and be able to determine what instrument is being played. There are some other data representations that can be used, which are slightly better that in a spectrogram made with stft, such as a wavelet, which can adjust its frequency and time resolution for a signal or the constant q transform. A data representation with more features or improvements to the encoder would allow for the encoder-decoder network to produce more accurate predictions.

Rewind's web site has the potential for adding new features and interfaces for new problems. A cool feature to add to Rewind's front end would be the ability to edit and remove notes from the piano roll. Rewind could be be expanded into an application that allows a user to edit existing music through transcribing the music.

Another addition would be to allow Rewind to recognize the lyrics of the music being played. One more thing that Rewind could provide is a way for users to collaborate and learn about music.

Bibliography

[1] James Allwright. ABC version of the nottingham music database. 2016. URL: http://abc.sourceforge.net/NMD/ (Last accessed 04/10/2016).

[2] James Allwright. Bach choral harmony data set. 2016. URL: http://archive.ics.uci.edu/ml/datasets/Bach+Choral+Harmony (Last accessed 04/10/2016).

[3] Arakisoftware. AmazingMIDI. July 20, 2003. URL: http://www.pluto.dti.ne.jp/araki/amazingmidi/ (Last accessed 04/10/2016).

[4] Pierre Baldi and Zhiqin Lu. Complex-valued autoencoders. *Neural netw.*, 33:136–147, September 2012. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2012.04.011. URL: http://dx.doi.org/10.1016/j.neunet.2012.04.011.

[5] Mert Bay, Andreas F. Ehmann, and J. Stephen Downie. Evaluation of multiple-f0 estimation and tracking systems. In *Proceedings of the 10th international society for music information retrieval conference*. http://ismir2009.ismir.net/proceedings/PS2-21.pdf. Kobe, Japan, 2009, pages 315–320.

[6] Peter J Billam. MIDI.lua. URL: http://www.pjb.com.au/comp/lua/MIDI.html (Last accessed 04/10/2016).

[7] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. High-dimensional sequence transduction. In *Acoustics, speech and signal processing (ICASSP), 2013 IEEE international conference on acoustics, speech and signal processing*, 2013, pages 3178–3182. DOI: 10.1109/ICASSP.2013.6638244.

[8] Nicolas Boulanger-Lewandowski, Jasha Droppo, Mike Seltzer, and Dong Yu. Phone sequence modeling with recurrent neural networks. In *ICASSP*. IEEE SPS, 2014. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=217321.

[9] Sebastian Bck and Markus Schedl. Polyphonic piano note transcription with recurrent neural networks. In *Acoustics, speech and signal processing (ICASSP), 2012 ieee international conference on*, 2012, pages 121–124. DOI: 10.1109/ICASSP.2012.6287832.

[10] Ilya Caulfield. Remodal v1. URL: http://vodkabears.github.io/remodal/ (Last accessed 04/10/2016).

[11] Center for Computer Assisted Research in the Humanities. Musedata. 2016. URL: http://musedata.stanford.edu/ (Last accessed 04/10/2016).

[12] Soumith Chintala. Audio library for torch. URL: https://github.com/soumith/lua---audio (Last accessed 04/10/2016).

[13] Kyunghyun Cho, Aaron Courville, and Yoshua Bengio. Describing multimedia content using attention-based encoder–decoder networks. 2015. eprint: arXiv:1507.01053.

[14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Corr*, abs/1406.1078, 2014. URL: http://arxiv.org/abs/1406.1078.

[15] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *Corr*, abs/1412.3555, 2014. URL: http://arxiv.org/abs/1412.3555.

[16] Michael Deal. MIDI.js. URL: https://github.com/mudcube/MIDI.js/ (Last accessed 03/09/2016).

[17] Li Deng, Mike Seltzer, Dong Yu, Alex Acero, Abdel rahman Mohamed, and Geoff Hinton. Binary coding of speech spectrograms using a deep auto-encoder. In *Interspeech 2010*. International Speech Communication Association, 2010. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=135405.

[18] Zhiyao Duan and Emmanouil Benetos. Automatic music transcription. ISMIR. 2015. URL: http://c4dm.eecs.qmul.ac.uk/ismir15-amt-tutorial/.

[19] Valentin Emiya. MAPS Database A piano database for multipitch estimation and automatic transcription of music. 2016. URL: http://www.tsi.telecom-paristech.fr/aao/en/2010/07/08/maps-database-a-piano-database-for-multipitch-estimation-and-automatic-transcription-of-music/ (Last accessed 04/10/2016).

[20] Hans de Goede. Timidity++. URL: https://sourceforge.net/projects/timidity/ (Last accessed 04/10/2016).

[21] Alex Graves. Sequence transduction with recurrent neural networks. 2012. eprint: arXiv:1211.3711.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[23] ImageLine. Fl studio. 2016. URL: https://www.image-line.com/flstudio/ (Last accessed 04/10/2016).

[24] Innovative Music Systems, Inc. MP3 to MIDI converter free download - convert WAV to MIDI. 2016. URL: http://www.intelliscore.net/ (Last accessed 04/10/2016).

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Corr*, abs/1412.6980, 2014. URL: http://arxiv.org/abs/1412.6980 (Last accessed 05/12/2016).

[26] Bernd Krueger. Classical piano MIDI page. 2016. URL: http://www.piano-midi.de/ (Last accessed 04/10/2016).

[27] AKoff Sound Labs. How to make a song? Hum a melody. Song making software does the rest. 2016. URL: http://www.akoff.com/index.html (Last accessed 04/10/2016).

[28] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in neural information processing systems 22*, pages 1096–1104, 2009. URL: http://books.nips.cc/papers/files/nips22/NIPS2009_1171.pdf.

[29] Nicholas Lonard, Sagar Waghmare, Yang Wang, and Jin-Hwa Kim. Rnn : recurrent library for torch. 2015. eprint: arXiv:1511.07889.

[30] Jacob Morgan and George Burdell. Online sequencer. 2016. URL: http://onlinesequencer.net/ (Last accessed 04/10/2016).

[31] Mozilla Developer Network. Web audio API. 2016. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API (Last accessed 04/10/2016).

[32] ofoct.com. Convert WAV (or MP3, OGG, AAC, WMA) to MIDI. 2016. URL: http://www.ofoct.com/audio-converter/convert-wav-or-mp3-ogg-aac-wma-to-midi.html (Last accessed 04/10/2016).

[33] Graham Poliner. Automatic piano transcription. 2016. URL: http://labrosa.ee.columbia.edu/projects/piano/ (Last accessed 04/10/2016).

[34] Armin Ronacher. Django the web framework for perfectionists with deadlines. 2016. URL: https://www.djangoproject.com/ (Last accessed 04/10/2016).

[35] Armin Ronacher. Flask web development, one drop at a time. 2016. URL: http://flask.pocoo.org/ (Last accessed 04/10/2016).

[36] Koray Kavukcuoglu Ronan Collobert Clment Farabet and Soumith Chintala. Torch. 2016. URL: http://torch.ch/ (Last accessed 04/10/2016).

[37] Andy M. Sarroff and Michael Casey. Musical audio synthesis using autoencoding neural nets. In *International computer music association (icmc 2014)*, 2014.

[38] Nicol N. Schraudolph and Terrence J. Sejnowski. Unsupervised discrimination of clustered data via optimization of binary information gain. In *Advances in neural information processing systems*. Morgan Kaufmann, 1993, pages 499–506.

[39] Srikanth Cherla Tillman Weyde Artur S. dAvila Garcez Siddharth Sigtia Emmanouil Benetos and Simon Dixon. An rnn-based music language model for improving automatic music transcription. In *15th international society for music information retrieval conference (ISMIR 2014)*, 2014.

[40] S. Sigtia, E. Benetos, and S. Dixon. An End-to-End Neural Network for Polyphonic Piano Music Transcription. *Arxiv e-prints*, August 2015. arXiv: 1508.01774 [stat.ML].

[41] Siddharth Sigtia, Emmanouil Benetos, Nicolas Boulanger-Lewandowski, Tillman Weyde, Artur S. d'Avila Garcez, and Simon Dixon. A hybrid recurrent neural network for music transcription. 2014. eprint: arXiv:1411.1623.

[42] Steven Smith. *Digital signal processing : a practical guide for engineers and scientists*. Newnes, Amsterdam Boston, 2003. ISBN: 075067444x.

[43] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. 2015. eprint: arXiv:1502.04681.

[44] The JQuery Foundation. Jquery user interface. URL: https://jqueryui.com/ (Last accessed 03/09/2016).

[45] The JQuery Foundation. Jquery write less, do more. URL: https://jquery.com/ (Last accessed 03/09/2016).

[46] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop. *Coursera: neural networks for machine learning*, 2012.

[47] W3C. Web MIDI API. 2016. URL: https://www.w3.org/TR/2015/WD-webmidi-20150317/ (Last accessed 04/10/2016).

[48] Wayne Wakeland. Learning general features from images and audio with stacked denoising autoencoders. Master's thesis. Portland State University, 2014.

[49] Widisoft. Widi recognition system. 2016. URL: http://www.widisoft.com/index.html (Last accessed 04/10/2016).

[50] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *Computing research repository (CoRR)*, abs/1212.5701, 2012. URL: http://arxiv.org/abs/1212.5701 (Last accessed 05/12/2016).