Multi-Step Chord Prediction: Neural Network-Enabled Chord Progressions with MLP and LSTM Architectures

Max Gupta, Columbia University
Advisors: Jérome Nika (PhD), IRCAM; Tristan Carsault, IRCAM

In collaboration with, and with generous support from:

Abstract

As the world undergoes a digital revolution, the musical arts are undergoing their own computer revolutions in making and sharing sound. Since the 90's, computers have become increasingly involved in all aspects of musical production. At first, it was the digitization of sound itself, transitioning from analog to digital and vinyl to MP3. Nowadays, most of the revolutions in computerized music are coming from artificial intelligence and machine learning. One of the central goals for artificial intelligence in music is getting a neural network to understand the cadence and pulse of a piece, an attribute that is often mapped by a musical term known as a *chord progression*. This research develops on state of the art models in this area, known as automatic composition, by improving upon data structures in vanilla MLP, LSTM, and encoder-decoder models in Nika et al. 2019.

In this comprehensive report, I begin with a theoretical background on the types of neural network architectures used in our study (some background in maths is required). I go on to introduce the field of automatic composition for chord progressions and provide a review of previous uses of our architecture types applied historically to generate chords. We then look at the real work of the project: improving upon state of the art works of Nika et Al. (2019) to produce better chord accuracies. In this section, we discuss changes made to the previous research and detail the reasons behind these changes (multi-step error propagation, data augmentation, temporal sensitivity). Finally we will discuss the results and suggest areas for further research and improvement in automatic composition with deep learning for real world applications.

Background

Max is a senior year undergraduate student at Columbia University, where he majors in applied mathematics and minors in computer science. He undertook this project in the summer of 2020 as a research assistant in the musical representations team at IRCAM, remotely from Vancouver due to COVID-19. Max first started at IRCAM in Paris in February of 2020 researching Markov chains in automatic composition and musical analysis, for which he completed a review of the state of the art on Markov chains in automatic composition, *"Musical Markov Chains: Stochastic Models in Music"* under the supervision of Dr. Mikhail Malt. Straight after completing this project, Max branched into neural networks as a more powerful method for automatic composition and started work with Dr. Jerome Nika and Tristan Carsault, improving upon key features of their previous paper, *"Multi-Step Chord Sequence Prediction Based on Aggregated Multi-Scale Encoder-Decoder Networks"*. Max holds a specialization in deep learning from DeepLearning.ai and is taking his first university courses in artificial intelligence in his final year at Columbia. Making the collaboration from Vancouver to Paris work would not have been

possible without the generous grant support from Columbia University, the Heinrich Family Fund, and the Spritz Family Fund. A big thank you goes out to the Columbia Center for Undergraduate Research and Fellowships and the Columbia Center for Career Education for supporting this work financially, to Jerome Nika and Tristan Carsault for supervising, and to my family and friends who supported this rewarding research project.

Introduction

Automated musical composition has been a computational challenge for decades spanning back to the 1940's and involving a plethora of mathematical formulae and algorithms with frequently mind-boggling complexities. Somewhat ironically, however, music is often thought of as a creative, 'intuitive' endeavour, at least it seems to be so for the world's most skilled artists, who often can't (and don't usually have to) explain the exact science of their music-making. At a high level, music production seems to become an incredibly intricate, personalized, and automatic internal creative process. In fact, most artists don't necessarily enjoy spelling out the why's and how's of their creative processes. "Works of art make rules; rules do not make works of art" claimed Claude Debussy, one of history's most talented and prolific pianists. Many musicians old and new will agree with Debussy, claiming that no set of rules, however intricate, and no algorithm, however nuanced, could ever imitate the lucid intuition of a brilliant artist. Algorithmic composition aims to do exactly the opposite: to boil creativity and musical skills down to fundamental, replicable patterns. But the intentions of algorithmic composition are in no way malicious in trying to disprove musicians, but rather to create new human-computer interfaces for artistic expression and performance: a musical man-machine synergy. Computer scientists, psychologists, and musciologists alike have put a vast amount of effort over the past few decades into this field, unraveling music cognition and the perception of pleasant sounding music. Increased performance in automatic music generation tasks have led to advanced applications in:

- Helping novice musicians compose pleasant sounding musical pieces
- Generating accompanying music for music, games, and programs based on scene context
- Accompanying musicians during live jam sessions, enabling a dynamic musical environment for solo musicians
- Musical pedagogy and practice environments

In algorithmic music composition, arguably the simplest technique involves constructing a song through the selection of notes sequentially according to a table of transitional probabilities that specifies the mathematical chance of the next note occurring as a function of the preceding note. This is what we call a Markov Chain for music, and is the mechanism I had previously used to compose basic songs while writing my junior thesis in applied mathematics at Paris with

IRCAM. Markov Chains are useful mathematical constructs, their use in music yields accuracy results that have long ago curtailed to a plateau. In the meantime, with the rise of deep learning, machine learning based algorithms and data processing techniques have since the late 90's taken over as the benchmark for musical generation tasks.

Since chord progressions are the underlying harmonic structure of most of today's music, automatic chord generation has immediate applications to all of these above areas. Real-time music improvisation systems thus need to be able to predict chords in real time along with a human musician across the timed duration of the song. Two aspects are important here: beat-to-beat accuracy (short term) and pattern to pattern accuracy (long term). The computer, like the human, needs to be able to track and play with both temporal horizons in mind.

While neural networks can already produce novel songs entirely algorithmically, the familiar chord cadence of mainstream music is usually difficult to attain. Currently available methods can predict musical chord sequence with high accuracy in single-step scenarios – that is, where the chord in a score depends only on the chord appearing just before it. However, most models fail to accurately predict chord progression based on a sequence of chords (Nika, Carsault, 2019). This lack of multi-step prediction inhibits the applicability of automatic music generation in more complex forms of musical scores. For example, when one chorus of a popular song is completely novel from the other, the listener notices and is usually thrown off. The networks need to understand the entire structure of the song in order to generate contextually appropriate melodies.

Accurate prediction of chord sequence is thus a necessity for automated music generation if the music thus generated aims to be appreciated by general audiences. Chord prediction could also be used as a key input in sound recognition, a service used intensively by streaming services like Spotify and SoundCloud. Most work in chord sequence prediction focuses on chord transitions from single chord to single chord, and does not include the duration of the chords. Such models include Hidden Markov Models (HMMs) and N-Gram models. However, these are both older examples of machine learning that have quickly become overclipsed by the rise of neural networks and deep learning.

Neural Nets and Deep Learning: A Theoretical Background for Automatic Composition

With the beginning of the 2000's came a widespread use of new data-driven techniques to generate musical chord progressions. With growing computational power and a larger amount of easily accessible, processable musical data came stronger "deep" learning models and higher accuracies. Machine learning performance increases proportionally with data volume and processing power and since both of the latter have exponentially increased with next-generation

semiconductors chips and high-speed GPU's, it's no wonder that deep learning has come to influence the music industry too.

In our experiments, we trained three distinct types of neural networks, each with varying computational complexities: vanilla Multilayer Perceptron (MLPs) networks, and word-based Long-Short Term Memory (LSTMs) networks. In this section, we will first go through the computational theory behind each network briefly, as it relates to automatic composition, before describing the training process in our experiments.

Previous Work in Neural Network Composition

Mozer, Neural Network Composition by Prediction

In 1994, Michael C. Mozer realized that Markov chains had reached a natural plateau in performance for musical creation. Ultimately, they were temporarily restricted by the Markov assumption that the current state should derive its identity from only the previous state. Thus, Markov chains end up following a form of musical *random walk,* regulated only by a table of transition probabilities. While this produces original and interesting examples, this linear technique of note-by-note composition lacks the architecture to scale up to entire song lengths. Mozer proposed a recurrent network architecture in CONCERT, his generative model, whose input is an entire melody instead of a single note, and whose output at each step is then a single next note in the melody. CONCERT made use of the most sophisticated RNN procedures of his time: log-likelihood objective functions, probabilistic interpretation of output values, and even 'psychologically-realistic' representations of melody, chords, and duration. The conclusions of the experiment revealed, however, that the back propagation mechanisms in the RNN were not sufficiently powerful, *especially for contingencies that span long temporal intervals*. The network was good to learn relationships between two events separated by only a few unrelated intervening events, but as the number of intervening events grows, a point is quickly reached where the relationship cannot be learned (Mozer, 1994).

Although RNN networks outperformed Markov approaches from the 50's and 60's, they failed in all cases to find global structure (ibid). A song is, after all, much more than just a linear string of notes.

It was soon discovered that the *real* reason for Mozer's findings had to do with the *vanishing gradient problem* inherent in RNNs, a technical issue that exponentially compiles error over longer time periods. This structural deficiency made RNNs incapable of dealing with extended time series data.

Our goal then, is to find and implement a neural network architecture that is capable of creating locally *and* globally coherent musical patterns. Just three years after Mozer's discovery of RNN limitation, there emerged a solution: Long-Short Term Memory, developed by the german computer scientists Sepp Hochreiter and Jurgen Schmidhuber. LSTM introduced a revolutionary change to the vanilla RNN architecture by introducing Constant Error Carousel (CEC) units to eliminate the *vanishing gradient problem*, which was causing the global incoherence in Mozer's experiments. LSTMs feature a more powerful method of back propagation, using *forget gates* that allow errors to flow backwards through unlimited numbers of virtual layers. With this added mechanism, LSTMs can, in theory, learn tasks that require memories of events that happened thousands or even millions of time steps in the past (Goodfellow, 2017). LSTMs soon became the new benchmark for time-series analysis with machine learning, replacing vanilla RNNs almost completely, and becoming the benchmark for word-based Natural Language Processing tasks used by Apple in Auto-Type, Google in Google Translate, and Facebook in their automatic translations .

Keunwoo Choi's text-based LSTM for automatic composition

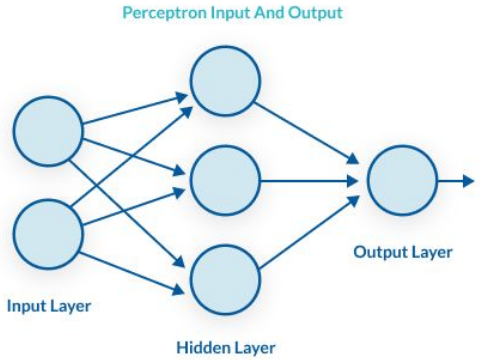
Recently, we have seen numerous studies in the automatic composition field attempt to harness the features of LSTMs for song generation. A recent success was implemented by Keunwoo Choi et al. of the Center For Digital Music, Queen Mary University of London. In this study, Choi implements character-based LSTMs and word-based LSTMs (pioneered by Andrej Karpathy) for automatic composition for the first time. Specifically, Choi trains the networks on chord progressions from the Jazz Realbook (the comprehensive compilation of lead sheets for jazz standards). In this way, Choi extracted all the chord information from the jazz dataset into text-based XLAB files and assigned an LSTM to learn musical patterns from the text itself.

Neural Network Architectures for Chord Generation

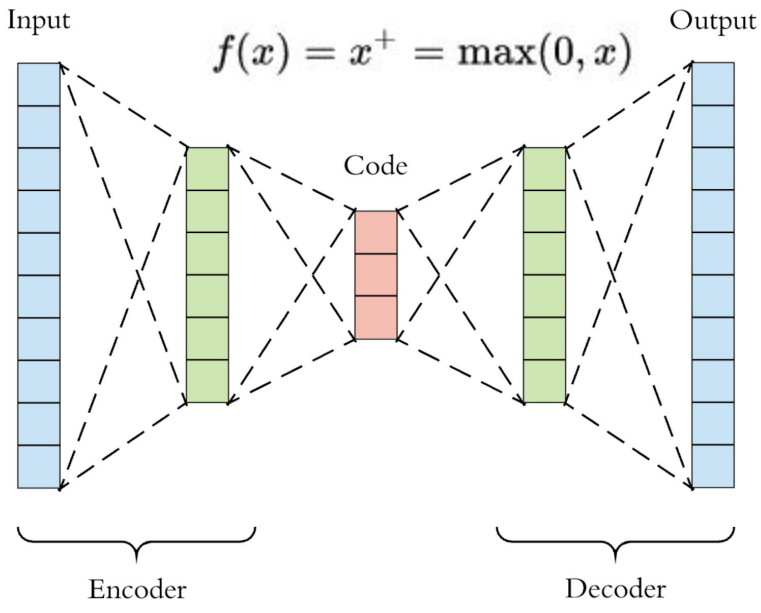
The Vanilla Multilayer Perceptron : MLP-ED

The Multilayer Perceptron is a term used to describe any type of feedforward artificial neural network (ANN) and is composed of any number of layers of *perceptrons*. The perceptron is the original fundamental building block of the most simple task in machine learning: binary classification. The perceptron acts as a simple linear classifier, combining a set of weights from its inputs, applying a mathematical activation function, and outputting a binary prediction. The original perceptron learning algorithm created by Frank Rosenblatt on behalf of the US Navy was modeled after the neurons in the human brain, with much mathematical simplification of biological processes. A famous roadblock in the career path of the perceptron, however, was its initial inability to learn an XOR function (a logical operation that outputs true only when its

inputs differ). After years of stagnation, it was revealed that *multiple layers* of these neuron-like perceptrons stacked on top of each other solved the XOR learning problem. This gave rise to the current multi-layer perceptron models and algorithms we use today, consisting of at least three feedforward layers, and utilizing the famous backpropagation technique for error optimization in training (Goodfellow, 2017).



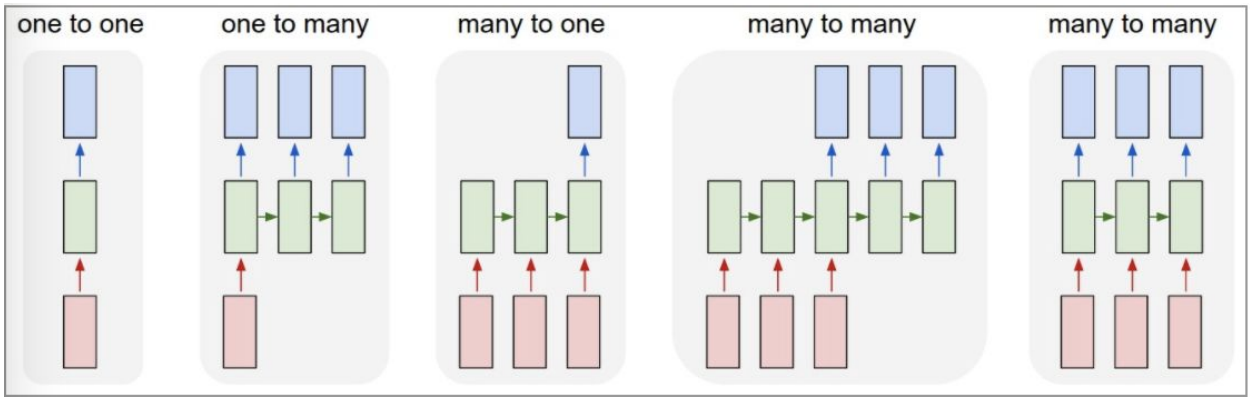
In our application, we are passing in multiple chord elements from a previous time frame to output multiple predicted chord elements to be placed in the current time frame of a song. However, we want flexibility in the number of chords we choose to input into the model and the number we choose to predict. A glaring limitation of vanilla MLP networks is that their API is too constrained for such problems, known as *sequence to sequence* prediction problems (Goodfellow, 2017). For such problems, we want to add encoder-decoder structure to our MLP architecture to encode input chord data into one-hot vector format before the decoder translates the vector back into chord format. In our experiments, we actually work with a purely *textual* dataset (described later), so the input is a chord string, EG "G:major", which is formatted into a one-hot vector, passed through an MLP encoder-decoder architecture (with ReLU activation) as shown below:



The architecture of the multi-layer perceptron encoder-decoder (MLP-ED) model used in this study is adopted from the MLP model proposed by Nika and Carsault in the 2019 paper, "Multi-Step Chord Sequence Prediction Based on Aggregated Multi-Scale Encoder-Decoder Networks". In this work, we use 2 encoder layers, 2 decoder layers, with a total of 500 hidden units. All encoder-decoder blocks are fully-connected layers with ReLU activation, with dropout layers (p = 0.5) between each layer and a Softmax layer at the output of the decoder block. The network is trained with ADAM at a learning rate of $1e^{-4}$.

The Word-RNN Based LSTM

While adding encoder-decoder functionality to vanilla MLPs is indeed one way to allow for sequence-to-sequence prediction, there exists another class of neural networks much better naturally fitted for our task: Recurrent Neural Networks or RNNs (Karpathy, 2015). What this means is that they accept a fixed-size vector as the input (e.g. our 8 beat jazz chord progressions) and produce a fixed-size vector as the output (e.g. our computer generated 8 beat chord progressions). Not only that, but these models perform this mapping using a fixed amount of computational steps (IE the number of *layers* in the network model). The core reason that recurrent neural nets are more useful is that they allow us to operate over *sequences* of vectors: sequences in the input, the output, or indeed in both. Below is a diagram illustrating these key differences between MLP's (one-to-one mapping) and RNN's (sequence mapping).



In the above diagram, each rectangle is a vector containing numeratized chord information (mostly binary integers, simply 1's and 0's) and arrows represent functions (e.g. matrix multiplication or vector addition). Input vectors are red, output vectors are blue, and "*hidden" states* (intermediary states between the output and input) are green. From left to right: **(1)** Vanilla MLP, processing from fixed-size input to fixed-size output (see our MLP-ED model in the next

section). **(2)** Sequence output RNN, processing a fixed-size input and outputting a many-element sequence output (we can input a single chord into our RNN model and come up with an entire predicted progression). **(3)** Sequence input RNN, processing a many-element sequence input and simplifying to a single-size output (conversely, we can input an entire chord progression and expect a single predicted chord). **(4)** Sequence input and sequence output RNN (this is the type of exercise we perform in our model: going from input chord progression to predicted next chord progression, both of varying and unlimited sizes). **(5)** Beat-aligned (synced) sequence input and output RNN (if we wish to beat-align our predicted chords, we sync the input, hidden, and output states in our RNN as shown).

The Mathematics of RNN's and LSTM's

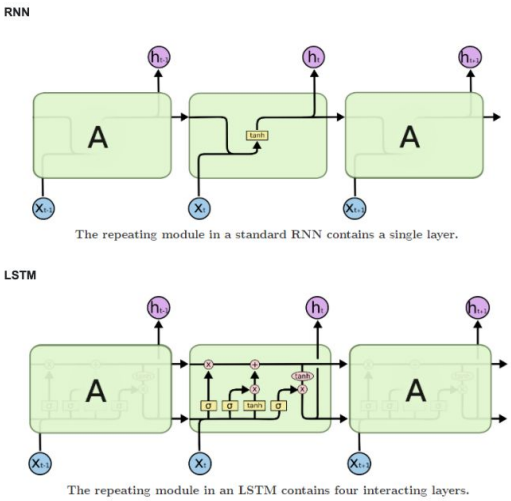
A Vanilla RNN has a simple API, accepting an input vector x with three matrices (call them *m1*, *m2*, and *m3*), performing matrix multiplication on *m1* and the previous hidden state, on *m2* and the current input, adding these results, and then 'squashing' the sum with a *tanh* activation function to get the current hidden state:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Then, the current hidden state is compounded by matrix multiplication with *m3*, the last input matrix, creating the output after one 'forward pass' (essentially a forward pass, also known as forward propagation is one iteration of the neural network). For best performance, as we are in the field of *deep* learning (several layers), we stack this operation on itself and compound the maths above many times. In our model, we use two RNN layers, each of which has 512 hidden states (units). In other words we have two separate RNNs: One RNN is receiving the input vectors and the second RNN is receiving the output of the first RNN as its input. Except neither of these RNNs know or care - it's all just vectors coming in and going out, with loss being lessened (ideally) at each step through the gradient-optimization process known as *backpropagation*.

Now in our model, we implement a particular, more powerful type of RNN, a *Long Short-Term Memory* (LSTM) network. The reason we do *not* use the vanilla RNN described above is that these networks have been shown previously to cause the gradient of the loss function to decay exponentially with time - a phenomenon known as the *vanishing gradient problem*. Without diving into the maths, this means that vanilla RNNs cannot handle long-term temporal dependencies, like those we find in music (or any other event carried out over a longer time-series, like stock prediction, weather reports, etc).

The LSTM solves the vanishing/exploding gradient problem by introducing new 'gates', such as input and 'forget' gates (pictured below at bottom, as compared to a vanilla RNN at top), which allow for better control over the gradient flow and enable better *preservation of long-range dependencies*. Thus, LSTMs are essentially RNN's specifically adapted for learning time-sensitive pattern information, like musical chords. In the literature, it is sometimes common to use the terms RNNs and LSTMs interchangeably, however, from hereon, we refer to this specific chord prediction mechanism as the *LSTM* (Karpathy, 2015).



This model is adopted from the paper "Text-Based LSTM Networks for Automatic Music Composition" by Keunwoo Choi et al. 2016 at the Center for Digital Music, Queen Mary University of London.

The Multi-Scale Autoencoder : MS-ED

In their paper, "Multi-Step Chord Sequence Prediction Based on Aggregated Multi-Scale Encoder-Decoder Networks", J. Nika and T. Carsault improved on the above word-based LSTM model for automatic composition by embedding multi-step prediction mechanisms to reduce *single-step error propagation*. In this study, we will not be concerned with the use of this model, however, it is explained in cursory detail below.

Specifically, this is achieved by merging input and ground truth chord labels into increasingly large temporal bags, trained with a family of encoder-decoder networks for each temporal scale. These 'bottle-necked' features are then aggregated and trained in a final encoder-decoder network. This initial model, created in 2019, *outperformed* existing methods in terms of accuracy and perplexity, while requiring relatively few parameters (Nika, Carsault, 2019).

The MS-ED model is based on a sequence-to-sequence architecture, which is defined by two major components. The first is a traditional encoder-decoder network, the second is an aggregation mechanism for pre-training the models at various time scales. Here, the authors define aggregation as an increase of the temporal span covered by one point of chord information in the input/target sequences. In order to train the whole architecture, Nika and Carsault used a two-step training procedure. In the first step, they separately train each network with inputs and targets aggregated at different ratios. The second step performs the training of the whole architecture where they concatenate the output of the previously trained encoders.

MLSP19

The model presented by Nika and Carsault in 2019 builds on Choi's earlier work, adding improved accuracy scores through a novel multi-step, aggregated encoder-decoder network system. Furthermore, Nika and Carsault were able to add a more refined layer of musical expertise in the structuring of chord alphabets and the musical analysis of error. Compared to LSTM models in Choi's work, the multi-scale encoder mechanism produced by Nika et al. produced a couple percentage points increase in accuracy across all three test chord alphabets.

MLSP20: A novel data structure in Pytorch

In this work, the authors developed further on the architectures of MLPS19 by changing the input chord data representation and improving upon the chord dictionaries to include all possible repetitions of each chord type. In this way, we train our models to recognize self-similar chunks of chord progressions and avoid single-step error propagation by avoiding chord processing at the beat level, and instead process at the segment level. As we noted with previous neural network architectures used throughout the history of automatic composition, global coherence and memory are the single most important factors for generating structured, sensible music. By avoiding route linear chord processing and step-by-step, beat-by-beat processing, and introducing temporal chunking and improved chord dictionary structures, we hope to achieve improved results over the state of the art for chord generation.

Dataset

In our experiments, we use the Realbook dataset, which contains 2,846 jazz songs based on band-in-a-box files. We use jazz chords as ground truth examples to train our models because of the incredibly high intricacy and complexity of chord transitions and long-term patterns. All files are in *xlab* format and contain time-aligned beat and chord information. We perform a 5-flod cross-validation by randomly splitting the song files into training (60%), validation (20%), and

test (20%) with 5 different random seeds for the splits. We report results as an average of the resulting 5 scores.

Previously, chord data was individually *beat-aligned*, meaning that each chord beat was contained as a singular data point. For example, the progression

[A:major, A:major, E:minor, E:minor, E:minor, D:major, D:major, D:major]

is the computer-read output when we input 2 beats of A major, 3 beats E minor, 3 beats D major from a jazz xlab file. In this paper, we built a model to parse the chord data from this rich dataset into temporal chunks, allowing us to specify the length and frequency of generated chord patterns. Our model parses the chord data from the jazz pieces into equal chunks of input sequences X, 8 beats long, and output sequences Y, 8 beats long. Furthermore, we experiment dividing the 8 beat input/output progressions into variable temporal sizes from chunks of sizes 1-6 to experiment with sizes of repeated chords and chord patterns. For example, we can now split the above example of 2 A major chords, followed by 3 E minor and 3 D major into the following sequence such that each temporal chunk is capped at 2 beats long:

[A:major 2, E:minor 2, E: minor 1, D:major 2, D:major 1]

We also experiment with processing the chord sequences with maximum repetitions being 3,4 or more beats long, as in the example above, with max. repetitions now changing to 3, we get:

[A:major 2, E:minor 3, D:major 3]

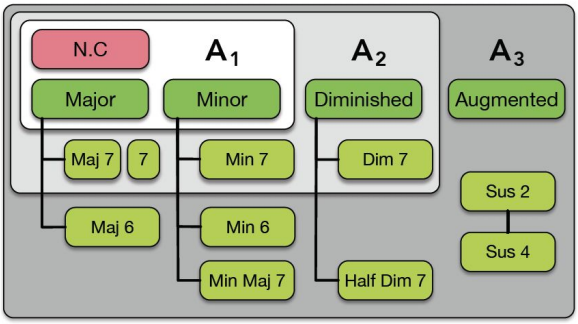
To ensure each vector is of the same size (for comparison purposes), we then pad each vector with the 'No Chord' symbol ('N') to attain a length of 8 beats. The above then becomes:

[A:major 2, E:minor 3, D:major 3, N, N, N, N, N]

We then map each 8-beat chord input and each 8-beat chord output to the associated chord alphabet such that each vector is mapped to a one-hot *torch* vector of size *[8, (maxReps*25)+1]*. This allows for a temporally flexible chord dictionary. As specified above, we work with a standard chord dictionary of 25 possible chord elements. If the maxReps (maximum repetitions) variable is set to 2, for example, we are working with tensors of size [8, 51]. The significance of this is that a larger repetition bagging size leads to a larger tensor size.

Chord Alphabets and Reduction

The dataset we use is composed by a total of 1259 chord labels. This great diversity comes from the precision level of the underlying jazz dataset and its syntax.



Here, we apply a hierarchical reduction of the original alphabet into three smaller alphabets as depicted in the diagram to the left, containing triads and tetrachords commonly used to write chord progressions. Each node in the diagram (except N.C. for No Chord) represents 12 chord symbols. Dark green represents the four standard triads: major, minor, diminished, and augmented. A0 contains 35 symbols, A2 contains 85 symbols, and A5 contains 169 symbols. The black lines represent chord reductions, and chord symbols not in a given alphabet are either reduced to the corresponding standard triad, or replaced by the no chord symbol.

In this paper, we study the prediction of a sequence of 8 repetition-segmented chords given the previous 8 repetition-segmented chords. For example, if we make the repetition segment 2 chords long, we study the prediction of a 4x2 chord progression from the previous 4x2 chord progression.

The majority of chord extraction and prediction studies rely on a fixed chord alphabet of 25 elements (major and minor chords), whereas some studies perform an exhaustive treatment of every unique set of notes as a different chord. Here, we use chord dictionaries of lengths determined by the temporal segmentation of the progression. In this way, a G chord repeated twice is its own distinct element from a G chord repeated three times. This creates more effective temporal ‘bagging’ of the chord progression into various sized groups.

Methodology

We test chord generation accuracy across two aforementioned models: a Multi-Layer Perceptron Encoder-Decoder (MLP-ED) network and a Long Short-Term Memory (LSTM) network. The MLP-ED model is a vanilla MLP network, using a standard encoder-decoder network with no memory augmentations or multi-step prediction mechanisms built-in. Our LSTM network is adopted from Keunwoo Choi’s LSTM, which is used to perform automatic music composition. The authors use different types of Recurrent Neural Networks (RNN’s), namely *word-RNN and char-RNN*, to generate beat-aligned symbolic chord sequences. Choi et al. found that the word-RNN LSTM was able to generate more context sensitive music with increased global coherence.

In our experiments, we rely on songs taken from the same data set as that of the jazz Realbook (Choi, 2016). We also apply reduction on the three different chord alphabets, as described in the section *Chord Alphabets and Reduction*. However, in order to clean the dataset, we removed all files that contain A3 chords repeated for more than 8 bars, leading to a total of 78 discarded songs. For all the remaining songs, we pad the beginning of the song with 7 beat-aligned *N.C. (No Chord)* symbols and extract all chord sub-sequences that contain 16 beat-aligned chords. Since the dataset is not exactly the same as in MLSP19, the accuracy scores are not directly comparable. However, we still present the scores of the previous models on this cleaned dataset below in *Results*.

Experiments

We run the training procedure across all 3 neural network architectures with a maximum repetition count of 2, a length of input sequence 8, and an identical length of output sequence, 8. We test the results for beat-aligned data inputs from the Realbook (Choi, 2016) against the results on the same networks for beat-grouped data inputs.

	MLP-ED	LSTM
# encoder layers	2	2
# decoder layers	2	2
# hidden units	500	500
# bottleneck dims	50	0

Results

	A1	A2	A3
mlpDecim	41.49±0.63	37.04±0.4	35.47±0.62
lstmDecim	40.82±0.63	37.22±0.44	35.29±0.43
mlpDecim(MaxRep=2)	40.94±0.54	36.89±0.43	35.48±0.51
lstmDecim(MaxRep=2)	40.33±0.46	37.13±0.52	34.91±2.47
mlpDecim(MaxRep=4)	36.67±0.6	32.54±0.17	31.39±0.32
lstmDecim(MaxRep=4)	36.5±0.36	32.54±0.35	31.52±0.49

After testing with a 5-fold cross validation, the above results show the percentage accuracies of the new and old chord representations across the MLP and LSTM models (leaving out the MS-ED model). The first two rows show 5-fold cross validation testing accuracies of the old representation from Nika and Carsault, 2019, the next two rows show the new representation with clusters of maximum two chords, and the final two with clusters of maximum four chords. Evidently, despite our initial hypothesis, the beat-grouped representation does not appear to outperform the beat-aligned representation from last year. This result, while surprising, can be due in part to any number of confounding variables. However, it seems the premise of our investigation may also have been ill-conceived. The accuracies across both models decrease significantly with increased chord group sizes, which seems counter-intuitive to our original hypothesis. This is probably due to the model not having the ability recognizing patterns over a longer time horizon. In this case, it would be worthwhile revisiting this tactic with a different model.

Discussion

The results of our experiments are shown as percentages above, but can be more accurately interpreted as levels of 'fit' between the original jazz chords and the jazz chord progressions interpreted by the machine learning models. Across the board, we see no sizable improvements across chord alphabets and models, LSTM or MLP. We observe that the LSTM model had the lowest accuracy on the A3 chord alphabet. This can be attributed to the fact that the A3 alphabet is the most complex in defining the chord families we test on. We see a general decrease in accuracy from A1 to A2 to A3 across our models with one exception (due to a lack of data points, most likely). Furthermore, as we observed, the LSTM models is functionally designed to be more context sensitive and has in-built gates to detect long-term dependencies, allowing it to be more accurate over longer chord sequences, in theory. However, the LSTM does not outperform the MLP model on longer range chord groupings (see maxReps = 4). This may also be due to the model being inappropriate for the task at hand. This will benefit from being revisited with models allowing for longer time dependencies and with a dataset more effectively chunked for training and testing.

Future Work

There rests a good amount of work we have yet to do to determine the full theoretical reasoning behind these accuracy improvements and a good deal more testing we have to do on industry-grade GPU's to really be certain of their accuracy. This work is yet to be completed.

Furthermore, I am very interested in further investigating upgraded neural architectures for data processing now that we have identified a new and improved data structure. Specifically, the concept of *attention* applied to RNN's has been the most interesting recent architectural innovation in neural networks for some years, recently displayed in full force with the latest state of the art language processor, GPT-3 by OpenAI. It would seem LSTMs are not ideal for this task. When compared to transformers, the newest language processing model, which applies *only* the attention mechanism, and rids the forget and transition gates of the LSTM, LSTMs are no longer state of the art. "It's called Long-Short Term Memory, but it's not that long." as Lukas Kaiszer, one of the authors of "attention is all you need", stated when proposing the transformer model to DeepMind. Frankly when comparing transformers and LSTMs, this is true. We already see abundant long-term context dependencies that LSTMs fail to pick up on and replicate. Furthermore, transformer-based music generation has proven massively successful, with the release of Musenet (OpenAI), Magenta (Google), Jukebox (OpenAI) and GPT-3 (Open AI) leading the way in novel music informatics processing models. In the next step, I will be investigating these applications with our new data representation proposed here.

References

Carsault, Tristan, et al. "Multi-Step Chord Sequence Prediction Based On Aggregated Multi-Scale Encoder-Decoder Networks." 2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP), 2019, doi:10.1109/mlsp.2019.8918813

Choi, Keunwoo & Fazekas, George & Sandler, Mark. (2016). Text-based LSTM networks for Automatic Music Composition

Eck, D., and J. Schmidhuber. "Finding Temporal Structure in Music: Blues Improvisation with LSTM Recurrent Networks." Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing, doi:10.1109/nnsp.2002.1030094

Giorgi, Bruno Di, et al. "A Data-Driven Model of Tonal Chord Sequence Complexity." IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2017, pp. 1–1., doi:10.1109/taslp.2017.2756443.

Goodfellow, Ian, et al. Deep Learning. The MIT Press, 2017

Karpathy, Andrej. The Unreasonable Effectiveness of Recurrent Neural Networks, karpathy.github.io/2015/05/21/rnn-effectiveness/

Mozer, Michael C. "Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-Scale Processing." Connection Science, vol. 6, no. 2-3, 1994, pp. 247–280., doi:10.1080/09540099408915726

Paiement, J., et al. [PDF] A Probabilistic Model for Chord Progressions: Semantic Scholar. 1 Jan.1970,www.semanticscholar.org/paper/A-Probabilistic-Model-for-Chord-Progressions-Paiement-Eck/7262b1592fef5e4aea6700b96d32390fde727724

Schmidhuber, Jürgen. "Deep Learning in Neural Networks: An Overview." Neural Networks, vol. 61, 2015, pp. 85–117., doi:10.1016/j.neunet.2014.09.003

Vaswani, Ashish, et al. "Attention Is All You Need." ArXiv.org, 6 Dec. 2017, arxiv.org/abs/1706.03762