Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Master's diploma thesis

in the field of study Data Science

Generating Jazz Chords Progressions Using Word Embeddings and Recurrent Neural Networks

Mateusz Dorobek

student record book number 277285

thesis supervisor

Tomasz Trzciński, DSc

WARSAW 2020

....

supervisor's signature author's signature

I want to dedicate this thesis to my wonderful parents **Halina** and **Andrzej Dorobek**, whose invaluable support and motivation allowed me to devote myself to science and gain priceless education while studying at Warsaw University of Technology.

Abstract

Generating Jazz Chords Progressions Using Word Embeddings and Recurrent Neural Networks

This thesis describes the problem of generating jazz music chords using recurrent neural networks (RNN) and numerical vector representations - embeddings. To create the embedding I use techniques known from the field of natural language processing (NLP). It is possible because of certain structural similarities between spoken language and music. I study the relationship between the chords in the hidden space generated by the algorithms: *Word2Vec*, *FastText* and *Multi-hot*. Visualization of chords in reduced vector representation space using *t-SNE* and *PCA* algorithms illustrates a lot of dependencies between the chords derived from the principles of jazz harmony. The test results confirmed the highest performance of chord generation when using the *Word2Vec* model in the *Skip-Gram* variant.

To generate chords I use their vector representation. I analyze the performance of sixteen models based on the recursive neural networks in terms of their hyper parameters and architecture. I also study the impact of the algorithm used to create the numerical representation on the results of recurrent models. To train neural networks I use a set of jazz standards obtained from publicly available Internet sources in a format allowing for the extraction of chord sequences.

The experiments shows the highest performance of deep models with recurrent layers - GRU and LSTM, but the one with the shortest runtime is the model with two GRU layers, Dropout regularization layer and Dense layer. I use trained models to build a simple program capable of generating continuous chord sequences in the unsupervised variant and with partially controlling this generation by the user. For implementation of all algorithms, visualization and data processing I use Python language and its libraries: `tensorflow`, `keras`, `gensim`, `scikit-learn`, `numpy`, `pychord`, `music21` etc.

Key words: recurrent neural networks, embeddings, generation, music, harmony, jazz, chords, LSTM, GRU, RNN, Word2Vec, FastText

Streszczenie

Generowanie Akordów Jazowych z Użyciem Reprezentacji Wektorowych Słów i Rekurencyjnych Sieci Neuronowych

Praca opisuje zagadnienie generowania jazzowych akordów muzycznych z wykorzystaniem rekurencyjnych sieci neuronowych (RNN) oraz wektorowych reprezentacji (ang. embeddings). Do stworzenia wielowymiarowej numerycznej reprezentacji akordów używam technik znanych z dziedziny przetwarzania języka naturalnego (NLP). Na ich skuteczne zastosowanie pozwoliły pewne podobieństwa strukturalne między językiem mówionym a muzyką. Badaniu poddane zostały zależności pomiędzy akordami w przestrzeni ukrytej wygenerowanej przez algorytmy: *Word2Vec*, *FastText* oraz *Multi-hot*. Wizualizacja akordów w przestrzeni reprezentacji wektorowych zredukowanej za pomocą algorytmów *t-SNE* oraz *PCA* zobrazowała wiele zależności między akordami wynikających z zasad harmonii jazzowej. Wyniki badań potwierdzają największą wydajność generacji akordów przy zastosowaniu modelu *Word2Vec* w wariancie *Skip-Gram*.

Do generowania akordów używam ich wektorowej reprezentacji. Analizuję wydajność szesnastu modeli opartych na rekurencyjnych sieciach neuronowych pod kątem ich hiper parametrów i architektury. Badam też wpływ algorytmu użytego do stworzenia reprezentacji numerycznej na wyniki modeli rekurencyjnych. Do treningu sieci neuronowych używam zbioru standardów jazzowych, uzyskanych z ogólnodostępnych źródeł internetowych w formacie pozwalającym na ekstrakcje sekwencji akordów. Przeprawodzone eksperymenty wskazują na wysoką wydajność głębokich modeli z warstwami rekurencyjnymi GRU oraz LSTM, a krótszy czas obliczeń przeważa na korzyść modelu składającego się z dwóch warstw GRU, warstwy regularyzującej Dropout oraz warstwy gęstej.

Wytrenowane modele służą do zbudowania prostego programu zdolnego do generowania ciągłych sekwencji akordów w wariancie nienadzorowanym oraz z możliwością częściowego kontrolowania tej generacji przez użytkownika. Do implementacji wszystkich algorytmów, wizualizacji i przetwarzania danych używam języka Python oraz jego bibliotek: `tensorflow`, `keras`, `gensim`, `scikit-learn`, `numpy`, `pychord`, `music21` itd.

Słowa kluczowe: rekurencyjne sieci neuronowe, embedding, generacja, muzyka, harmonia, jazz, akordy, LSTM, GRU, RNN, Word2Vec, FastText

Warsaw, ...

Declaration

I hereby declare that the thesis entitled „Generating Jazz Chords Progressions Using Word Embeddings and Recurrent Neural Networks", submitted for the Master degree, supervised by DSc Tomasz Trzciński, is entirely my original work apart from the recognized reference.

.....

Contents

Introduction

In my thesis I study the problem of jazz chords prediction. The task is to generate a chord, or propose a set of chords to choose from, based on the preceding sequence. To clarify what is a chord and create its formal definition I describe several musical concepts (see Section 1.3).

My approach is to find the numerical representation of the chord symbols, use this representation to encode the chords in sequences that I extract from the downloaded collection of jazz songs. Then I create the dataset that I use to train generative model. First I train the embedding model and use it to encode the chord symbols to numerical form. Next I use encoded sequences to create training data, which consists of training samples. Each training sample $x \in X$ is a matrix of size $n \times e$, where n is a number of chords and e is a size of embedding (numerical representation). The expected output $y \in Y$ is the chord (vector of size e) that follows the input sequence in a song that the sample is extracted from. I use this dataset mainly to train the neural network. I use it also to initiate the generation process by sampling the initiating sequence from it. Due to the sequential character of the chords progressions I focus on recurrent neural networks architectures [27]. After tuning the architecture and hyper-parameters I use the model to create a simple program that generates continuous chord progression of chords based on initiation sequence.

Music harmony as well as spoken language has strong short and long context dependencies [24]. Chords as well as words have different variants and substitutions (see Section 1.3) These similarities are the main reason why I test different numerical chord representations using word embeddings used in NLP. I also create and test my own method based on harmonic structure of chords - *Multi-hot* (see Definition 1.8) as a baseline model to compare with word embeddings. During my research I found multiple papers on chords prediction and embeddings. Unfortunately a lot of these works have very strong simplification assumptions about chords. In my thesis I do not use significant simplifications to test how different models perform when operating on a fully complex jazz vocabulary.

In Chapter 1 - Proposed Music I introduce a brief history of music, explaining how it was influenced by jazz, and why I narrow down the scope of this work to this genre of music. I also define some music elements in mathematical form, and introduce different chord representations

used later along this work. In Chapter 2 I summarise different works in chords embeddings and prediction, pointing out their strong sides and aspects that can be improved. Chapter 3 describes how I gather and process all the data using formats described in previous chapters. There are also data set statistics, which I use to justify my assumptions and simplifications. This chapter also describes the embedding techniques that I use for chord encoding and all recurrent neural networks models used in chords prediction. Chapter 4 is a summary of all the experiments performed in my thesis. Comparison of embedding techniques and results of RNN models, along with advanced visualisation gives a view on the performance of these models and summarises achieved results. In Chapter 5 I summarise my contribution to this thesis, describe the problems that I had during this project and state possible improvements in future work.

1. Music

Music as an art form is one of the most important elements of human culture. It is a universal language that can be understood across the world. It evolved in different cultures but still shares the same fundamental rules. The listeners experience similar feelings when listening to music from different parts of the world [26]. This means that music carried by an acoustic wave has some information that people are able to interpret as emotions.

To describe music more precisely we need to define the basics elements of music. Melody is an element that describes sequences of tones, pitch, and duration. In European music one of the most important factor is melody. In southern music (Africa, Latin and South America) rhythm is the element that makes this music so unique. Rhythm organizes the time structure of music material. Harmony, which plays a crucial role in eastern music defines the relationships between tones, and how they change in time. There are other elements: dynamics, articulation, tempo and timbre, that also can be used to describe and analyze music in a more structured manner.

1.1. Harmony

In this thesis, I work with harmony because it is s wide, complex and interesting component of jazz. Music from a technical point of view is nothing more, than just periodic disturbances of the acoustic medium. Acoustic wave can be described in three dimensions: frequency, amplitude and time. Harmony is a set of rules that describes relationships between frequencies in time and frequencies itself. Each genre of music has its own harmony style and common patterns. The vast majority of created songs are examples of tonal music, but there is the second category of atonal music [19] that I not consider in this work. Tonal music is oriented around a key center (or several key centers) that implies the existence of harmonic tension. Some harmonic structures like dominants build it up and the others like majors resolve it [24]. In western music, pieces are build from consecutive harmonic tension and releases. People tends to perceive similar structures as consonances or dissonance even if they are from different cultures [26]. That phenomenon is easily explainable. Harmonic structure is more stable and consonant if it has more common

harmonics of its components[1]. I have described harmony structure of a chord in Section 1.3 - Jazz Chords Grammar.

1.2. Jazz

Roots of jazz music are in New Orleans, which was one of the biggest maritime transport nodes at that time. In the XIX century due to slavery a lot of people kidnapped from Africa, Latin and South America were transported to port cities. This created a unique society on a world scale. In multicultural cities like New Orleans, Louisiana, Kansas, Chicago and another developed new kind of music - jazz. Jazz and blues are the ancestors of every modern genre of music. The golden ages of jazz were in the middle of the last century. Nowadays it is not as popular but developed in a variety of forms. The reason why I focus on jazz music in this work is its variety and complex structure that is worth analyzing. Jazz harmony is a wide topic, that sometimes cannot be explained by formal rules. The richness of jazz harmony gives inspiration to numerous artists, not only musicians. In this work, I focus on jazz music because of its completeness in terms of harmony in tonal music.

1.3. Jazz Chords Grammar

To make the subject of this work clear I define the basic harmonic structure of a chord and other musical concepts that I use in this thesis.

Definition 1.1 (Note). *A note* is represented by three main features: pitch, duration and dynamics, but I focus only on pitch, duration and dynamics are less important in harmonic context.

Definition 1.2 (Pitch). *A pitch* is a psycho-acoustic feature of a tone wave of a certain frequency. Frequency can be represented as a numerical value in Hz and vertical position (height) on a music score. In this work I will use musical notation, which consist of note name (pitch class), e.g. $C, C\sharp, D$, etc. and an octave in range $[-1, 9]$.

[1] harmonics are higher components of tone, their frequencies are integer multiples of the fundamental tone.

Definition 1.3 (Interval). *An interval* is a relative distance between two notes represented by number of semitones e.g. Interval $C \rightarrow E$ has 4 semitones and in musical terms it is called a major third. Major thirteenth[2] is the highest chord extension that is present in downloaded jazz songs.

Example		**Main Intervals**		**Compound intervals (higher by octave)**	
From	**To**	**Semitones**	**Interval**	**Semitones**	**Interval**
C	C	0	Perfect unison	12	Perfect octave
C	$C\sharp$	1	Minor second	13	Minor ninth
C	D	2	Major second	14	Major ninth
C	$D\sharp$	3	Minor third	15	Minor tenth
C	E	4	Major third	16	Major tenth
C	F	5	Perfect fourth	17	Perfect eleventh
C	$F\sharp$	6	Tritone	18	Augmented eleventh
C	G	7	Perfect fifth	19	Perfect twelfth
C	$G\sharp$	8	Minor sixth	20	Minor thirteenth
C	A	9	Major sixth	21	Major thirteenth
C	$A\sharp$	10	Minor seventh	22	Minor fourteenth
C	B	11	Major seventh	23	Major fourteenth
C	C	12	Perfect octave	24	Perfect fifteenth

Table 1.1: Intervals

Definition 1.4 (Chord). *A chord* is an ordered set of at least 3 notes. Jazz chords usually have 4 to 6 different notes in its voicing. Chords can be represented **implicitly** by chord symbol, or **explicitly** by its specific application called voicing.

Definition 1.5 (Chord Substitution). *A chord substitution* is a chord that can replace another one that has the same harmonic function. Usually chord substitutions have two common components [1] e.g. tritone substitution C^7 - $G\flat^7$.

[2] It is present for example in B^{13} chord.

Definition 1.6 (Chord symbol). *A chord symbol* is its implicit representation defined by:

- its root note[3] (e.g. $C7, E\flat 7, F\sharp 7$) and
- *type* (e.g. $\mathrm{C}\Delta^{\sharp 11}$, Cm^{9}, $\mathrm{C}^{7\sharp 5}$).

Definition 1.7 (Chord voicing). *A chord voicing* is a specific arrangement of notes for a given chord. Voicing can be created, by enumerating the exact notes composing the chord,
(e.g. $\mathrm{E}^{7}(^{\sharp 5}_{\sharp 9}) :=< 0, 4, 8, 12, 14, 19 >$ see Figure 1.1)

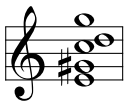

Figure 1.1: Sample chord in music notation

In order to represent a chord in numerical representation, we can use a set of specific notes. Such a set can be created using voicing of the chord. Root position voicing implies that notes appear in order from the lowest pitch to the highest. The notes in a voicing can be numbered like in table 1.1 - Intervals. This explicit representation has a variable length. To create chords representation with fixed length I define *Multi-hot* encoding in Definition 1.8.

Definition 1.8 (Multi-hot). *Multi-hot* encoding is a vector representation where each bit corresponds to one of possible values of encoded vector.
Multi-hot vector ($c = \{c_1, c_2, \ldots, c_{\mathbb{N}}\} \in \{0, 1\}^{\mathbb{N}}$) bits are set to 1 if corresponding value in encoded vector is present and 0 otherwise, e.g. Cm^{7} chord ($< 0, 3, 7, 10 >$ in explicit representation) encoded as multi-hot: Figure 1.2

[1	0	0	1	0	0	0	1	0	0	1	0	...	0]
C			E♭				G			B♭			

Figure 1.2: Sample Multi-hot encoded chord

[3] I'll ignore enharmonic due to its insignificance in this work

2. Related works

This Chapter describes works related to the music data processing, chord embeddings and sequence generation. I present the results, conclusions and describe the architecture of each of the models used in these papers.

2.1. Data Format

Musical content can be stored in multiple ways. Some of them like audio wave (mp3 and wav files) are common to everyone that owns a phone and uses physical copy of a song. Other formats like spectrogram are used in scientific applications. This representation consist of frequencies spectrum over time providing better view on audio than audio wave 2D representation. Spectrogram has frequency is its third dimension. This representation is far too complex if one wants to analyze musical content. The spectrogram contains information about the timbre of a given note, and for the harmony analysis this information is completely unnecessary. To prevent this, it is enough to change the frequency dimension from continuous to discrete. This way piano-roll format is created. I used this format in my previous project *Application of Deep Neural Networks to Music Composition Based on MIDI Datasets and Graphical Representation* [8, 9], to generate music fragments using DCGAN architecture. In this work I focus not on specific sounds, but on the harmony rules that describe the song. Although it may be hard to imagine the harmonic structures in a song, they are very well described by chords. Therefore, in this work I use this representation of musical content which is a sequence of chord symbols.

2.2. Embeddings

Embedding is a term that determines projection of input data into other representation space. They have applications in natural language processing [20]. I use embeddings to encode chords symbols in a numerical form because of fixed size of the encoded vector and useful properties of that encoding in its latent space [21].

2.2.1. Word2Vec

Word2Vec [20] by Mikolov et al. is a model that learns word embeddings. It is a two-layer neural network trained to reconstruct the words with similar context and meaning to embedded space. The values of hidden layer of a trained *Word2Vec* are a numerical representation of a given word. There are two possible versions: Continuous Bag of Words (CBOW) where network predicts word from context words surrounding it and Skip-Gram, that works in the opposite way predicting context from a word.

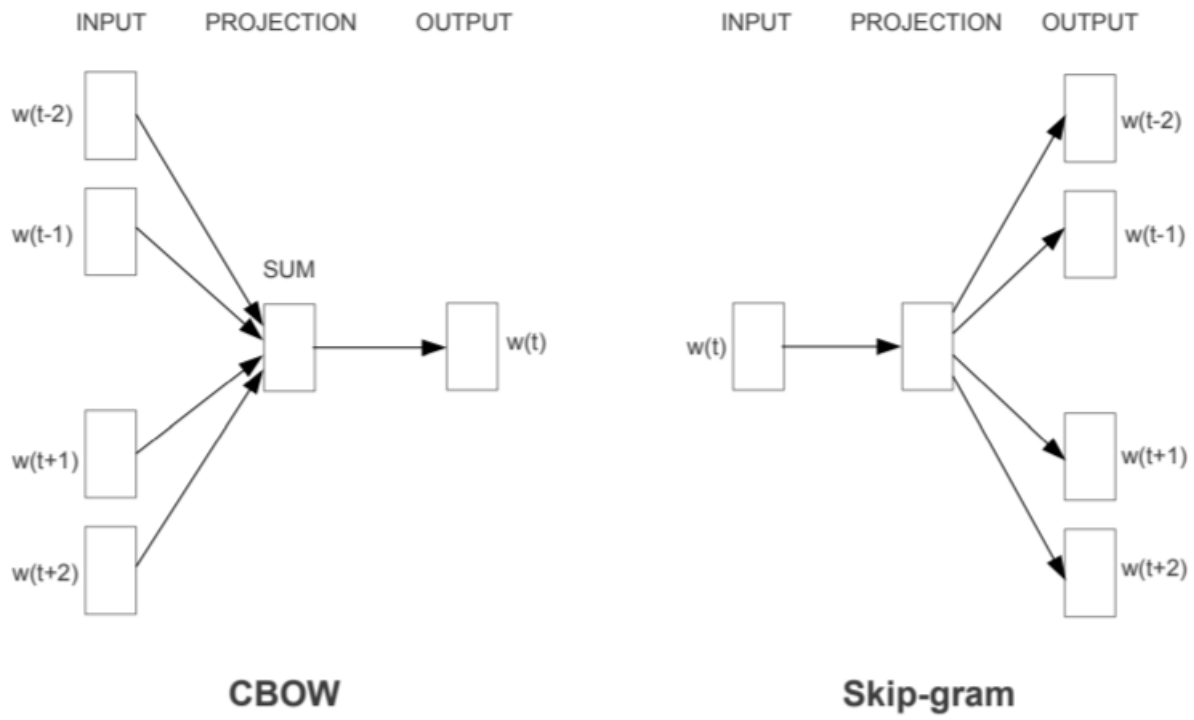

Figure 2.1: CBOW cs Skip-Gram [21]

2.2.2. FastText

FastText [13] by Mikolov et al. extends the *Word2Vec* architecture by using several N-grams instead of words only. N-grams are sub-words that consists of n letters of a given word, e.g. word "*chord*" have three tri-grams: "*cho*", "*hor*" and "*ord*". Embedding of that word is a sum of its n-gram embeddings. *FastText* is better suited for embedding rare words because it exploits subword information [2].

2.2.3. Seq2Seq

In [30] Sutskever et al. show the sequence to sequence models. These architectures are modeling probability $\mathbf{P}(\mathbf{y}_j|\mathbf{Y}_{<j}, \mathbf{X})$, where $\mathbf{X}$ is an input sequence, $\mathbf{Y}$ is an output sequence

and $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_J)$. Model consists of two deep *LSTMs Encoder-Decoder* networks first for input sequence and the second for the output sequence. *Sequence-to-sequence* models find various applications in NLP problems like dialogue systems, question answering or machine translation.

Embedded representations are found useful in sequence modelling. Similar contextual dependencies in chord sequences are described in the theory of the music harmony [1]. As well as short term dependencies, long ones play important role in harmony. Similarities between chord and word sequences resulted in multiple works on chord embeddings.

2.2.4. Chord2Vec

In [27] Madjiheurem et al. describe the use of *bilinear* and *sequence-to-sequence* based models to create chord embeddings. *Bilinear* model is based on *Word2Vec Skip-gram* architecture [20] with output layer slightly changed. Using *Multi-hot* representation this model assumes independence between notes in a chord, which is not true due to the chords nature, e.g. major chords almost never have the altered *9'th*. *Sequence-to-sequence* model proposed by Sutskever et al.[30] is often used in language, image and audio models to predict one sequence from another. For example an input music audio sequence and melodic line.

This approach uses explicit representation of a chord (see Definition 1.4). Neighborhood of an input chord is used as an output sequence. Each chord from an output sequence is separated by a marker symbol ϵ which denotes the end of the sequence. For example:

$$\begin{array}{rccl} \textit{Input:} & (\mathrm{C}\Delta^9) & \Rightarrow & \{0, 4, 7, 11, 14\} \\ \textit{Output:} & (\mathrm{Dm}^7,\ \mathrm{G}^7) & \Rightarrow & \{2, 5, 9, 12, \epsilon, 7, 11, 14, 17, \epsilon\} \end{array}$$

Experiments performed on these models proved that *sequence-to-sequence* based models have better performance. Nevertheless the *skip-gram* based models are mentioned as a matter of future work planned by the authors. It is due to its speed in text classification and comparable accuracy with much simpler architectures than the deep learning models.

2.3. Sequence Prediction

In order to generate a word (chord in my case) that is a part of a sequence we need to predict it from the previous context. Otherwise it will be a random guess from vocabulary.

2.3.1. N-Grams

N-Grams language models use fixed length history (N-1) to predict the next symbol. Fixed size simplifies the architecture, allowing to store the probabilities in table. This entails consequences in a form of its limited lookup and long term dependencies skipping. Sometimes with larger N a zero-frequency problem may occur but additive smoothing is a suitable solution in this case.

2.3.2. RNN

Recurrent Neural Networks originated in 1986 in Rumelhart work [25]. They are based on regular neural networks but each RNN layer consist of successively connected neurons organized in a directed path. RNN uses internal state to store dependencies from history. It is a form of "memory" [10]. It makes it possible to process variable length input sequences. Input sequence data $x = \langle \ldots, x_{t-1}, x_t, x_{t+1}, \ldots \rangle$ is passed through this layer in such a way, that hidden state from cell with x_{t-1} as an input is passed to the next cell with x_t input (see Figure 2.2).

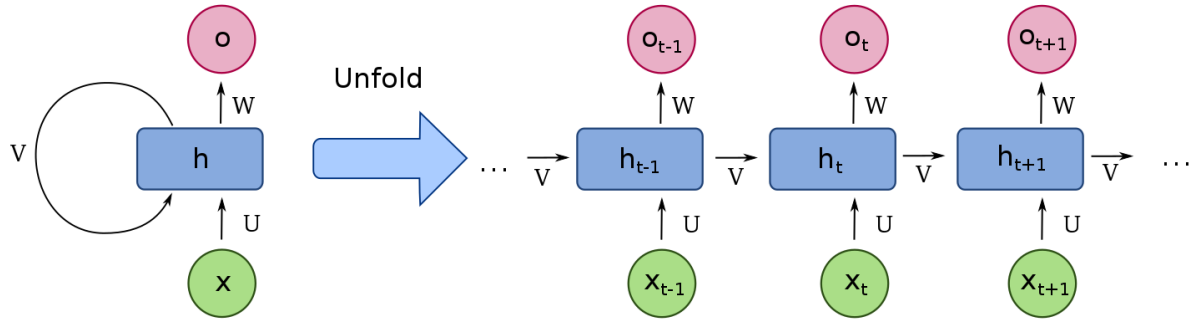

Figure 2.2: RNN structure [7]

2.3.3. LSTM

Long Short Term Memory introduced in [12] by Hochreiter and Schmidhuber is a special kind of Recurrent Neural Network capable of learning long term dependencies. Just like the RNN, *LSTM* has a structure consisting of repeating cells. A set of gates (see Figure 2.3) allow information from past (in a mater of time in a sequence) to pass through *LSTM* layers without interruption and if needed it can be strengthened. It is proven that LSTM networks are more effective than conventional RNNs, in particular when they have several layers for each time step [16].

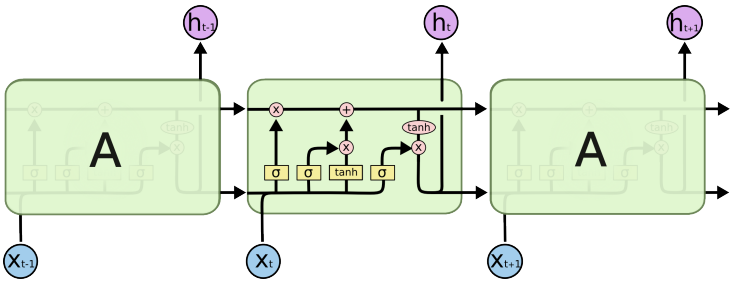

Figure 2.3: LSTM structure [23]

Basic LSTM elements:

- **Forget gate layer** at this level network decides which information will be used.
- **Input gate layer** decides how input values will influence state vector.
- **Output gate layer** splits information into three streams: direct output, long term stream (state) and short term stream that will be concatenated with the input in the next *LSTM* cell.
- **Connecting Input and Forget layer** is an extension which modifies *LSTM* structure, introducing a forgetting mechanism that allows to remove the old dependencies.
- **Peephole connection** allows other gates to have insight into the state of the cell.

2.3.4. GRU

Gated Recurrent Unit introduced in 2014 by Kyunghyun Cho et al. [5] is an improvement of LSTM network. Its architecture requires less parameters, because of its lack of the output gate and the combination of forget and input gate into the update gate (see Figure 2.4). In [6] Chung concludes that simpler architecture of *GRU* performs comparable to *LSTM*. "However we could not make concrete conclusion on which of the two gating units was better."

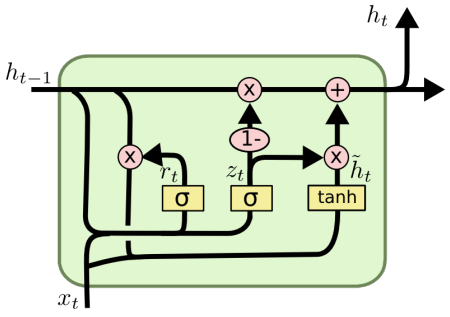

Figure 2.4: GRU structure [23]

2.3.5. Comparison of N-grams and RNN Models

In [15] authors perform a comparison between N-grams (5-grams) and recurrent neural networks (RNN). In their work authors build models that are meant to cover whole songs and learn long term context from the begging of a piece. This paper assumes a simplification that only two types of chords are taken into account, which significantly reduce the size of the chords vocabulary. Authors perform data augmentation by transposing data to all twelve keys and adding it to the base data set. Before training the language model itself they test two embedding methods: one-hot encoding, fixed size vector encoded with *Word2vec Skip-Gram* model. Embedding size space is small 4, 8, 16, 24 probably due to small size of chord vocabulary - 25 possible chords symbols. Numbers of hidden layers chosen in the final models are 3 and 5 with size 256 and 512 respectively. Finally, two architectures that performed the best are *LSTM* and *GRU* (based on average log-probability). An interesting part of the pipeline is the Hyperband - optimisation scheme used to find the best configurations of hyper-parameters for each model. As a result all RNN based models performed much better than N-gram models. *GRU* and *LSTM* performed better than simple RNN.

3. Proposed Method

In Section 3.1 I describe data flow and system architecture. In Section 3.2 I describe how I gather and process the data, and what assumptions and simplifications I make. To understand the data better I also plot distributions of chords statistics. In order to generate chord progressions, chords needs to be represented in a numerical form. I use NLP embeddings to encode chord symbols to a numerical vector. Visualisation of the embedded space (reduced by *t-SNE* and *PCA* algorithms to two dimensions) shows that some of the dependencies between chords occur in a circular geometrical shapes in the latent space. In Section 4.1 I describe how to evaluate embeddings. To predict chords from the previous sequence I use recurrent models based on *SimpleRNN*, *GRU* and *LSTM* layers (see Section 3.4). In chord progressions both short and long dependencies are equally important [1], but SimpleRNN models consider only the short range context [6], which is noticeable during the evaluation. In Section 3.3.4 describe experiments and sets of parameters that I use in order to obtain the best embedding.

3.1. System Architecture

The data flow in this project is described on Figure 3.1. After obtaining raw jazz songs I process them (see Section 3.2) to extract chord sequences. Then I use trained embedding algorithm (*Word2Vec Skip-Gram* - see Section 3.3.2) to encode chord symbols into numerical representation, and sample training dataset (see Section 3.2.7) from encoded songs using Algorithm 1. Next I use trained Neural Network (*GRU6432* model - see Section 4.3.5) to generate chords. Chord generation (described in Section 4.4) may be performed in multiple ways, with or without user control. At this level chords are still in numerical from, thus I use the same embedding model I use for encoding to decode chord sequences into a symbolic form. Symbolic form can be interpreted by a musician and played on the instruments like piano or guitar. Generated chords are not a complete song, there is no melody and rhythm in chord sequence, thus a performance based on this musical content is only a backing track, that can be used to create complete song by adding lyrics, melody and other musical elements.

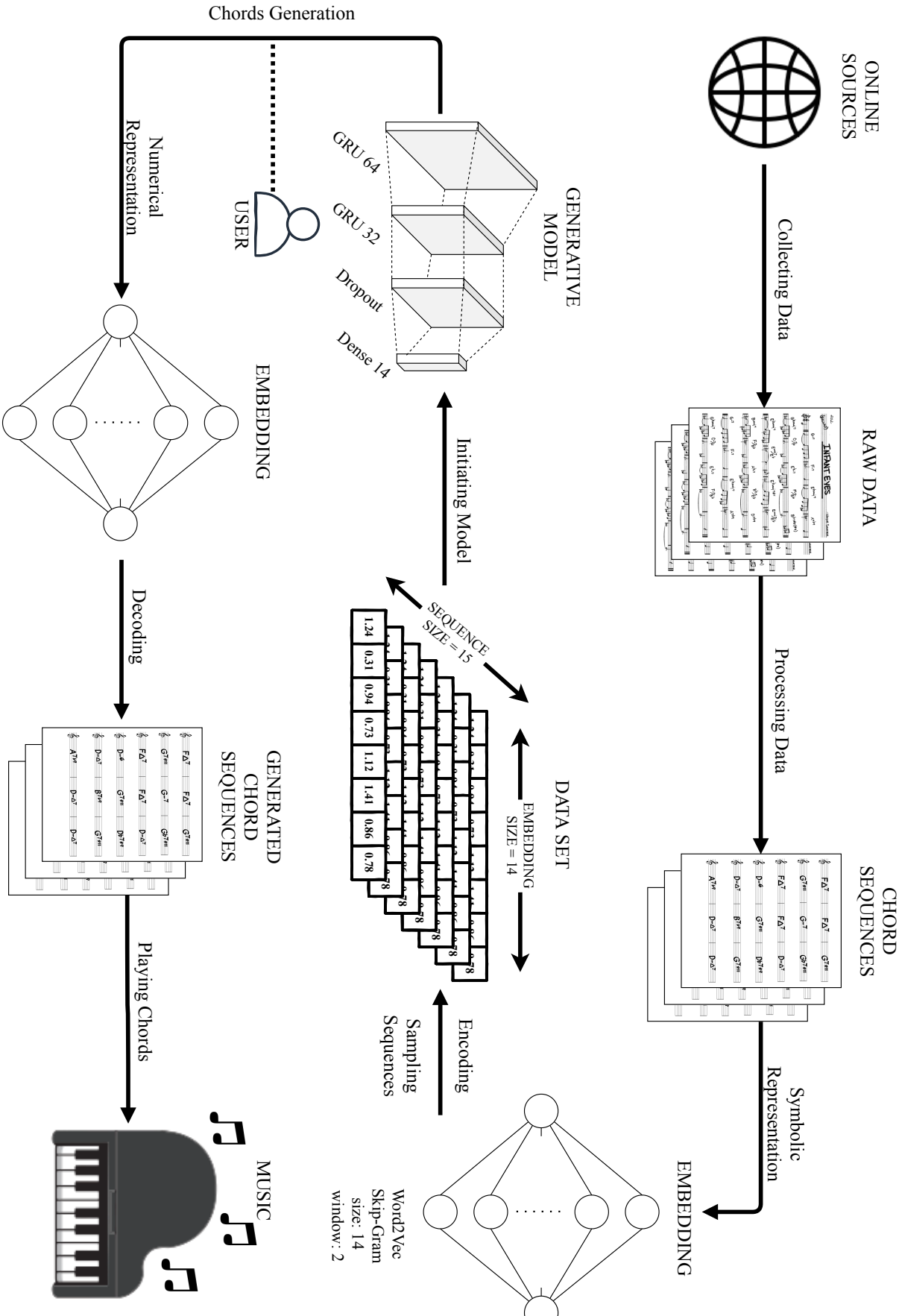

Figure 3.1: Data flow and system diagram

3.2. Data

Jazz musicians often shorten harmony to symbolic chord notation. Chord progressions along main theme are called the lead sheet (sample lead sheet in Figure 3.2). It is a useful guide for improvisation and accompanying. I use word "guide", because chord progression can be interpreted in many various ways and reharmonised during performance. Nevertheless it was appreciated by musicians to have that guide while learning jazz standards [1]. During 1970s students of Berklee College of Music gathered and published series of Real Books (link in Appendix 5.3), that become one of the biggest part of jazz music publication of all time.

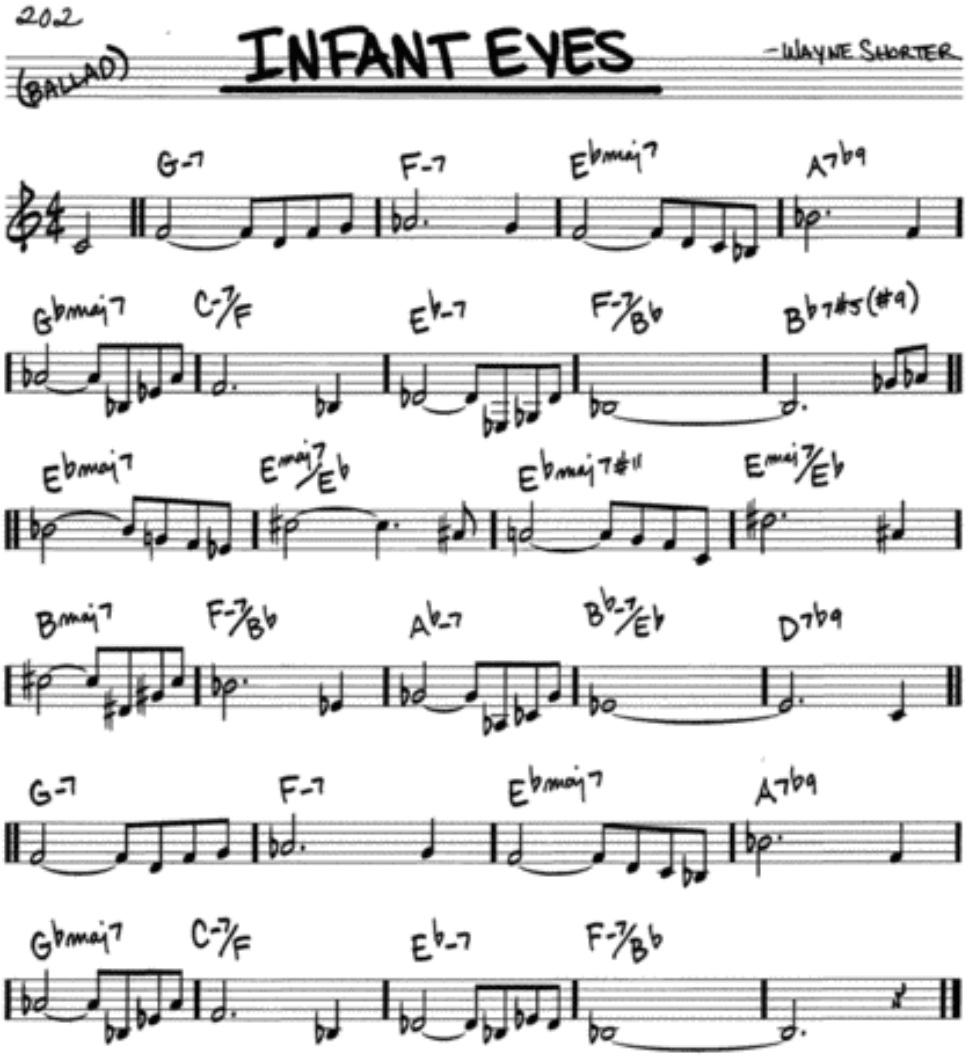

Figure 3.2: Sample jazz lead sheet [17]

[1] Well known compositions widely performed and being a part of every jazz musician repertoire. There is no definitive list of jazz standards, but most of them are well-known in jazz environment.

3.2.1. Data Collection

I gather jazz songs from *iReal Pro* forum (links in Appendix 5.3). On that website users post song chords progressions and group them in multiple categories. Search for all jazz related posts allows to download 2137 unique jazz standards. For the purpose of web scrapping, handing HTML, and parsing responses I use Python libraries: `urllib`, `beautifulsoup`, `pandas`, `numpy`, and others (see Appendix 5.3 for links to Python packages). Format of downloaded songs is a custom URL (see detailed description on the website in Appendix 5.3). I use *pyRealParser* library to decode the URL and extract measures with chords. Sample song chords from *iReal Pro* below.

```
| F^7Ab7  | Db^7E7 | A^7C7 | C-7F7 |
| Bb^7Db7 | Gb^7A7 | D-7G7 | G-7C7 |
| F^7Ab7  | Db^7E7 | A^7C7 | C-7F7 |
| Bb^7Ab7 | Db^7E7 | A^7C7 | F^7   |
| C-7F7   | E-7A7  | D^7F7 | Bb^7  |
| Eb-7    | Ab7    | Db^7  | G-7C7 |
| F^7Ab7  | Db^7E7 | A^7C7 | C-7F7 |
```

3.2.2. Raw Data Structure

Apart from chords each song contain other metadata: *title, composer, style, key, bmp, time signature*, etc. As we can see in example above there can be more than one chord in a measure. Sample chords extracted from measures below.

```
{'Root': 'F',  'Ext': '^7', 'Bass': ''},
{'Root': 'Ab', 'Ext': '7',  'Bass': ''},
{'Root': 'Db', 'Ext': '^7', 'Bass': ''},
{'Root': 'E',  'Ext': '7',  'Bass': ''},
{'Root': 'A',  'Ext': '^7', 'Bass': ''},
{'Root': 'C',  'Ext': '7',  'Bass': ''},
...
```

3.2.3. Preprocessing Assumptions

Some chords have a bass note, denoted by a slash notation. Bass note is present in less than 4% of chords, thus I decide to remove it from chord symbols. This is the first simplification that I assume about the chord structure in this work. Other simplifications that I use while creating

a dataset is that I do not differ chord duration. All chords in my dataset are consider the same length. I also remove all repetitions, what makes this chord predicting model not sensible for harmony rhythm. If user of an application that is based on that model wants to create a chords composition, he has to arrange chords by himself. My assumption is that the model would not be trained to propose different duration of chords, because it is a part of the musical arrangement not the harmony itself. This simplification makes my data more perspicuous to model. Moreover models may perform better on a simplified data.

3.2.4. Chord Types Variety

I encode all components of these chords according to Table 3.1. In my work I use 62 types of chords (61 from iRealPro documentation (see Appendix 5.3 for more information) and one that is not included (empty symbol) that denotes a *major triad*). All these chords forms a full vocabulary of jazz standards downloaded from online sources from Section 3.2.1. My vocabulary is significantly more detailed than in works I describe in Chapter 2.

Chord type	**Components**	**Chord type**	**Components**	**Chord type**	**Components**	**Chord type**	**Components**
5	(0, 7)	o7	(0, 3, 6, 9)	h9	(0, 3, 6, 10, 14)	-11	(0, 3, 7, 10, 14, 17)
2	(0, 2, 7)	^7♯5	(0, 4, 8, 11)	7b9sus	(0, 5, 7, 10, 13)	7♯11	(0, 4, 7, 10, 14, 18)
+	(0, 4, 8)	6	(0, 4, 7, 9)	7susadd3	(0, 5, 7, 10, 16)	9♯11	(0, 4, 7, 10, 14, 18)
(empty)	(0, 4, 7)	-6	(0, 3, 7, 9)	9	(0, 4, 7, 10, 14)	7♯9♯11	(0, 4, 7, 10, 15, 18)
o	(0, 3, 6)	-7b5	(0, 3, 6, 10)	7b9	(0, 4, 7, 10, 13)	7b9♯11	(0, 4, 7, 10, 13, 18)
h	(0, 3, 6)	-b6	(0, 3, 7, 8)	7♯9	(0, 4, 7, 10, 15)	7b9♯9	(0, 4, 7, 10, 13, 15)
sus	(0, 5, 7)	-♯5	(0, 3, 7, 8)	9b5	(0, 4, 6, 10, 14)	9sus	(0, 5, 7, 10, 14, 16)
^	(0, 4, 7)	7b5	(0, 4, 6, 10)	9♯5	(0, 4, 8, 10, 14)	11	(0, 4, 7, 10, 14, 17)
-	(0, 3, 7)	7♯5	(0, 4, 8, 10)	7♯9♯5	(0, 4, 8, 10, 15)	7b9b13	(0, 4, 7, 10, 13, 17, 20)
add9	(0, 4, 7, 14)	-^7	(0, 3, 7, 11)	7♯9b5	(0, 4, 6, 10, 15)	7b13	(0, 4, 7, 10, 14, 17, 21)
^7	(0, 4, 7, 11)	-^9	(0, 3, 7, 11, 14)	7b9b5	(0, 4, 6, 10, 13)	13	(0, 4, 7, 10, 14, 17, 21)
-7	(0, 3, 7, 10)	^9	(0, 4, 7, 11, 14)	7b9♯5	(0, 4, 8, 10, 13)	13♯11	(0, 4, 7, 10, 18, 17, 21)
7	(0, 4, 7, 10)	-69	(0, 3, 7, 9, 14)	7alt	(0, 4, 8, 10, 15)	13b9	(0, 4, 7, 10, 13, 17, 21)
7sus	(0, 5, 7, 10)	-9	(0, 3, 7, 10, 14)	^13	(0, 4, 7, 11, 14, 21)	13♯9	(0, 4, 7, 10, 15, 17, 21)
h7	(0, 3, 6, 10)	69	(0, 4, 7, 9, 14)	^9♯11	(0, 4, 7, 11, 14, 18)	13sus	(0, 5, 7, 10, 14, 17, 21)
		^7♯11	(0, 4, 7, 11, 18)			7b13sus	(0, 5, 7, 10, 14, 17, 20)

Table 3.1: Chord Types

The chords are not evenly distributed. The basic types are most commonly used among downloaded jazz songs. Chords with a complex structure are used less frequently. Below I present a comparison of the distribution of 15 most (fig. 3.3), and 15 least (fig. 3.4) popular chords in the context of a set of all collected pieces. The uneven distribution of chords have influence on what chords will be generated by recurrent model. A generator that returns a flat distribution will not correctly model the actual characteristics of this data.

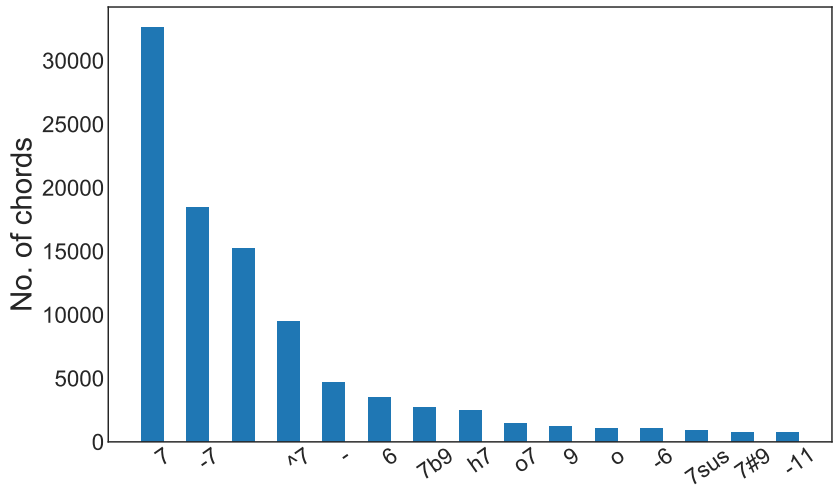

Figure 3.3: Most common chords types

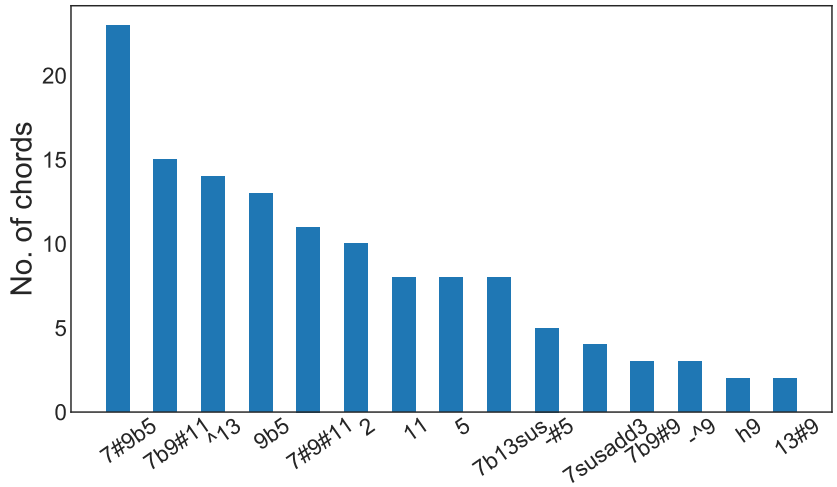

Figure 3.4: Least common chords types

3.2.5. Chord Roots Distribution

Besides of chord types, I plot the distribution of the chord root notes. As it turned out this distribution is also not uniform. The vast majority of chords have Eb, Bb, F, C, G and D root note. These chords have less key accidentals. The closer to C on the circle of fifths the fewer key accidentals. However, this distribution is centered on the F note. It is most likely so, because a significant part of jazz songs have a blues form, which is often traditionally played in the F major key.

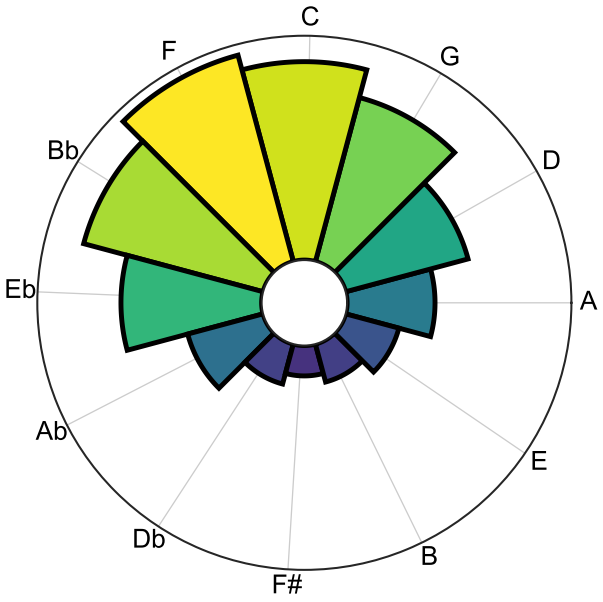

Figure 3.5: Roots distribution

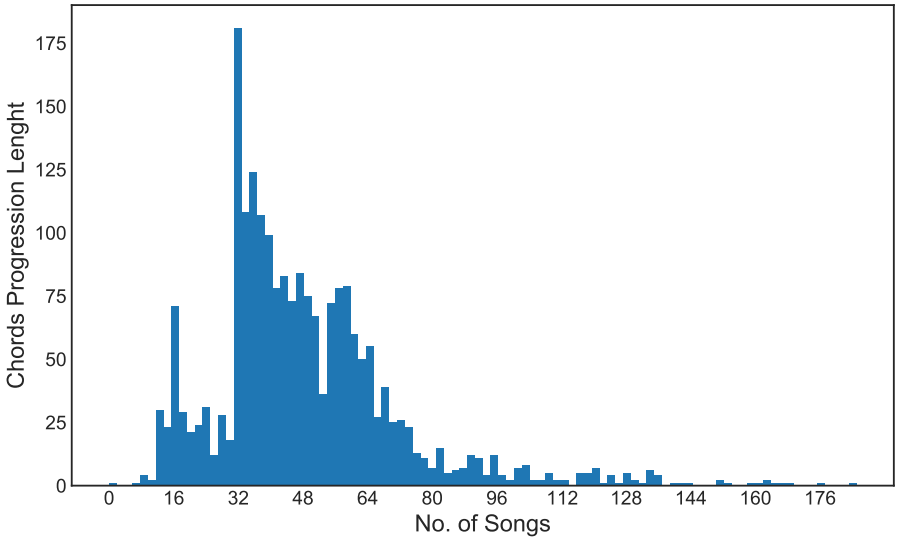

Figure 3.6: Chords progression lengths distribution

3.2.6. Progression Length Distribution

The mean length of chord sequence (without form repetitions) after aforementioned preprocessing is 48.5 chords per song. Jazz songs tends to have more complex harmonic structure than other genres of music. In Figure 3.6 we can see distribution of chord sequence lengths in songs. There is a peak in 32, which is very common jazz song length in measures. Some of these songs have more than one chord in measure, they are represented as a dense right side of that peak.

3.2.7. Generating Training Dataset

To train recurrent neural network I use two sets X as an input sequence and Y as an true label - the expected output. To create this training dataset I use Algorithm 1. This algorithm requires known size of the sequence in order to generate training samples. The size of the embedding does not affect the procedure of the algorithm.

Algorithm 1: Training dataset generation

Data: *songs* - chord sequences extracted from downloaded data
n - length of an input sequence,
e - size of the embedding vector

Result: *training dataset* $= (X, Y)$ where
X - matrix with size $n \times e$,
Y - vector with length e

```
INITIALIZATION: X =list(), Y=list()
for song in songs if len(song>=n) do
    //To generate sequence of size n I need song that is longer or equal n
    for i in range(len(song)) do
        if i<len(song)-n+1 then
            in_seq=song[i:i+n] //I take all n-length subsequence from song
        else
            //In case of last k measures of song I take
            in_seq_beg=song[i:len(song)-1] //k last chords of song and
            in_seq_end=song[:(i+n+1)%len(song)] //n − k chords from beginning and
            in_seq=np.concatenate([in_seq_beg, in_seq_end]) //merge them into
              continuous sequence wth size of n
        out_chord=song[(i+n+1)%len(song)] // the output is the chord that follows
         corresponding in_seq. % is modulo operator.
        X.append(in_seq)
        Y.append(out_chord)
return X Y
```

3.3. Embedding

In this Section I describe my own trivial chord embedding - *Multi-hot*. I create this baseline model to compare with other embeddings. In Section 4.1 I justify the use of the *stat-of-the-art* algorithms described in Section 2.2 in chords progression embedding problem, underlining the analogies between word and chord sequences.

3.3.1. Multi-hot

In first experiments I use one of the simplest possible embedding which is my own baseline model - *Multi-hot* encoding (see Definition 1.8). To transform chord from symbolic representation with *Multi-hot* we need to define chords components. To get proper numeric chord components (explicit representation 1.7) we need to transpose chord type components (using Table 3.1 - Chord Types) by interval from base note. For example: chord DMaj^7 (stored as ^7) has 4 components: $(0, 4, 7, 11)$ (see Table 3.1 - Chord Types), root note D is 2 semitones higher than base note C (see Table 1.1) - Intervals). I transpose chord components by interval from base note C, which gives in this particular example vector: $(2, 6, 9, 13)$. Explicit representation of chord in definition 1.4 shows that it can be held as a numerical vector, with values $n \in \mathbb{N}$ such that $0 \leq n \leq N$. I have assumed 0 - the lowest note as C and N is the highest possible note. To calculate N we need:

- the highest possible extension, which is *major thirteenth*, that gives us 21 semitones,
- highest pitch class: B that is 11 semitones higher than C (see Table 1.1 - Intervals)

That gives us a range $0 \leq n \leq 32$ of 33 possible components of a chord. To create *Multi-hot* encoding I follow algorithm described in Definition 1.8. This is the first embedding that I test. Its main advantage is that *Multi-hot* vectors representing chords are easy to interpret. Each bit of this representation corresponds to a certain note on a piano keyboard. It is a binary encoding and each of the features can take one of only two values, which makes it not very effective representation. Relation of type is easy to extract because it is only distances between the following ones in vector representation, but there are more other complex relations, that comes from context of the real chords sequences. These features can be defined based on music harmony theory, but we cannot guarantee that these features will be suitable for chords generation problem, and this approach is not well justified in representation of that kind of data. *Multi-hot* representation is sparse, because median chord polyphony is equal 5 (see Figure 3.7) and length of encoded vector is 33 which gives around 80% of zeros in each chord encoding.

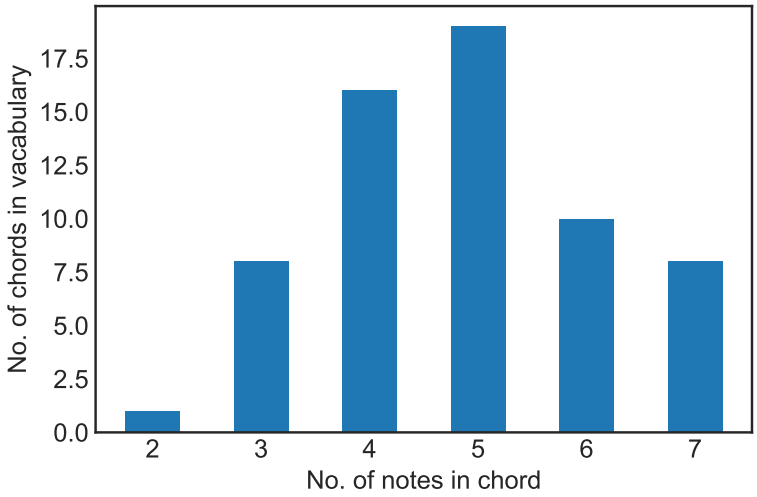

Figure 3.7: Polyphony of chords distribution

To compare *Multi-hot* and others embeddings available in *Gensim* library (see Appendix 5.3 for link), I have also implemented it in this API. This allowed me to test the models simultaneously in a single pipeline.

3.3.2. Word2Vec

Word2Vec is an embedding that takes into account contextual dependencies between words. I test it because these dependencies in chords progressions are very strong. Some chords occurs in context of a given one often than others. In jazz harmony there are constructions as *Tritone Substitution* where some chords may be substituted by others (see Definition 1.5). Using embedding that exploits context of a chord in sequence is very promising.

3.3.3. FastText

In this work I also test *FastText* because its main deference from *Word2Vec* - *n-grams* is also justified in chords progressions. As I have explained in Section Data 3.2 (see Table 3.1) there are over 60 types of chords. Some types differ only in their higher components. For example chords $D^{7\sharp 9}$ and $D^{7}(^{\sharp 5}_{\sharp 9})$ differs only in 2 notes. Their role is the same and they may be used as a substitution in some cases. *FastText n-grams* are prepared to detect these similarities using *bi-grams* of there chords are (D7, 7♯, ♯9) and (D7, 7♯, ♯5, 5♯, ♯9). They have two of their *bi-grams* in common. That means the model will treat them as more similar. This approach has it downside of treating not related chords as similar in some cases (e.g. $D^{7\sharp 9}$ and $F\Delta^{\sharp 11}$).

3.3.4. Testing Embeddings

To test embeddings I create a test set of analogies. This set consist of over 700 testing examples. Each of them have structure $a \quad b \rightarrow c \quad d$, where a and b are in the same kind of relationship as c and d. For example *woman, queen → man, king.* There is the same relation of "royalty" in both pairs, thus this is an analogy. This method is described precisely in [22] by Mikolov et al. Authors use this test on words in different forms, singular and plural. In my case I group test in six categories (see Table 3.2), each corresponding to a different kind of relation.

Algorithm 2: Analogy test

Data: a, b, c, d

Result: Evaluation score - percent of analogies that are close enough to original value

1 INITIALIZATION:
  Find numerical representations of all chords in vocabulary using embedding model that this metric is calculated for. score := 0
2 **for** *analogy in TestSet* **do**
3 a,b,c,d := analogy // extract 4 words from line
4 A,B,C,D := numRepr(a,b,c,d) // get numerical representations using model
5 V := B-A // vector from point A to B
6 D* := C+V // calculated analogy value D*
7 **if** *D in mostSim(D*, 5)* **then**
8 score += 1 // if real value of D is close (top 5 closest points) to D*
9 **return** *score/size(TestSet)*

Name	All Possible	No. of Analogies	Example	Description
root change	90k	132	*C7 D7 F7 G7*	Using the most popular chord type '7' (Dominant 7 chord)
chord type change	500k	132	*C-7 C^7 D-7 D^7*	Checking if direction of chod type change vector is preserved
V-I progression	132	132	*C7 F^7 D7 G^7*	The most common and important music chords pair
II-V progression	132	132	*C-7 F7 D-7 G7*	Part of the most important jazz chord sequence
V/V-V progression	132	132	*C7 F7 D7 G7*	Other important jazz chords pattern
less common progression	5M	132	*F-7 Db7b9 Ab-7 E7b9*	Random chord type and interval combinations

Table 3.2: Analogies tests

Each test is created combining root notes (C, D, E etc.) and chord types (7, ^ 7, -7, etc.) Some of the tests results in numerous combinations, so I randomly subsample them. In tests: V-I progression, II-V progression, V/V-V progression number of all possible analogies is 132 (12 x 11). In order to have all test equally impact the final score the size of each test analogies should be as similar as possible. That is why I have reduced number of analogies in all test to 132. In each test I perform algorithm 2 to calculate the mean score of analogy test [22].

3.4. Recurrent Models

In this Section I describe recurrent models that I test in chord generation problem. This task is a regression problem. Network predicts the continuous numerical vector representing a single chord. The input of the network is a sequence of numerical vectors, representing chords as well. I train the model using sequences of chords that are sampled from real jazz songs. First n chords are used as an input and $n + 1$ chord is used as a true value. That way the model learns to generate chords that can be used as a extension of the input sequence. Just like the training data where input and output are part of the same continuous sequence. I use three recurrent layers to build the models: `SimpleRNN` [25], `LSTM` [12] and `GRU` [5], all implemented in `Keras` library. In this work I study both shallow and deep architectures.

3.4.1. Shallow Models

I create three shallow models, each consisting of one recurrent layer. The `size` of each layer is equal to `output_size` which denotes the number of features in a single embedding vector. Parameter `input_shape` denotes the size of an input data (`[time_steps, features]`), where `time_steps` is number of chords in the sequence that I predict from. The `features` parameter is a size of the embedding, same as `output_size`. Default value of `activation` parameter is `'tanh'`, which denotes the hyperbolic tangent, that ranges from -1 to 1. Figure 3.8 shows the architecture of these models.

```
SimpleRNN_shallow = Sequential([SimpleRNN(output_shape, input_shape,
                                          activation)])
LSTM_shallow = Sequential([LSTM(output_shape, input_shape, activation)])
GRU_shallow = Sequential([GRU(output_shape, input_shape, activation)])
```

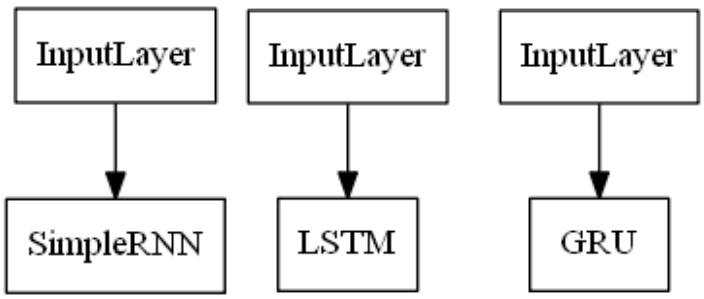

Figure 3.8: Shallow models architecture

3.4.2. Deep Models

In deep models I introduce `Dropout` and `Dense` layers. First two consist of one recurrent layer. I set the `size` parameter of the recurrent layers to 32. `Dense` layer as the output layer gives this models more complexity. `Dropout` layer is a regularization mechanism. With `rate=0.2` `Dropout` layer removes 20% of connections between adjacent layers, preventing overfitting. This method is applied only in training procedure, no connections are removed during validation [29]. Figure 3.9 shows the architecture of these models.

```
GRU32 = Sequential([
    GRU(size=32, input_shape),
    Dropout(rate=0.2),
    Dense(size=output_shape, activation),
])

LSTM32 = Sequential([
    LSTM(size=32, input_shape),
    Dropout(rate=0.2),
    Dense(size=output_shape, activation),
])
```

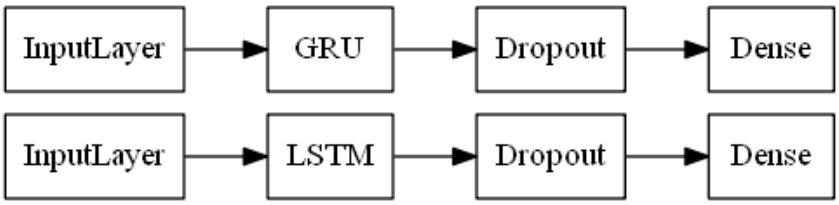

Figure 3.9: Deep models architecture

Another two models consists of one additional recurrent layer. Another difference is that first recurrent layer has parameter `return_sequences` set as `True`, which means, that each of the cells from that layer will return values while unfolded. That makes the following layer will work on sequences as well. Higher complexity of these models have been regularised by `Dropout` layer with `rate=0.4`. Also I omit `ShallowRNN` layer, because of its lower performance. Figure 3.10 shows the architecture of these models.

```
GRU3264 = Sequential([
    GRU(size=64, input_shape, return_sequences=True),
    GRU(size=32),
    Dropout(rate=0.4),
    Dense(size=output_shape, activation),
])
LSTM3264 = Sequential([
    LSTM(size=64, input_shape, return_sequences=True),
    LSTM(size=32),
    Dropout(rate=0.4),
    Dense(size=output_shape, activation),
])
```

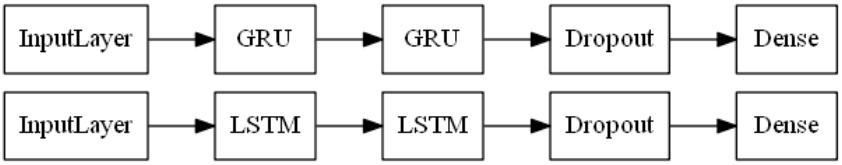

Figure 3.10: Deep models with double recurrent layer architecture

3.4.3. Configuring Models

I use state-of-the-art Adam [14] algorithm to optimize a Stochastic Gradient Method using adaptive estimation of first and second-order moments. I use is MSE (mean-squared-error) as a loss function. It is often used in regression problems [3]. In this case continuous embeddings representation makes the generation problem a regression that takes input sequence and expects continuous vector at the output. Additional metric that I have used is MCD (mean-cosine-distance), that has an application in embeddings manipulations [18]. This metric measures distance between two points just like MSE, but not in terms of euclidean distance, but in terms of angle between vectors formed by them with coordinate centre. This metric may be a better approximation of the models performance, because in high dimension spaces the euclidean distance tends to grow fast, and becomes a less relevant measure than the angle between vectors [18]. In order to implement this metric and use it during training `Keras` models I create `Metric` class and included it in model during `fit()` method by parameter `callbacks`.

```
class Metrics(Callback):
    def __init__(self, tr_data, val_data):
        self.validation_data = val_data
        self.train_data = tr_data
    def cosine_distance_sum(self, A, B):
        cos_dist = []
        for a,b in zip(A, B):
            nom = 1.0 - np.dot(a, b)
            denom = np.linalg.norm(a)*np.linalg.norm(b)
            cos_dist.append(nom/denom)
        return sum(cos_dist)
    def mean_cos_dist_sum(self, X_vali, y_vali):
        y_pred = model.predict(X_vali)
        return self.cosine_distance_sum(y_pred, y_vali)/y_vali.shape[0]
    def on_train_begin(self, logs={}):
            self.mean_cos_dist = []
            self.val_mean_cos_dist = []
    def on_epoch_end(self, epoch, logs={}):
        x_train = self.train_data[0]
        y_train = self.train_data[1]
        score = self.mean_cos_dist_sum(x_train, y_train)
        x_val = self.validation_data[0]
        y_val = self.validation_data[1]
        score_val = self.mean_cos_dist_sum(x_val, y_val)
        self.mean_cos_dist.append(score)
        self.val_mean_cos_dist.append(score_val)
        print( f"epoch{epoch},loss{logs['loss']},val_loss{logs['val_loss']},
            mean_cos_dist{score.item()},val_mean_cos_dist{score_val.item()}"
        )
    def on_train_end(self, logs={}):
        self.mean_cos_dist = {
            'mean_cos_dist': self.mean_cos_dist
        }
        self.val_mean_cos_dist = {
            'val_mean_cos_dist': self.val_mean_cos_dist
        }
    def get_train_data(self):
        return self.train_data
    def get_val_data(self):
        return self.validation_data
```

4. Results

4.1. Embeddings Comparison

To test and compare embeddings models I use analogy test described in Section 3.3.4. I will refer to it as *Evaluation score* on the plots below. In the first test I compare the size of the embedding (size of the latent vector). In case of *Multi-hot* embedding this size is fixed (see Definition 1.3 - Interval) and would not be tuned. I test embedding size only on *Word2Vec CBOW*, *Word2Vec Skip-Gram* and *FastText* models. First experiment on range of 1 to 96 with step equals 4 (1, 5, 9, 13, ...) see Figure 4.1

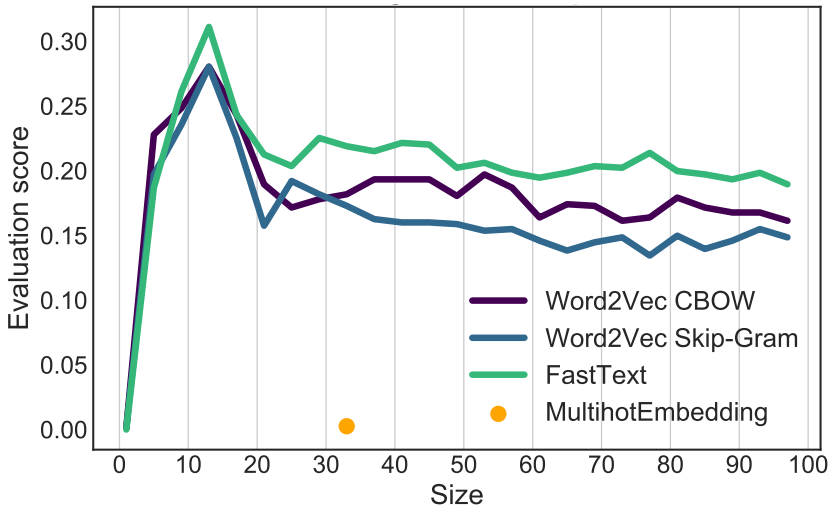

Figure 4.1: Embedding size comparison with step of 4.

The best results of word embeddings are around 15, all greater embedding sizes tends to lower the score. The results show that *Multi-hot* embedding has noticeably lower value than other embeddings. With size 33, the achieved score is 0.25%. In order to find the best model I perform additional tests with all possible sizes in the range of 1 to 34 (see Figure 4.2).

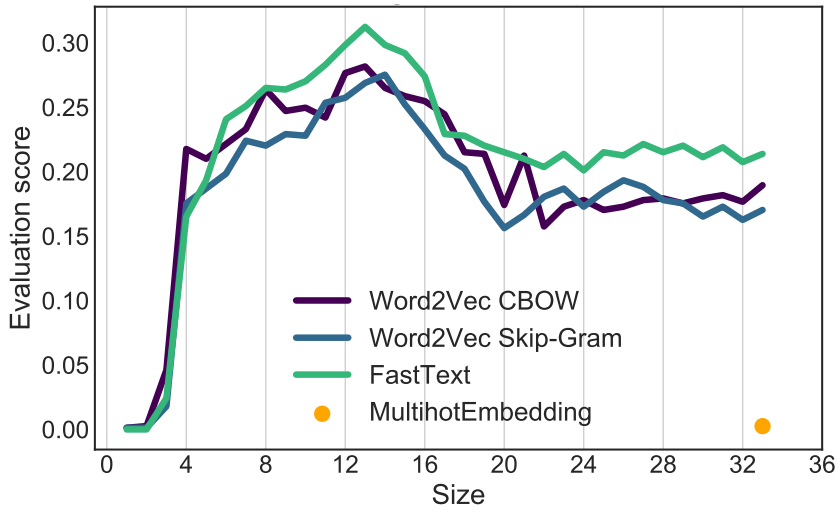

Figure 4.2: Embedding size comparison with smaller size step.

Word2Vec with *CBOW* architecture scored 28.17% with embedding size of 13. *FastText* is slightly better with a score of 31.24% and the same embedding size of 13. *Word2Vec* with *Skip-Gram* architecture scored 27.53% with embedding size of 14. Both of the tests are performed using window parameter with size of 2. The results of this test tell that all three tested embeddings perform the best within the size range of 10 to 15. Increasing the size of the embedding only lowers this score. In further experiments I use embeddings with sizes described at the beginning of this paragraph. Knowing the optimal values of each embedding size I perform comparison of different window sizes within tested models (see Figure 4.3).

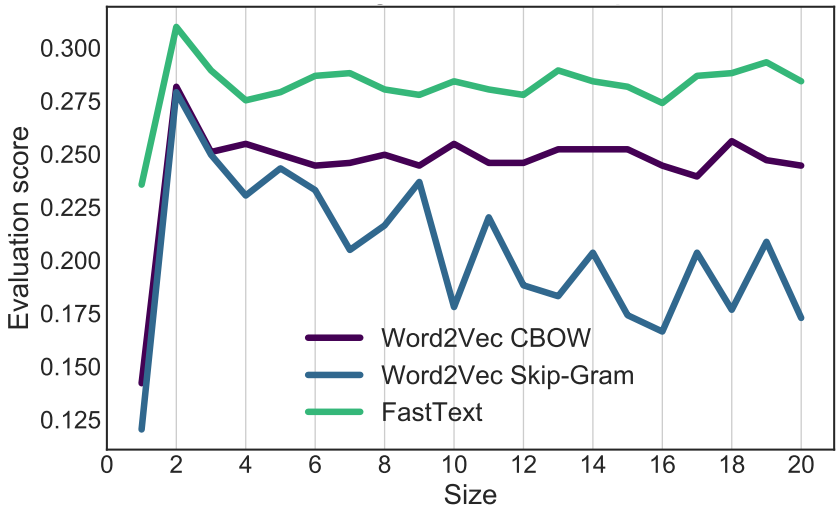

Figure 4.3: Embedding window comparison

In the range of 1 to 20, all embedding models scored the best result with window of size 2. In summary the best embedding according to analogies test (see Section 3.3.4) is *FastText* with latent vector of size 13, and window of size 2.

In conclusion word embeddings perform much better than *Multi-hot* baseline model. I present the high-level relations in embedded space in visualisations in the following Section. We can see them only in word embeddings. Larger size of embeddings surprisingly do not increase the performance in analogy tests. It is interesting that all three word embeddings followed the same pattern on Figure 4.1 and 4.2. The optimal point is almost the same in all embeddings (size equals 13 or 14). Sizes larger than 20 slowly decrease the evaluation score on analogy tests. One of the reasons of that behaviour is small size of the vocabulary of jazz chords (62 types $\times$ 12 keys equals 744 possible chords).

4.2. Embeddings Visualisation

To visualize the embeddings of chords I decided to plot all 744 chords in 2D and 3D space. More than 3 dimensions are hard to plot and analyze visually. Each of the embeddings has more than 3 dimensions, *Word2Vec CBOW* and *FastText* have the size of 13 and *Word2Vec Skip-Gram* has the size of 14. To reduce the dimensionality and plot the embedding on a 2D space I use *t-SNE* technique. Formal interpretation of the *t-SNE* results is complex, but this dimensionality-reduction algorithm can find hidden structures in multidimensional space, which other algorithms cannot [31]. In Section 4.2.6 I present 2D visualisation using *PCA* algorithm to give another perspective on the embedded space. Displaying labels of all chords would certainly not brighten up the image of the embedding, therefore I plot only certain groups: *Dominants, Majors, Minors, and II-V-I Progression Chords*. The chords in each group are strongly dependent on each other, because of their type or progression that they belong to. Below I present comparison of each chord group and models.

4.2.1. Word2Vec Continuous bag of Words

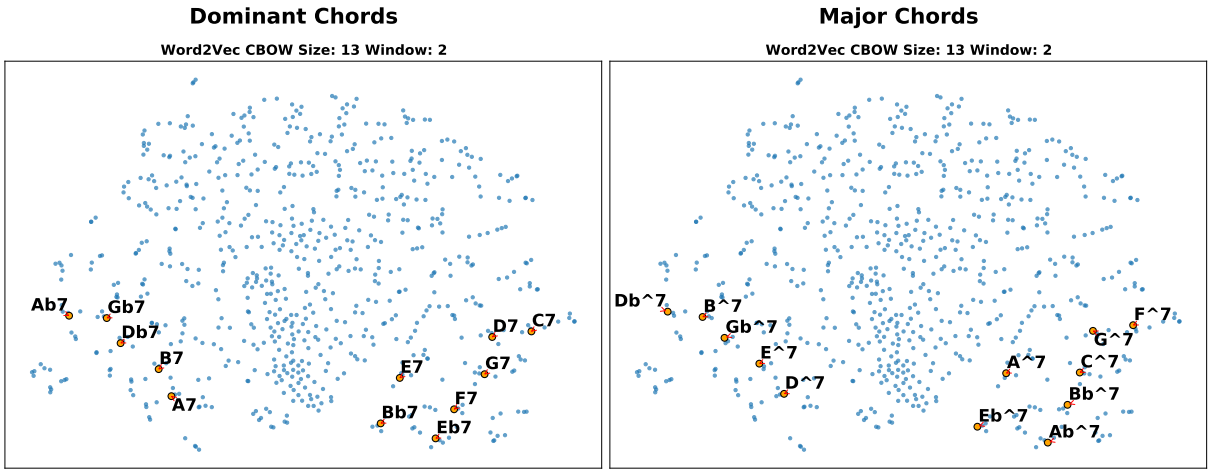

Figure 4.4: Word2Vec CBOW dominant and major chords embedding visualisation

In Figure 4.4 we can see that Dominant and Minor Chords are separated into two groups. The criterion of that split is popularity. Chords in groups on the right side have roots that occurs in the chords dataset often, and chords on the left side have less common roots (see Figure 3.5 with distribution of chord roots). To see the relations between the Major and Dominant chords placement I plot them together (see Figure 4.5).

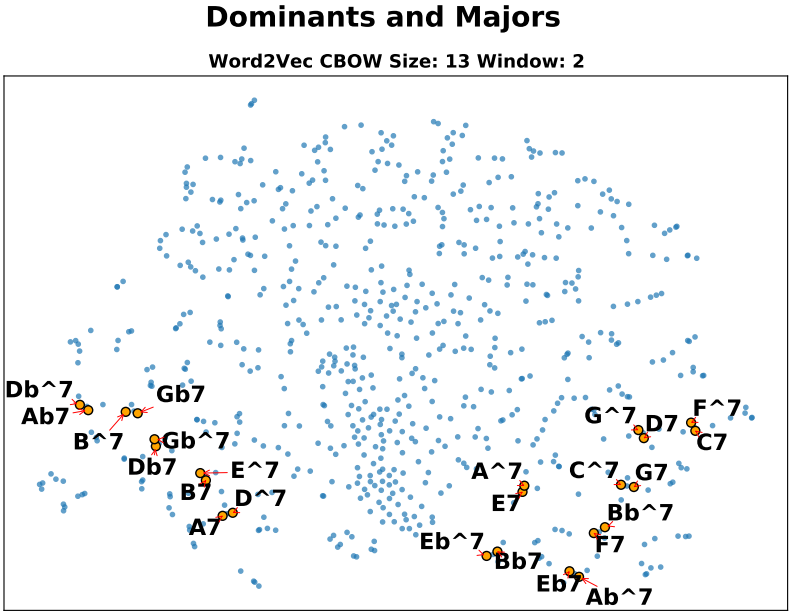

Figure 4.5: Word2Vec CBOW dominant and major chords embedding visualisation

Dominant and Major chords occur in pairs, that are close to each other. These pairs, however, are not accidental. The chords that form a pair belong to the V7 - I^7 progression (the most popular progression in music). In Figure 4.6 I plot chord that belongs to the most common jazz progression II-7 - V7 - I^7.

As we can see in Figure 4.6 these chords are also grouped. For example triplet D-7 - G7 - C^7 is placed in the right size of the plot.

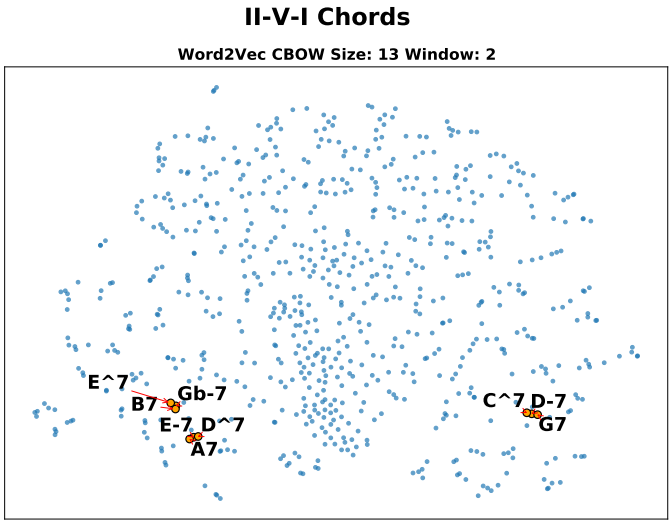

Figure 4.6: Word2Vec CBOW II-V-I chords embedding visualisation

4.2.2. Word2Vec Skip-Gram

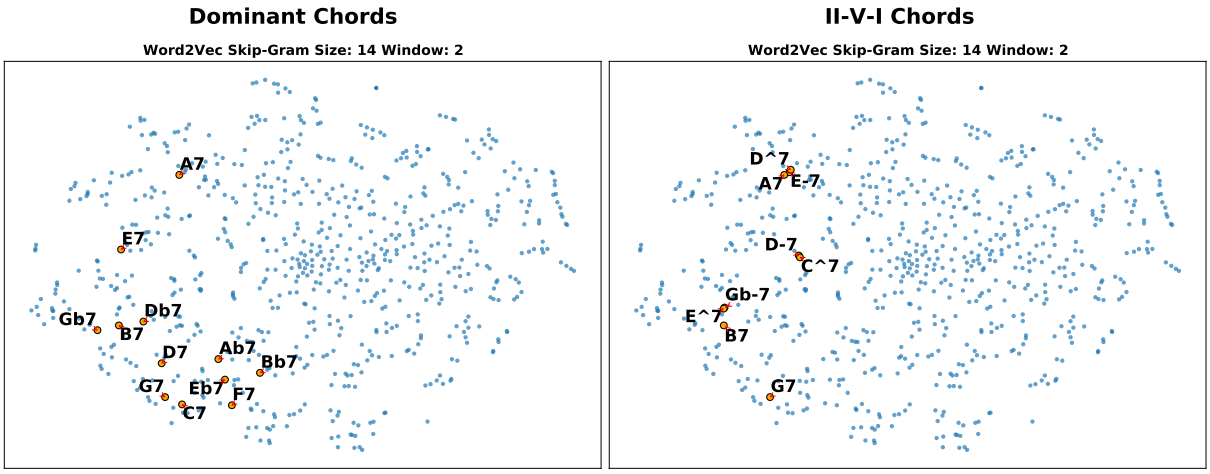

Figure 4.7: Word2Vec Skip-Gram dominant and II-V-I chords embedding visualisation

The Skip-Gram architecture does not have such precise structures, but still shows these relationships (see Figure 4.7). Not all II-V-I progression chords are correctly grouped.

4.2.3. FastText

FastText architecture (size = 13, window = 2) performs the best in analogies test. The latent space places Dominant chords in a specific area of the reduced embedding. In II-V-I progression it is visible that corresponding chords are also grouped. We can see some circular structures in the left side of the plot. Each of them contain 12 points. This may suggest that they contain some specific chord types.

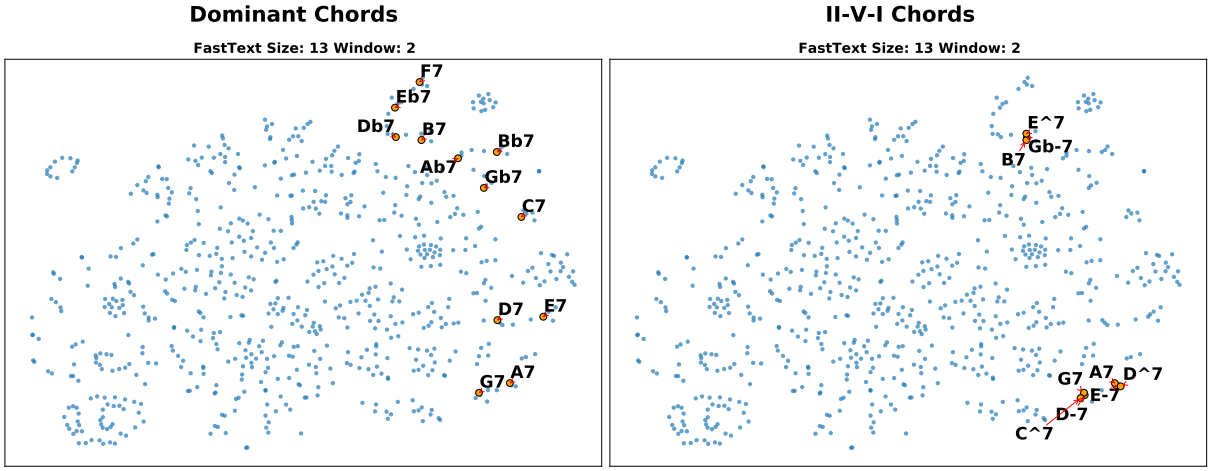

Figure 4.8: FastText dominant and II-V-I chords embedding visualisation

4.2.4. Multi-hot

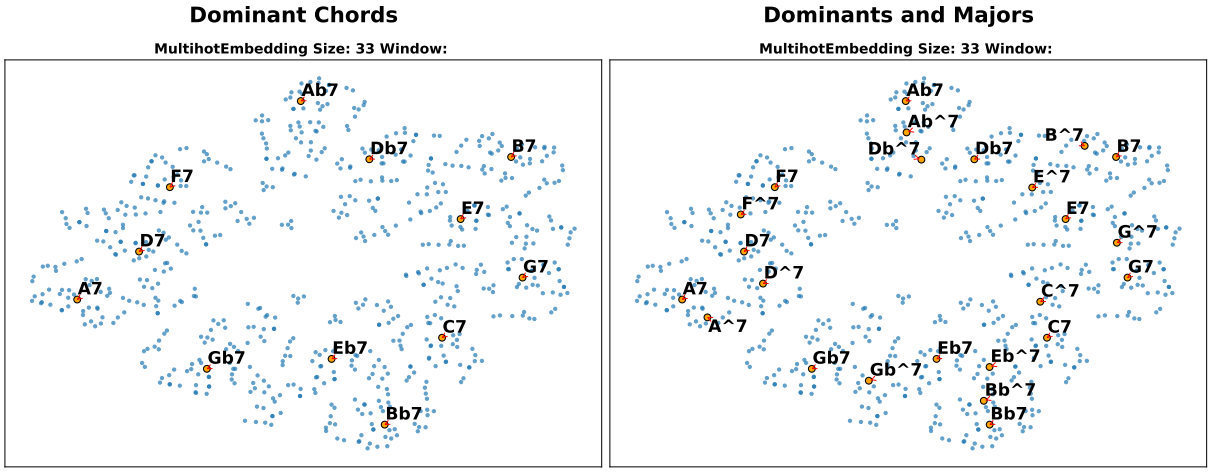

Figure 4.9: Multi-hot dominant and majors chords embedding visualisation

Multi-hot is different from other embeddings, because it is a binary vector. Every point is placed on a corner of 33D hyper-cube, making it a sparse representation. That may be the reason of unusual arrangement with void in the middle (see Figure 4.9). Dominant chords are placed evenly across the space, and the corresponding major chords are not their corresponding tonics. Instead of pair F7 Bb^7 there is F7 F^7. In MultiHot representation F7 and F^7 are closer than

corresponding V-I progression. That is why this important V to I relation is not present in both full and reduced space of a *Multi-hot*.

4.2.5. Word2Vec Skip-Gram Larger Model

In Section 4.1 larger sizes of embedding resulted in lower score from analogies tests. Nevertheless, I visualize larger size embeddings and it turned out that they have other very interesting properties.

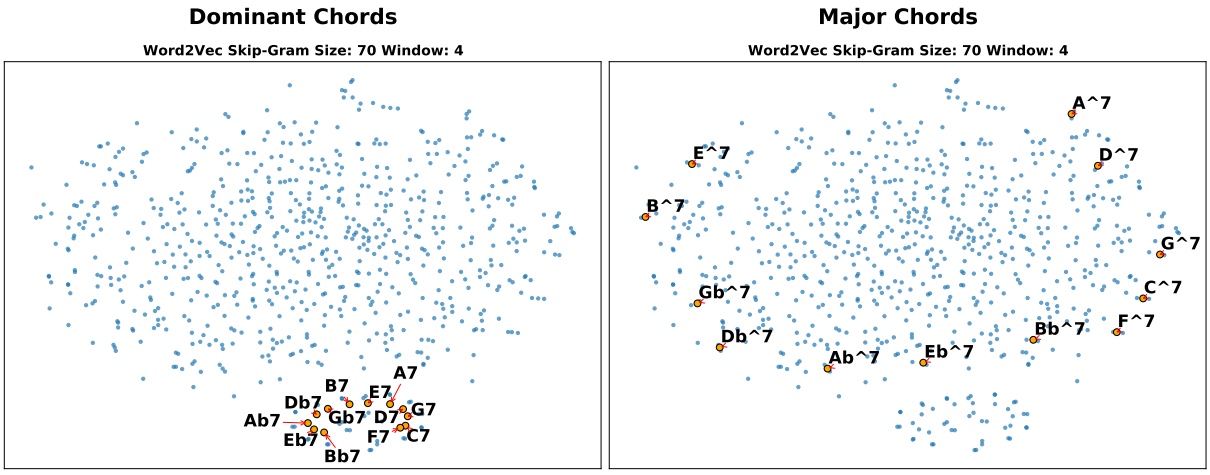

Figure 4.10: Word2Vec Skip-Gram size 70 dominant and majors embedding visualisation

The Figure 4.10 shows the *Word2Vec Skip-Gram* model (size = 70, window = 4). Both Dominant and Major chords are evenly distributed on a circle, but that is not the only good property of that embedding. Chords occurs in perfect order just like in a circle-of-fifths (see Figure 4.12). The sequence A, D, G, C, F, Bb, Eb, Ab, Db, Gb, B, E is a sequence of notes distant from each other by an interval of a perfect fifth see Table 1.1.

4.2.6. Tensorflow Embedding Projector

To visualise structures present in embedding representation I also use Tensorflow tool - Embedding Projector [28]. It allows to load word embeddings and metadata to describe each word, and visualise a 2D or 3D plot using *PCA* or *t-SNE* algorithm. Thanks to its visualisation features I include snapshots with interesting structures that are present in embeddings that I have created.

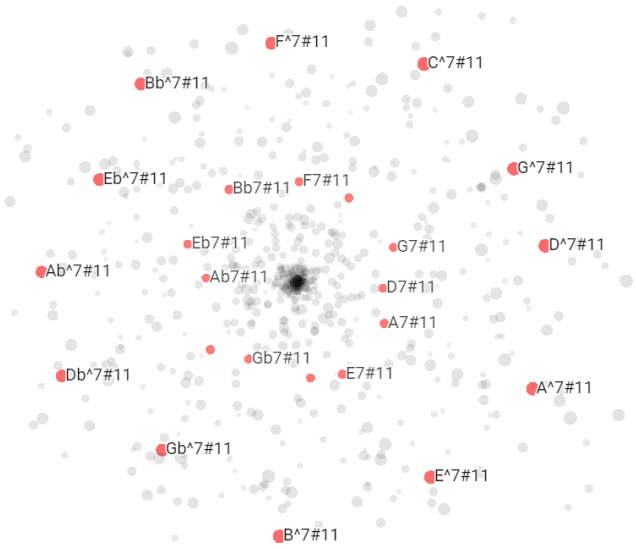

Figure 4.11: Word2Vec CBOW size 13 - ♯11 chords

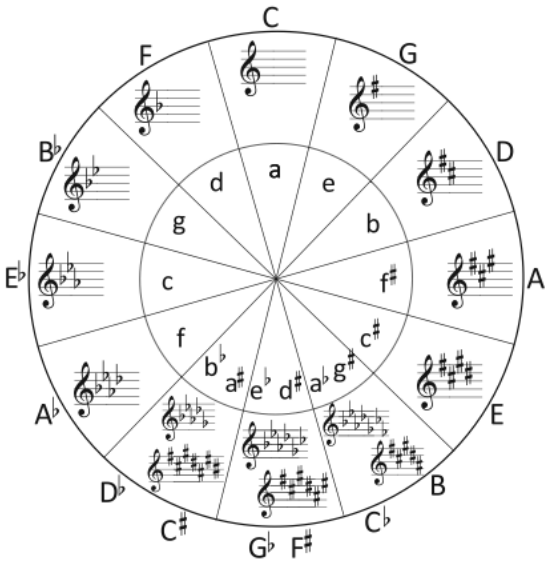

Figure 4.12: Circle of fifths [11]

Both ^7♯11 and 7♯11 chords (even they have very different role in chord progression) tends to have similar structure in embedded space (see Figure 4.11) It is worth mentioning that they are in circle-of-fifths order, even the size of embedding is small - 13. This structure is not visible in my previous visualisations, which makes Embedding Projector a useful visualisation tool.

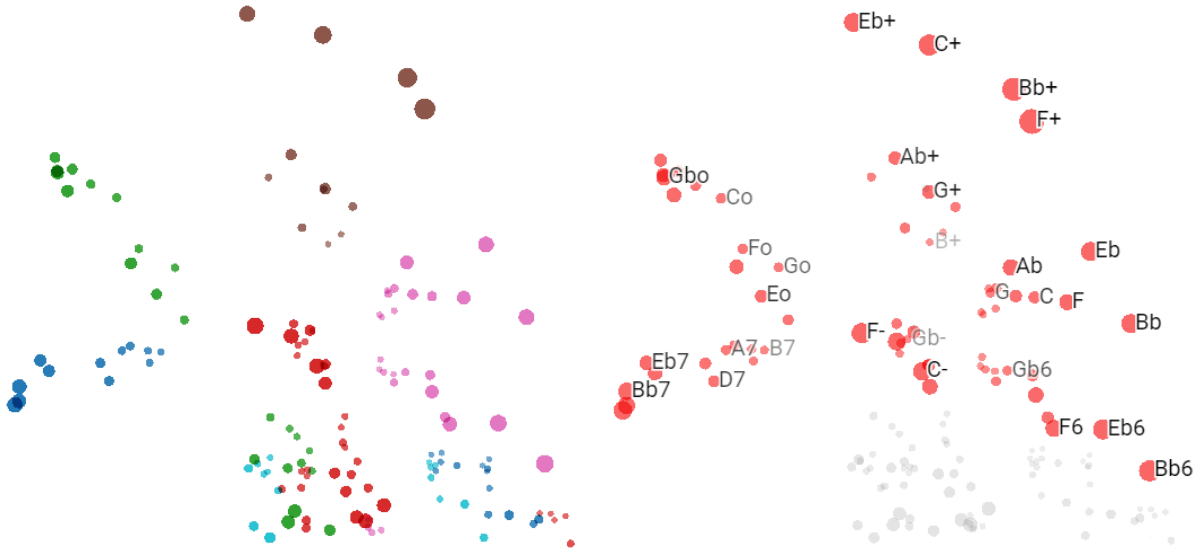

Figure 4.13: Visible chord classes separation Word2Vec Skip-Gram size 14

In Figure 4.13 on the left side we can see how different classes of chords (Dominant, Major, Minor, etc) marked by color are grouped. After selecting them and showing their labels we can see what chord class corresponds to each of the structures.

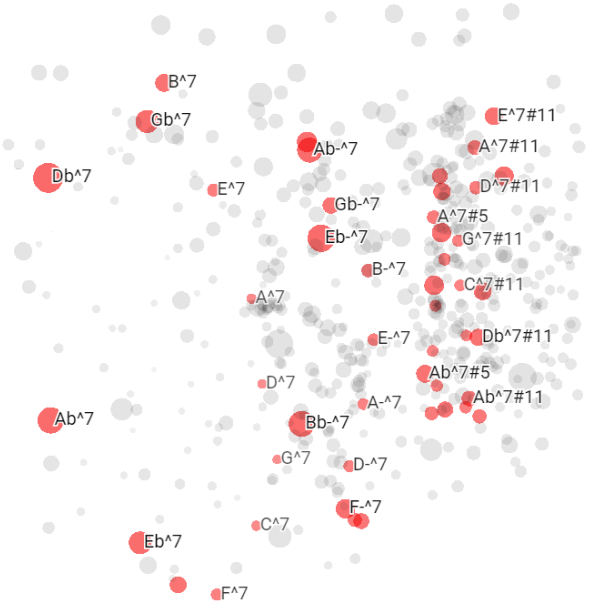

Figure 4.14: Major sevenths - FastText 13

Figure 4.15: Altered dominants FastText 70

Figure 4.14 shows how major seventh chords are arranged in relation to each other. Extracted from chaotic structure inside the embedding cloud of points these chords form circular structures oriented in the same plane. All previous visualisations are performed using PCA algorithm built in Embedding Projector. To plot embeddings in the following models with bigger latent vector size - 70, I have used T-SNE and UMAP algorithms from Embedding Projector.

Visible group of chords marked on Figure 4.15 shows that embeddings are capable of detecting similarities on a higher level, because every chord in this group is Altered Dominant. They are visibly separated from other chords, and grouped close to each other. It is possible that this behavior results from the nature of those chords in jazz music. Altered Dominants can be used as substitution of each other. A common practice is using long sequences of these chords called secondary dominants [24]. Altered dominant is often used in modal interchange and changing keys [24].

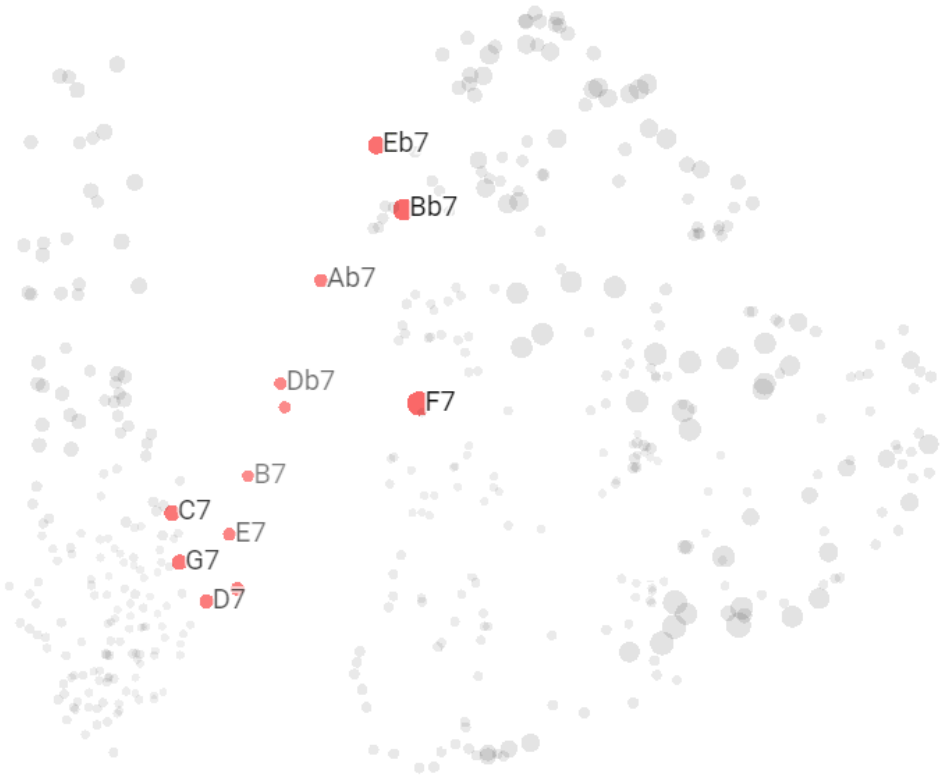

Figure 4.16: Dominant spiral Word2Vec CBOW size 70

Dominants plays the important role of transient chords in harmony, often linking different tonal centres. In Figure 4.16 created using UMAP algorithm Dominant Chords formed a spiral connecting two big groups of chords, which is a certain analogy to their role in music, that connects different key centres, allowing to link two different parts of a song [24, 1].

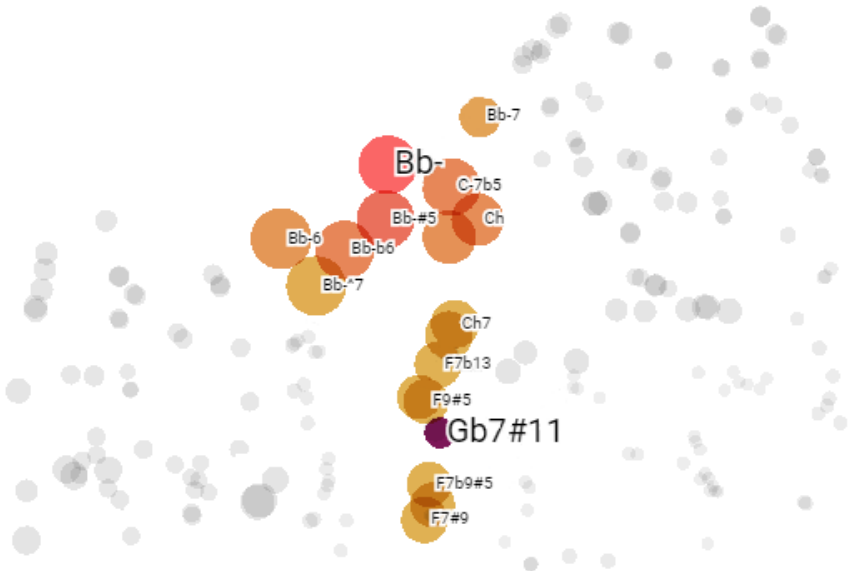

Figure 4.17: Bb Minor centre Word2Vec CBOW size 70

To show that chord type is not the only factor in all those embeddings, the last example - Figure 4.17 show the most important chords from key B♭Minor. I have to zoom the visualisation to the maximal limit to select all tightly spaced chords. In this group we can observe a big variation of extensions and even substitutions of commonly used chords in that tonal centre. I have selected B♭- chord as a point to compare the similarity with other points from selected group. Colors represent similarity to B♭- chord: red - very similar, orange - similar, purple - less similar. All the interactive Embedding Projector visualisations are available in Appendix 5.3.

4.3. Models Comparison

In this Section I present the results of the recurrent models in terms of chord generation problem. I use models architectures described in Section 3.4 - Recurrent Models. To compare the models, I perform a series of tests measuring the performance of each model, help to select the appropriate sets of hyper parameters and decide what size of the input sequence results in the best prediction performance. I describe several difficulties encountered during the training and how I manage to overcome them.

4.3.1. Choosing Embedding

In Section 4.1 the best embedding according to the analogies test is *FastText* with size 13, and window size 2. In this Section I compare embeddings once again but this time in generation problem and decide which one to use in further recurrent model analysis. I compare three best architectures from Section 4.1: *Word2Vec CBOW* size 13, *Word2Vec Skip-Gram* size 14, *FastText* size 13, with window size 2. As a recurrent model I use SimpleRNN shallow network and sequence with size 3 to predict from. Size 3 means, that I will use three chords in order to predict the fourth chord from sequence of four chords that I sample from the data set.

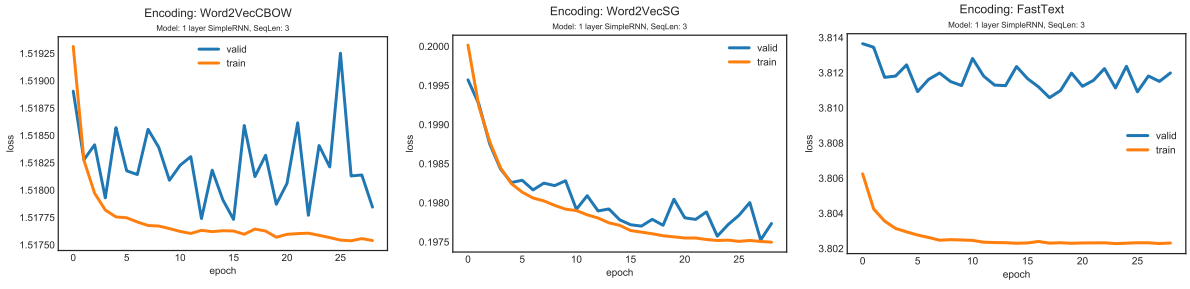

Figure 4.18: Embedding comparison

The *Word2Vec Skip-Gram* model have the lowest values on both train and valid set. *FastText* embedding underfits the problem with ten times worse performance than Skip-Gram. *Word2Vec*

CBOW is very unstable on validation set.

4.3.2. Shallow Networks

I assume that *Word2Vec Skip-Gram* embedding performed the best and preform shallow networks comparision test using this embedding.

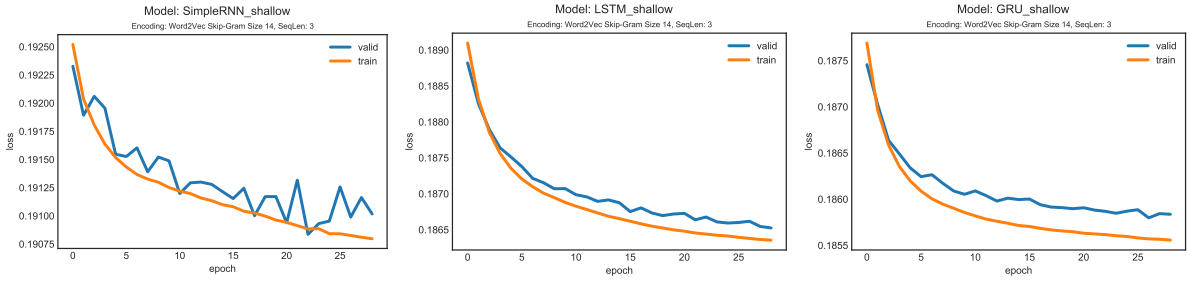

Figure 4.19: Shallow networks comparison

In this test runtime differences are more visible. 30 epochs training lasted 6 minutes for *SimpleRNN*, 9 minutes for *LSTM* and 10 minutes for *GRU*. Both on validation and test set loss results are very similar. Shallow models seem to behave more similar in this test, but LSTM loss plot on validation set is stable and the difference between validation and training loss is the smallest, thus I use it in the following test.

4.3.3. Sequence Size

A parameter that is not tested so far is sequence length. Using *LSTM* model with *Word2Vec Skip-Gram* embedding I test three sequence lengths: 3, 7 and 15. They corresponds to test in which from progressions of length 4, 8 and 16 accordingly we predict the last chord based on the previous 3, 7 and 15 chords. These lengths are not random. Most music progressions have lengths equal to a multiple of 4 (see Figure 3.6)

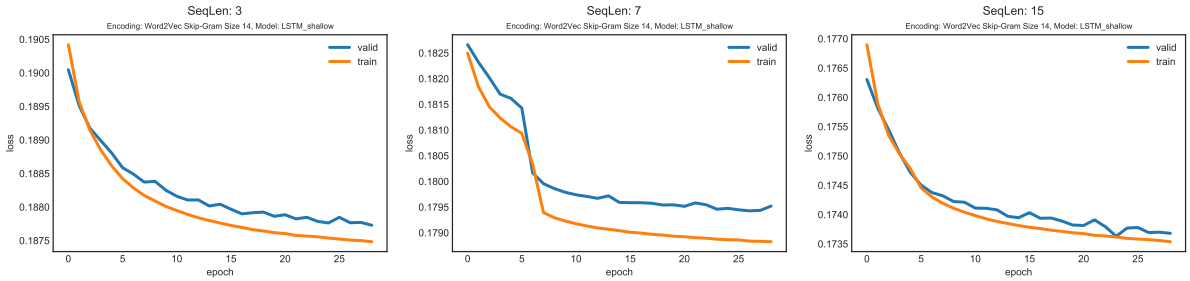

Figure 4.20: Sequence length comparison

The longer the history of a chord sequence, the wider the context on which a neural network can base its prediction. The above charts confirms that statement with the lower loss function

values which come with greater sequence length. Moreover the longer the sequence, the closer is validation loss curve to training one, meaning that these models are less overfitted [4].

4.3.4. Activation Function

After comparing the distribution of values of all embedding dimensions and the predicted vectors it turned out that the range of values in embedding is much wider that in predicted output (see Figure 4.21). It is the result of using default activation function - the hyperbolic tangent, that ranges from -1 to 1.

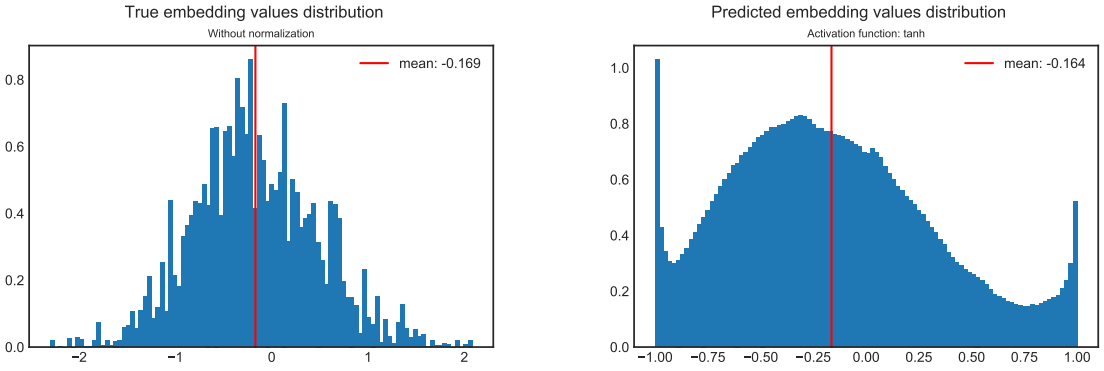

Figure 4.21: Tanh activation distribution of values

In order to make neural network capable of generating output that is not limited by activation function, I use a linear activation function which have unlimited range. Once again I perform embeddings test this time using *LSTM* shallow network and sequence length 15. As a result the range of the output values distribution change and is very similar the input distribution range (see Figure 4.22). Notice that mean value is closer when linear activation function is applied. In `Keras API` linear activation function is denoted as `None` because no additional operation on the output needs to be performed.

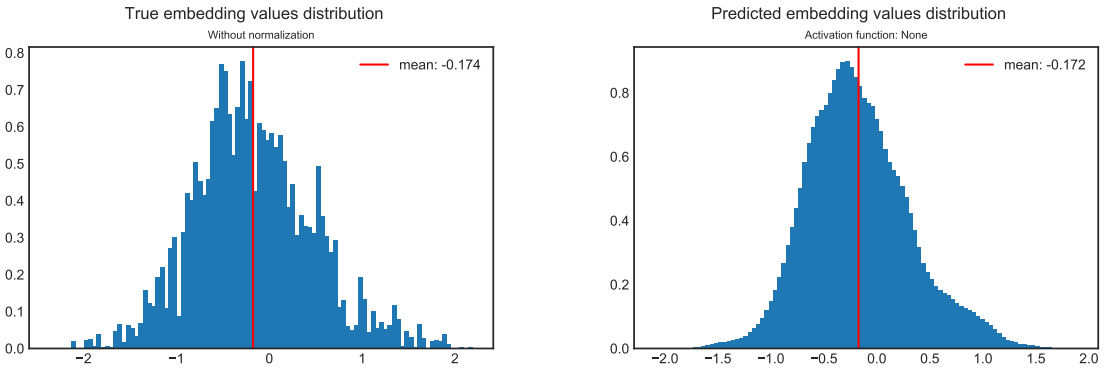

Figure 4.22: Tanh activation distribution of values

I compare embeddings results on models with linear activation (see Figure 4.23).

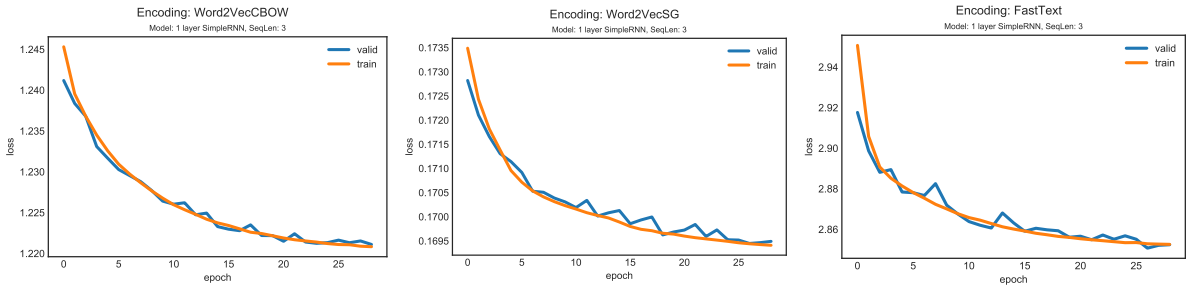

Figure 4.23: Linear activation embeddings comparision

Once again *Word2Vec Skip-Gram* embedding achieved the best results, but this time all models are more stable and not overfitted.

4.3.5. Deep Networks

In this experiment I compare deep architectures. First I compare *GRU32* and *LSTM32* architectures described in section 4.3 I use *Word2Vec Skip-Gram* embedding and sequence length equals 15.

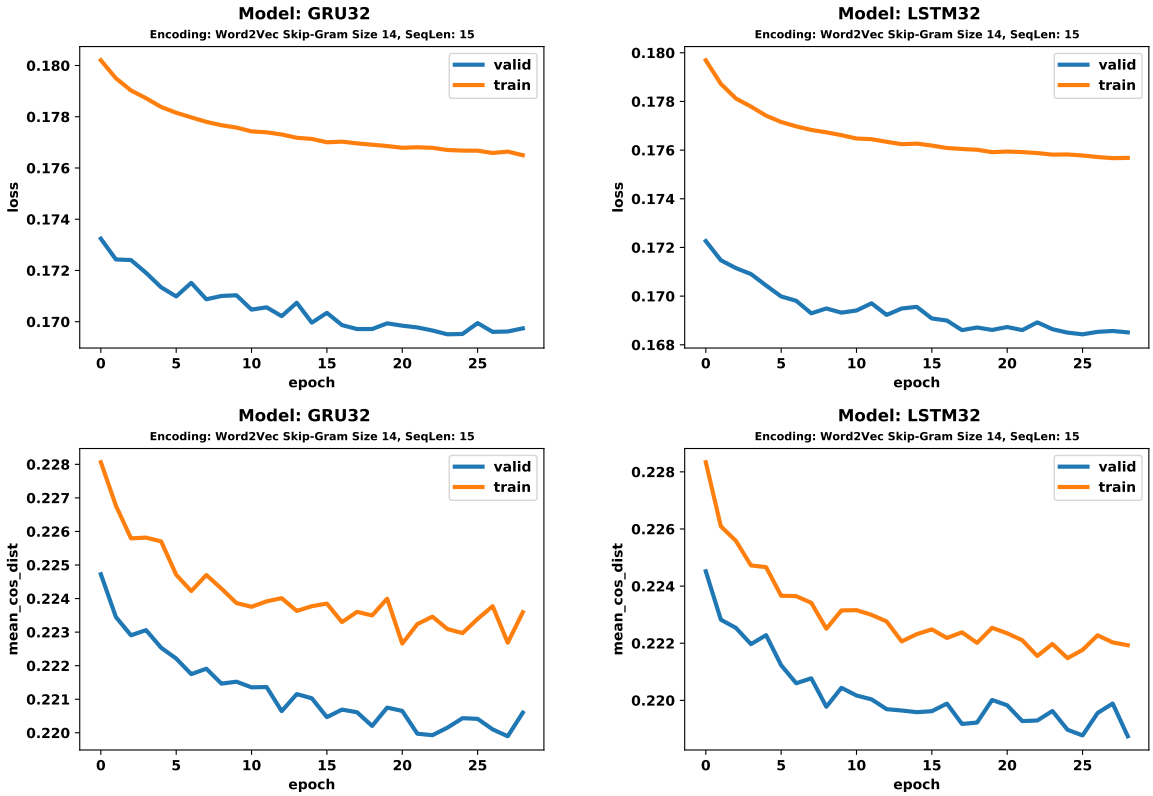

Figure 4.24: LSTM32 vs GRU32

As we can see in Figure 4.24 both architectures perform similar. The only significant difference in these algorithms is their runtime (*GRU* 8 minutes, *LSTM* 10 minutes). The main advantage of *GRU* model is its speed. It is on average 20% faster. In this experiment the validation loss curve is much lower that training one. That strange behaviour is because of using Dropout layer, that removes some of the connection between Recurrent and Dense layer. This regularisation method improves the training of the model preventing overfitting is active only during training lowering the train scores. This temporary drop in performance results in significant improvements on validation set. The problem of lower validation loss in networks using dropout is described in [29] "We reported test error of the model that had smallest validation error."

4.3.6. Final Models

The final training is performed on two models with two recurrent layers: *GRU3264* and *LSTM3264*. This time I set the training epochs number to 200, and use a `EarlyStopping` callback from *Keras* to stop training when model stops improving. I use *Word2Vec Skip-Gram* with size 14 and window 2 since it is the best embedding in previous tests and train the models on a sequence length 15, also the best value in previous test.

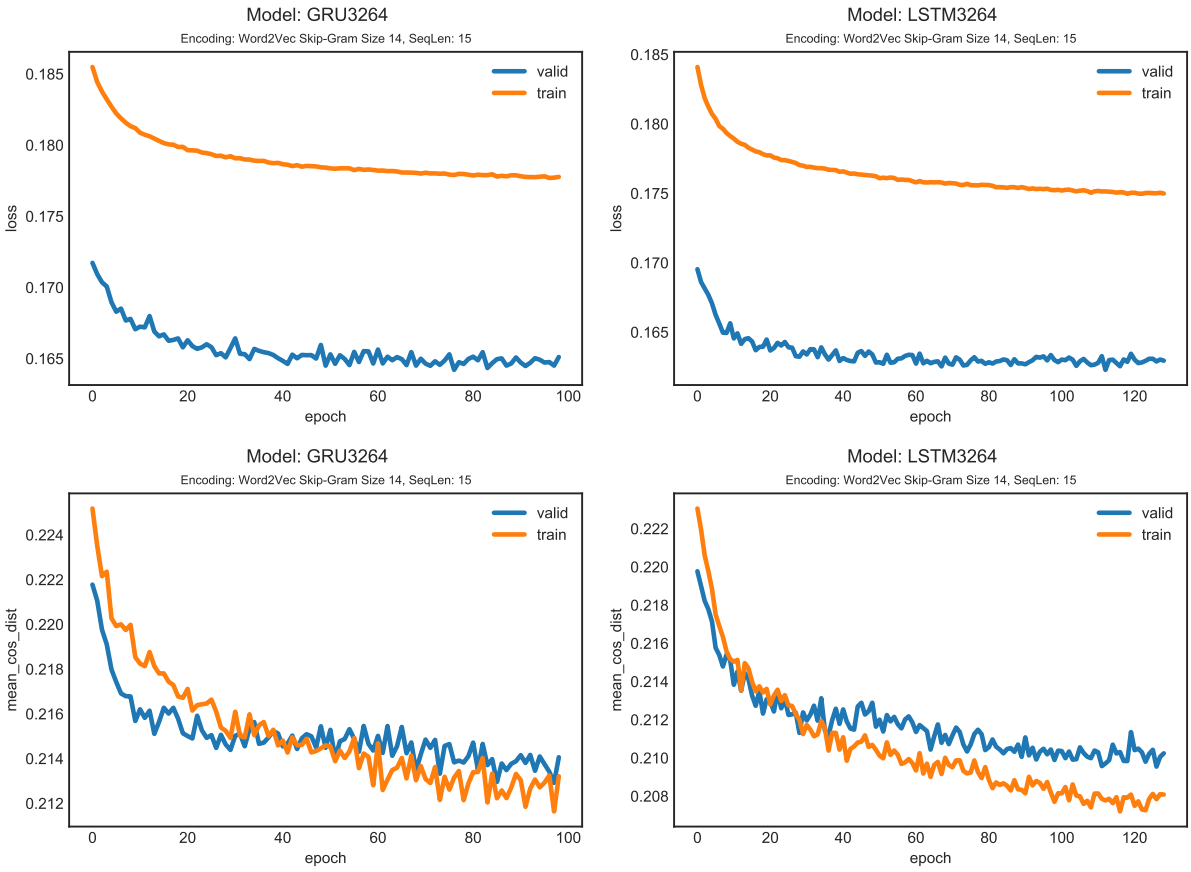

Figure 4.25: LSTM3264 vs GRU3264 final training

As we can see these models also performs similar to each other, but achieve better results that one layer models overall. *GRU* achieved the same result in smaller number of epochs and in a shorter time. I gather results from all experiments in tabular form (see Table 4.1).

Test	Model Name	Epochs	Runtime	Embedding	Seq Len	Mean Cos Dist	Loss	Params	Activation
Choosing Embedding	*SimpleRNN_shallow*	30	**595**	*Word2Vec CBOW*	3	1,96909	1,51750	351	tanh
			608	*Word2Vec Skip-Gram*		**0,25746**	**0,19770**	406	
			610	*FastText*		4,74240	3,80260	351	
Shallow Networks	*SimpleRNN_shallow*	30	634	*Word2Vec Skip-Gram*	3	0,24989	0,19096	406	tanh
	LSTM_shallow		484			**0,23719**	0,18651	1624	
	GRU_shallow		**450**			0,24259	**0,18572**	1260	
Sequence Size	*LSTM_shallow*	30	***485***	*Word2Vec Skip-Gram*	3	0,24433	0,18761	1624	tanh
			519		7	0,23677	0,17917	1624	
			549		15	**0,22757**	**0,17336**	1624	
Activation Function	*LSTM_shallow*	30	**3919**	*Word2Vec CBOW*	15	1,55270	1,21977	1404	Linear
			3942	*Word2Vec Skip-Gram*		**0,22344**	**0,16921**	1624	
			3951	*FastText*		3,73853	2,84151	1404	
Deep Networks	*GRU32*	30	**480**	*Word2Vec Skip-Gram*	15	0,21990	0,16949	5070	Linear
	LSTM32		558			**0,21874**	**0,16832**	6478	
Final Models	*GRU3264*	100	**2340**	*Word2Vec Skip-Gram*	15	0,21292	**0,16275**	25230	Linear
	LSTM3264	130	3268			**0,20955**	0,16490	33102	

Table 4.1: Results table

I perform six experiments on recurrent networks in this work. In each of them, I make a significant change compared to previous experiment. I limit the number of tested embeddings to three models that did best in the embedding experiments (see Section 4.18). In the first experiment, it turned out that *Word2Vec Skip-Gram* is performing better then other embeddings. It is worth noting that in this experiment I use the simplest model - *SimpleRNN_shallow* and the tanh activation function. This activation function limits the output range, which have negative influence on the model performance. The embedding comparison is repeated in another experiment to confirm the advantage of Skip-Gram architecture.

In the second experiment, I compare shallow networks. You can see the advantages in runtime for networks with a *GRU* layer. *SimpleRNN_shallow* network, despite the simpler architecture have the longest runtime. This is due to the lack of hardware acceleration that can be achieved in *LSTM* and *GRU* layers thanks to the *cuDNN* library and the *NVIDIA GeForce 1050Ti* graphics card I use for the calculations.

Another experiment compares how models that are trained on longer chord sequences behave. Longer sequences extends the calculation time, fortunately, it is not a linear increase. Longer sequences also improve the quality of prediction in both Mean Square Error and Mean Cosine Distance metrics.

Next experiment is to compare the quality of prediction after changing the activation function to linear. You can immediately see a significant increase in the runtime of the algorithm. It is caused by the fact that the *LSTM* layer is trained without hardware acceleration from the

cuDNN library because one of the requirements of this acceleration is that the activation function should be set as default value - *"tanh"*. In this experiment, I repeat the embedding test using the *LSTM_shallow* layer and the sequence length equals 15. The result is a better loss and mean cosine distance values. *Skip-Gram* architecture also proved to be the best embedding in all metrics, thus I use it as a final embedding, even though it performed slightly worse than *FastText* in analogies test (see Section 4.1).

In this experiment I test models with more layers: *GRU32* and *LSTM32* both models contain Dropout and Dense layers. I chose the best parameters from previous experiments: Linear activation, length 15 sequences and Skip-Gram architecture. Here *LSTM32* turned out to be better architecture, but at the expense of longer runtime.

Adding another recursive layer turned out to be a good idea, so I repeat this procedure to create the final models. *GRU3264* and *LSTM3264* models have two recurrent layers. In this experiment I also set a larger number of epochs - 200, but the Early Stopping mechanism stops the algorithms after a smaller number of epochs, 100 and 130 respectively. As a result of final experiment, I obtain two architectures that are better than any of the previously tested models. The performance of both algorithms is comparable, but *GRU* model is much faster, thus I choose it as a final model.

4.4. Chord Generation

At the beginning of this project I created a simple *LSTM* network based on *Multi-hot* embedding. It is intended to be a baseline model to compare the results of chord generation with the final model obtained using word embeddings and multilayer recurrent networks. Whole process of chords generation is different in case of that model, so I describe it separately for *Multi-hot* and *NLP* embedding based generators.[1]

4.5. Baseline Model Chords Generation

As a result of baseline model prediction I receive a vector of probabilities that and then select a threshold to get a proper *Multi-hot* binary vector. I use a threshold calculated to get an exact number of notes, in example below 4 notes. I encode *Multi-hot* representation to chord voicing, and then decode the chord symbol if it exist. Chords could be decoded using top-1 similar

[1]This section will be particularly interesting to people familiar with jazz harmony rules, and music in general, but I will explain all needed musical terminology as simple as I can.

method, but I encode voicings using `pyChord` library, because this binary representation can be a certain chord representation, so I do not need to find existing embedding in my vocabulary. It is enough to just find the symbol behind this binary representation. Below is the process of extracting chord symbol from the output of the baseline model.

```
output: [0.42, 0.41, 0.23, 0.45, 0.13, 0.58, 0.19, 0.22, 0.51, 0.14, 0.38, 0.17]
threshold = 0.42              [1 0 0 1 0 1 0 0 1 0 0 0]
get components:                        [0,3,5,8]
get chord name:                           Fm7
```

There is no way to compare this model in metrics that I use along *NLP* embedding based models because of the format of *Multi-hot* representation, so I did not included any metrics. In Figure 4.26 I present the results of a continuous generation by a trained model, where generated sequence become a source to generate the next chord. First 8-chord sequence are randomly sampled from training set.

Fm/C	|Fm/C	|C7	|Fm/CaddB-
Gm/D	|B-7/D	|Fm/C	|Gm7/D
E-	|A-7/C	|A-7/C	|A-/CaddA
Gm7/D	|C7	|F/C	|F/CaddB-
E-	|Gm/D	|Cm	|Cm7
C7	|F-+/CaddB-	|F/C	|B-/C
B-/D	|E-+	|C7	|Chord Symbol Cannot Be Identified
F7/C	|B-/D	|B-/D	|B-/C
E-m	|A♯m7/C♯	|E-m	|Chord Symbol Cannot Be Identified
Gm7/D	|C7	|F/C	|B-/C

Figure 4.26: Sample generated sequence - baseline model

For an untrained musician it is hard to analyse this output, thus I comment these chord sequences based on jazz harmony rules, and evaluate their musical quality.

There are multiple slash chords (much more than 4% in training set) that is because model generated not popular voicings, that can be only denoted as a slash chords. I use *pyChord* library to get symbolic name out of explicit vector notation. There are chords, that could not be identified by this external library. That means that my model generates a structures that cannot be interpreted as regular chords. It is worth mentioning that not all possible combination of notes are a proper chords. Some of them would sound very dissonant, and probably would not find a usage in music. There are also not many progressions like $II - V$ or $V - I$ very popular in jazz music.

Apart from the downsides, there are few good qualities of generated sequence: There is a visibly variety in chords types, roots, bass notes and even keys. Sometimes there are chord repetitions, and it is common in the real compositions.

4.6. Final Model Chords Generation

The Final model is constructed using two *GRU* layers, with sizes 64 and 32, Dropout layer with rate equals 0.4 and Dense output layer (see Figure 4.27). Embedding used to encode chords symbols is *Word2Vec* with *Skip-Gram* architecture with size 14 and window 2.

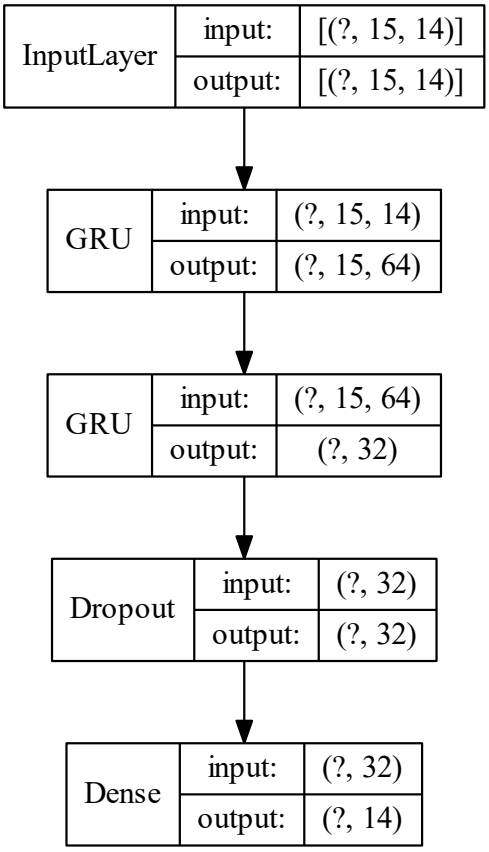

Figure 4.27: Final model architecture

I perform first generation in following manner: From test set I get sequence of 4 chords, print their symbols (last chord in brackets) print top 5 similar chords to predicted value.

(1)	E7	E7	B-7	(B-7)	[B-7	A6	E7	D6	Gb7b13	]
(2)	Ab	Ab	Db	(Ab)	[Ab	Db	Eb7	Bb7	Eb+	]
(3)	G	G	E7	(A7)	[A7	D	A+	D+	G	]
(4)	Eb-7	E9	A13	(D7)	[Db7b9	Dbh9	Gbh9	Abh9	Ebh9	]
(5)	Eb-7	Ab7	Db^7	(Eb-7)	[Eb-7	Db6	Ab7	Db^7	Bb7b13	]
(6)	C	Dbo	D-7	(C)	[C	G7	G+	C+	F	]
(7)	A7	A7	D	(D)	[D	A+	A7	Ebo	G	]
(8)	Db	Bb7	Bb7	(Eb7)	[Ab7	Bb7	Eb7	Db	Eb+	]
(9)	D7	E^7	C7	(E^7)	[Gb-7	B7	E6	E^7	Fo7	]
(10)	G7	C	Dbo	(Abo)	[G	G+	D7	C	D+	]
(11)	G-7	Eb7#11	Db7#11	(C^7)	[D-7	C6	C^7	A9#5	A7b13	]
(12)	Bb-7	Eb7	Bb-7	(Ab)	[Bb-7	Eb7	Ab6	Ab^7	F-7	]
(13)	F	G7	C7	(D-)	[F	C7	C+	Gbo	F+	]
(14)	B6	E7#11	Eb-7	(Ab7)	[B6	Eb-7	Db-7	Fh	Db7	]
(15)	Ab-7	Db-7	Eb7	(E9)	[Bb-7	Eb7	Ab6	Ab^7	Ao7	]
(16)	Ab7	Db-7	Gb7	(Eb-7)	[Db-7	Gb7	B6	B^7	Gb+	]
(17)	Ab^7	Bb-7	Eb7	(Ab^7)	[Bb-7	Ab6	Ab^7	Eb7	F-7	]
(18)	Gb^7	F-7	Gb^7	(F-7)	[Eb-7	Bb7b13	Bb7b9	F7b9#9	Gb^7	]
(19)	Db7	Gb	Bb7	(B)	[Db	Eb+	Ab	Eb7	Ab7	]
(20)	Db7	Gb^7	G-7	(C7)	[Ebh9	D-7	Ch9	Ebo7	Gbh9	]

Above examples are generated using init samples that are randomly sampled from the training dataset described in Section 3.2.7. I only sort them for easier commenting. Examples (1-7) first most similar chord to predicted vector is actually the one that is present as a label in test set (in brackets). That precise prediction almost never occurred in a baseline model. Here in 20 random samples I have 7 perfectly predicted one. In music there is never only one option, that why I analyze the quality of the rest of the predictions according to jazz harmony rules.

In example (8) every chord in prediction plays important role in key of E♭. Each of them could be used and create very different paths in this harmonic passage. In samples (9-11) we can see that 3 chord and top-1 chord from prediction are in *triton* or *minor second*. These are very common intervals in jazz chords. In examples (11-16) in top-2 predictions there is a repetition of third chord. My model proposes repetitions in sensible rate, because too many of them obviously will make music boring, and lack of repetition bring chaos to the composition. In examples (17-20) and all previous as well I can notice a lot of interesting jazz progressions: modal interchange, backdoor dominants, secondary dominants, substitutions, and interesting passing chords.

At the end of this chapter I present result of a generation of a longer sequence. This time I predict on a full 15 chords sequence to generate the next chord. I repeat this process several times. After each prediction I extend the sequence by last predicted chord, and use last 15 chords of that sequence to predict a new chord. Below I present some of my favourite songs created with my model. Symbol ''||'' separates generated chords from initial sequence.

```
Song: 20442
0:        F7      |    D-7     |    F-7     |    A9#5    |
4:        C-7     |    D-7     |    Eb^7    |    A7b13   |
8:        C-7     |    D-7     |    Eb6     |    D-7     |
12:       F7      |    C6      |    Eb^7    ||   G7b13   |
16:       D-7     |    C-7     |    G7b13   |    Eb^7    |
-----------------------------------------------------------
Song: 13752
0:        Ch      |    F9#5    |    Bb-^7   |    Bb-^7   |
4:        Gh7     |    C7#5    |    G-7     |    F^7     |
8:        F7b9    |    F9#5    |    Bb-6    |    Bb-#5   |
12:       C7b9    |    C7b13   |    C7#5    ||   F6      |
16:       C7#5    |    C7#5    |    G-7     |    F^7     |
-----------------------------------------------------------
Song: 9027
0:        D7sus   |    C-7     |    A       |    Ab+     |
4:        A-7     |    Ah7     |    D7#5    |    D7      |
8:        Ab^7    |    E7      |    E7b13   |    Gbh7    |
12:       D7      |    D7#5    |    C7#11   ||   A-7     |
16:       G6      |    D7      |    A-7     |    Gbh7    |
-----------------------------------------------------------
Song: 2345
0:        G^7     |    G6      |    D7b13   |    F6      |
4:        E7b13   |    G^7     |    Bb^7    |    C-7     |
8:        G^7     |    Ab6     |    Eb7     |    D7      |
12:       G6      |    G^7     |    Ab^7    ||   Ao7     |
16:       D7      |    Gbo7    |    C7      |    G7      |
```

5. Conclusions

In this Chapter I summarise my contribution and conclusions I draw while working on this project. I also include a list of improvements that can be done to further develop this project.

5.1. My Contribution

My work stands out from the ones discussed in Chapter 2 in that I define the musical matter precisely, and create a full dictionary of symbols and chord components used in jazz standards without any significant simplification. I study the structure of chords, the distributions of various values to draw conclusions about the construction, analysis of embedding, models, and generated sequences. I describe a number of musical terms in detail in order to explain the basic elements of jazz music to the reader, and allow him to understand achieved results.

In this thesis, I create embedding for jazz chords. I use existing solutions solutions (see Section 3.3) and implement my baseline encoding in the Gensim API. I visualize the latent space of the chords embeddings in multiple ways and describe the relations in this space referring to the principles of the jazz harmony. I test embeddings for their hyper parameters and generate a set of analogies to test embeddings response to those analogies created using the most important jazz chords progressions.

I performed experiments on recurrent models studying their structure and hyper parameters. I use the *cuDNN* library for hardware acceleration on long-lasting calculations on neural networks. I use the potential of the GPU in my notebook to perform these calculations with acceleration. I implement the mean cosine distance metric using Keras API.

I generate and describe many examples that testify the quality of my model and confirm the validity of my assumptions. Finally, I incorporate the results into clear visualizations, cross-sectional tables, and extensive descriptions, to share the results in a clear form.

5.2. Significant Findings

The scope of the work in this project was very wide, from analysing existing solutions, collecting and processing data to creating and testing models and interpreting the results. I learned about two very important artificial intelligence methods: words embeddings and recurrent neural networks. I had to explore some API elements of two machine learning tools - *Keras* and *Gensim*. I learned how important it is to predict what results will be needed before a long learning process of the neural network is started. Writing down all the results is also very important because recreating the experiments can be very time-consuming. I learned that a lot depends on the choice of metrics that we are testing our model with. None of the metrics are universal, and the right choice of metrics can influence on the choice of the model and hyper parameters. Watching interactive visualizations in Embedding Projector I noticed a lot of interesting relationships between some of the chords that I would not have thought about before. It follows that we can learn a lot about music by studying how it is interpreted by the computer.

The results of the experiments show that recurrent neural networks perform better when recurrent layers are stacked. Using proper activation function is significant to obtain the expected output values. Larger size of the history sequence results in better performance of the prediction. Networks with long term 'memory': *LSTM* and *GRU* have better performance in particular when the input sequence is large. Word embeddings: *Word2Vec* and *FastText* perform significantly better than embedding based on music harmony rules. Small size of the vocabulary caused the lack of improvement with increase of the the embedding size above certain value. Visualisation of reduced latent space of the embeddings shows circular geometric relationships between groups of chords that are strongly connected in terms of jazz harmony.

5.3. Future Work

This project has a great potential to be further developed not only from a research but also from a business perspective. It is possible to use a model in an application that would tell the musician what chords he or she can use in his or her composition. It would mainly be directed to amateur musicians, as they represent a large part of the market and often need more inspiration to complete their songs. From a research point of view, more advanced embedding research can be conducted using those integrated with the generating model. It is also possible to collect more songs and categorize them by genre, which would have an interesting effect on the analysis of different statistics and breakdowns and would be a considerable improvement for the application.

Bibliography

[1] Alfred Blatter. *Revisiting Music Theory: A Guide to the Practice*. Routledge, 2007.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2016.

[3] Alexei Botchkarev. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology, 2018.

[4] Rich Caruana, Steve Lawrence, and C. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. volume 13, pages 402–408, 01 2000.

[5] Kyunghyun Cho, Bart van Merrienboer, and et al. Gulcehre. Learning phrase representations using rnn encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[7] F. DeLoche. *RNN unfold image source*. `https://upload.wikimedia.org/wikipedia/commons/thumb/b/b5/Recurrent_neural_network_unfold.svg/440px-Recurrent_neural_network_unfold.svg.png`.

[8] Mateusz Dorobek. Wykorzystanie sztucznej inteligencji do generowania treści muzycznych., 2019.

[9] Mateusz Dorobek and Mateusz Modrzejewski. Application of deep neural networks to music composition based on MIDI datasets and graphical representation. In *Artificial Intelligence and Soft Computing*, pages 143–152. Springer International Publishing, 2019.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[11] Garrett Grimm. *Understanding the Circle of Fifths and Why It's a Powerful Tool*. `http://musictheorysite.com/the-circle-of-fifths/`.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[13] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, 2017.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[15] Filip Korzeniowski, David R. W. Sears, and Gerhard Widmer. A large-scale study of language models for chord prediction. *CoRR*, abs/1804.01849, 2018.

[16] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[17] Hal Leonard. *Infant Eyes by Wayne Shorter*. Digital Sheet Music.
`www.sheetmusicplus.com/title/infant-eyes-digital-sheet-music/19425111`.

[18] Chunjie Luo, Jianfeng Zhan, Lei Wang, and Qiang Yang. Cosine normalization: Using cosine similarity instead of dot product in neural networks, 2017.

[19] Iris Mencke, Diana Omigie, Melanie Wald-Fuhrmann, and Elvira Brattico. Atonal music: Can uncertainty lead to pleasure? *Frontiers in Neuroscience*, 12, 2019.

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[21] Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation, 2013.

[22] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North A merican Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[23] Christopher Olah. *Understanding LSTM Networks*.
`https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[24] Vincent Persichetti. *Twentieth-Century Harmony: Creative Aspects and Practice*. W.W. Norton, 1961.

[25] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[26] Thomas Schäfer, Peter Sedlmeier, Christine Städtler, and David Huron. The psychological functions of music listening. *Frontiers in psychology vol. 4 511*, 2013.

[27] Sephora Madjiheurem, Lizhen Qu, and Christian Walder. Chord2vec: Learning musical chord embeddings. 2016.

[28] Daniel Smilkov, Nikhil Thorat, Charles Nicholson, Emily Reif, Fernanda B. Viégas, and Martin Wattenberg. Embedding projector: Interactive visualization and interpretation of embeddings, 2016.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1954, January 2014.

[30] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[31] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016.

List of appendices

Embedding Projector links to all my embedding models visualisations

- https://projector.tensorflow.org/?config=https://gist.githubusercontent.com/SaxMan96/b04b97652ae888341feb300a0123d8cf/raw/c6a507268bfb3417c2d11d717e65239e7d2b7504/Word2VecSkipGramSize14_projector_config.json,
- https://projector.tensorflow.org/?config=https://gist.githubusercontent.com/SaxMan96/b82b2072cd200067f75bb54518b7a5a1/raw/474eb4f97bf8c67c74b951c3977ae3532c1e7d75/Word2VecCBOWSize13_projector_config.json,
- https://projector.tensorflow.org/?config=https://gist.githubusercontent.com/SaxMan96/c969c06ef0f255cc3613c5769152ab30/raw/dba7880791b569df7a304fbb82036671338b696d/FastTextSize13_projector_config.json,
- https://projector.tensorflow.org/?config=https://gist.githubusercontent.com/SaxMan96/b3ecca37320f23e448b51d2e377fc0b3/raw/2257d74e6de1c25d6ff9bb443c6e64b929cd6a27/Word2VecCBOWSize70_projector_config.json,
- https://projector.tensorflow.org/?config=https://gist.githubusercontent.com/SaxMan96/5262d6c44dca7db3df284c9657b2d6cc/raw/7a37a987d3cafc010a48ce3072067e4a025360bb/Word2VecSkipGramSize70_projector_config.json,
- https://projector.tensorflow.org/?config=https://gist.githubusercontent.com/SaxMan96/8587062e23a8a4edda38c0fc586a192d/raw/d948417b2a8fb790033709be4a1e607e57ca90b0/FastTextSize70_projector_config.json

iRealPro custom URL format containing songs

https://irealpro.com/ireal-pro-file-format/

All sources that I have gathered data from.

Links from iRealPro forum.

- https://www.irealb.com/forums/showthread.php?215-Gypsy-Jazz,
- https://www.irealb.com/forums/showthread.php?209-Pat-Metheny-songs,
- https://www.irealb.com/forums/showthread.php?204-Contemporary-Jazz,
- https://www.irealb.com/forums/showthread.php?210-Fusion-and-Smooth-Jazz,
- https://www.irealb.com/forums/showthread.php?10591-Dixieland-Trad-Playlists,
- https://www.irealb.com/forums/showthread.php?4522-Jazz-1350-Standards,

All chord symbols used in iReal Pro:

https://technimo.helpshift.com/a/ireal-pro/?s=editor&f=chord-symbols-used-in-ireal-pro

Selected python libraries

- pyRealParser - https://pypi.org/project/pyRealParser/
- gensim - https://pypi.org/project/gensim/
- pychord - https://pypi.org/project/pychord/
- tensorflow - https://www.tensorflow.org/

The official Real Book site

https://officialrealbook.com/