Generating Guitar Tablatures with Neural Networks

Elias Mistler

Master of Science

Data Science

School of Informatics

University of Edinburgh

2017

Abstract

In contemporary guitar music, guitar tablatures represent an efficient way of writing down playing instructions for a guitarist. Contrary to classical sheet music notation, tablatures describe one distinct, ideally easy way of playing a melody. This makes tablatures an effective and intuitive notation system, but makes transcribing sheet music to a suitable tablature a tedious task requiring expert knowledge. The project at hand aims to automate this procedure and predict the optimal tablature for any input melody.

We show two different approaches, both based on Machine Learning:

In our first approach, guitar frettings are directly predicted based on previously played frettings. This is accomplished by a *Long Short-Term Memory Recurrent Neural Network*, enhanced by a notion of *intention* of the musical outcome.

Our second approach predicts the difficulty of a fretting in terms of a cost function, rather than predicting the ideal fretting directly. The cost function is based on conditional probabilities in the training data and estimated by a Feed-forward Neural Network.

The agreement between our best approach and published tablature is measured at 72.9% median validation accuracy, higher than what we achieve by applying a heuristic similar to previous approaches. Subjective evaluation shows that all generated tablatures are playable and that in many cases, divergence from the published tablature is well justified.

Acknowledgements

I would like to thank my supervisor Dr. Paul Anderson, who had originally suggested a project similar to the one at hand and was very open towards my approaches and ideas. Thanks to my family, love, and friends for supporting me and being patient and understanding. A big thank you to my fellow musicians who helped me with their feedback on my ideas and the system. Finally, thank you to *Irn Bru* - I couldn't have done this without you!

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Elias Mistler*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

In contemporary guitar music, reading and writing traditional sheet music is uncommon and often seen as unnecessary overhead for both composer and guitarist. In fact, many renowned composers and guitarists are said to be unable to even read sheet music, including Jimi Hendrix, Eric Clapton, Stevie Ray Vaughan and Tom Morello [Barnes, 2014], as well as Eddie Van Halen, The Beatles, Elvis Presley, Slash, Tommy Emmanuel and many more [Obacom, 2015].

Guitarists argue that sheet music notation is very much targeted to keyboard instruments such as the piano, whereas it is generally of little use for the guitar, especially when considering the possibility of using different tunings on the guitar and transposing. The tablature notation on the other hand is optimised for stringed and fretted instruments and are therefore claimed to have a bigger use to guitarists. [Reid, 2016]

While contemporary guitarists seem to prefer tablatures over sheet notation, or are not even able to read sheet music, a huge amount of music has only ever been written down in standard sheet music notation. Transcribing a piece from sheet notation to a suitable tablature however is a tedious process requiring expert knowledge. Consequently, the material is inaccessible to aspiring guitarists who would like to enhance their repertoire by adding classical pieces or playing melodies which were originally intended for other instruments.

The project at hand aims to automate the transcription from standard sheet music to suitable guitar tablatures through the use of Machine Learning. We hope to enable a wider range of guitarists to extend their repertoire and to be creative with music originally published in sheet music notation. This could facilitate the creation of interesting

re-interpretations, similar to Jimi Hendrix' version of the *Star Spangled Banner* (48 tablatures online on `ultimate-guitar.com`) or *Canon Rock* (100+ tablatures), the famous rock version of Johann Pachelbel's *Canon in D Major* with countless videos online and tens of millions of views.

1.2 Focus of the project

This project aims to automate the transcription from staff notation to guitar tablatures. As we will show in chapter 2, there have been approaches to solve the fretting problem, mainly along with the guitar fingering problem, i.e. which fingers should be used to play a given fretting. These approaches are largely based on modelling biomechanical difficulty and explicit finger positions. We will not model these finger settings and instead focus on learning solutions to the fretting problem, i.e. we will predict the strings and frets to be used, but no recommendation as to which finger to use. We do this for two main reasons:

1. In the guitar tablature community, it is highly unusual to annotate tabs with fingering instructions. Musicians are expected to play from tablatures without actively thinking about which fingers to use. Furthermore, for a single tablature, there can be different, equally good, fingerings.

2. As it is unusual to annotate tablatures with fingerings, we could only find a small number of such tablatures. We judged this number to be insufficient for a Machine Learning approach. For the fretting problem, however, plenty of usable tablatures are available online.

Contrary to previous approaches, we keep the system implementation as independent as possible from specifics of the used instruments, such as the number of strings and frets or the tuning applied. We hope to enable the system to generate tablatures for any stringed instrument, such as guitars in different tunings, bass guitars, mandolins and ukuleles. The system handles polyphonic music and is robust, i.e. always outputs a valid tablature for a given input piece, given that the piece is playable on the guitar.

1.3 Structure of the report

Chapter 2 will give a detailed description of the guitar fretting problem and the associated challenges. We will further present important algorithms and give an overview of

existing approaches to the guitar fretting and fingering problems. In chapter 3, we will elaborate on our specific approach to the problem and on its software implementation. Chapter 4 covers the evaluation of our results and chapter 5 summarises the project, its findings and gives directions for future research.

Chapter 2

Background

In this chapter, we summarise important background knowledge to the proposed approach. In section 2.1, we explain the guitar fretting problem in detail, highlighting issues to be addressed in our system. Section 2.2 gives a brief introduction to the underlying Machine Learning algorithms. Section 2.3 is concerned with past papers on the guitar fretting problem.

2.1 The Guitar Fretting Problem

In this section, we will elaborate on the guitar fretting problem. As we can closely relate the fretting problem to the difference between standard sheet music notation and guitar tablatures, we will first give a basic explanation of both notation systems, as well as the MIDI representation of pitches.

2.1.1 Sheet Music Notation

Sheet music notation, or *staff notation*, is the most common form of notation across instruments and genres. It is concerned with describing the musical intention, i.e. a description of a musical outcome. 2.1 shows a simple example of sheet music notation.

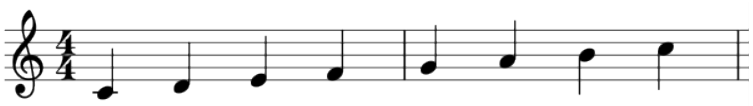


Figure 2.1: The ascending C-major scale in sheet music notation, from *middle C*, 261.6Hz to *tenor C*, 523.2Hz)

On a five-line grid, pitches are organised vertically. A note can either be located on a line or between two lines. This is done for reading purposes and has no musical implications. The vertical dimension is however based on the notion of a C major scale rather than physical pitch distance.

Importantly, this structure corresponds to the keyboard layout of a piano but has no direct connection to the way pitches are organised on a guitar fretboard. Figure 2.1 shows how the C major scale fits into the five-line grid with equal distances. Figure 2.2 shows how the C major scale and thus the sheet music notation directly correlate to the piano keyboard in that the C major scale is represented by the white keys.

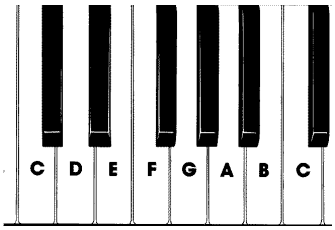


Figure 2.2: A piano keyboard with the annotated white keys constituting a C major scale.
Source: `www.playpiano.com`

In a major scale, the pitch distance between the third and fourth, as well as seventh and eighth note is a semitone, all other distances are whole tones, i.e. two semitones. To account for the semitones in between, the accidentals ♯ and ♭ are used to increment or decrement a note by a semitone. Figure 2.3 shows the chromatic C scale, consisting of all available tones within one octave. ♯ and ♭ historically have different musical implications with respect to key and harmonics in a piece. In the contemporary, equal-tempered tuning, however, an incremented lower and a decremented higher note (e.g. $F♯$ and $G♭$) correspond to the same physical frequency, the same *pitch*.

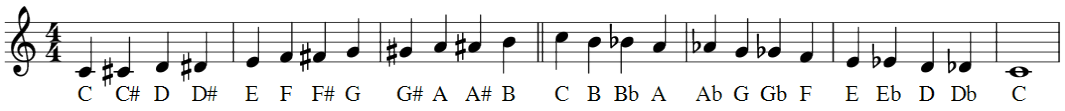


Figure 2.3: The chromatic C scale in sheet music notation. Note that the ascending and descending parts consist of the same pitches, only notated with different accidentals.

Doubling the physical frequency results in the perception of a higher version of the same pitch, called the *octave*. Octaves can be explicitly marked with ascending octave indices, e.g. C_2 and C_3.

The time dimension is represented by the horizontal axis. Note lengths are encoded by the note shape, symbolising a relative note duration with respect to an underlying beat. Additional annotations can determine time signature, dynamics, tempo, tempo variations and other musical properties.

2.1.2 MIDI Notation

A straightforward representation of physical pitches is given by simply enumerating pitches as a series of semitones, starting from $C_0 = 0$ at $8.18Hz$. As this approach is used in the widespread *Musical Instrument Digital Interface (MIDI)* format [MIDI, 1996], it is often referred to as *MIDI pitch notation*. Enumerating the pitches rids the notation of musical connotations and ambiguities in sheet music. It instead allows for straightforward computation of pitches and pitch distances in semitone space. Table 2.1 illustrates how the MIDI notation assigns a single, distinct representation to every note.

name	C_3	$C\sharp_3$	D_3	$D\sharp_3$	E_3	F_3	$F\sharp_3$	G_3	$G\sharp_3$	A_3	$A\sharp_3$	B_3	C_4
alt.		$D\flat_3$		$E\flat_3$			$G\flat_3$		$A\flat_3$		$B\flat_3$		
MIDI	36	37	38	39	40	41	42	43	44	45	46	47	48

Table 2.1: Octave of tones with their sheet music names and the assigned MIDI pitch

The purely semitone-based nature of the MIDI notation corresponds well with the guitar fretboard, as moving one fret up on the fretboard increases the resulting pitch by a semitone. Consequently, a MIDI pitch can be calculated from the guitar fretboard as the sum of the string base pitch and the number of the depressed fret. Playing the second string in the third fret thus yields: $45(A_3) + 3 = 48(C_4)$

2.1.3 Guitar Tablatures

While sheet music notation describes the musical content, it can also be useful to give direct instructions on how to play a piece. *Tablatures* capture these instructions and are especially popular among beginner guitar players, but also have great use for more experienced guitarists in that they are straightforward and efficient to read and write. Figure 2.4 shows an example of a tablature.

A tablature is made of six horizontal lines symbolising the guitar's strings, where the lowest line represents the lowest sounding string. In the following, we will always

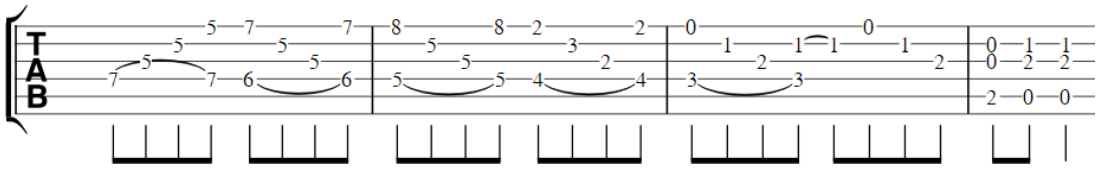


Figure 2.4: Beginning of a tablature for Led Zeppelin's *Stairway to Heaven* as a text tablature. Source: `ultimate-guitar.com`, rendered in *MuseScore*

enumerate the strings from low to high, i.e. calling the lowest-sounding string first string.[1]

As the lines in tablatures represent strings, numbers on these lines represent the fret on which to play the string. When a string should be picked without depressing a fret on the fretboard, this is represented by a 0 on the respective line. A tablature may have additional annotations, mostly either taken over from sheet music or introducing specific instructions on guitar techniques, such as pinch harmonics, palm muting, bendings, hammer-ons and pull-offs.

In their simplest form, tablatures can be represented by mono-spaced ASCII characters, as shown in figure 2.5. These simple text tablatures allow for quick notation and easy sharing of tablatures which we assume to be an important reason for the great popularity of guitar tablatures.

```
E|-------5-7-----7-|-8-----8-2-----2-|-0---------0-----|-------
B|-----5-----5-----|---5-------3-----|---1---1-----1---|-0-1-1-
G|---5---------5---|-----5-------2---|-----2---------2-|-0-2-2-
D|-7-------6-------|-5-------4-------|-3---------------|-------
A|-----------------|-----------------|-----------------|-2-0-0-
E|-----------------|-----------------|-----------------|-------
```

Figure 2.5: Beginning of a text tablature for Led Zeppelin's *Stairway to Heaven* as a text tablature. Source: `ultimate-guitar.com`

2.1.4 Tablature Transcription from Sheet Music

On a guitar, there are usually six strings, tuned E_3, A_3, D_4, G_4, B_4 and E_5, or, in MIDI notation, $[40, 45, 50, 55, 59, 64]$. On a guitar with 24 frets, we can therefore play an E_5 (pitch 64) on multiple locations, from the 24^{th} fret on the first string to the open sixth

[1] Note that often, the strings are instead enumerated from high-sounding to low-sounding. For ease of computation, however, the enumeration from low-sounding to high-sounding makes more sense. Note further that the name "lowest string" may lead to confusion, as the lowest-sounding string is in the highest physical location.

string. Figure 2.6 shows all these locations on the fretboard, the possible *note frettings* of E_5.

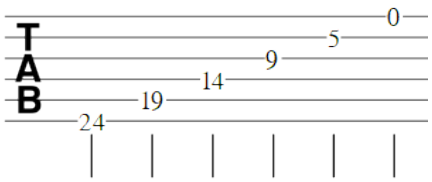


Figure 2.6: The six different ways of playing a high E on a 24-fret guitar in standard tuning

Similarly to E_5, most other pitches can be played in different locations on the fretboard. Table 2.2 shows where each pitch can be played on a 24-fret guitar in standard tuning.

from	to	#strings	E_3	A_3	D_4	G_4	B_4	E_5
E_3	$G\sharp_3$	1	+					
A_3	$C\sharp_4$	2	+	+				
D_4	$F\sharp_4$	3	+	+	+			
G_4	$A\sharp_4$	4	+	+	+	+		
B_4	$D\sharp_5$	5	+	+	+	+	+	
E_5	E_5	6	+	+	+	+	+	+
F_5	A_5	5		+	+	+	+	+
$A\sharp_5$	D_6	4			+	+	+	+
$D\sharp_6$	G_6	3				+	+	+
$G\sharp_6$	B_6	2					+	+
C_7	E_7	1						+

Table 2.2: Number of fretting possibilities and associated strings per pitch range on a 24-fret guitar in standard tuning

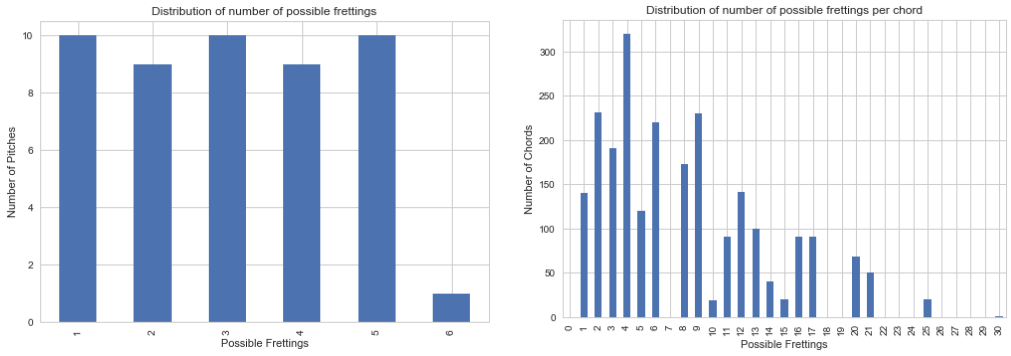
Table 2.3 shows the schematics of MIDI pitches on the guitar fretboard. The highlighted frets indicate the crossover point with the respective next string, i.e. where the resulting pitch is equal to the pitch of the open next string.

Figure 2.7 shows an aggregated distribution view of the number of possible frettings: A single pitch is almost uniformly likely to have any number of possible frettings between 1 and 5 (figure 2.7(a)). Two simultaneously played pitches have a much wider

string: 6^{th}	64	65	66	67	68	69	70	71	72	...
5^{th}	59	60	61	62	63	**64**	65	66	67	...
4^{th}	55	56	57	58	**59**	60	61	62	63	...
3^{rd}	50	51	52	53	54	**55**	56	57	58	...
2^{nd}	45	46	47	48	49	**50**	51	52	53	...
1^{st}	40	41	42	43	44	**45**	46	47	48	...
fret:	0	1	2	3	4	5	6	7	8	...

Table 2.3: Systematics of a guitar fret board with MIDI pitches

distribution of possible frettings (figure 2.7(b)). We will further consider any collection of simultaneous pitches a *chord* and therefore, every combination of note frettings resulting in the chord a *chord fretting*.



(a) Single note frettings

(b) Two-note chord frettings

Figure 2.7: Distribution of the number of possible frettings

As we can see from figure 2.7(b), there exists a single two-pitch chord with 30 theoretically possible frettings. Similarly, three-pitch chords can have up to 120 frettings and four-pitch chords up to 360.[2]

The maximum number of possible frettings N can thus be approximated by the binomial coefficient of the number of strings s and the number of simultaneous pitches p, as shown in equation 2.1. The number of possible fretting sequences increases exponentially with the length of the sequence.

$$N \approx \binom{s}{p} = \frac{s!}{p!(s-p)!} \tag{2.1}$$

[2]These numbers were retrieved from explicitly running our fretting finder algorithm on all possible combinations of pitches on a 24 fret guitar in standard tuning.

Figure 2.8 shows the eight different possible frettings of the A5 chord on a guitar in standard tuning. These frettings are manually sorted by ease of play, where the first two are very good frettings, the following three are inconvenient and the last three frettings are completely unfeasible to play as the frets span a much larger range than a human hand can support.

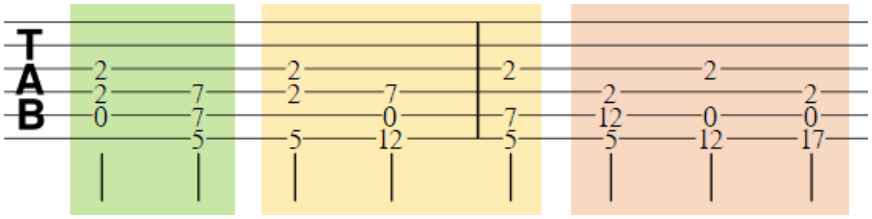


Figure 2.8: The possible 8 frettings of an A5 ”power chord” on a 24-fret guitar in standard tuning, sorted manually by ease of play

The task of tablature transcription from sheet music is concerned with finding a viable fretting sequence from these theoretical possibilities. Importantly, the majority of frettings is actually unfeasible to play.

Due to the complexity of the fretting space, finding a good guitar fretting is a complex, high-dimensional problem. In section 2.3, we will show that past research on this field was mainly concerned with manually finding simple rules and heuristics, capturing the difficulty of the possible frettings. The project at hand is however concerned with inferring viable frettings automatically from data, through the use of Machine Learning.

2.2 Machine Learning

This section is concerned with the basics of Machine Learning. It loosely follows the structuring and explanations from "Pattern Recognition and Machine Learning" [Bishop, 2006], a popular and well-written introduction to a broad range of Machine Learning approaches. The book should be consulted to gain a detailed understanding, as we can only give a conceptual introduction of the field in the format of this dissertation.

2.2.1 Overview

Machine Learning is a discipline in the study of Artificial Intelligence concerned with algorithms which learn from data. The *learning* procedure can be understood as inferring patterns, or even rules, from examples. Typical tasks in Machine Learning include classification of images or music recordings, speech recognition, financial risk assessment, and strategy learning in games or simulations. Typically, Machine Learning approaches are structured by their learning goal as follows:

1. *Supervised Learning*: An algorithm is trained to predict a dependent variable from a number of explanatory variables. Typically, the dependent variable is known for a set of (historical) data and to be predicted for new, unseen data.

   (a) *Regression* models predict a numerical outcome, e.g. a scoring function or a credit limit to assign to a customer of a financial institute.

   (b) *Classification* models have distinct outcomes, known as *classes*. Typical examples include face recognition in images or musical style recognition from audio recordings. Classification models can often be built on top of regression models by redefining regressions in form of a probability distribution.

2. *Unsupervised Learning*: An algorithm is trained without an explicit target variable. This approach is generally useful for exploratory purposes, e.g. uncovering internal structures and patterns of the data. Results on unsupervised learning can also be useful as a preprocessing step, as inferred representations may well correspond to meaningful high-level features of the data.

   For example, [Le, 2013] observed how an unsupervised system learned to recognise the presence of a human face in images. Unsupervised learning is often

applied in the form of *clustering* data points and reducing the dimensionality of data based on identified clusters or patterns.

3. *Reinforcement Learning*: An agent is trained to make decisions based on a reward signal from its environment. During the learning procedure, the agent interacts with the environment and adjusts its own behaviour to maximise the expected reward.

   A typical application example of reinforcement learning is autonomous navigation in robot agents or self-driving cars. For example, [Tamar et al., 2016] propose a reinforcement learning system for robots which is able to plan ahead and thus make reusable learning progress.

In the rest of this section, we will focus on supervised learning, which our approach to the guitar fretting problem is based on.

2.2.2 Linear Regression

Linear regression is a comparatively simple statistical method to relate vectors of d input variables, $\vec{x}$, to a dependent target variable, y. y is approximated as a linear combination $\hat{y}$ of the input variables x_i. Equation 2.2 shows the linear model with a weight vector $\vec{w}$. x_0 is set to 1 and the corresponding weight w_0 is called the *bias* of the model.

$$\hat{y} = \sum_{i=0}^{d} w_i x_i = \vec{w}^T \vec{x} \tag{2.2}$$

We call the error of a single prediction, $y - \hat{y}$, *residual* of the prediction. The error across a data set is usually referred to as *prediction loss* and can be calculated as a function of these residuals, commonly using the mean squared error function over all n training examples:

$$O(\vec{w}) = \frac{1}{n} \sum_{j=1}^{n} (y_j - \vec{w}^T \vec{x}_j)^2 \tag{2.3}$$

Using the mean squared error as loss function, we can solve for the optimal linear regression weights algebraically by using the pseudo-inverse of the $n \times d$ matrix X of all training input vectors $\vec{x}_i^T$:

$$\hat{\vec{w}} = (X^T X)^{-1} X^T \vec{y} \tag{2.4}$$

Linear regression will always find the best linear fit to the data. However, interesting dependencies are rarely linear. As a simple solution to making the model more flexible, the input data can be preprocessed, e.g. by adding polynomials of the original input as new features (sometimes called *Polynomial Regression*).

2.2.3 Logistic Regression

For classification tasks, the linear model can be adapted to produce class probabilities $P(c|\vec{x}) = \sigma(\vec{w}^T\vec{x})$, where σ is the *sigmoid*, or *logistic squashing function*:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

Due to the interpretation as probability, the loss function needs to reflect the confidence of the predicted classification. Logistic regression can thus be fit by optimising for the log likelihood $L(\vec{w})$ given by equation 2.6.

$$L(\vec{w}) = -\log P(X,\vec{y}|\vec{w}) = -\sum_{i=1}^{n} \log P(\vec{x}_i, y_i|\vec{w}) \tag{2.6}$$

Unlike in linear regression, the optimisation in logistic regression cannot be done explicitly. The error function is however *convex* and can therefore be optimised with guaranteed convergence to the global optimum by using a *gradient descent* algorithm: Starting from a random point in weight space, the weights are iteratively adjusted to achieve a movement in direction of smaller errors.

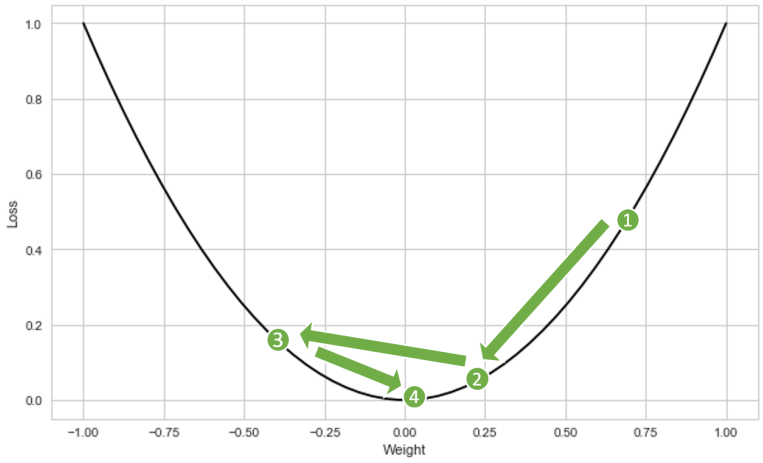


Figure 2.9: The gradient descent algorithm visualised

The algorithm is visualised in figure 2.9: From a random starting point (1), the weight is adjusted in direction of smaller gradients, following down to (2). At the

step from (2) to (3), the step size is too big and the algorithm leaps over the optimum. As the loss increases, the direction is inverted and the algorithm moves down to (4). This procedure is repeated until the gradient of the loss functions is less than a given threshold.

While the logistic regression model is very similar to linear regression in its systematics, there is an important difference in that the logistic squashing function results in the nonlinear mapping $\vec{x} \rightarrow \sigma(\vec{w}^T\vec{x})$.

2.2.4 Neural Networks

An *Artificial Neural Network* is a graph structure consisting of small computation units called *neurons*, vaguely resembling axons in a human brain. Typically, a neuron implements a logistic regression or a similar, nonlinear mapping with learned weights. The principle remains the same: A vector of input values is weighted with the neuron's weight vector and then squashed using an *activation function*. Neurons are usually organised in *layers* on a directed acyclic graph as displayed in figure 2.10.

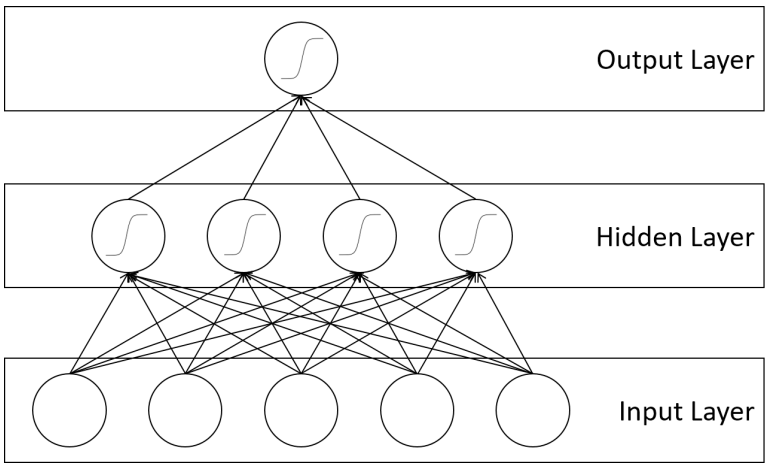


Figure 2.10: A simple feed-forward Neural Network layout with 5 input variables, one hidden layer with 4 hidden units, and a single output variable

Input and output are connected to *input layer* and *output layer*, respectively. A number of *hidden layers* are added in between and every layer is fully connected to its successor, i.e. every node in layer l supplies its *activation* as an input for every node in layer $l+1$.

Even though each neuron only performs a simple piece of logic, the combination of nonlinear mappings and multiple layers enables Neural Networks to describe complex functions. Neural Networks can be arbitrarily wide and deep, where wide refers

to the number of neurons per layer and deep to the number of hidden layers. The term *Deep Learning* [Goodfellow et al., 2016] was formed to describe especially deep Neural Network architectures. A common interpretation suggests that deeper networks reach a higher abstraction in that the hidden units learn more abstract features as a function of rather simple inputs.

Like a single logistic regression module, Neural Networks are usually trained with a variant of the *gradient descent* algorithm. Unlike in logistic regression, the overall loss function of a Neural Network is however not convex. There are generally many local optima in the weight space and therefore, convergence to the global optimum is not guaranteed - in fact, it is extremely rare. However, Neural Networks do often work well even with local minima.

There are several variations of the gradient descent algorithm, aiming to achieve a better, and faster, training. A common variant is the *Adam solver* [Kingma and Ba, 2014] which adapts automatically to the steepness of gradients and uses early weight updates (”*Stochastic Gradient Descent*”) to accelerate the training procedure.

During training, the overall prediction loss is measured at the output layer and consequently used to train the neurons of the output layer. The hidden layers are recursively trained on the prediction loss, redistributed by the weights between hidden layer and output layer, or between output layers for deep architectures. This algorithm, known as *Backpropagation*, is the de-facto standard for training Neural Networks.

2.2.5 Recurrent Neural Networks

There exist a number of variations and enhancements of the ”standard” architecture of Feed-forward Neural Networks, mostly originating from targeted attempts to improve the learning performance with respect to a specific task. An important variation for time-series based domains, such as natural language and music applications, are *Recurrent Neural Networks (RNN)*.

Recurrent Neural Networks have cyclic structures, i.e. contrary to the feed-forward architecture, the output of a neuron can, directly or indirectly, influence its own input. For the purpose of our project, we will only consider the direct case, where a neuron at time step t has access to its own activation at time step $t-1$.

Figure 2.11 shows a single, recurrent neuron: On the left, a loop connection indicates the recurrence. The right image displays an unfolded view of the neuron through time, showing how every neuron’s output only depends on its previous activation and

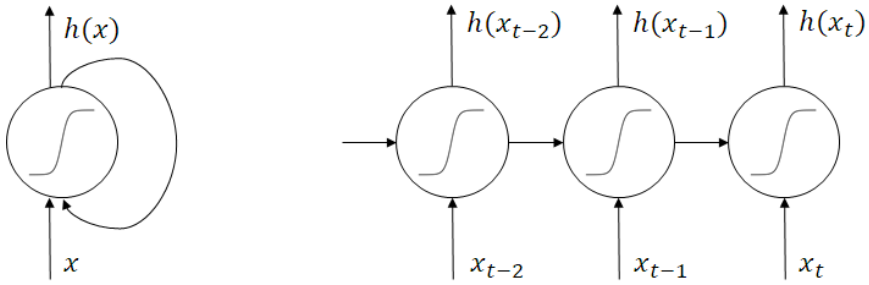


Figure 2.11: A single recurrent neuron. *Left*: static view, *right*: unfolded in time

the current output. To train a RNN, gradients are propagated back along this unfolded view. This procedure is often referred to as *backpropagation through time*.

2.2.6 Long Short-Term Memory

A popular flavour of RNN are *Long Short-Term Memory (LSTM)* networks. In LSTMs, a neuron has an incoming connection from its previous value before applying the activation function. This value is named *memory cell* of the LSTM neuron. The error is propagated through the memory cell time steps as a constant, rather than being redistributed at every time step. This way, problems with vanishing and exploding gradients are prevented.[3]

A single LSTM neuron is actually a module consisting of four "standard" neurons: A *forget gate F* and *input gate I* determine how strongly the new cell value $c(x_t)$ is influenced by the previous cell value $c(x_{t-1})$ and the current input x_t. The cell value is squashed by activation function a, corresponding to a standard feed-forward neuron. An *output gate O* determines how much of the activation is used as the overall output of the LSTM module. The gates are neurons themselves, with the inputs being x_t and $h(x_{t-1})$ and the outputs being the control signals to the LSTM module.

LSTMs can be trained much faster and with better outcomes than other Recurrent Neural Networks. Consequently, they have become very popular in time-series prediction and deliver state-of-the-art results. For more details about LSTMs and their training procedure, please refer to the original paper, [Hochreiter and Schmidhuber, 1997]. A comprehensive and popular introduction to LSTMs is also given by [Olah, 2015].

[3] *Vanishing gradients* describes a problem where repeated weighting with a small weight along a computation graph leads to very fast decay of the error gradient, i.e. the effective time context in the RNN would be very limited. Similarly, *exploding gradients* describe the opposite case, where large weights cause an exponential growth of the gradient.

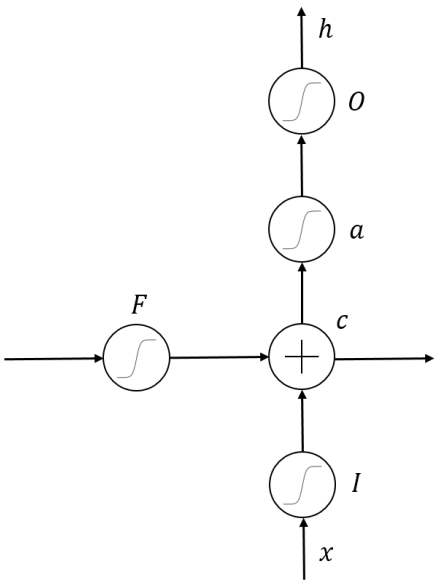


Figure 2.12: A single LSTM neuron, consisting of memory cell, activation function, and the three gates: input, output and forget

2.3 Related Research

In this section, we will give a comprehensive review of past work on the guitar fretting problem and the closely related guitar fingering problem, where not only strings and frets are predicted, but also detailed instructions on which finger to use for which note.[4] We systematised, to our best knowledge, a complete view of relevant past research papers in table 2.4. The table is structured by the scoring function, i.e. how the goodness of a fretting or fingering is evaluated, and by its optimisation methods, i.e. how the best sequence of frettings or fingerings is found. We will elaborate on a selection of these approaches below.

2.3.1 Explicit Modelling of Guitar Fretting Difficulty

The majority of past approaches to guitar fretting and fingering are based on explicitly modelling biomechanical difficulty of the hand positions and movements. This approach naturally solves both fretting and fingering problem at the same time. It is however dependent on the expert knowledge used to create the difficulty model and may thus have a strong bias towards the designer's musical background and playing

[4]Some of the past papers use *fretting* and *fingering* interchangeably which can lead to confusion as to which problem is addressed. We made sure to keep to the distinct terms in this report, i.e. *fretting* when string and fret are predicted and *fingering* when the hand and/or finger position are predicted

research paper	scoring	optimisation
[West, 1993, Kehling et al., 2014]	heuristic	(chunk-wise) enumeration
[Bygrave, 1996, Miura et al., 2004, Fiss and Kwasinski, 2011, Alcabasa and Marcos, 2012, Burlet and Fujinaga, 2013, Yazawa et al., 2013, Yazawa et al., 2014, Ramos et al., 2016]	heuristic	optimised tree search / dynamic programming
[Tuohy and Potter, 2005, Tuohy and Potter, 2006b, Rutherford, 2009, Ramos et al., 2016]	heuristic	genetic algorithm
[Sayegh and Tenorio, 1988, Sayegh, 1989, Radisavljevic and Driessen, 2004, Barbancho et al., 2012a, Hori et al., 2013]	learned	optimised tree search / dynamic programming
[Tuohy and Potter, 2006d, Tuohy and Potter, 2006c]	hybrid	genetic algorithm
[Tuohy and Potter, 2006a, Tuohy and Potter, 2006b]	Neural Network + genetic algorithm	

Table 2.4: A systematic overview of past research papers addressing the guitar fretting and guitar fingering problems

style, or be overly simplified. It is further to be acknowledged that many of the papers discussed here focus on the transcription of music from audio and only consider the fingering problem marginally and with comparatively simple methods.

[West, 1993] uses the sum of Manhattan distances between all fingers as an estimate of chord fretting difficulty. Equation 2.7 shows a formula for the resulting cost function. The cost of moving between two fingerings is similarly calculated as the sum of Manhattan distances travelled by each finger. The resulting fingerings were reported to be overall playable and better than randomly generated fingerings.

$$c(F) = \frac{1}{2} \sum_{x \in F} \sum_{y \in F} |x_1 - y_1| + |x_2 - y_2| \tag{2.7}$$

[Ramos et al., 2016] consider monophonic music only and base the difficulty on the Euclidean distance in fret-string-space. Equation 2.8 shows the corresponding distance measure, with x and y being two consecutive note frettings, defined as vectors of the format $(string, fret)$. [5] The focus of the paper is, however, on evaluating different optimisation approaches, rather than finding the optimal cost function. In the chosen scenario, several optimised genetic algorithms outperform the *A** (*A-star*, [Hart et al., 1968]) tree search algorithm, which may indicate a useful tendency for future research.

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \tag{2.8}$$

[Bygrave, 1996] uses a similar heuristic, minimising finger spread and travel distance. Additionally, some unwanted fretting situations are penalised, e.g. string crossings or the use of the little finger. The resulting heuristic is not explicitly stated, but some generated fingerings are shown and related to the scoring function, prompting future research to investigate more intricate heuristics.

Similarly, [Burlet and Fujinaga, 2013] define their cost function as a weighted sum of finger spread, travel distance and a general penalty for playing chords in high fret positions. Contrary to [West, 1993], the distance between two chords is calculated as the distance of string and fret means, rather than the sum of all individual distances. Burlet claims to use the A* search algorithm [Hart et al., 1968] for optimisation. While the central benefit of A* is that heuristics for the remaining path at any time allow for a quicker search, the heuristic is actually always set to 0, reducing the algorithm to a Viterbi tree search. The resulting tablatures are only evaluated with respect to their estimated cost. As the cost is also used to produce the tablatures, this is effectively an evaluation of the search algorithm, rather than the actual tablatures.

[Tuohy and Potter, 2005] uses a hand-crafted heuristic together with genetic optimisation to generate tablatures. The results are reported to be very close to published tablature and sometimes even easier to play, and consistently outperforming commercial software. The results are only judged informally. Unfortunately, neither the used heuristic, nor the achieved accuracy is explicitly given.

[5] In the original paper, the explanation of the variables is slightly erroneous and thus not very clear. We adjusted the formula accordingly.

[Yazawa et al., 2013] interweaves the tablature generation and pitch recognition from audio. The frettings are mainly generated to verify the plausibility of a set of pitches being played. Playability is checked with two simple constraints: A maximum of four fingers can be used and the finger spread is limited to four frets. Duplicate pitches are eliminated and barre chords[6] are modelled explicitly to estimate the number of fingers used correctly. The approach was refined later in order to find the best fingering. The evaluation of biomechanical difficulty is done similarly to other approaches we discussed, with the important difference of the wrist being modelled separately from the hand, thus eliminating the false implicit assumption that fingers move independently. Furthermore, the different timbre of the different strings is compared against the timbre in the input audio to get an additional hint of where on the guitar each note was played. The cost function is formulated as a weighted feature vector, i.e. as a linear model. These weights are manually set but suggested to be learned in future research. [Yazawa et al., 2014]

[Kehling et al., 2014] uses a set of constraints related to specific guitar techniques such as bendings and hammer-ons to describe tablatures from audio. They also limit the finger spread range to four. Smaller distances to a computed centroid on the fretboard are preferred, both within a chord and between consecutive chords and notes.

Based on the results shown in the heuristics-based papers mentioned, it is our impression that there have been quite promising results in general, but most approaches have rather strong limitations to their effective use.

We expect a new, rule-based and carefully tuned approach to be published by [Stavropoulou, 2017]. The project is running in parallel to our project and will be used to compare results in chapter 4.

2.3.2 Estimating Guitar Tablature and Fingering Quality from Data

The first approach to estimate the quality of guitar fingerings from data dates back to 1988. [Sayegh and Tenorio, 1988] / [Sayegh, 1989] describe a procedure of fitting a search tree to the probability of guitar fingerings, given the intended pitch. The probabilities are estimated from data by counting the relative occurrences in some training data. By doing so, Sayegh implicitly models the fingerings with a *Hidden Markov Model*.

A *Hidden Markov Model (HMM)* assumes a sequence of hidden states that generate

[6] Barre chords are chords where the index finger is used to push down several strings at once

observable outputs. Every output is only dependent on its generating state and every state is only dependent on its preceding state, as visualised in figure 2.13. In the case of guitar frettings, the hidden states would be the applied fingering, the outputs would be the resulting pitch. HMMs can be solved using the Viterbi algorithm which infers the most probable sequences of hidden states for sets of annotated input data. Once the HMM is fitted, it can be used to assess the likelihood of different fingering sequences for the observed pitches.

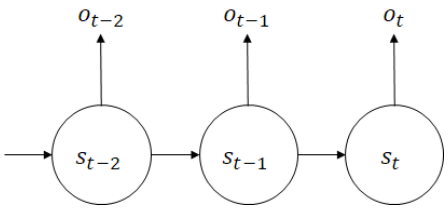


Figure 2.13: A Hidden Markov Model with a sequence of three states s_{t-2}, s_{t-1} and s_t, producing the output sequence $[o_{t-2}, o_{t-1}, o_t]$

While Sayegh only experimented with a small, monophonic toy version of the problem, a number of other authors have built on top of the suggested approach, e.g. by first segmenting the piece to transcribe into phrases, thus reducing the complexity of the search. [Radicioni et al., 2004]

[Radisavljevic and Driessen, 2004] extend the approach by enabling polyphonic music and modelling the cost function as a weighted combination of features, i.e. a linear regression model. The evaluation shows that in fact, the system learns useful information. However, due to lack of data, only 7 songs were used and the accuracy score was only reported on the training data. Radisavljevic points to multi-layer feed-forward neural networks as a possible direction for future research.

[Hori et al., 2013] formulate the fretting problem as an Input-Output HMM [Bengio and Frasconi, 1995], an extension to the standard HMM model which allows for better modelling of the intended pitches. Due to lack of applicable training data, however, the cost function is manually modelled by setting the HMM probabilities based on heuristics, rather than training data. After careful hand-tuning, the system was able to produce some tablatures which were judged easy to play.

Following up on their previous approaches purely based on heuristics, [Tuohy and Potter, 2006c] / [Tuohy and Potter, 2006d] implemented a hybrid cost function which is based on a large set of heuristics. The heuristics are weighted based on training data, resulting in a linear model similar to [Radisavljevic and Driessen, 2004],

except that it was trained with genetic algorithms, rather than the explicit solution usually applied in Machine Learning (as described in section 2.2).

The most advanced approach in terms of Machine Learning was presented by [Tuohy and Potter, 2006a]. In the paper, a feed-forward Neural Network is trained to predict the fret of a single note depending on previous and later notes. The combination of string and fret is inferred by selecting the valid fretting closest to the predicted fret. Chords are not treated separately from single notes. Instead, a chord is effectively treated like an ascending arpeggio, i.e. all its note frettings are individually and sequentially predicted from low to high.

A genetic algorithm is used to perform feature selection on a set of 64 features derived from human-written tablatures. The Neural Network itself is trained with gradient descent and can be directly applied to the input chords, using a genetic algorithm to find the best sequence of frettings. Afterwards, a single pass of post-processing, based on the previously shown heuristics, is applied to remove obvious errors. The highest training accuracy, comparing the generated tabs with the human-written ones, was reported at 94%.

2.3.3 Other Related Research

A small number of alternative approaches to tablature transcription have been published. While these approaches do not address the same problem as the project at hand, they do offer alternative angles which may be of interest to the reader.

[Humphrey and Bello, 2014] recognise chords from audio and match them to predefined chord shapes with a Convolutional Neural Network. The result is a chord transcription which can be used to support guitar accompaniment. It does however not detect melodies and guitar riffs, consequently not creating tablatures.

[McVicar et al., 2015] propose a composition system for guitar music. The composition is done with a Hidden Markov Model in tablature space and is consequently based on some of the algorithms discussed above. The outputs are style-specific, playable guitar tablatures for both rhythm and lead guitar.

There exist several approaches [O'Grady and Rickard, 2009, Barbancho et al., 2012b, Barbancho et al., 2012a, Yazawa et al., 2013] which aim to recognise the used frettings from audio recordings of a guitar. As the different strings have different strengths, materials and tensions, they produce different timbres. The approaches attempt to recognise the timbres and thus infer the string used to generate each pitch.

[Burns and Wanderley, 2006] and [Paleari et al., 2008] extend this approach by using visual cues collected from video recorded simultaneously with the audio recordings.

Chapter 3

Methods

In this chapter, we present two approaches to finding an optimal tablature for a given sheet music input. The first approach models and predicts frettings directly. The second approach assesses the quality of each possible fretting, rather than trying to directly predict the optimal fretting.

3.1 Direct Fretting Prediction

Our first approach aims to predict frettings directly. In the following sections, We will discuss different aspects of the guitar fretting problem and how they are addressed in the direct prediction approach.

3.1.1 Time Line Setting

A tablature can be understood as a time series of guitar frettings used to realise a sequence of chords and notes. As discussed in section 2.2.6, LSTM networks are commonly used to model sequential behaviour and predict new elements in the time series. This is usually done solely based on previous outputs, not on any additional input. Consequently, if we run a typical LSTM on a fretting sequence, it may predict frettings which are easily accessible but do not result in the intended chord.

We therefore propose adding the notion of *intention* into LSTMs, similarly to the inputs in Input-Output-HMMs [Bengio and Frasconi, 1995, Hori et al., 2013]. Effectively, every input vector x_t is extended by one or more *intention features*, capturing the intention, in our approach the intended chord, at time step $t+1$. Importantly, these features do not exist in the output vector, i.e. we do not predict the chord at time $t+2$.

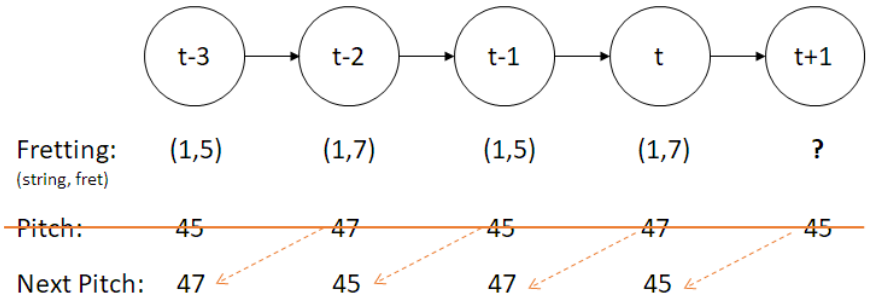


Figure 3.1: Pitch and fretting in a timeline. Shifting pitch features one step into the past allows the fretting at time step $t+1$ to be dependent on the intended pitch.

Figure 3.1 depicts our solution for a sequence of single notes to be played: In the typical setting of time series prediction, the pitch at time step $t+1$ would not be available for the prediction of the fretting y_{t+1}, as predictions are only based on previous time steps. To model the dependency correctly, we shift the pitch features into the past by one time step, i.e. the input vector x_t at time step t contains the pitch features for time step $t+1$. Consequently, the fretting output at time step $t+1$ will depend both on the previous frettings and the intended pitch.

3.1.2 The Dimensionality of Chords

So far, we have only considered monophonic music, i.e. mapping a pitch to a *note fretting*, where a note fretting is defined as a pair of string and fret:

$$pitch \rightarrow (string, fret) \tag{3.1}$$

A *chord* can however consist of up to six pitches on a six-string guitar. We have investigated different possibilities of modelling chords and chord frettings accordingly. Figure 3.2 shows the canonical fretting of a full C Major chord, which we will use to illustrate some of our feature sets.

1. *Vector Representation*. We can re-interpret the variables in scalar equation 3.1 so that $pitch$, $string$ and $fret$ represent vectors of length $n_{strings}$, where $n_{strings}$ is the maximum number of strings (i.e. 6 for standard guitars). The binary vector of strings indicates which strings were struck, the fret vector captures the frets held on each string. While this representation is meaningful and robust for frettings, the pitch feature vector turns out to be unsuitable as it may change drastically for small changes in the meaning.

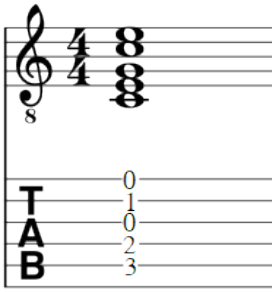


Figure 3.2: C Major chord in its most typical shape on a six-string guitar in standard tuning, in sheet music and tablature notation

If we add an E_3 (pitch 40, open low E string) to the C major chord depicted in figure 3.3, the resulting pitch vector will be $[40, 48, 52, 55, 60, 64]$, i.e. the value in every cell of the vector changes. This mismatch in logical and representative difference can possibly result in very poor training behaviour of a Machine Learning model. We will therefore use the vectorised representation only for strings and frets, but not for pitches.

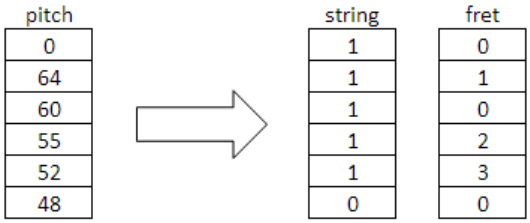


Figure 3.3: C Major chord in vector representation

2. *Sparse Encoding*. Instead of encoding the pitches as a simple vector of length $n_{strings}$, we can use a sparse vector counting the occurrences of all possible pitches in a predefined range. Similarly, we can encode the frets in a binary $n_{strings} \times (n_{frets} + 1)$ matrix, indicating for every pair $(fret, string)$ if it was played or not.[1]. The sparse encoding is visualised in figure 3.4.

   The sparse encoding for pitches has the desired properties, i.e. the amount of change in meaning is reflected by the amount of change in the representation. However, its size has to be fixed to the anticipated range of possible pitches. With standard six-string tuning, pitches between 40 and 88 can be realised.

   While the simple, vectorised representation of frets has an explicit ordering of frets, and thus implicit distances between frets, this notion of proximity is not

[1] including fret 0 to indicate the open string, thus $n_{frets} + 1$

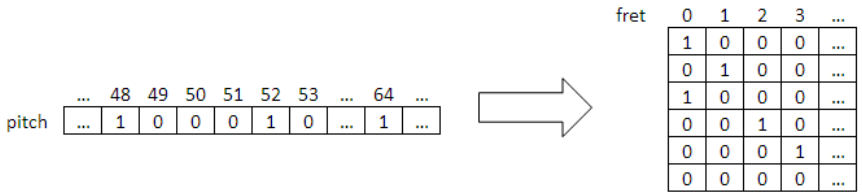


Figure 3.4: C Major chord in sparse representation

captured in the sparse representation. At first, this might seem like a disadvantage, as close frets are more likely to occur in a good chord fretting. Decoupling the explicit encoding could however have a beneficial effect on the learning procedure and potentially allow for better handling of open strings, i.e. distances between fret 0 and other frets are effectively modelled independently of other inter-fret distances.

The sparse encoding needs a large amount of numeric values and is specific to the number of strings and frets on an instrument, as well as its total pitch range.

3. *Statistical Descriptors*: As a simple alternative to the detailed sparse encodings, we introduce a vector of statistical measures which describe the distribution of strings and frets. These measures include the mean string and fret in a chord as used by [Burlet and Fujinaga, 2013], capturing where on the fretboard a chord may be played. Along with the means, we report the standard deviation of string and fret distribution, effectively capturing the spread over frets and strings, closely relating to the centroid distances as used by [Kehling et al., 2014]. Additionally, we include the correlation coefficient to capture the hand orientation on the fretboard. Figure 3.5 illustrates how chords can have very different shapes on the fretboard, relating to the guitarist's hand position.

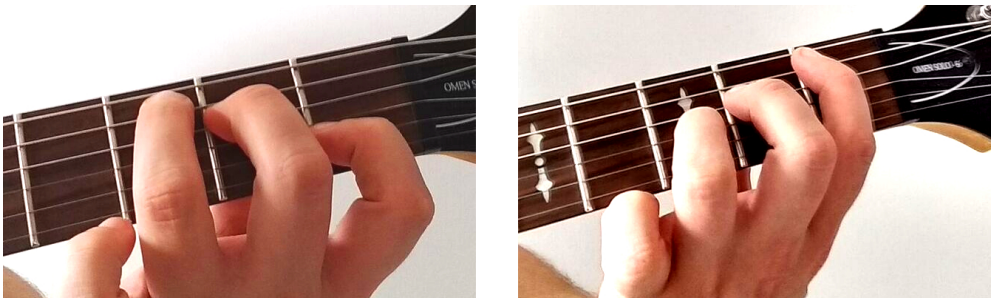


Figure 3.5: Two chords with different hand orientations. Left: *C*, with the fingers closer to the thumb grasping higher strings. Right: *B♭dim*, with the the fingers closer to the thumb grasping lower strings.

The descriptor method has a direct graphical interpretation on the guitar fretboard, as depicted in figure 3.6. Two possible fretting combinations are shown in blue and green. The orange area visualises the descriptors' expressivity: a) shows how the mean captures the position on the fretboard. b) shows how the variance additionally captures the size, or *spread*, of the fretting. c) visualises how the correlation coefficient captures more information about the shape of the fretting.

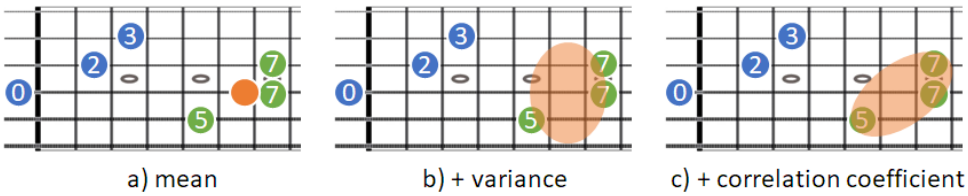


Figure 3.6: Two possibilities of playing a D5 chord on a guitar fretboard, and a coarse fretting prediction (red)

While potentially, more measures can give a more accurate description, we expect a small number of descriptors to carry sufficient information. Due to the nature of the coarse approximation, statistical descriptors can however only be used to identify the best from a list of frettings, not to directly predict a fretting. Section 3.1.3 will cover our validation approach for fretting predictions.

4. *Sequential Interpretation*. Instead of explicitly modelling chords, we can treat a chord as equal to the ascending arpeggio, i.e. sequence, of its pitches, as done by [Tuohy and Potter, 2006a] and visualised in figure 3.7. While the simplicity of this approach is appealing, it may fail to capture important patterns in chords which we assume not to be equivalent to sequential patterns.

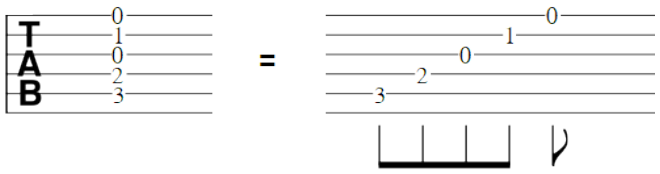


Figure 3.7: The sequential interpretation visualised with a C Major chord

All the mentioned features can be either used directly or in *delta mode*. In delta mode, not the features themselves are used for training and prediction, but their first derivation. This captures the movement, rather than static positions. While less information is captured, the delta features may generalise better: For example, moving up

by a pitch distance of seven could in many cases be translated to moving up one string and two frets, regardless of where the first event was situated.

3.1.3 Validation of Frettings

When directly predicting frettings, there is no guarantee that the predicted fretting will actually realise the intended chord, especially if there is only a coarse approximation in the form of a vector of statistical descriptors. To make the prediction robust, we therefore compare it to all possible frettings associated with the chord and select the possible fretting closest to the prediction. This way, we guarantee that the generated tab always realises the intended chords - even if the prediction is far from a reasonable fretting.

As an additional validation step, we only consider frettings with a maximum fret range of four, as suggested in [Yazawa et al., 2013]. This pre-filtering makes the system more robust against obvious mistakes and improves the computing performance, as fewer possible frettings have to be generated and evaluated.

3.1.4 Different Instruments and Tunings

Most previous approaches to the guitar fretting and fingering problems only consider a six-string guitar in standard tuning. As our approach learns from data, it will be able to learn different tunings if sufficient training data is provided.

Furthermore, some tunings can be understood as transpositions of others. A popular example is the $E\flat$ tuning, where all strings are tuned one semitone lower than in standard tuning. We can easily enable transfer learning between these tunings by transposing the input chords in opposite direction and then assuming standard tuning. Similarly, we can model a capo as a higher tuning combined with a decreased number of available frets.

3.1.5 Training the Neural Network

The training data was acquired from the 100 top-rated Guitar Pro tablatures on *Ultimate Guitar*[2], a popular community platform for guitar tablatures. While Guitar Pro is a proprietary file format, it is widely used and keeps tablatures in a structured, machine-readable format. The structured format has two major advantages over text tablatures

[2] `https://www.ultimate-guitar.com/top/?rating&filter=guitar_pro`

for our purpose:

1. Text tablatures can be difficult to parse, as there are different conventions on how tunings are represented or how playing techniques are notated. The structured format of Guitar Pro can be converted to XML formats which are straightforward to parse.

2. The quality of text tablatures varies wildly and may not even be properly represented by user ratings, as investigated by [Macrae and Dixon, 2011]. The use of the structured Guitar Pro format requires a specialised software and thus requires a higher level of commitment, leading to a smaller number, but presumably higher quality of structured tablatures compared to plain text tablatures. Additionally, most notation software comes with a playback function and other amenities, thus reducing notation mistakes.

We converted the Guitar Pro tablatures to XML by using the command line interface of *MuseScore*, a free and flexible notation software supporting guitar tablatures.[3] From the XML files, we extracted the different feature sets described in section 3.1.2. Some of the tablatures were corrupted[4] and thus excluded in the data cleansing process. The remaining 82 tablatures made up for a total of 300,000 chord and note events.

We trained different versions of our fretting prediction model. 74 tablatures were chosen for the training procedure. We used the *Adam* solver and a *mean squared error (MSE)* loss function with the L2 regularisation term $\lambda|W^T W|$ which penalises large weights and thus prevents overfitting. λ was set to 0.1. Each network was trained in 10 epochs, i.e. 10 full iterations over the training data, with a batch size of 128. One tablature was used as a preliminary validation set for the prediction loss. The fully trained LSTM was then evaluated on the remaining 7 tablatures. These results will be presented in chapter 4.

$$mse_{L2}(\hat{\vec{f}}, \vec{f}^{(i)}|W) = \frac{1}{K}\sum_{k=1}^{K}\left(\hat{f}_k - f_k^{(i)}\right)^2 + \lambda|W^T W| \tag{3.2}$$

[3] All sheet music and tablature snippets in this document were created using MuseScore.

[4] In most cases, an instrument was declared as 24-fret instrument, but higher frets were played in the actual score.

3.2 Fretting Prediction by Cost Estimation

Our second approach is based on the estimation of the difficulty of possible frettings, rather than the direct prediction of an optimal fretting. Similarly to the vast majority of past research, as summarised in section 2.3 and table 2.4, we estimate the fretting difficulty in form of a cost function and use an optimisation algorithm to find the fretting sequence with the lowest total cost. The cost function is however learned, rather than explicitly designed.

3.2.1 Cost Function

Our cost function is based on the conditional probabilities of frettings, given their context. We calculate the conditional probability estimate $P(f_t|f_{t-1}, p_t)$ of each fretting f_t to occur after the fretting f_{t-1} and the pitch intention p_t as:

$$P(f_t|f_{t-1}, p_t) = \frac{count(f_{t-1}, p_t, f_t)}{count(f_{t-1}, p_t)} \tag{3.3}$$

Above equation implies independence of f_t from anything before time step $t-1$, i.e. f_{t-2} has no influence on the probability of f_t. We do however assume that f_{t-2} can carry useful information. For example, $f_{t=2}$ could be useful to calculate the first derivative of fret positions, i.e. the hand or finger movement speed in fret direction. Similarly, alternating patterns can be easily described by $f_t = f_{t-2}$.

We therefore decided to calculate conditional probability estimates for different context lengths. We get the following, generalised formula for conditional probabilities with context length T:

$$P(f_t|f_{t-T}, ..., f_{t-2}, f_{t-1}, p_t) = \frac{count(f_{t-T}, ..., f_{t-2}, f_{t-1}, p_t, f_t)}{count(f_{t-T}, ..., f_{t-2}, f_{t-1}, p_t)} \tag{3.4}$$

We use the negative log probabilities as costs, as shown in equation 3.5. The logarithmic function allows for computation of very small probabilities which could otherwise be lost due to arithmetic underflow. Negating the log probabilities yields a number which is smaller for better frettings and thus constitutes a viable cost function to be minimised. We use $P_T(f)$ as a shorthand for the conditional probability of f given its context of length T.

$$c(f) = -\log P_T(f) \tag{3.5}$$

We define the cost $c(s)$ of a fretting sequence s as the sum of individual fretting costs $c(f)$.:

$$c(s) = \sum_{f \in s} c(f) \tag{3.6}$$

For context length $T = 1$, this effectively returns a log likelihood estimate for s:

$$c(s) = \sum_{f \in s} c(f) = \sum_{f \in s} -\log P_T(f) = -\log \prod_{f \in s} P_T(f) \tag{3.7}$$

With a context length $T > 1$, the total cost does not represent the likelihood of the fretting sequence. We do however expect it to yield similar results and thus lead to a meaningful assessment of candidate frettings.

3.2.2 Cost Prediction for Unseen Data

Due to the complexity of the fretting space, the calculated probabilities and costs only cover a small fraction of the possible frettings and fretting sequences. To generalise to unseen data, we train a Feed-forward Neural Network to predict the cost of a fretting, i.e. learn a mapping from a fretting to its cost:

$$f \rightarrow c(f) \tag{3.8}$$

The Neural Network uses the same features as the direct prediction network in section 3.1. To assess the cost of a possible fretting at time step t, the input includes features of that fretting and the intended chord, rather than only including previous frettings.[5] Similar to the direct prediction LSTM, the cost predictor can take inputs of variable context length T.

3.2.3 Optimisation

We can formulate the search for an optimal tablature as the optimisation problem stated in equation 3.9, with S being the set of all possible fretting sequences.

$$\min_{s \in S} c(s) \tag{3.9}$$

[5]The look-ahead mechanism is not needed here, as we evaluate a known, possible fretting, rather than predicting it. Consequently, the pitch features are time-aligned with the frettings, not shifted.

Many commonly used search algorithms, like the Viterbi algorithm, A*, and D*, are based on the assumption that the cost of a node only depends on its predecessor node. With this assumption, the cost optimisation problem can be reduced to finding the optimal path in a graph. [Sayegh, 1989]

Due to the variable context length chosen in section 3.2.1 and the resulting dependencies between distant frettings, these assumptions do not apply. Consequently, we decided to implement the optimisation as a simple tree search.

However, the search space for frettings and fretting sequences is prohibitively large to perform a full tree search. Consequently, we prune the tree while traversing it. The pruning is realised on two different levels:

1. *Candidate pruning*. The cost function $c(f)$ is evaluated for every possible candidate fretting f in context. Each candidate fretting is kept for further evaluation only if its cost is both within γ standard deviations of the best candidate fretting's cost and it is among the ρ lowest cost estimates:

   $C = [c(f)\ for\ f \in fretting\_candidates]$;
   forall $f \in fretting\_candidates$ **do**
   | **if** $c(f) > \min(C) + \gamma\, std(C)$ **then**
   | | $C.remove(f)$;
   | **else if** $rank(c(f), C) > \rho$ **then**
   | | $C.remove(f)$;
   end

   Algorithm 1: Candidate pruning

2. *Sequence pruning*. Similarly to candidate pruning, the cost function $c(s)$ is evaluated for every considered sequence s. The sequence is kept for further evaluation only if it is both within γ standard deviations of the best-rated sequence and it is among the top ρ lowest cost estimates.

The parameters γ_c, γ_s, ρ_c and ρ_s can be tweaked to optimise the search run time while still receiving good search results.

On top of the pruning, we add a simple rule to the optimisation algorithm: If a fretting f_t is equal to its predecessor f_{t-1}, its cost is set to zero. While this does not enforce taking the same fretting for the repetition of a note or chord, it does express a strong preference and therefore makes the system more robust. This *zero join cost* technique is commonly applied in HMM-based waveform concatenation speech synthesis. There,

it gives a preference to concatenate speech segments which were recorded together, thus keeping longer recording fragments intact. [Hunt and Black, 1996]

When the graph is fully traversed, the fretting sequence with the lowest total cost is selected. Due to the pruning, there is no guarantee that the sequence is globally optimal, but we still expect even strongly pruned trees to work reasonably well.

3.3 System design

Figure 3.8 shows the data flow in *tabgen*, our tablature generation system: A set of tablatures in the MuseScore XML format is used to extract the training features which are then used to train a Neural Network. To generate a new tablature for a piece, a Muse Score file is fed into the system. Chord information is extracted and fed into the optimisation algorithm. The algorithm performs its pruned tree search, evaluating the possible frettings at each stage with the trained Neural Network. In the direct prediction case, the tree is pruned to a width of 1, always selecting the fretting corresponding closest to the prediction.

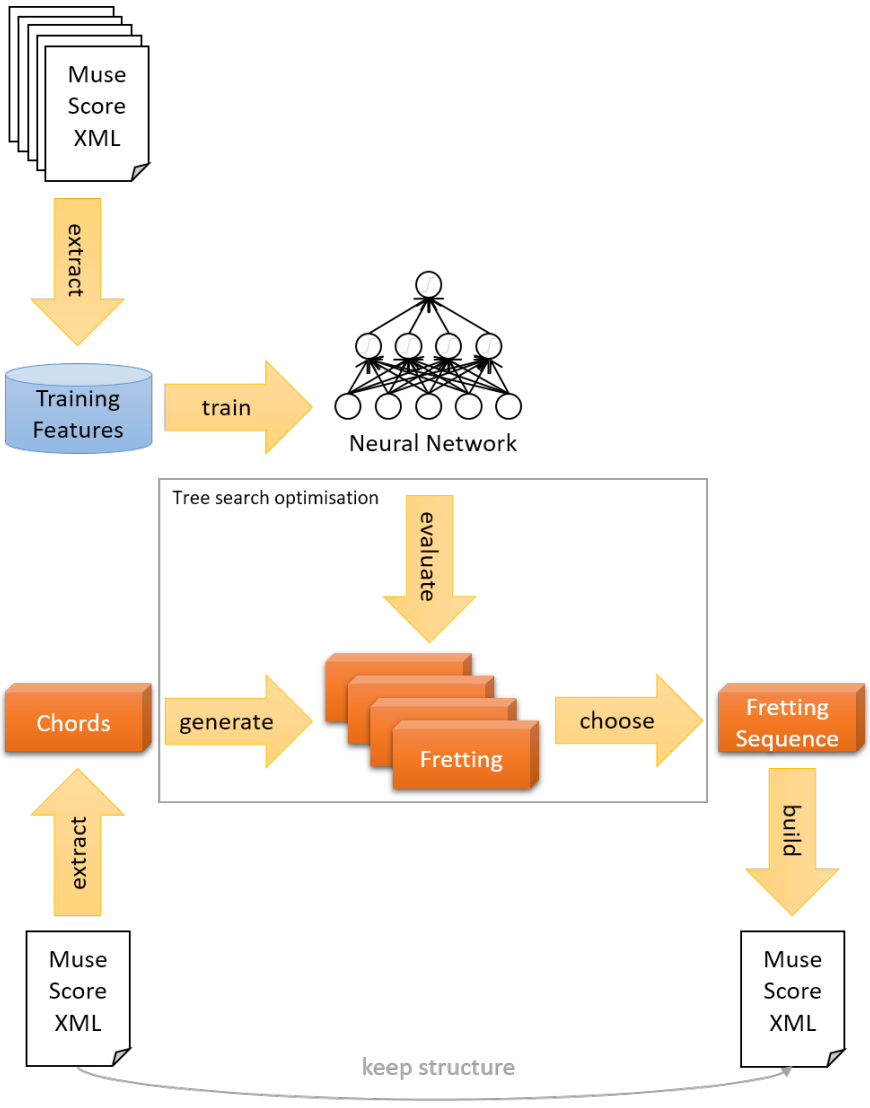


Figure 3.8: *tabgen* system design

The best fretting sequence is saved into a copy of the input XML file, preserv-

ing the general structure of the piece and any additional annotations, such as playing techniques, which are not in the focus of the project.

We implemented *tabgen* in Python 3.5, using a combination of the popular Machine Learning libraries `tensorflow` and `keras` for training the Neural Networks. The `xml` library was used as basis of our parser, `pandas` and `numpy` for data handling and performance optimisation. We designed the system to be object-oriented and modular. It is therefore rather simple to configure the system, and to implement enhancements, for example alternative cost functions.

The documentation of *tabgen* gives a deeper insight into the architecture and design of the system, without discussing the code in detail. The documentation is attached as appendix A. Technical details about classes and methods can be found in the form of inline-comments in the code available from GitHub.[6]

[6] `https://github.com/gitarreneli/tabgen/`

Chapter 4

Evaluation

In this chapter, we evaluate the proposed system. Section 4.1 is concerned with evaluation in an objective, key figure based manner. Section 4.2 evaluates the system output subjectively, discussing strengths and shortcomings of the approach.

4.1 Objective Evaluation

This section is concerned with an objective, key figure based evaluation of our two Machine Learning approaches to the guitar fretting problem. We define an accuracy measure, introduce a number of variations of our model and then report the accuracies for the models and variations.

4.1.1 Prediction Accuracy

We define the prediction accuracy on a sequence of frettings as the agreement between a published fretting sequence s and the predicted fretting sequence $\hat{s}$, as formulated in equation 4.1. $\sqcap$ is a binary function returning 1 if the input is $\vec{0}$ and 1 otherwise. $\hat{s}_t = \hat{\vec{f}}$ is the predicted chord fretting at time step t.

$$accuracy(\hat{s}, s) = \frac{1}{t} \sum_{t} \sqcap(\hat{s}_t - s_t) \tag{4.1}$$

The prediction accuracy relates the generated tablature to previously published tablature. Importantly, the published tablature may not be optimal, or they may be more than one optimal tablature. The reported accuracy can therefore only be an indicator of the system performance, not an absolute measure of output quality.

4.1.2 Evaluated Models

We evaluated a number of different variations of our models, along with some simple baseline heuristics to compare the performance against.

4.1.2.1 Baseline Models

1. *Random Tablature*. As a first baseline, we create random tablatures: For every note or chord, we randomly sample a fretting from the list of possible frettings with a maximum finger spread of four. The performance of this model should be seen as absolute lower boundary for any other model.

2. *Distance Heuristic - Sequential Mode*. This heuristic follows the sequential interpretation of chords, as described in section 3.1.2. It simply measures the distance between individual note frettings, as used in other approaches before.

3. *Distance Heuristic - Chord Mode*. A heuristic based on the notion of fretboard distance, both within a chord fretting and between chord frettings. Equation 4.2 shows the heuristic as a weighted sum of the sum of distances within a fretting, d_{steady}, the mean distance to the previous fretting, $d(\vec{f}^{(t)}, \vec{f}^{(t-1)})$, the number of skipped strings within the chord, and the fret mean. The weights were set manually, based on the subjective quality of tablatures generated from a small subset of the training data.

$$c(\vec{f}^{(t)}) = 0.2 d_{steady}(\vec{f}^{(t)}) + 0.7 d(\vec{f}^{(t)}, \vec{f}^{(t-1)}) + 3.0 f^{(t)}_{strings\_skipped} + 0.01 f^{(t)}_{fret\_mean} \tag{4.2}$$

4. *Probability Lookup*. As a baseline for our cost prediction model, we implemented a lookup mechanism, using the pre-calculated probabilities from the training data. If the sequence is not present in the training data, a low probability value ($\lambda = 0.001$) will be used, as described in algorithm 2.

if $(f_{t-1}, f_t) \in training\_data$ **then**
  $P(f_t|f_{t-1}) = P_{train}(f_t|f_{t-1})$;
else
  $P(f_t|f_{t-1}) = \lambda$;
$c(f_t) = -logP(f_t|f_{t-1})$;

Algorithm 2: Probability Lookup Baseline

4.1.2.2 Machine Learning Models

We trained different variants of the prediction models, using the different feature sets introduced in section 3.1.2. The following list summarises the features used for the direct prediction model. The cost model was trained on the same features, with an adjusted temporal structure for the input as discussed in section 3.2.2.

1. *Descriptors*. Using descriptors for the intended pitch, we predict a set of descriptors for strings and frets. This yields a rather coarse model with comparatively few features.

2. *Vectorised*. This approach predicts the vectorised features for strings and frets. As previously explained, the vectorised representation of pitches is not considered suitable for a Machine Learning approach. The sparse pitch representation is used instead.

3. *Full Sparse*. This approach uses the detailed, sparse representation for both the pitches and the frettings.

4. *Sequential*. Following the sequential model, a string and fretting are directly predicted from the intended pitch.

5. *Sequential Sparse*. We interpret the chords as sequential, but still use sparse representations of features.

The models were also trained and tested in delta mode and with variable context length. We generally used two hidden layers, with the first hidden layer containing 1024 hidden units and 512 hidden units in the second layer. In case of the direct prediction model, the first layer consisted of 1024 LSTM cells. All other units were modelled as standard, feed-forward neurons with a *tanh* activation function.

4.1.3 Results

In the following, we show our results with the different models and baselines defined above. For visualisation of the results, we use violin plots. Violin plot shows the median and interquartile range, similar to boxplots. Additionally, they give a kernel density estimate, visualising how the data is distributed.[1]

[1] `http://seaborn.pydata.org/generated/seaborn.violinplot.html`

4.1.3.1 Baseline

Figure 4.1 shows the results achieved with the baseline heuristics. The hand-crafted distance metric performs very well, with a median accuracy of 68.9% and a rather narrow distribution. In a sequential scenario, the distance heuristic performs worse, presumably because too much contextual information about the chord fretting shape and the individual note fretting distances is lost. The probability lookup yields an even lower accuracy which can be linked to the sparsity of training data: For over 66% of the evaluated frettings, no probability estimate was available from the training data. As expected, the random baseline generates tablatures which agree with the published tablature significantly less than the other baselines.

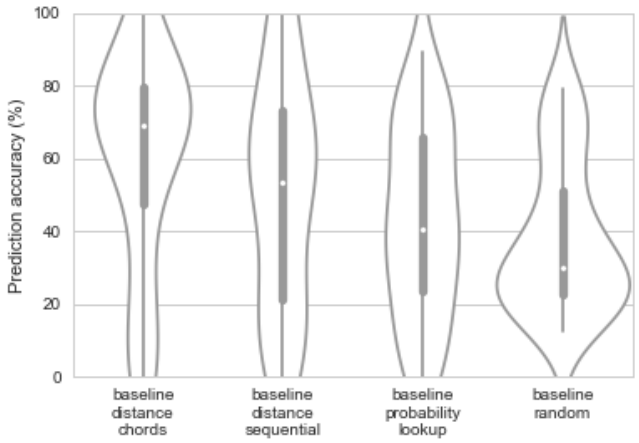


Figure 4.1: Accuracy of the baseline heuristics

4.1.3.2 Direct prediction

Figure 4.2 shows the result using different versions of the direct prediction model, trained with a context length of 1. Out of the direct prediction approaches, the sequential model performed best at 72.9%, closely followed by the descriptor model at 69.1%. The accuracy distribution of both approaches is very similar to the distance baseline. This may suggest that the models learn a similar distance function which seems likely, given the integer representation of string and fret position and shape.

The sparse and vectorised models perform at lower median accuracies of 59.0% and 50.8%, respectively. The sparse encoding does not capture the intrinsic distance between frets, which may explain its lower accuracy. For the vectorised model, we represented the pitches in a sparse representation. Consequently, input and output format are quite different, which may have a negative impact on the learning procedure.

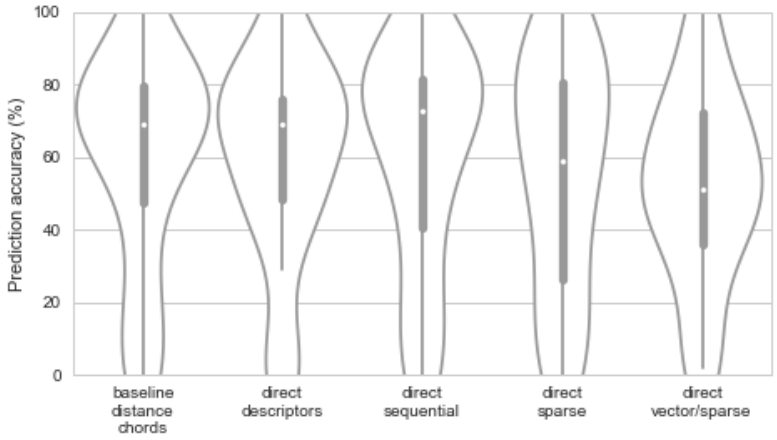


Figure 4.2: Accuracy of the direct prediction model with context length 1

Experiments with the delta mode consistently yielded lower accuracy by more than 10%. Consequently, we will not consider the delta scenario further for the direct model.

Figure 4.3 shows how changing the context length to 2 resulted in a lower median accuracy, but with a narrower distribution. Doubling the number of training epochs for context length 2 brought the accuracy up to a similar level as with context length 1. After more than 20 epochs, the accuracy started going down, presumably due to overfitting. Similarly, the accuracy for context length 1 decreased after more than 10 training epochs.

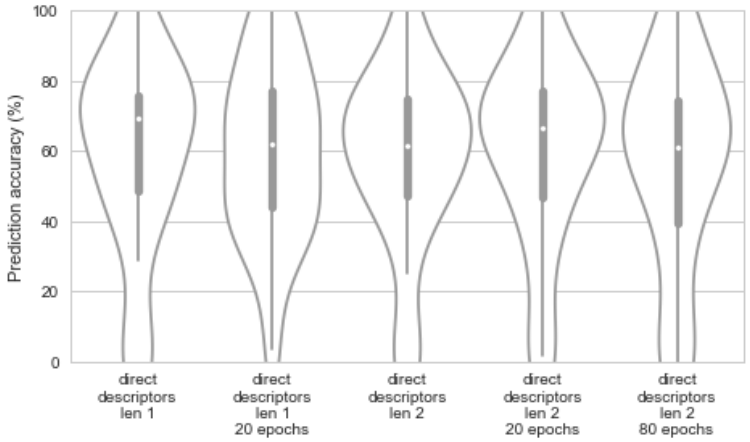


Figure 4.3: Accuracy of the direct prediction model with varying context length and number of epochs

We conclude that, in the direct setting and with the given, small amount of training data, the simpler models, descriptor model and sequential model, with a context length of 1 perform best. The lower performance of longer context lengths and the sparse

model representations can be linked to sparsity in the training data and may outperform the other approaches given enough training data.

[Tuohy and Potter, 2006a] reported an agreement with published tablature of up to 94%, using a combination of Neural Network and a genetic algorithm. While the accuracy is considerably higher than ours, it is to be acknowledged that Tuohy used a set of tablatures retrieved from `classtab.org`, a website for classical music tablatures. Our training data consists of the 100 most popular guitar tablatures published on `ultimate-guitar.com`, thus spanning a much wider range of musical genres, starting from pop and singer-songwriter music to rock and heavy metal. As the different musical styles typically come with different playing styles and fretboard patterns, we assume there is a limit to cross-genre learning of tablatures.

A melody line may, for example, be played on low frets and open strings in a classical setting, where the guitarist can let the strings ring and create a fuller sound. A rock guitarist, on the other hand, may prefer to play melodies on higher frets, where the notes will cut through the mix. In our experience, higher frets are more comfortable to play on an electric guitar due to the lower distance between strings and fretboard, as well as typically more available frets overall. We will discuss this further, with respect to our system's outputs, in section 4.2.

4.1.3.3 Cost prediction

Experiments with the cost prediction model showed considerably lower results than with the direct model. Figure 4.4 shows the results for the cost prediction model, trained with a context length of 1. All the results were obtained with aggressive pruning settings of $\rho_c = 2$ and $\rho_s = 3$.

While the cost model accuracy is considerably lower than the direct prediction accuracy, we can observe that it is in most cases higher than the probability lookup baseline. This implies that our Neural Network is able to generalise probabilities from the seen examples. However, most probabilities in the training data were rather high, while we would expect low probabilities for most theoretically possible fretting sequences. Naturally, frettings with low probabilities are less likely to occur in the training data, making it difficult to make predictions about low probabilities.

The lack of low-probability examples in the training data leads to probabilities of unseen combinations being estimated as too high, and consequently, a poor overall performance. We expect that the performance could strongly benefit from a larger and more diverse set of training data. Additionally, adding small amounts of intentionally

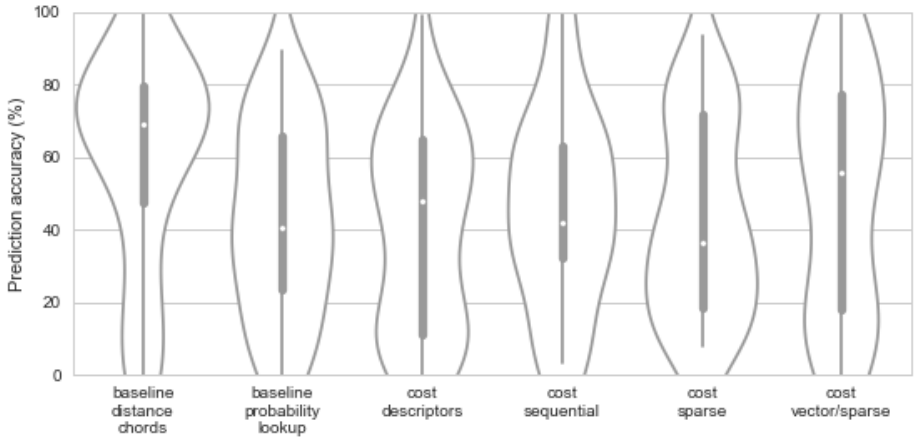


Figure 4.4: Accuracy of the cost prediction model with context length 1

bad tablature into the training data could help to estimate the difference between good and bad frettings.

4.2 Subjective Evaluation

In this section, we give some examples of our system's output and compare it to published tablature, discussing some key differences. Our first example will also be compared to a new, rule-based approach about to be published.

Figure 4.5 shows two short passages of Alec Templeton's *Bach Goes To Town*, as transcribed by a human expert. The transcription is easy to play, as it makes appropriate use of the relation between frets and strings. The last note of each phrase is played on an open string, producing a long-ringing sound.

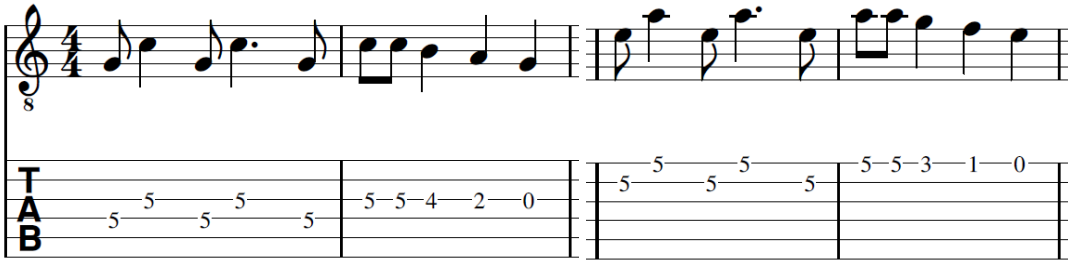


Figure 4.5: Two passages from Alec Templeton's *Bach Goes to town*, as transcribed by a human expert

[Stavropoulou, 2017] proposes an approach using more than 15 manually weighted rules to determine the ideal fretting positions. Most importantly, low fret positions and open strings are preferred, hand position changes avoided. The approach generates both fretboard positions and fingering instructions. Rules and weightings are mainly influenced by guitarists' fingering decisions for classical guitar pieces.

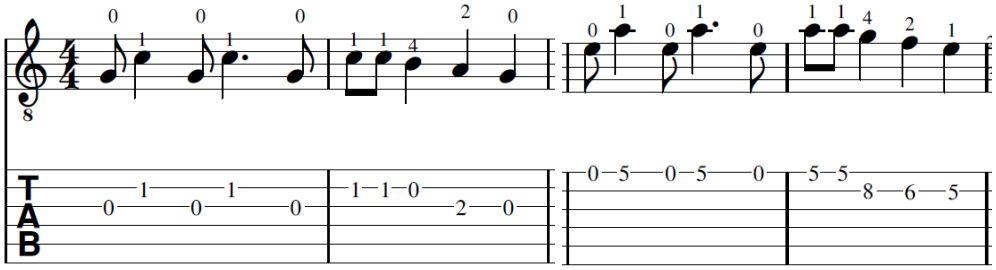


Figure 4.6: Two passages from Alec Templeton's *Bach Goes to town*, as transcribed by the rule-based system proposed in [Stavropoulou, 2017]

Figure 4.6 shows the tablature as generated by this rule-based approach, including the associated fingerings. The strong preference of open strings works out very well in the first passage, making the melody very easy to play and, due to the ringing open

strings, produces a full sound on an acoustic guitar. The second passage is easy to play as well, but will lead to a quite different sound than the first passage, as no suitable open string is available.

Figure 4.7 shows our system's output for the same excerpts. The most noticeable difference to the other approaches is that it is played in noticeably higher frets and with no open strings. This may be preferred on an electric guitar, in order to have the guitar stand out more against the accompaniment and to keep the sound crisp and clear. As a considerable part of our training data consists of contemporary, band-centric guitar music, this behaviour is easily justified and may be desired for similar settings.

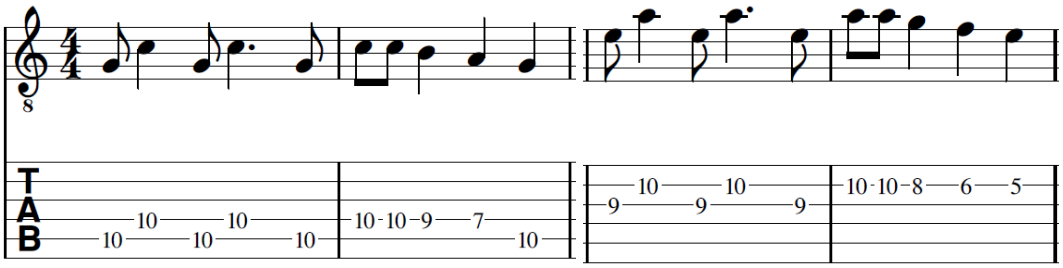


Figure 4.7: Two passages from Alec Templeton's *Bach Goes to town*, as transcribed by our fretting prediction approach

In our opinion, none of the three shown tablature excerpts can be seen as the *correct* or *best* way to play the given melody. The choice of a good tablature certainly is based on both the playing difficulty and the achieved sound. The difficulty itself is partially dependent on the choice of instrument. The intended sound can be dependent on several factors, including the instrument, style, accompaniment and personal preference.

Consequently, for generating a tablature which can be genuinely deemed optimal, the style intention and instrumentation choice of the guitarist should be considered.

Another example for the style dependency can be illustrated by the generated tablature for Isaac Albéniz' *Asturias*, as displayed in figure 4.8. In the original playing instructions (left), the open string is used alternating with the lower melody line. In the generated tablature (right), the notes are played in a compact region on the fretboard. While this does not capture the original sound intention of the piece, it may describe a viable way of re-interpreting the piece in a contemporary context.

The system does however still make some obvious mistakes. Figure 4.9 shows a short piece of generated tablature for the rhythm guitar in Chuck Berry's *Johnny B. Goode*. We would generally expect the chord 5^{th} and 6^{th}, which are typical in Blues

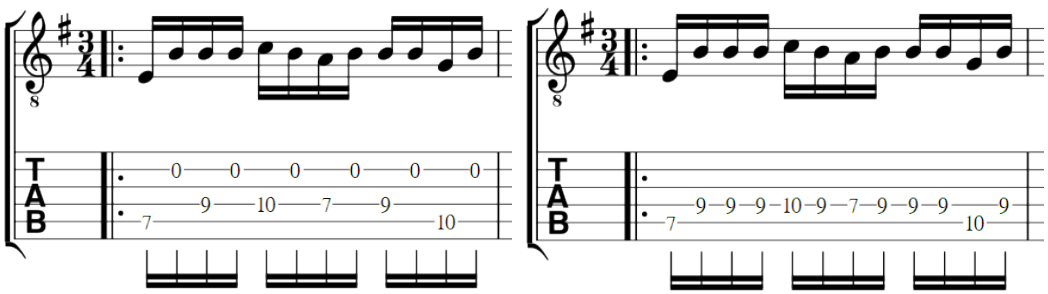


Figure 4.8: Beginning of Isaac Albéniz' *Asturias*. Left: original notation. Right: generated tablature

and Rock'n'Roll music, to be played on the same string. While this works fine in the first measure, the 6^{th} in the second measure is played on the B-string. The decision to play there can potentially be explained by the smaller distance between G and B string. It does however result in an untypical ringing of the sixth on the B string and is comparatively uncomfortable to play when using a guitar plectrum.

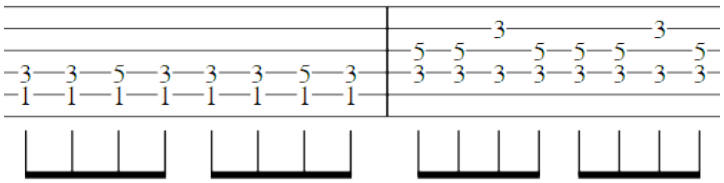


Figure 4.9: A generated rhythm guitar tablature excerpt from Chuck Berry's *Johnny B. Goode*

The excerpt of generated tablature Green Day's *American Idiot* displayed in figure 4.10 is almost identical with published tablature. Only the last chord is arguably notated oddly. In the original piece, the three high strings are struck open, in order to generate noise, rather than to play a chord of harmonic meaning. Of course, this intention cannot simply be inferred from the sheet music notation. However, the fretting is also unfortunate to reach from the previous fretting, as it becomes necessary to cross over fingers while moving to different strings.

Similar to this example, we found that many disagreements with published tablature could easily be justified by lacking information about the musical intentions. Another shortcoming was found to be that the system does not recognise potential repeating string patterns independently of the frets. Figure 4.11 shows how for Aerosmith's *Dream On*, the alternating pattern between the two highest strings and the G

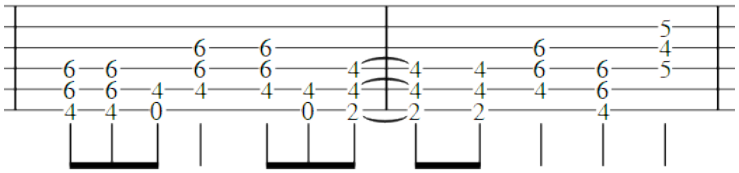


Figure 4.10: A generated tablature excerpt from Green Day's *American Idiot*

string is first predicted correctly, and then suddenly interrupted. Again, the musical intention was not captured in the sheet music and consequently. The sound will be different from the original, but the tablature is valid and playable.

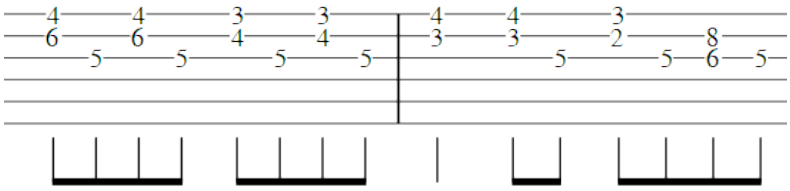


Figure 4.11: A generated tablature excerpt from Aerosmith's *Dream On*

We judged all generated tablatures to be playable and thus useful to quickly and easily learn a piece which was written in sheet music notation.

Chapter 5

Conclusion

5.1 Summary

We have shown two approaches to Neural Network based tablature generation from sheet music, one based on predicting a cost function, one based on the direct prediction of frettings. Past research was used as a starting point for a comprehensive system addressing all major aspects of the guitar fretting problem: The system can handle instruments with nearly arbitrary numbers of strings and frets, as well as all possible, strictly ascending tunings, and capos.

The generated tablatures agree with published tablature by 72.9%, measured for our best approach across a range of musical styles. Where the outputs disagree with published tablature, in many cases a musical intention was not expressible in the sheet music notation, or the system output simply suggested playing in a different fretboard location overall, but with similar shapes and patterns. While there were some unfortunate frettings, all tablatures were considered playable. Consequently, the proposed approach and the developed system can fulfil the purpose of generating tablature for sheet music input, allowing guitarists of different backgrounds to benefit from a wide range of written music.

5.2 Current Limitations

Our approach transcribes tablatures exactly from the pitches used in sheet music. Depending on which instrument the sheet music was written for, there may not be a good way of playing the piece on a single guitar. There is currently no mechanism in place which can separate the input from one line of sheet music to tablatures for multiple

guitars.

Another important limitation is the amount of used training data. The providers of `ultimate-guitar.com` had promised to supply a set of 10,000 GuitarPro tablatures for the project, which was never delivered. The terms and conditions of the website prohibit the automatic, large-scale download of tablatures (*scraping*), which would otherwise have been a viable option. Consequently, a small set of tablatures was downloaded and used for the project. We suspect that with more training data, the approach will perform noticeably better.

One potential shortcoming of the approach itself is the detachment of the Neural Networks' prediction loss from the tablature prediction accuracy. Due to the complexity of the fretting problem, however, we assume it is unfeasible to train a Machine Learning approach which fully implements the fretting problem and needs no further validation, i.e. a system like *tabgen* is necessary to validate and optimise the fretting sequences. Using the system accuracy to train the Neural Network would however make the training procedure prohibitively slow.

5.3 Future Work

Our approach, as well as the software system *tabgen*, can be used as the basis for a variety of possible future work:

A future approach could incorporate specific guitar technique annotations, like bendings and hammer-ons, into the tablature generation. As these annotations have no direct equivalent in sheet music notation, predicting their adequate may turn out as an interesting, but difficult project.

As our approach can deal with different tunings, a straightforward extension could suggest the most suitable tuning for a given musical piece.

The training of the cost prediction model could potentially be improved by maximising the difference between best rated fretting and second best rated fretting, rather than fitting directly to the probability. A larger set of training data would help improving the cost estimator, as less likely frettings will occur and serve as "negative examples" with low conditional probabilities. The current, tree-based optimisation method could potentially be enhanced, or replaced with a more powerful optimisation algorithm.

Potentially, our approaches can benefit from adding heuristic measures as additional input features. Ideally, this may combine the benefits of the Machine Learning

approach and previous, heuristic-based approaches. Alternatively, it may be beneficial to group frettings together by shape, as in many cases, a fretting shape is applicable on different locations of the fretboard. A special handling for empty strings may turn out useful as well. Furthermore, advanced Machine Learning techniques like Adversarial Neural Networks could be applied to use the training data more effectively.

Another angle towards improving the tablature generation could be capturing more high-level information about a piece, such as the overall melody shape, structure and repetitions, which could give valuable cues for an ideal tablature.

The accuracy evaluation may be improved by comparing a generated tablature against multiple published versions of the same song, as in many cases, there are several viable tablatures for a single musical piece. Furthermore, the system could be trained and evaluated on several musical genres separately, as it can learn genre-specific patterns which may not transfer well between genres.

Appendix A

Technical Documentation

This technical documentation is to accompany the *tabgen* tablature generation system. The full source code is available on GitHub[1]. Additional comments and interface documentations can be found as inline comments in the code.

A.1 Prerequisites

In order to fully run *tabgen*, you need to install the following software (recommended versions in brackets):

- Python[2] 3 (3.5), including the libraries:
  - tensorflow[3] (1.2), ideally as tensorflow-gpu[4] on top of CUDA[5]
  - keras[6] (2.0.5)
  - scikit-learn[7] (0.18.1)
  - numpy[8] (1.13.0)
  - pandas[9] (0.20.2)
  - tqdm[10] (4.14.0)
- MuseScore[11] (2.0.3)

[1] <https://github.com/gitarreneli/tabgen>
[2] <https://www.python.org/>
[3] <https://www.tensorflow.org/>
[4] <https://pypi.python.org/pypi/tensorflow-gpu/>
[5] <https://developer.nvidia.com/cuda-downloads>
[6] <https://keras.io/>
[7] <http://scikit-learn.org/stable/>
[8] <http://www.numpy.org/>
[9] <http://pandas.pydata.org/>
[10] <https://pypi.python.org/pypi/tqdm>
[11] <https://musescore.org>

A.2 Running the system

To run the system, ...

- Make sure your tablature files are split into the folders *training_input* and *evaluation_input*.
- Edit the enumeration class *tabgen.definitions.FeatureConfig* to support the desired instrument and features.
- Make sure that *tabgen.definitions.Path.MSCORE* points to a working installation of MuseScore or MuseScore portable.
- Run the *do_preprocessing* script.
- Run the *estimate_accuracy* script. You may alternatively want to use *run_sample* or make a copy of it and adjust the settings.

A.3 The *tabgen* package

The tabgen package consist of the following six modules:

A.3.1 tabgen.definititions

This is the basic configuration file, containing global settings.

The enumeration class *Path* contains file paths as strings. *Path.MSCORE* has to point to an executable binary file of *MuseScore* in order for the system to work with input files other than *.mscx.

The enumeration class *FeatureConfig* contains the configuration of features to extract from training data and to use during evaluation time. A short info is given inline. For detailed information on the features, please refer to the project report.

A.3.2 tabgen.preprocessing

The preprocessing module defines functions for batch-processing of tablatures, turning them into usable training data. Simply run this module using the *do_preprocessing* executable.

A.3.3 tabgen.base

A collection of base classes:

- ChordFrettingEvaluatorBase: The abstract base class for any evaluator. If you wish to add a new evaluator, e.g. a hand-tuned heuristic, inherit from this class and implement the *evaluate(fretting)* method.
- StringConfigBase: An abstract class for StringConfig, to decouple dependencies
- PruningConfig: A configuration object for pruning in the tree search.

A.3.4 tabgen.modelling

This is the core package, modelling the nature of the guitar fretting problem. We conceptually divide classes into *pitch view*, capturing the musical intention or output, and *fretting view*, capturing the mechanics of the fretboard.

Pitch view and *fretting view* are connected by the *StringConfig* class, which defines the strings and frets of an instrument. Conceptually, an object from the *fretting view* can be applied to a *StringConfig*, yielding an object in *Pitch view*. Similarly, the *StringConfig* can be used to generate possible *fretting view* objects from a *pitch view* object. The implementation of *StringConfig* pre-defines some typical instruments, e.g. *StringConfig.STANDARD_24_FRETS* for a six-string guitar with 24 frets in standard tuning.

The following *pitch view* classes are implemented:

- Pitch: represents a single pitch. The pitch class is based on the MIDI integer representation, but can generate note names with the method *get_note_name()*. It can further produce viable *NoteFretting*s with the method *get_note_frettings(string_config)*.
- Chord: A chord is implemented as a collection of *Pitch*es with a duration. It can produce viable *ChordFretting*s with the method *get_note_frettings(string_config, ...)*.

The following *fretting view* classes are implemented:

- NoteFretting: Captures a single note fretting, defined by a combination of string and fret. The corresponding *Pitch* object can be generated by calling *get_pitch(string_config)*.
- ChordFretting: Captures a fretting of a *Chord*. *ChordFretting* can be understood as the main class in *tabgen.modelling*.
  - The features are extracted in this class and are available through the property *ChordFretting.features*.
  - The property *ChordFretting.cost* captures the cost of the fretting, depending on the evaluator (class *ChordFrettingEvaluatorBase*) the *ChordFretting* was initialised with.

  - The property *next_pitches* implements the pitch lookahead feature for LSTM predictions.
  - The property *previous* can be used to traverse backwards in a tree of *ChordFrettings*.
  - The method *get_chord(string_config)* yields the associated chord.
- ChordFrettingSequence: A sequence of *ChordFrettings*, i.e. a potential tablature. Offers a simple text tablature printout with *to_ascii_tab()* and implements the model-based aspects of the tree search implemented by *tabgen.processing.Solver*.

A.3.5 tabgen.evaluation

The evaluation module is a collection of evaluation classes, i.e. subclasses of *tabgen.base.ChordFrettingEvaluatorBase* which return a cost function *tabgen* will optimise for. The following implementations are available:

- RandomChordFrettingEvaluator: Returns a random number, resulting in a random tablature. Use as a lowest baseline for your experiments.
- BaselineChordFrettingEvaluator: A generic class for heuristics, implemented as a hand-tuned linear model. Use the *weights* dictionary to assign weights to features. If you set *tabgen.definitions.FeatureConfig.heuristics = True*, you will have access to a predefined set of heuristics, available as features *'heuristics_*'*. For example, use *dict(heuristic_distance_move=1.0)* to generate frettings based on the distance travelled between consecutive frettings.
- ProbabilityLookupEvaluator: Looks up probabilities from preprocessing. Falls back to a low probability for unseen data.
- RegressionChordFrettingEvaluator: Estimates the cost function as a predicted negative log probability. Refer to the project report for details.
- LSTMChordFrettingEvaluator: Predicts a new fretting based on previous frettings. As this does not clearly identify a "second best" match, you may want to set your *PruningConfig* to (0,1,0,1). This effectively disables the tree search and selects the best fit at every time step.

A.3.6 tabgen.processing

This module implements two classes tying the whole system together:

- Parser: Parse MuseScore XML files to an internal class construct with *parse(file_path)* or write a previously parsed class construct back to an XML file with *save(file_path)*.

Note that *save* does not generate a new file, but copy the existing structure and change only the tablature to the generated one. The parser is tested with XML files converted from Guitar Pro with MuseScore 2.0.3, but may not fully work with other input formats.

- Solver: The *Solver* implements the tree search used to find the best fretting sequence for a chord sequence input. Call *Solver.solve(chord_sequence, string_config)* to create a new tablature from a list of *Chord*s, or use *Solver.solve_multi(...)* to process a batch of input files with the tablature generator.

A.4 Executables

- do_preprocessing: Scans files in *Path.TRAINING_INPUT* and generates a training data file *Path.FEATURE_FILE*
- estimate_accuracy: trains models on *Path.FEATURE_FILE* and evaluates them. Change the *evaluators* dictionary to select or define the models to train. Note that only one cost neural network and one direct prediction network can be trained at the same time. You may however backup the weight files.
- run_sample: Use this file as a skeleton for running tabgen with your own settings.

Bibliography

[Alcabasa and Marcos, 2012] Alcabasa, L. and Marcos, N. (2012). Automatic guitar music transcription. In *Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on*, pages 197–202. IEEE.

[Barbancho et al., 2012a] Barbancho, A. M., Klapuri, A., Tardón, L. J., and Barbancho, I. (2012a). Automatic transcription of guitar chords and fingering from audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(3):915–921.

[Barbancho et al., 2012b] Barbancho, I., Tardon, L. J., Sammartino, S., and Barbancho, A. M. (2012b). Inharmonicity-based method for the automatic generation of guitar tablature. *IEEE transactions on audio, speech, and language processing*, 20(6):1857–1868.

[Barnes, 2014] Barnes, T. (2014). Why not being able to read music means nothing about your musical ability. `https://mic.com/articles/101250`. Accessed: 2017-07-26.

[Bengio and Frasconi, 1995] Bengio, Y. and Frasconi, P. (1995). An input output hmm architecture. In *Advances in neural information processing systems*, pages 427–434.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[Burlet and Fujinaga, 2013] Burlet, G. and Fujinaga, I. (2013). Robotaba guitar tablature transcription framework. In *ISMIR*, pages 517–522.

[Burns and Wanderley, 2006] Burns, A.-M. and Wanderley, M. M. (2006). Visual methods for the retrieval of guitarist fingering. In *Proceedings of the 2006 conference on New interfaces for musical expression*, pages 196–199. IRCAMCentre Pompidou.

[Bygrave, 1996] Bygrave, G. (1996). Fingering for stringed instruments. BSc Honours Year Dissertation.

[Fiss and Kwasinski, 2011] Fiss, X. and Kwasinski, A. (2011). Automatic real-time electric guitar audio transcription. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 373–376. IEEE.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

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

[Hori et al., 2013] Hori, G., Kameoka, H., and Sagayama, S. (2013). Input-output hmm applied to automatic arrangement for guitars. *Information and Media Technologies*, 8(2):477–484.

[Humphrey and Bello, 2014] Humphrey, E. J. and Bello, J. P. (2014). From music audio to chord tablature: Teaching deep convolutional networks to play guitar. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6974–6978. IEEE.

[Hunt and Black, 1996] Hunt, A. J. and Black, A. W. (1996). Unit selection in a concatenative speech synthesis system using a large speech database. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 373–376. IEEE.

[Kehling et al., 2014] Kehling, C., Abeßer, J., Dittmar, C., and Schuller, G. (2014). Automatic tablature transcription of electric guitar recordings by estimation of score-and instrument-related parameters. In *DAFx*, pages 219–226.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Le, 2013] Le, Q. V. (2013). Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE.

[Macrae and Dixon, 2011] Macrae, R. and Dixon, S. (2011). Guitar tab mining, analysis and ranking. In *ISMIR*, pages 453–458.

[McVicar et al., 2015] McVicar, M., Fukayama, S., and Goto, M. (2015). Autoguitartab: computer-aided composition of rhythm and lead guitar parts in the tablature space. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(7):1105–1117.

[MIDI, 1996] MIDI, M. A. (1996). The complete midi 1.0 detailed specification. *Los Angeles, CA, The MIDI Manufacturers Association*.

[Miura et al., 2004] Miura, M., Hirota, I., Hama, N., and Yanagida, M. (2004). Constructing a system for finger-position determination and tablature generation for playing melodies on guitars. *Systems and Computers in Japan*, 35(6):10–19.

[Obacom, 2015] Obacom, A. (2015). 10 legendary musicians who never learned how to read music. `http://www.nairaland.com/2408051`. Accessed: 2017-07-26.

[O'Grady and Rickard, 2009] O'Grady, P. D. and Rickard, S. T. (2009). Automatic hexaphonic guitar transcription using non-negative constraints.

[Olah, 2015] Olah, C. (2015). Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2017-08-07.

[Paleari et al., 2008] Paleari, M., Huet, B., Schutz, A., and Slock, D. (2008). A multimodal approach to music transcription. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 93–96. IEEE.

[Radicioni et al., 2004] Radicioni, D., Anselma, L., and Lombardo, V. (2004). A segmentation-based prototype to compute string instruments fingering. In *Proceedings of the Conference on Interdisciplinary Musicology*, volume 17, page 97.

[Radisavljevic and Driessen, 2004] Radisavljevic, A. and Driessen, P. F. (2004). Path difference learning for guitar fingering problem. In *ICMC*, volume 28.

[Ramos et al., 2016] Ramos, J. V., Ramos, A. S., Silla, C. N., and Sanches, D. S. (2016). An evaluation of different evolutionary approaches applied in the process of automatic transcription of music scores into tablatures. In *Tools with Artificial Intelligence (ICTAI), 2016 IEEE 28th International Conference on*, pages 663–669. IEEE.

[Reid, 2016] Reid, H. (2016). Why aren't guitar players better at reading music? `http://www.woodpecker.com/blogs/reading_music.html`. Accessed: 2017-07-26.

[Rutherford, 2009] Rutherford, N. (2009). Fingar, a genetic algorithm approach to producing playable guitar tablature with fingering instructions. *Undergraduate project dissertation, Dept. of Computer Sci., Univ. of Sheffield*, 15.

[Sayegh, 1989] Sayegh, S. I. (1989). Fingering for string instruments with the optimum path paradigm. *Computer Music Journal*, 13(3):76–84.

[Sayegh and Tenorio, 1988] Sayegh, S. I. and Tenorio, M. F. (1988). Inverse viterbi algorithm as learning procedure and application to optimization in the string instrument fingering problem.

[Stavropoulou, 2017] Stavropoulou, N. (2017). Computing guitar fingerings. MSc Dissertation. School of Informatics, University of Edinburgh.

[Tamar et al., 2016] Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. (2016). Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162.

[Tuohy and Potter, 2005] Tuohy, D. R. and Potter, W. D. (2005). A genetic algorithm for the automatic generation of playable guitar tablature. In *ICMC*, pages 499–502.

[Tuohy and Potter, 2006a] Tuohy, D. R. and Potter, W. D. (2006a). An evolved neural network/hc hybrid for tablature creation in ga-based guitar arranging. In *ICMC*.

[Tuohy and Potter, 2006b] Tuohy, D. R. and Potter, W. D. (2006b). Ga-based music arranging for guitar. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1065–1070. IEEE.

[Tuohy and Potter, 2006c] Tuohy, D. R. and Potter, W. D. (2006c). Generating guitar tablature with lhf notation via dga and ann. *Advances in applied artificial intelligence*, pages 244–253.

[Tuohy and Potter, 2006d] Tuohy, D. R. and Potter, W. D. (2006d). Guitar tablature creation with neural networks and distributed genetic search. In *Proc. of the 19th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA-AIE06, Annecy, France*.

[West, 1993] West, A. (1993). Determining playable guitar fingerings from sheet music. Project Report.

[Yazawa et al., 2014] Yazawa, K., Itoyama, K., and Okuno, H. G. (2014). Automatic transcription of guitar tablature from audio signals in accordance with player's proficiency. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 3122–3126. IEEE.

[Yazawa et al., 2013] Yazawa, K., Sakaue, D., Nagira, K., Itoyama, K., and Okuno, H. G. (2013). Audio-based guitar tablature transcription using multipitch analysis and playability constraints. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 196–200. IEEE.