UiT The Arctic University of Norway

Department of Computer Science and Computational Engineering

Investigating representation of tablature data for NLP music prediction

Tor Eldby

Master's Thesis in Applied Computer Science...DTE3900...May 2021

Contents

List of Figures

List of Tables

Acknowledgement

I would first like to thank Sasha, for all her love and support through this heavy work period, and all her words of wisdom and guidance in times where I felt lost and exhausted in my work, not only through this thesis period, but throughout these now 5 years of studies. Her support has been the deciding factor many times in my life when times were hard, for me to persevere through it all. I look forward to many more years of mutual love and support.

I would also like to thank Shayan Dadman and Professor Bernt Bremdal for their support and knowledge, which was invaluable and crucial for the past few months' work to really manage to go anywhere. They were available at any time I needed them, and our discussions were one of the crucial factors for this thesis work's progress to accelerate the way it did in slow periods.

Finally I would like to thank my parents who, with love, support, and sage-advice, have helped me continue to progress in my life as a student and a man, and I'm sure will still be a large factor of my life as I move out into the big world to carve my own path.

Thank you, all.

Investigating representation of tablature data for NLP music prediction

Tor Eldby

May 15, 2021

Abstract

In this thesis, the ability of CharRNN models learning to compose guitar music using varying representations of guitar tablature is explored.

I utilize a well-versed sequential model of LSTM cells, and investigate the ability of said model to input, and predict both character to character, and sequence to sequence, following the principles of natural language processing and music information retrieval. The study was conducted on datasets consisting of data naïvely retrieved from a subset of classical guitar tablature.

With regards to tablature structure, the experiments uncover a clearly superior form for character to character prediction, producing a model capable of composing seemingly musically coherent phrases. The work is not fully able to compare the character predictor with the sequence predictor and further details how this could potentially be alleviated.

1 Introduction

Automatic music generation is the task of using algorithmic composition, machine learning and artificial intelligence methods to generate music. In recent years, with the growth of powerful deep learning architectures and methods like *RNN* [5, 6] and *CNN* [7, 8], it has become a widely researched field of study. With rapidly growing progress we have seen several models emerge like WaveNet[1], for general audio waveforms, and MuseNet[2], specifically for music composition. There are a few different ways of which researchers has approached these problems. Some of

[1]`https://deepmind.com/blog/article/wavenet-generative-model-raw-audio`
[2]`https://openai.com/blog/musenet/`

which form algorithms by looking at the rules of music theory, like [9, 10], whilst others approach the generation from a machine learning perspective by training a deep neural network through large amounts of musical data [7, 11, 12, 13]. These all follow the concept of *Music Information Retrieval*[3](MIR), which covers a vast field of backgrounds and applications concerning music. In machine learning, problems boil down to data representation, extracting and learning features, and generating music that sound pleasing to human ears as well as being coherent with regards to music theory. The vast majority of artificial music generators are deep neural networks based on RNN architectures that generate music step-by-step using MIDI with promising results [14, 13, 9].

In this paper I choose to focus on leveraging recurrent deep learning concepts like RNN to investigate data representation and form for composition of guitar music in tablature space, using a CharRNN-based model to iteratively generate music as a prediction by training upon tablature data, taking inspiration from [15]. I wish to answer the questions of what would be the best performing form and representation of data, along with how well both a character to character predictor, and a sequence to sequence predictor, well suited for natural language applications, would perform with regards to tablature music.

All the Python code used is made available on my project repository[4]

1.1 Data representation

Musical representation comes in various forms and flavours, and each offer their advantages and disadvantages. The most widely used for music representation and generation is MIDI and ABC, whilst the most commonly used by end-users are pure audio data like MP3 and waveform. With regards to machine learning, the advantages and disadvantages of the different forms of data comes down to information density, ease of use and complexity in computing. This section will discuss the different forms of representation for audio/music data, and conclude with a reasoning for choosing tabs for this thesis work.

Audio is by far the most information dense datatype with regards to music. Simple waveform can be interpreted by the brain to form an understanding of musical intention and projection like story, feelings, memories and so on, all from a combination of base musical features like notes, chords, progressions, timbre, harmonies, and structure, which can again be decomposed further into frequencies, beats, rhythm,

[3]`https://en.wikipedia.org/wiki/Music_information_retrieval`

[4]`https://source.coderefinery.org/Ascended/master-thesis-tor-eldby`

etc. This is of course an incredibly complex process[5]. Although this data is information dense, extensive preprocessing upon the raw waveform is necessary to obtain these base features. For brevity I refer to [16, 17] for a deeper look into the techniques and uses for audio data processing. Another drawback to audio data is that to process all these features for a machine learning system would take large amounts of computations if one would hope to model an artificial composer able to compose at human levels.

MIDI and ABC are promising middle-ground representations, as they inherently highlight the musical features of the data, which makes them quite flexible and robust for different machine learning methods with regards to preprocessing into an understandable input format. They contain information that helps with sampling and encoding data into time-dependent sequences that can be input into a machine learning algorithm for fitting and prediction [7], although MIDI tends to work best with keyboard-like instruments [13]. Similarly, ABC notation has been used for artificial music generation [15, 18], although not as exhaustively as MIDI.

Tablature[6] is a type of music notation designed for string-instruments, in particular guitar, and is meant to represent tuning, finger positions, strum direction, transitions, chords, and so on [19]. As with piano, guitar is a highly popular instrument and as such it's musical notation has evolved to become efficient and robust for humans to read and play with specifically as a stringed instrument [13]. Additionally there are some larger datasets of guitar music available for download, although tablatures usually come as TXT [20] and MIDI [21] format. With some preprocessing, the TXT format could become as useful information-wise as MIDI and ABC. A part of the complexity of tablature springs up from how several notes of the same pitch can be represented by several fingerings, posing a challenge for generating tablature with human-playable fingering.

Both because of the lack of research with regards to using tablature to generate music, aside from the few previously mentioned, as well as the promising results from [15], I believe it would be interesting to look into using tablature character data and investigate how one could optimally form and represent it for a character prediction model.

[5]`https://neuro.hms.harvard.edu/centers-and-initiatives/harvard-mahoney-neuroscience-institute/about-hmni/archive-brain-1`

[6]`https://en.wikipedia.org/wiki/Tablature`

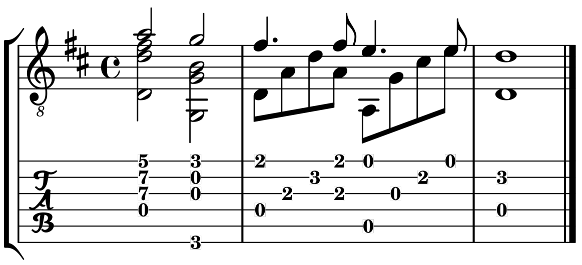


Figure 1: Simple classical music notation compared with it's equal tablature.

1.2 Related works

Although it is little, there is some research and production that focuses on AI music composition in tablature space.

Using a genetic algorithm, Tuohy et al.[10] analyse and converges guitar tablature towards more playable formats using a set of valid guitar tablature for a specific song (e.g. any given note can be represented by several different strings and fret positions), and selecting and breeding genetic populations of said tabs based on a heuristic fitness function that analyses and scores hand and finger movement throughout the generated tabs. What results from this is guitar tablature which is musically indistinguishable from the original, but more easily playable, according to their heuristic. Their primary challenge and background for this work was that often times, guitar tablature is created to be musically accurate, as it relies too heavily on music theory, and not necessarily easily playable for guitarists. When that is the case, they wished to be able to automatically discover or generate good and valid equivalents of the oriinal tablature. In genetic algorithms, the main problem is representing the data as a set of genes and other biological equivalents. The researchers solved this by first representing each fretboard position as a gene, and any two arbitrary pair of different tablature as the parent's set of genes. Then they cut their gene-sets at multiple points and crossed those splices between each, and finally randomly mutated some frets/genes into their potential musical equivalents. Alongside this implementation, they needed to understand how they score the different tablature for their playability. The researchers acknowledged this difficulty and that their approach isn't necessarily as accurate as some other methods could be. Rather than utilizing some fitness function, they found it more practical to devise a heuristic for hand movement and hand manipulation. They each basically estimate the total movement accross the fretboard, and the potential difficulty of playing a specific chord at a time, respectively. I believe their approach is well viable for their work, and I agree with their chosen method for generating and approaching viable tablature. It stays within the musicality of the original music, and with their fitness function approaches some "best playability". Although I understand their choice for heuristic function, I wonder how their algorithm would perform if their fitness function was more like what they avoided in trade for their heuristic method. How fast would their algorithm converge if the fitness function was more precise? I believe it that were the case, it take less iterations for the music to reach some maximum playability, although the increase computation cost for the fitness function could potentially be detrimental.

For generating guitar solos, [22] detects musical phrases and generates fitting guitar solos accordingly[7]. Their motivation was that allowing less skilled music composers to use guitar solos in their compositions. In part, this paper faces challenges regarding phrase boundary detection primarily, and guitar music composition secondarily, but it's methods diverge into using hidden markov models for detecting these boundaries and generating a distribution of initial phrases and phrase transitions. With this, their methods for tuning these phrases and exploiting them comes down to a stochastic random walk upon this state space. They utilized input data in the MusicXML format that they got from GuitarPro[8]. Their results are favorable and I do not believe I have much to say about it or their methods. The linked demonstration speaks for itself.

Shayan Dadman [15] uses NLP techniques to generate new music from input examples using data represented as ABC notation, specifically using jazz music, amongst other methods. His paper addresses several different challenges and forms of music information retrieval in experiments. I highlight the final experiment, of which uses recurrent neural networks to generate music from jazz examples. The challenges relating to this work is that of representing the data in such a way as to be able to efficiently perform NLP techniques upon it, alonside doing neural network computations whilst avoiding common issues like fitting and insufficient regularization. As the data is pure ASCII text, this is solved by using a form of conversion vocabulary, where every character present in the dataset is converted to some number within the range 0 to N, where N is the amount of unique characters in the dataset. With this, the researcher trains a variety of models using a sizeable dataset. According to the results, the best trained model achieved sufficient accuracy and managed to produce music given a small input string.

[7]Demonstration: `https://www.youtube.com/channel/UC7jBPwdo3oyqgzihSj_kB6Q`

[8]GuitarPro: `http://www.gprotab.neti/`

2 Artificial Neural Networks

An artificial neural network is classified as a network of artificial neurons built as simple input-output computation machines. It is fundamentally designed around our current simplest understanding of how neurons in the brain function as a network [23, p. 280]. For instance, the simplest form of a neural network is a single perceptron[9].

Using machine learning concepts, like input and output, activation, bias, errors, and backpropagation, it is capable of learning to perform linear regression tasks quite confidently. This is thanks to the concept of gradient descent, which the aforementioned backpropagation performs as the perceptron iteratively trains to classify it's input.

The following sections will be brief, but required, explanations and definitions of the concepts and tools used in this project with regards to algorithmic music composition with NLP techniques, which were utilized in the thesis work. First an introduction to feed forward networks and network architecture, moving on to sequential models like recurrent neural networks (RNN) and long-short term memory (LSTM), along with how they learn. Finally some information about Natural Language Processing (NLP), and Keras and Tensorflow, which were the primary methods and tools used in this work.

2.1 Feed-forward Neural Networks

The fundamental element of processing for artificial neural networks is the concept of feeding forward data. In short, data moves from input towards output between weighted paths, passing through previously mentioned neurons where brief weighted-sum calculations are performed and processed with activation functions, "feeding" the data forward through the network, neuron-to-neuron. The final result is obtained from the neurons at the output layer. The calculation before activation is:

$$r = b + \sum_{i=1}^{n} W_i X_i = W^T X + b \tag{1}$$

Here, W is the weights of the inputs to the node, and X is the values being input into the node on these weighted paths. b is a standard bias applied to every neuron to guide the network towards some specific behaviour. r is the result of this weighted sum and will pass on into an activation function before finally passing as output from the neuron (And maybe then be the input for the next layer of neurons). You

[9]`https://en.wikipedia.org/wiki/Perceptron`

can also see that the weighted sum calculation gets defined as a linear multiplication between a row matrix W and a column matrix X. Figure 2 illustrates this process.

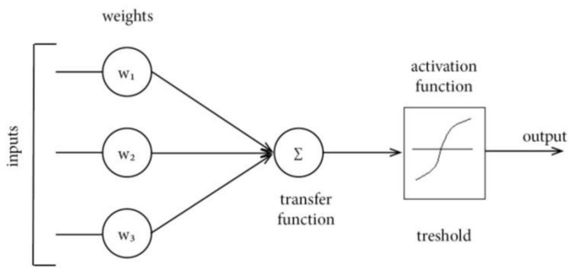


Figure 2: A neuron with 3 inputs, weighted sum function and sigmoid activation. Source: [1, Figure. 6]

Activation functions are used to guide the output of neurons for different purposes, and a comprehensive list of these can be found on Wikipedia[10]. It is important to note that the choice of activation function for neurons is crucial for a network to function optimally. The most commonly used are ReLu [24], Sigmoid and tanh.

During training, every epoch is defined as taking input and feeding it forward into the network. The final output $\hat{y}$ is compared with the expected output y (In the case of supervised learning) and results in some error e. Backpropagation "propagates" this error backwards through the network and updates each of the weights accordingly. The choice of activation function is important because of the *Gradient Descent* performed during backpropagation, meaning it is dependent upon the derivative of the activation function of each neuron [23, p. 289-292].

2.2 Network Architecture

Traditional neural networks occur when you string together a series of these computing neurons, for instance a series of perceptrons, into a linear network. The final output of the network is dependent upon the relationship of the calculations between the nodes, and as such is capable of storing multiple instances/dimensions of learned information based on the training sets the network is provided. As such, these networks always come in a layered form, like that of shallow- and deep neural networks. The neurons of a neural network does not necessarily have to be simple perceptrons, and the behaviour and functionality of these nodes can vary. The choice of which to use is dependent upon the task the model is designed for.

[10]`https://en.wikipedia.org/wiki/Activation_function`

Neural networks generally come in the form of an input layer, hidden layers, and an output layer. The amount of hidden layers is what determines whether a network is deep or shallow, typically 1 or 2 layers is the max for a shallow network [25]. The simplest form of neural networks are those with only two layers, the input and output layers, known as **single-layer neural networks**. **Multi-layer neural networks** are known as Deep Neural Networks (DNN), and they consist of the rest of the set of neural networks, where hidden layer counts reach above 2. These hidden layers allow a network to perform higher order, more complex, functions [26]. Figure 3 shows an example of a deep network. In a shallow network, only 1 or 2 hidden layers would be present in the figure. Notice how layer size does not matter for the definitions.

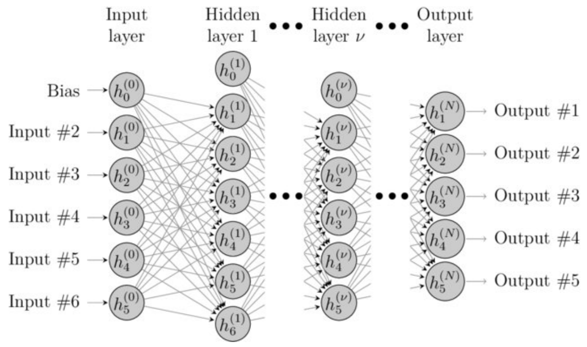


Figure 3: Shallow and deep neural networks. Source: [2, Figure. 2.1]

2.3 Sequential Models

Sequential models are the most common type of neural network models. Their general form is comprised of sequentially connected layers of neurons, as described in the previous section, and they are widely used with architectures all over the field of neural networks. In it's essence, a sequential model only defines how it's layers are connected, so it can really be applied in any context. For instance, figure 3 depicts a sequential neural network.

The behaviour of a model comes down to it's neurons and how they function and relate to one another. In the case of working with sequential data (i.e: Data that has some form of temporal or otherwise sequential dependency), recurrent neural networks are well versed and will be discussed next.

2.3.1 RNN

Recurrent Neural Networks [6] are classes of artificial neural networks where connected nodes have recurrent output connections to their own inputs, along with the standard input. Because of this, an RNN node in a network not only uses the input of the current time step for it's calculations, but also takes into account it's previous output. This connectedness allows for the node itself to learn temporal dependencies in the input data. Figure 4 helps visualise this. Notice the unfolded view is a stepwise series of nodes through time. The output of the previous state of the cell h_{t-1} is the input to h_t, along with the second input x_t from the dataset X

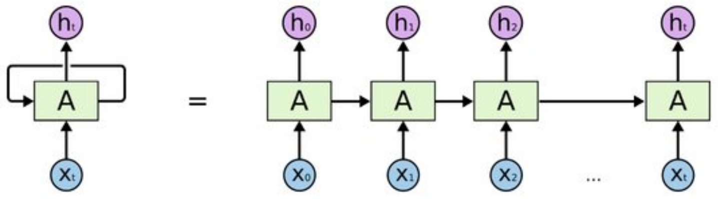


Figure 4: An RNN node, depicted as a recurring cell and an unfolded sequential graph. Source: [3, Figure. 18]

2.3.2 LSTM

An LSTM [5] is a more complex RNN architecture, in which several feedback connections and gates filter, manage, and keep information that both passes through and is being kept in the cell. Figure 5 illustrates the most common LSTM architecture with it's gates, inputs and outputs.

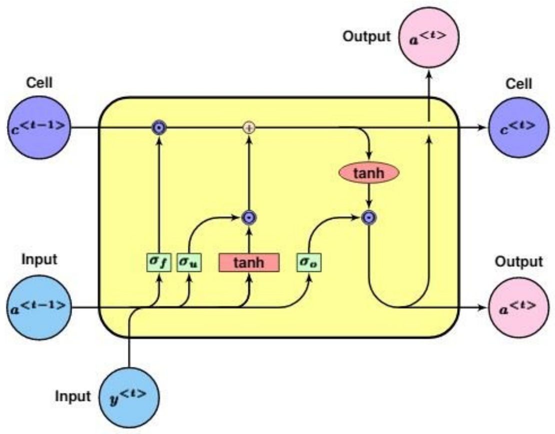


Figure 5: An LSTM cell with its gates. Source: [4, Figure. 1]

The general LSTM cell contains 3 controlling logistic gates, whose task it is to regulate the information that the LSTM has seen before, simultaneously with the input it gets at the current step. The values of these gates range between 0 and 1 and they function as follows:

- **Forget gate**: Learns to control when or how the long-term state should be erased
- **Input gate**: Learns to control which parts of the input to add into the long-term state
- **Output gate**: Learns to control what part of the long-term state should be given as output

Put simply these gates learn to recognise important input, store it long term, preserve it until it is not needed, and output it when needed [5, 23].

Described briefly, the process of how an LSTM cell functions can be seen as a small feed-forward network in and of itself, and the following equations are gotten from[23, p. 517]. The cell has two inputs feeding into 4 nodes with activations $\sigma_f, \sigma_u, \sigma_o, tanh$. There, weighted sum and activation is applied and the values are passed into the different logistic gates and used to modify the long-term state c. The modified state is then copied, where one copy is passed on to the next iteration for the current cell, whilst the other copy is given a $tanh$ activation and gated through the output gate, before being copied again and passed as both output and recursive input $a^{<t>}$

2.4 Keras and Tensorflow

Tensorflow[11] is a machine learning framework built for researchers and development engineers to perform the entire process of machine learning, being data loading and preprocessing, model creation, training and management, and full scale deployment and management of functioning machine learning models. It offers powerful and flexible tools and solutions for both Python and Javascript, along with well made tutorials and API documentation. On top of it's development frameworks, it is also built to be applicable on mobile devices and offers a lite version for low-power devices.

Keras[12] is a Python API built on top of Tensorflow. It is primarily marketed as a high-level API for Tensorflow and offers massive efficiency and scalability. It is a widely used platform for quick and simple machine learning prototyping and testing and allows for even faster and more flexible usage of Tensorflow, along with its solid implementation by many high-profile corporations and experts[13]. It offers curated tutorials dependent upon your usage of it.

2.5 Natural Language Processing

A **Natural Language Processor** is a machine which is capable of taking in sequences of data and learning features from the input text. An NLP model is used in a variety of linguistic applications[14], like translation, text-to-speech, command and voice detection, grammar tagging, summarisation tasks, and so on. Famously, Alan Turing's[15] Turing Test, also known as the Imitation Game, is in it's essence a NLP task for a machine to seem so human that it is capable of fooling a human to think it is itself a human, and in the case of Turing's description it is quantified using written language. Nowadays, this would be known as a chatbot [23, p. 525]. Crucially, linguistic data like words, sentences, paragraphs, and so on, are similar to music data in many ways. In written form, they are represented by sequences of characters. Along with this, they are both temporally dependent (references in time) and contain deterministic rules (grammar and music theory). Therefore, I believe implementing NLP techniques to the task of both generating music from tablature character data, along with using their results in investigating representations and forms of this character data is possible.

[11] `https://www.tensorflow.org/about`

[12] `https://keras.io/about/`

[13] `https://keras.io/why_keras/`

[14] `https://www.ibm.com/cloud/learn/natural-language-processing`

[15] `https://en.wikipedia.org/wiki/Alan_Turing`

3 Methods, Tools and Experiments

The following sections will detail the tools and methods created for and used in the experiments towards investigating representations for the tablature character data. First, a talk about the preprocessing and loading of the datasets acquired for the experiments, along with the dataset itself. Following this will be a description of the 4 experiments conducted and their respective results and discussions.

3.1 Data

The data was acquired from the Classtab[16] set of files, which has, as of the time it was downloaded, a total of approximately 3060 .txt files of tablatures. Giving the data a quick look-through it has lots of redundant data, with regards to musical information. Introductory information like history, name, dates, etc. Following will be descriptions of the tasks and solutions for the data preprocessor and loader created during this thesis work.

3.1.1 Preprocessor

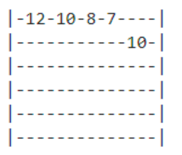
As each of the classtab .txt files carries parts of no value to the music of the tabs whatsoever, the preprocessor's task is to filter all the information in the data such that there is only tablature data left. Along with this, the preprocessor outputs 4 distinct formats for the tablature data, illustrated in figure 6, where the same bar of the same song is illustrated.

The preprocessor uses regular expressions to take a raw tab file, containing both tablature and other raw-text and non-music information, and extracts (very strictly) only the pure tablature parts. As the whole classtab dataset is around 3000 files and regular expressions are effective, yet hard to design, I chose to let it be very strict in what it deems as proper tablature. If at any point the tabs are detected to be not purely tabs containing alphanumerics surrounded by $|$ and $-$, the simplest and fastest decision is to reject the tabs and continue with the next tabs, and this is what it does.

With these tabs it then generates the different forms of the data. The data is either 6 rows of N characters (row-heavy), like usual tabs, or N rows of 6 characters (column-heavy), which can be seen as N chords where N is the length of the tablature. Along with this the data can also be multiple tracks split into widths of tabs between the barlines $|$. If filtering is desirable, the preprocessor also filters

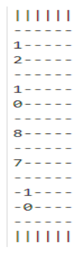
[16] `https://www.classtab.org/`

out any characters that are not wanted. Either the tabs can have barlines removed, have maximum widths of pure silence (chords of only non-activated strings), all non-numeric characters replaced with − (Except for the − character, of course), or any combination of these filters. Note that filtering is only ever output as column-heavy tabs.



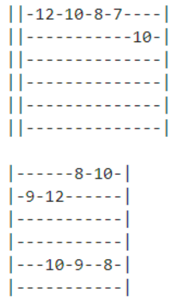
```
|-12-10-8-7----|
|-----------10-|
|--------------|
|--------------|
|--------------|
|--------------|
```

(a) Row heavy, normal tabs



```
||||||
------
1-----
2-----
------
1-----
0-----
------
8-----
------
7-----
------
-1----
-0----
------
||||||
```

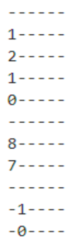
(b) Column heavy, transpose of row-heavy



```
||-12-10-8-7----|
||-----------10-|
||--------------|
||--------------|
||--------------|
||--------------|

|------8-10-|
|-9-12------|
|-----------|
|-----------|
|---10-9--8-|
|-----------|
```

(c) Multi-track, neatly divided rows to be more human readable



```
------
1-----
2-----
1-----
0-----
------
8-----
7-----
------
-1----
-0----
```

(d) Filtered, no unnecessary data

Figure 6: The different representations of the tabs produced by the preprocessor

3.1.2 Loader

The loader loads a given preprocessed file into a tensorflow dataset suitable for the model's architecture (character to character, or sequence to sequence), along with generating statistics about the preprocessed file, like the amount of activations for strings, most used characters in the vocabulary, and so on. The loader also takes in a specific split ratio for it's dataset and requires the ratios to sum up to 1. This makes it easy to generate any number of unique splits of the data into several datasets.

Table 1 is an example of the stats extracted by the loader as it reads through a source file. This file in particular seems to have 54% silence, which is particularly interesting for the purposes of analysing why generated music seems to have quite a bit of silence within it

Barline count:	3068
Silent chord count:	18190
Chord count:	15388

String activation count					
E	B	G	D	A	E
5904	6360	4950	3776	3004	1828

Fret activation count									
Open string	Fret 1	Fret 2	Fret 3	Fret 4	Fret 5	Fret 6	Fret 7	Fret 8	Fret 9
7267	2733	4718	2997	1867	1920	776	1907	825	812

Vocabulary usage												
|	\n	–	0	1	2	3	4	5	6	7	8	9
18408	0	175640	7267	2733	4718	2998	1867	1920	776	1907	825	812

Table 1: The statistics from a data file comprised of tablatures extracted from 80 files from the main set of classtab files

3.2 Model

The model is based on a promising model from [15], and is essentially a sequential model of a few large layers of LSTM cells with regularization methods applied. Although I deem it necessary to describe a few parts of this model, I refer to the original paper for more detailed technical information about them

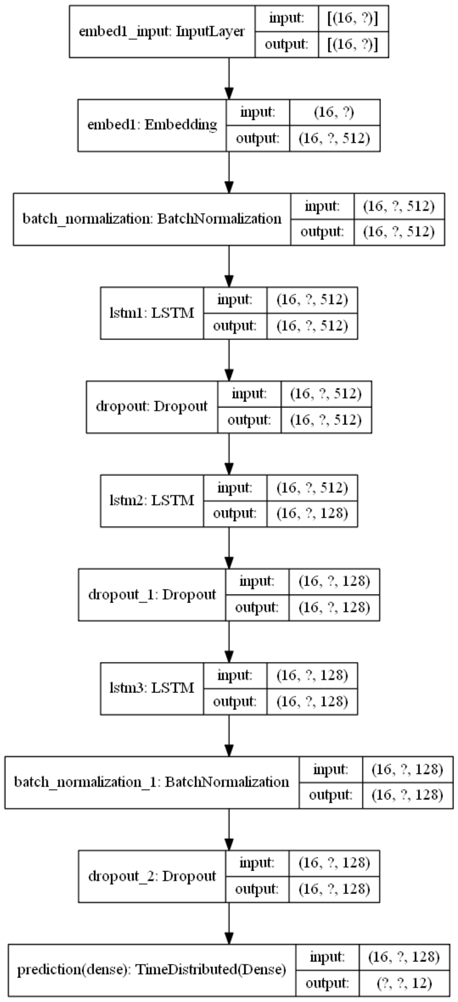


Figure 7: All the layers of the CharRNN training model

For each of the layers, the input and output parts of the plot are dependent on their specification and/or previous layers. For example, the embedding layer "embed1" takes in a 2-dimensional input batch of size 16, but outputs a 3-dimensional set of batches, as the lstm1 layer requires a 3-dimensional input, there are 512 nodes on it's layer, and the batch normalization layer preceding lstm1 simply takes an input and normalizes it's content (i.e: scales it's content to between 0 and 1, which helps to control the gradients of the model [27]).

The dropouts after each LSTM layer are meant to avoid over-fitting, which is a problem for many recurrent architectures, by increasing the regularization of the weights in the network [23, p. 365-367] and [28]. Essentially it helps to distribute training to weights that, if they weren't dropped, would potentially be favored against in most training instances, along with helping to avoid the chance of weights becoming overfitted.

3.2.1 Input representation

Inputting to the model is done as a set of vocabularized characters. It is again similar to the vectorization method of [15], but with a few differences with regards to the tablature itself.

With regards to the character to character model, as a set of characters in sliding-window sequences, where the following character is predicted based on the input set of characters. Once predicted, the "window" slides one character to the right and predicts the next character again, keeping the width of this window at a constant size. For sequence to sequence, this same method applies but instead of a sequence of characters there is a pair of sequences of 6 characters each, wherein the next sequence is predicted based on the input pair. These sequences are chords, so the representations are arbitrary, since any one sequence will be the same 6 characters.

Regardless of the structure, the tablatures being input into the model both for training and generation will be linear sequences of character-indices, i.e. they look like sequences of vocabular character indices with periodic or semi-periodic instances of newline character indices.

3.3 Training

Training is straight forward. An amount of epochs, batch size, sequence lengths and other specifications are used to generate a dataset using the aforementioned loader. The training model is loaded, loss, optimizer, and metrics are compiled in (For now only sparse categorical crossentropy as a loss function, and adam as an optimizer has been tested), and finally the model is fit with training and validation data, along with using callbacks for saving the weights as it trains. In the end a sample prediction and training history is visualised for loss and accuracy. If desirable the user can then save the logs for future evaluation. For character to character training, a sequence length of 140 was default and used on most all experiments as preliminary tests gave favourable results with that sequence length

3.4 Composition

For composition, the appropriate weights from the desired trained model are loaded and built, then the composition method for the model is run. Depending on the model, composition either predicts and categorically samples single characters or sequences of 6 characters for any desired amount of predicted chords. After composition the user can visualise and choose to save the music if so desired.

What is crucial for the composition of the tablature to be deemed as valid, is that it upholds the structural trend of the training data. Figure 6 shows all the different representational forms I use for the original tablature data. As such, the generated tabs need to be representative of these structures in every detail. For example, the multi-track compositions need to be formed as an arbitrary amount of chunks of 6 lines with width N. Alternatively, the row-heavy representation is simply 6 lines of tabs, all of width N, whilst column-heavy is row's transpose of N lines of width 6. This can be referred to as a representation's structural dependency, and it must be upheld in tablature generation.

4 Training and composition experiments

The following will be a few brief subsections about the different trainings and compositions from the different data representations. Afterwards, the results of each of these will be compared and discussed.

For all data representations a character to character model was utilised with a sequence length of 140 and 50 epochs. Batch sizes varied from experiment to experiment, as it was limited by the power of my hardware, but the final results depict the batch sizes for the chosen models. For the sequence to sequence model, only a simple sliding window of sequence pairs was used.

4.1 Multi track

Training the model was straight forward for either amount of epochs and the history can be seen in the following figure. Please note that because of hardware limitations, the training only lasted for around 20 epochs before it crashed.

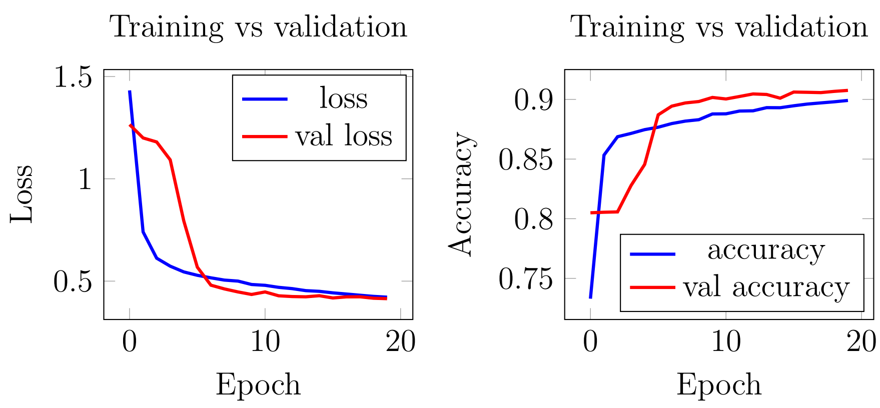
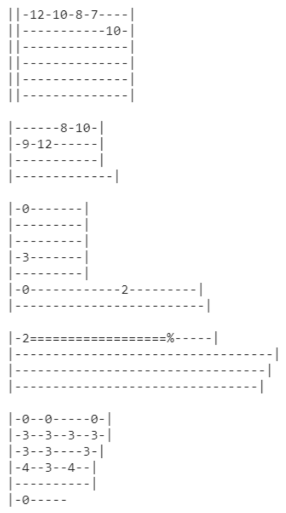


Figure 8: Multi-track model training history

Figure 9 shows the composition results and I'd like to highlight the formatting problem of the prediction. It seems to somewhat understand that newline is important, but it does not manage to reliably predict chunks with the limitations implied by the multi-track format.



```
||-12-10-8-7----|
||-----------10-|
||--------------|
||--------------|
||--------------|
||--------------|

|------8-10-|
|-9-12------|
|-----------|
|-------------|

|-0-------|
|---------|
|---------|
|-3-------|
|---------|
|-0------------2----------|
|-------------------------|

|-2==================%-----|
|----------------------------------|
|---------------------------------|
|--------------------------------|

|-0--0-----0-|
|-3--3--3--3-|
|-3--3----3-|
|-4--3--4--|
|----------|
|-0-----
```

Figure 9: Composition results for a multi-track input representation. The first few lines are input

4.2 Row-heavy

Following is a figure of the training history of the model trained upon row formatted data.

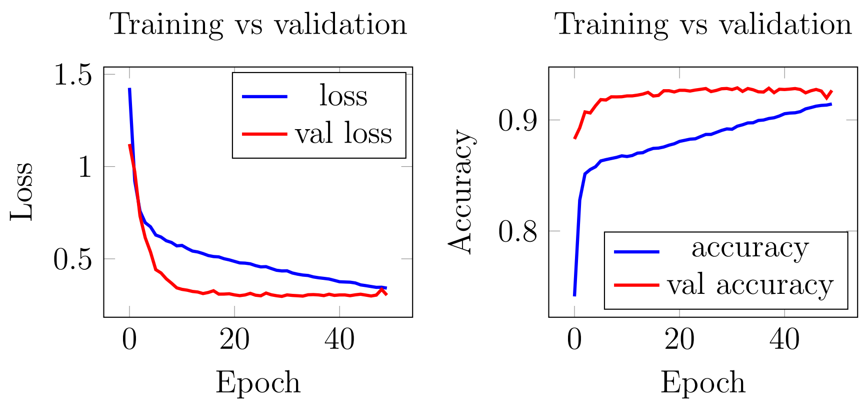
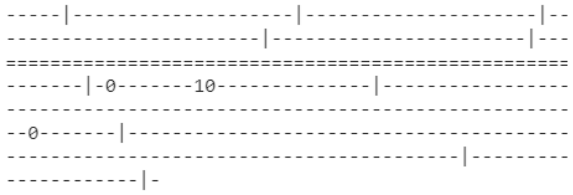


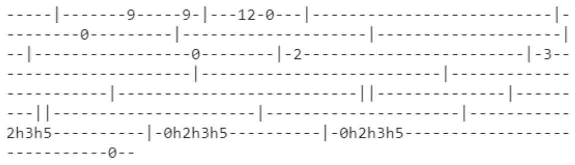
Figure 10: Row model training history

Once again a promising convergence between the training and validation results for each epoch. Figure 11 shows a comparison of composition results between this model and one trained on 20 epochs.



```
-----|--------------------|---------------------|--
-----------------------|-----------------------|---
===================================================
-------|-0-------10--------------|-----------------
---------------------------------------------------
--0-------|----------------------------------------
-----------------------------------------|---------
------------|-
```

(a) 20 epochs



```
-----|-------9-----9-|---12-0---|--------------------------|-
--------0---------|---------------------|---------------------|
--|-----------------0---------|-2-------------------------|-3--
--------------------|---------------------------|-------------
-----------|---------------------------||---------------|------
---||-----------------------|---------------------|-----------
2h3h5----------|-0h2h3h5-----------|-0h2h3h5------------------
-----------0--
```

(b) 50 epochs

Figure 11: Comparison of 2 generated tabs by a row-heavy trained model. Multiple lines are a result of line wrapping. They are both in fact only a single line.

4.3 Column-heavy

Columnated datasets is where results start to show some promise beyond that of simple training performance. Figure 6b shows how they are styled, and as you can also see it bears a striking resemblance to 6d, which is the filtered datasets.

4.3.1 Method

Thanks to the regularity of the format, the data can be easily interpreted for either character to character, or sequence to sequence datasets. Each row in the matrix would be a chord, and given how musical structure follows sequences of chords it is safe to decide that sequences would be 6 characters representing the 6 strings on each chord. With this I created and trained models using datasets designed for character prediction and sequence prediction.

Training both models for 50 epochs on the largest possible dataset of 69268 rows (415608 characters) was done only a small number of times due to time constraints. Nevertheless, according to the following accuracy plots, there wouldn't be much need for training beyond 20-30 epochs, as there were diminishing returns beyond these points.

Character to character was trained using a sequence length of 140, whilst both models were trained with a batch size of 16. The reason the sequence length is a multiple of 7 is because I wanted the model to learn using sets of chords, 6 characters + 1 newline character

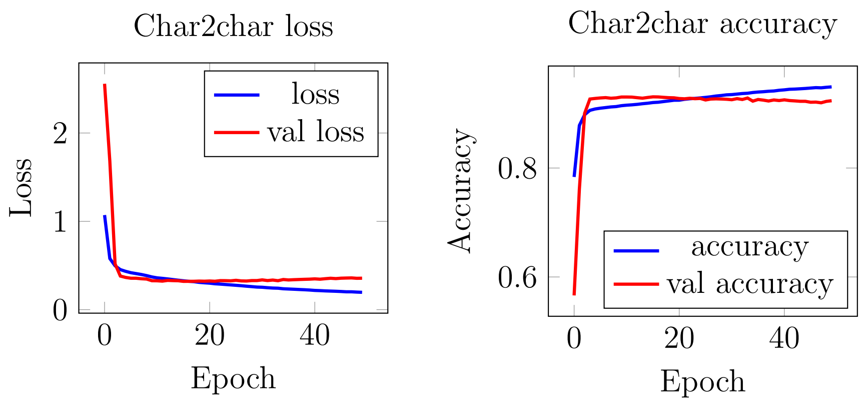


Figure 12: Character model training accuracy

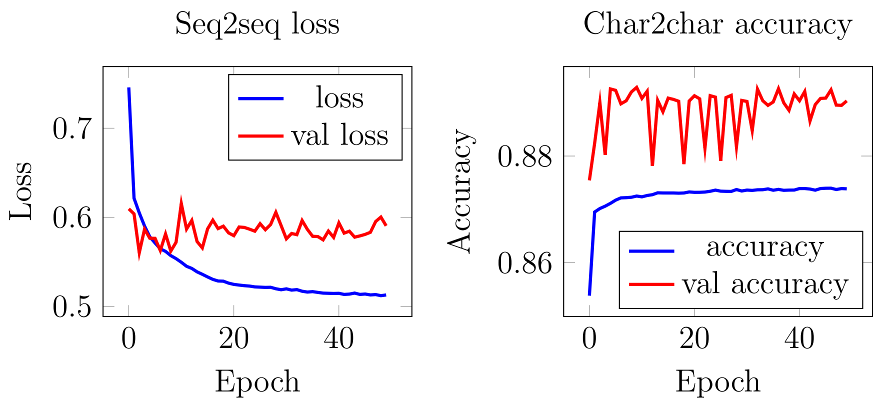
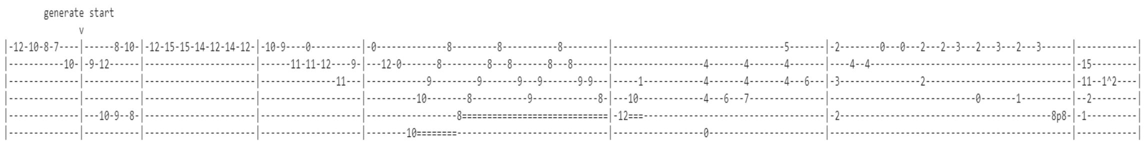


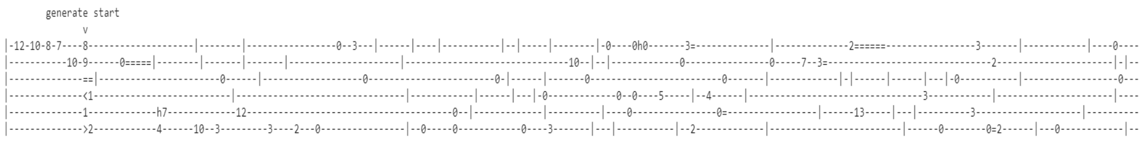
Figure 13: Sequence model training accuracy

Character to character seems to show promise, whilst the sequence to sequence model is very erratic during it's validation steps. I do not believe that model manages to learn anything with regards to seeing unexplored data. Disregarding those results, the predictions using each trained model at a temperature of 0.9 can be seen in figure 14



```
      generate start
            v
|-12-10-8-7----|------8-10-|-12-15-15-14-12-14-12-|-10-9----0---------|-0------------8--------8----------8---------|------------------------------5-------|-2-------0--0--2--2-3--2--3--2--3-----|-----------|
|----------10-|-9-12------|---------------------|------11-11-12----9-|---12-0-------8--------8--8-------8--8-------|---------------4------4------4------|---4-4------------------------------------|-15--------|
|-------------|-----------|---------------------|--------------11---|-----------9--------9------9--9------9-9---|----1-----------4------4------4--6---|-3-------------2---------------------------|-11--1^2----|
|-------------|-----------|---------------------|-------------------|---------10-------8---------9-----------8-|---10-----------4--6--7--------------|--------------------------0-----1---------|--2--------|
|-------------|---10-9--8-|---------------------|-------------------|----------------8==========================|-12===---------------------------------|-2-------------------------------------8p8-|-1---------|
|-------------|-----------|---------------------|-------------------|-------10=======------------------------------|---------------0----------------------|------------------------------------------|-----------|
```

(a) Character to character



```
      generate start
            v
|-12-10-8-7---8------------------|--------|---------------0--3--|------|----|----------|--|-----|--------|-0---0h0-------3=------------|------------2=====----------------3------|-----------|---0----
|----------10-9-----0=====|--------|-------|-------|---------------------|-------------------------10--|--|-----------0-------------0----7--3=-------------------------2-------------------|-|--
|-------------==|--------------------0-----|------------------0----------------------0-|------|------0-------------------------0------|-----------|-|-----|-----|---|-0---------|----------------0---
|------------<1----------------------|----------------------------|-----------|-----|---|-0-----------0--0---5----|--4-----|----------------------------------3----------|------------------|---
|------------1-----------h7-----------12-----------------------------0--|-----------|---------|---0--------------0=---------------|-----13----|--|---------3------------------|---------
|------------>2----------4-----10--3-------3---2--0--------------|--0----0----------0---3------|---|---------|--2-----------|-----------------------|-----0------0=2----|--0---------|--
```

(b) Sequence to sequence

Figure 14: Comparison of composition for character to character, and sequence to sequence

5 Results

First, a point to make is that in all the different training sessions for the character to character models, we see that the plots are quite promising with regards to training progress and stability. Table 2 shows the configuration and final results of these trainings.

Data form	Seq. length	batch size	epochs	Loss (train|val)		Accuracy (train|val)	
Multi	140	32	20	0.42	0.41	0.90	0.91
Row	140	32	50	0.34	0.30	0.91	0.92
Column	140	16	50	0.19	0.35	0.95	0.92

Table 2: Final results from training the different character predictor models

Something to note is that these results are quite good. High accuracies, with minimal loss, along with favorable convergence between training and validation. However, the resulting compositions for both the multi-track, and the row-heavy representations failed to uphold the structural requirements for them to be deemed valid tablature.

In the case of the multi-track model's composition, we can observe that there is an attempt by the model to uphold the structure it has been learning from the data, but it fails to really learn these patterns, as I can only assume is because of how irregular it is. For row-heavy, the model completely fails at even trying to understand the crucial 6-line structure of the data it is trained upon, producing only a single line of tablature characters when composing.

The column-heavy model, though, produces not only favorable structure, but also parts that seem to uphold some musical rules. Using an online tool[17], I played the composed tabs and personally do not think it sounds good. But, disregarding if the music sounds *good*, I believe that at least the tablatures contain elements of musicality. There are parts that harmonise nicely, having both a short span of repeating fingering, along with chords harmonising at regular intervals. I believe the model managed to pick up on elements of musical harmony and temporal dependence, on top of the physical structure of the training data.

The reason for how these results came to be comes down to a combination of the physical structure of the data the models are trained upon, along with the sequence length, or window size, given to the models upon training. A CharRNN model inherently reads text from left to right, and as such the sequencing must be adjusted in such a way that it envelops the parts of the data that is crucial for the model to learn, i.e. recognisable patterns such as the usage of \n in tablature structure. This, is infeasible for the structures that had large, unclear patterns like that of the row-heavy representation, or many smaller patterns like that of the unclear block width

[17]toFret: `http://www.tofret.com/tablature-player.php`

of the multi-track representation. In fact, what was clear about the column-heavy representation was that because of the simple and highly repetitious pattern of the data, i.e. 6 characters followed by \n, the model was able to seemingly perfectly represent it when being used to generate new music. Along with this, it seemed to learn other patterns from the music itself, as seen in figure 14a.

Additionally, there is the results of the sequence to sequence predictor. As it was not limited to learning the structure of the tablature, it would be expected to better learn the musical features of the data. But with regards to the training results, and the generated tablature, it does not seem like it had really learned anything at all, and the generated tablature is simply some random distribution of the characters present in the vocabulary of the training data. The structure is, of course, simply an emergence of how it runs. It always generates chords of 6 characters, and isn't limited by the structure of the tablature it reads.

6 Discussion

Something to note is that the multi-track representation didn't completely fail in it's attempts at forming the tablature structures. As mentioned about the dependency of sequence length to structure form, I believe that had the model been more stable during training it would be able to potentially learn the structure, somewhat. The most glaring issue with the structure of the representation is that it is quite complex. Not only are there clear 6-line chunks, but each chunk seem to be of arbitrary width. This turns the problem of representation into an issue which is different to that of the issues with the row-heavy data. For the row-heavy data, there simply is no feasible way to adjust the sequence lengths such that the structure can be learned, because enveloping an entire line in a single sequence potentially thousands of characters long is not efficient model input management. For the multi-track representation, the sequence length *could* be adjusted such that it envelops several of these blocks, in theory, but it would also make the size of training batches incredibly large, and again mean that the model has an abnormally large input, similar to that of the infeasible row-heavy suggestion.

Something to think about for the column-heavy dataset is that the way it is formed helps to highlight the chord-like structure of music itself, and not only the simple 6 characters + \n structure. Along with this, I believe it was a mistake to form it's dataset as pairs of sequential chords. Thinking about it with regards to the character predictor input structure, it would rather be better to take several sequential chords as input to the sequence predictor. This would be more similar to the character predictor with the column-heavy dataset, in that logically speaking, sequences of 140 characters as input means 20 chords are being input every 7 intervals of inputs (i.e. every input, the 140-width window slides one character along the data). Essentially, the character predictor sees entire sets of chords as well as their transitions quite clearly. This would be replicated in the sequence predictor if the inputs were of sets of chords, rather than pairs.

7 Further work

The data that is used is made from scouring through a set of around 3000 files from ClassTab that are strictly evaluated only for their tablature, with no regards towards musical style, theme, genre, etc. As such, because classical music is varied and vast, there is an arbitrary mix of musical styles, themes, lengths, etc. that the models are introduced to. Because of this, they are given a mix of musical features to learn from, forcing them to expend learning capacity towards more features than potentially necessary. Whilst recurrent networks excel at learning from data with many features, it would be wise to limit unnecessary complexity. Further work would be needed with gathering and more carefully selecting data for testing the model's capability of better specialising to specific musical features, e.g. music made by specific artists.

Alongside this, more work is needed to solve other issues and limitations for the preprocessor itself, like improving it's ability to filter out non-tab data from provided files.

The multi-track model seemed to show promise with it's prediction results, and I believe with more training, a variety of sequence lengths, and less complex data it could viably learn to predict tabs with multiple tracks. One future test could be to make sure the widths of each chunk stays constant, such that the model can learn to predict them more easily.

The sequence predictor, whilst faulty, shows promise in concept and I believe with more experimentation upon it's implementation it could go further. I believe one mistake was to make the input data come in pairs, rather than sets of chords in some matrix form. I believe it would be valuable to look into this and correct it. As previously stated, if the models weren't constrained to learn the tablature structure on top of the musicality, they would more readily expend resources towards learning the latter and might produce better results. On top of that, I believe fixing the mistake would result in the model being much more receptive of the music it learns from, with regards to temporal dependencies.

With the promising results from the models trained on columnated datasets, it would be beneficial to look into training models on data with more limited features.

Measures to manipulate and filter the datasets are implemented in the preprocessor. I hypothesize that the sequence to sequence model might perform better if it were to train upon data with less redundant data, like non-musical characters. Also, as mentioned in the loader's section, the datasets invariably have a lot of silence inside it, which could contribute to the machines' predilection towards predicting silence. Using the filtered datasets, this should be looked into further.

8 Conclusion

These experiments have been used to uncover how different potential representations of tablature can be used to train models for prediction of tablature music using character datasets. Whilst training showed promising results, tests show that for representations that have complex or non-regular structure, like the standard multi-track and rows, the model fails to pick up the structural features of the data. Whilst predicting characters alone went quite well for the models, understanding tablature structure was too complex if there weren't enough structural indicators like newline patterns to denote the frequency and format of the other musical characters.

In conclusion, I believe that the best and most promising representation of tablatures for a character to character predictor of music is the columnated tabs. The structure of this representation is regular, and allows the model to quickly pick up on it and focus on learning the more complex musical features of the tabs. Furthermore, the model trained upon this columnated data seems to pick up on musical features like repetition, timing and harmony, and with a more carefully selected dataset might even be able to learn more musical features like styles and themes.

9 Final discussion

Some points can be made about approximations and limitations of some of the tools. The preprocessor functions quite well, although it is crude and strict and can potentially miss some useful tabs or other items. Analysing some of the filtered files shows that there could be some slight fault in the filtering methods, and should be looked into further to generate more robustly filtered files. Additionally, the loader works well, but it misses some characters when it statistically analyses the tab file for activations and vocabulary usages, particularly special characters like newline and space, although this isn't too big of a problem since these characters do not affect the musical value of the tabs at all.

I think that the work as it stands now only partially answers the original questions stated in the introduction. Whilst I am certain of my claims regarding the row and multi-track representations being inferior to the column-heavy one, I believe that the sequence predictor should be given further work towards having more representative training data and could potentially rival the character predictor model if it managed to train properly. A fact to consider about the sequence predictor is that it doesn't need to also learn the structure of the tabs. It's input and output format means that the structure is given. This, as previously stated, means the model can focus on learning musical structure instead. The limiting factor is it's ability to train with properly constructed training and validation data. I cannot make a conclusion on whichever of the models are the superior to one another, as I feel the sequence model is unjustly represented in the work. With this, further work is needed. Regardless, I feel it is still clear that the columnated tablature is superior to the other representations in the case of the *character to character* predictor

References

[1] A. Ibrahim, R. Alexander, M. Shahid, U. Sanghar, R. Dsouza, and D. Souza, “Control systems in robotics: A review,” *International Journal of Engineering Inventions*, vol. 5, pp. 2278–7461, 04 2016.

[2] T. Epelbaum, F. Gamboa, J.-M. Loubes, and J. Martin, “Deep learning applied to road traffic speed forecasting,” 09 2017.

[3] L. Galante and R. Banisch, *A Comparative Evaluation of Anomaly Detection Techniques on Multivariate Time Series Data*. PhD thesis, 01 2019.

[4] B. Soni, D. Patel, and M. López-Benítez, “Long short-term memory based spectrum sensing scheme for cognitive radio using primary activity statistics,” *IEEE Access*, vol. PP, pp. 1–1, 05 2020.

[5] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[6] A. Sherstinsky, “Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network,” *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, Mar 2020.

[7] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, “Midinet: A convolutional generative adversarial network for symbolic-domain music generation,” 2017.

[8] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” 2016.

[9] Y. Deng, Z. Xu, L. Zhou, H. Liu, and A. Huang, “Research on ai composition recognition based on music rules,” 2020.

[10] D. Tuohy and W. Potter, “A genetic algorithm for the automatic generation of playable guitar tablature,” 2005.

[11] G. Li, S. Ding, Y. Li, and K. Zhang, “Music generation and human voice conversion based on lstm,” 2021.

[12] J.-Y. Liu and Y.-H. Yang, “Event localization in music auto-tagging,” 2016.

[13] Y.-H. Chen, Y.-H. Huang, W.-Y. Hsiao, and Y.-H. Yang, “Automatic composition of guitar tabs by transformers and groove modeling,” 2020.

[14] M. Majidi and R. M. Toroghi, "Music harmony generation, through deep learning and using a multi-objective evolutionary algorithm," 2021.

[15] S. Dadman, "Synthetic composition of music," Master's thesis, UiT The Arctic University of Tromsø, June 2020.

[16] C. Hausner, "Design and evaluation of a simple chord detection algorithm," 2014.

[17] A. Lerch, "Audio content analysis," 2021.

[18] B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova, "Music transcription modelling and composition using deep learning," 2016.

[19] "Reading guitar tabs for beginners." `https://www.schoolofrock.com/resources/guitar/reading-guitar-tabs-for-beginners`.

[20] "Classtab: Classical guitar tablature." `https://www.classtab.org/`.

[21] Q. Xi, R. M. Bittner, J. Pauwels, X. Ye, and J. P. Bello, "Guitarset," Aug. 2019. Changes from version 1.0.1: - Added Key Annotation in each jams file - Removed impossible notes (negative frets) - Updated Pitch Contour annotations to have the correct '"index"' field in 'obs.value'. - Cleaned up .zip files.

[22] M. McVicar, S. Fukayama, and M. Goto, "Autoleadguitar: Automatic generation of guitar solo phrases in the tablature space," *2014 12th International Conference on Signal Processing (ICSP)*, pp. 599–604, 2014.

[23] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow.* 1005 Gravenstein Highway North, Sebastopol: O'Rielly Media, 2019.

[24] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," 2015.

[25] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 8, pp. 1553–1565, 2014.

[26] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Pearson Prentice Hall, 1994.

[27] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.