SAM-Bach: A Deep Generative Model for Bach Chorale Generation

Shuonan Dong

School of Computer Science,
McGill University, Montreal, Canada

A thesis submitted to McGill University
in partial fulfillment of the requirements of the degree of

Master of Science

July 2018

2018/07/11

Abstract

During the past few decades, we have seen the invasion of deep generative models in the areas of image processing and text generating. Tremendous practical applications are invented accordingly. The progressing techniques of generative models have brought many exciting changes to our life. Recently, music information retrieval(MIR) started gaining attention from both deep learning research community and industrial circle. Polyphonic music generation, as one of the most popular topics in MIR, has many in-demand applications such as music composition, music retrieval and music recommendation etc.

In this paper, we address polyphonic music generation by using J.S. Bach chorales and leveraging deep generative models and Monte Carlo Markov chain(MCMC) samplings. Specifically, we propose the SAM-Bach system to conduct a Bach-like polyphonic chorales generation task. The whole generation process can be decomposed into three steps: training, inference and sampling. Training and inference are accomplished by a deep generative network containing several subnetworks. Bach-like Chorales are generated via SAM sampling algorithm which is essentially a Monte Carlo Markov chain method. We use J.S. Bach chorales as our music data from a musicology toolkit called music21. Methods implemented in our work require very little expert knowledge of either music composition or Bach harmonization. We conduct two types of experiments under two different settings and the results are evaluated from three perspectives. Chorales generated via SAM-Bach enjoy the major preference of the 248 participants of an online survey compared with other recent advanced works. Besides, we also consult Bach music experts to comment on the chorales generated by SAM sampling. Our chorales are believed to have better melodic generations and fewer dissonances compared with other methods. In terms of the performance of proposed SAM sampling, we implement an element-wise pseudo-KL divergence(CP ratio) as the measure of our sampling process. We compare it with the most popular MCMC sampling method for Bach chorale generation – Gibbs sampling. The CP curves show that SAM sampling indeed has faster convergence property. We also have the sampling methods of SAM and Gibbs evaluated by the human. The responses of the online survey indicate the quality of chorales generated by these two methods are quite compatible and share almost even preference.

Résumé

Les dernières décennies ont été marquées par l'invasion de modèles générateurs profonds dans les domaines du traitement de l'image et de la génération de texte. Ainsi, on a inventé en conséquence d'énormes applications pratiques et commerciales de grandes valeurs. Les techniques progressives des modèles génératifs ont apporté beaucoup de changements passionnants à notre vie. Récemment, la recherche d'information sur la musique a commencé à attirer l'attention à la fois de la part de la communauté de recherche en apprentissage profond et des milieux industriels. La génération de musique polyphonique bénéficie de larges applications dans la demande allant de la composition musicale à la récupération de musique à la recommandation de musique, etc. Dans cet article, nous adressons la génération de musique polyphonique à l'aide de chorales de J.S. Bach et des modèles génératifs profonds et de l'échantillonnage de la chaîne de Markov Monte Carlo. Le processus de génération entière peut être décomposé en trois étapes : la formation, l'inférence et l'échantillonnage. La formation et l'inférence sont accomplies par un réseau génératif profond composé de plusieurs sous-réseaux conçus à des fins différentes. Les chorales sont générées via un algorithme d'échantillonnage SAM. Nous utilisons les chorales de J.S. Bach comme nos données de musique d'une boîte à outils de musicologie musique21. Les méthodes implémentées dans notre travail requièrent peu de connaissances spécialisées sur la composition musicale ou l'harmonisation de Bach. Nous menons deux types d'expériences dans deux contextes différents et les résultats sont évalués à partir de trois perspectives. Nos chorales de Bach apprécient la préférence de la majorité des 248 participants de l'enquête en ligne par rapport à d'autres travaux récents. Nos chorals sont soupçonnés d'avoir une meilleure génération mélodique avec moins de dissonances par rapport à d'autres travaux existants. En termes de performance de l'échantillonnage SAM proposé, nous appliquons une divergence pseudo-KL par élément(ratio CP) comme mesures de notre processus d'échantillonnage et la comparons avec la méthode d'échantillonnage MCMC la plus populaire pour la génération de chorale de Bach-Gibbs. Les courbes CP montrent que l'échantillonnage SAM a une propriété de convergence plus rapide. Les réponses du sondage en ligne indiquent que la qualité des chorales générés par ces deux méthodes est tout à fait compatible et partage presque la même préférence.

Acknowledgment

I would like to thank Professor Xue Liu for his expert advice and guidance on my master's thesis topic. I am sincerely grateful for his trust and support throughout my study period at McGill University.

Special thanks to (ordered by departments) Peng Tang(MS[1], McGill), Dr. José A. Correa (MS, McGill), Yue Dong (CS, McGill), Qinglong Wang (CS, McGill), Chen Ma (CS, McGill), Landu (CS, McGill), Yu Bai (CS, UdeM), Ju Yaolong (Music, McGill), Edward Wingell(OSM)[2] for taking time to help me and offering me innumerable valuable suggestions on statistical inference, deep learning and Bach chorales studies. Your advice contributed to the rigorousness of my work and saved me tons of time.

To my friends, I feel genuinely blessed and fortunate to know you all. Thank you for the countless happy moments and the unconditional supports. I received too much and learned a lot from you all. I would not be me without your presences in my life.

Last but not the least, I would like to express my deepest love for my parents and families who are thousands of mile away. Your unconditional and endless love make me behave stronger than I ever could imagine.

[1]Mathematics and Statistics Department
[2]Orchestre Symphonique de Montréal

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Ever since the introduction of the deep belief network by Geoffrey Hinton in 2006, the study of deep learning has entered a brand new era. Meanwhile, the significant growths in both computational power and contemporary techniques like dropouts etc. have made complex generative models more affordable for real-world tasks. Due to these achievements, Boltzmann machine, long short-term memory networks(LSTMs) and bidirectional RNNs etc. started regaining their popularities within the deep learning community. However, the deep generative model is still believed to be a less-developed sub-branch of deep learning study. There remain many ambitious goals and obstacles need to be achieved and overcame by making efforts.

We have seen plenty of exciting changes that deep generative models have brought to our life. Interior design Ghosh et al. [2017], DNA sequence design Killoran et al. [2017], human-face generation Goodfellow et al. [2014], semantic generation and translations between captions and images Pu et al. [2016] Reed et al. [2016] etc. all became realities due to the revival of deep learning studies. Compared with images and text, music is less investigated but enjoys many more potential applications such as music composition, music retrieval, music recommendation, melody recognition which are of great practical values.

Music composition is deemed as an artistic creation that computers would never have a chance to get involved with. It is still unpromising to expect machine-generated music would be as equally appreciated as those human composed. In spite of this bias, many convincing music pieces have been produced by computers using deep generative techniques. The most recent and impressive example would be the AI-composed Star Wars serenade symphony played at nVidia's Titan V launch party NIPS 2017[1]. Although the final work is tuned manually based on the AI generation, it shows a great success of AI music composition with the collaboration of the human and AI. AI is never designed to take over the world from human. Instead, by investigating advanced AI-generation techniques, we hope to unveil and overcome more and more challenging problems with the assist of artificial intelligence. Likewise, AI-generated music will never replace the authentic one but will help us better understand music and accomplish music-related tasks more efficiently.

The study scope of this thesis is limited to one of the most popular topics in automatic music composition – polyphony composition. Polyphony is a kind of music texture consisting of multiple independent melodies being played at the same time. A famous example of polyphony would be J.S. Bach chorales which contain four independent parts interplayed with each other. Polyphony is usually well-structured and therefore is suitable for pattern learning. However, it has very complex interactions between each part. The more parts it has, the harder to ensure its harmonic property. All voices put constraints on each other. Because of its rigid composing rules, polyphonic has less freedom in terms of composing. In other words, there would be more rules and patterns can be found in its creation. All these reasons make polyphony a suitable data type of building music generative models.

Specifically, J.S. Bach chorale, as one of the most typical polyphonic music is chose as our dataset for the following reasons:

1. Bach chorales have four distinguishable part structures: soprano part as leading melody which is harmonized by Alto, Tenor, and Bass parts. Besides, it has rigid rhythmical rules and there are exactly four subdivisions corresponding to a beat length in Bach chorales. These features allow us to introduce time segmentation into the data representation. Based on this design, blank segments could be used to represent pauses at any time slice without considering other parts' time alignments.

[1] https://blogs.nvidia.com/blog/2017/12/07/titan-v-launch/

2. Many preprocessed corpus of Bach chorales are already available within the community, which will accelerate the whole experiment cycle and enable us to compare the experimental results with others existing work.

3. J.S.Bach Chorale is one of the most popular corpora being used and tested in polyphonic music generation, which is deemed equivalently as MNIST and ImageNet when it comes to image processing.

It seems to be somewhat promising to deal with music generation given that such many advanced techniques are already available. However, it is not easy to apply the above techniques of processing images and text to music generation. Music is a unique data type and has its own structures and rules which makes it intrinsically different than images and text. For examples, a paragraph of text has a clear grammatical structure, but a music piece could enjoy a large variety of composing structures depending on composers' preferences. Besides, the ordering might not be a vital issue in some text-related tasks such as automatic semantic summarization, which enables the model to pay more attention to the meaningful keywords instead of the whole sentence. However, the same rule does not apply in the case of music. Any tiny permutation or modification of music notes may significantly affect the style and the quality of the music. Likewise, it is not that applicable to employ the data representation and corresponding models of images to solve music generation problems. As a common practice, an image is always treated as a matrix and is further unrolled into a multi-dimensional vector. The value of an element in the vector represents the basic color degree of the certain position. Taking operations of these values is often of practical usages such as dimension reduction via averaging neighboring pixels' values. However, if one concatenates different parts of music scores into one vector, the timing and rhythm information embedded in different parts will be damaged. Moreover, the number used to represent a certain note merely means the order of that note occurs in the bag of notes, which is not as meaningful as that of a color degree. Thus, there exist many limitations if we directly use available representations and methods to solve music generation tasks. This concern calls for more novel approaches to constructing suitable generative models for music generations.

Figure 1.1 (By Ian Goodfellow Goodfellow et al. [2014])presents a summary of existing generative methods of deep learning. As shown in the graph, the core of generative models is to maximize likelihood, i.e., to generate the most probable outcome given a 'target' density. There are mainly two subbranches to solve the problem depending on how much we know and how much we hope the samples rely on the 'target'. Examples of two extremes are fully visible nets and GAN(Goodfellow et al. [2014]). Both of them consider a tractable density and an implicit density respectively. The Markov chain method lies between the two approaches predefines a density as its target. Instead of strictly reproducing samples tat follow the target distribution, it aims to generate samples to approximate the target. Due to the limiting behavior of Markov chains, it is a very suitable method in music generation. More detailed reasons are shown below:

1. It is unpractical to know and learn the exact distribution of Bach chorale, no one can ensure the obtained dependencies are as accurate as J.S. Bach employed when he composed Lutheran hymns. Therefore, our target itself is an approximation of the real probability distribution.

2. Furthermore, we only hope to generate Bach-like chorales instead of exact Bach chorales. We need generated chorales to be of non-plagiarism property. Using MC sampling can introduce enough randomnesses to guarantee the diversity of the generated chorales.

3. From implementation and theoretical perspectives, sampling is the most tractable option among all choices. There are many statistical tools available for the tuning process of generation Krauth [2006].

More and more state-of-the-art studies of Bach chorales generation have been proposed in recent years using deep learning techniques. Some employ pure optimization tools to generate Bach-like chorales, other employ deep generative networks with statistical sampling methods. Some are developed based on expert knowledge of Bach chorales, yet some are completely agnostic involving very little background information of music.

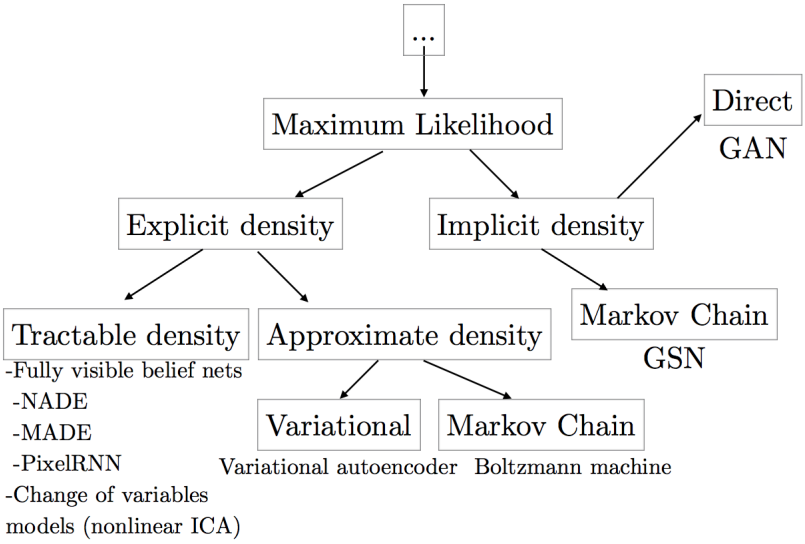


Figure 1.1 Generative Models

However, only few existing work pay attention to the feature of interaction between the generation system and its users. Composers may hope to be able to further modify the generated music. DeepBach is the only existing work that provides user-friendly interfaces for composers enabling further modification of the generated results. That is why we choose its plug-in to implement our work.

The study scope of this thesis is limited to agnostic learning-sampling for Bach chorale generation with recomposing interfaces. We proposed a Bach-like chorale generation system called SAM-Bach which consists of a stacked deep neural network and the SAM sampling. We construct a deep neural network structure to extract the composing patterns of the training chorales. Besides, it is capable of capturing the melodic and rhythmical styles of certain notes on certain parts which are fundamentally useful for the reharmonization tasks. Since during reharmonization, we need to generate three other parts given first part fixed. The ability of accurate style extraction will significantly affect the quality of the obtained chorale. Experimentally speaking, the SAM generated Bach-like chorales indeed enjoy major preference compared with other existing work as shown in the result of a survey with 248 human participants.

Moreover, the proposed SAM sampling is also superior in efficiency performance and convergence properties compared with the most commonly used method in Bach chorale generation – Gibbs sampling. Transitions in Gibbs sampling are made to approximate the high-probability area which belongs to a greedy strategy. Instead, SAM sampling allows accepting 'worse' proposal with a certain probability. The introduction of randomness enables SA sometimes to escape uncompetitive local optima (Givens and Hoeting [2012] section 3.3.1.2) and yields better results.

1.2 Contributions

The main contributions of this thesis can be summarized as follows:

1) To generate Bach-like chorales, we propose SAM-Bach system which contains two parts: a deep neural network for training and the SAM algorithm for sampling. There are total four voice parts in Bach chorales and a common practice is to contain four notes from each part per time slice and deem each part equivalently during the training process. However, notes from the same part actually play more roles while composing, especially in terms of melody and rhythm capturing. SAM system is constructed in a way where the notes on the same part are more emphasized than other parts, thus the predictions could preserve more information from the same voice compare with other models. That gives us the results with better melodic properties and especially maintain the leading melody flow when we reharmonize the soprano part. The second part of the system is SAM sampling, which is a combination of simulated annealing and Metropolis sampler.

2) We complete two tasks by using SAM-Bach system: 1) Generating Bach-like chorales from random noises; 2) Harmonizing three other parts given soprano melody. SAM-Bach is also able to extract metadata information (such as time signature, the fermeta and segmentations etc.) from any type of music score and therefore it is able to process any type of melody and generate Bach type chorales accordingly. We also reharmonized several pop songs and choose a typical example: *shape of you* to present the reharmonization performance of SAM-Bach.

3) Music evaluation depends on the knowledge, experience and tastes etc. of the evaluators. To evaluate a computer-generated music piece is no doubly even tricker. To thoroughly examine the value of our work, we conduct evaluations from three different perspectives.

(a) The first evaluation is a peer-oriented comparison. We compare our results with recent advance studies and original Bach chorale pieces via online survey. In the first group of comparison, among 248 valid responses, over 50% participants choose SAM-Bach chorales over others. Approximately 40% participants prefer SAM-Bach to the real Bach chorales, even including Bach experts.

(b) The second is a Bach-oriented assessment by Bach expert. The generated chorale and Bach reharmonization are evaluated by 147 and 248 participants respectively via a online survey. The generated chorales produced by SAM and Gibbs share fairly equal aesthetic appreciation among the participants. However, in terms of Bach reharmonization task, SAM-Bach wins slightly more votes than other sampling methods as shown in figure 5.5.

(c) The third evaluation is about sampling performance. We compare the convergence property and computation time of both SAM and Gibbs sampling. Common metrics for computing statistical difference such as KL divergence and total variation distance are not applicable in this case because the distribution to be approximate here is a conditional distribution which varies as the condition changes. Our target distribution is dynamically changing during the sampling process and it is meaningless to compare two items under different settings. To be able to quantify the convergence performance, we proposed a CP Ratio as the metric which is essentially a modified KL divergence indicator. Besides, to make sure the SAM algorithm does not degrade the quality of generated chorales, we conduct another online survey for this evaluation. The corresponding result shows that chorales created by SAM and Gibbs share almost the same preference in the human evaluation of 147 people. SAM sampling shows faster convergence performance while maintaining the quality of generated Bach-like chorales.

1.3 Organization

The remaining chapters are organized as follows:

Chapter 2 presents several current state-of-the-art approaches of music generation by category. General methods and related work are summarized and compared using various formats. Differences between our work and existing work are also discussed.

Chapter 3 covers the background of multi-fields from probability theory to statistic inference, and machine learning to music theory etc. which consist the knowledge system of the proposed SAM-Bach architecture. A knowledge graph is provided to visualize the relations between methodologies and concepts.

In chapter 4, the SAM-Bach system is presented and illustrated. Bach-like chorale generation is conducted in two steps: distribution estimation and sampling corresponding to the two parts of the architecture. Date preprocessing is also mentioned before introducing the system.

Chapter 5 provides details about our experiment implementations and three types of evaluations.

Conclusions and future work are addressed in the final chapter.

Chapter 2

Related Work

2.1 Polyphonic Music Generation

Polyphony has been deemed as a peak of Western culture and it usually refers to the music during the Renaissance period. Polyphony is a kind of music texture consisting of multiple independent melodies being played at the same time. The overall sound and quality of a polyphony are shaped by the combination of its melodies, rhythms, and harmonies. Nowadays, the term polyphony is used in a broader way, in which it can refer to any musical textures that are not monophonic. Polyphonic music is often obtained by composing several harmony parts given certain leading melody. Different combinations of the melody and harmony produce various styles of music. Different composers may have different composition patterns. There is no exact ruleS of harmonization. The abstract nature of polyphony composition makes it tricky to learn and evaluate. It has been considered as an artistic task which is achievable only by the human with solid relevant knowledge and experience.

Not until middle 19th century, automatic polyphonic music composing become more and more achievable due to prosperous development of deep learning (Todd [1988],Todd [1989]). This chapter provides an overview of significant milestones and recent state-of-the-art results in both deep generative modeling and polyphonic music generation. Deep generative models can be seen as the major branch of deep learning providing solid theory skeleton in various generative tasks. The green texts in figure 2.1 correspond to the previous milestones of deep learning since 1980s. The blue ones are the representatives of polyphonic music generation.

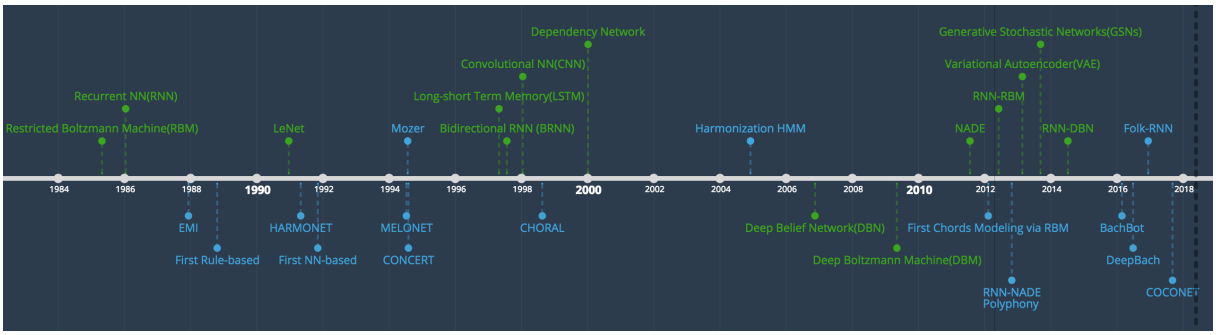


Figure 2.1 Milestones of deep learning and polyphonic music generation

As mentioned earlier, polyphonic music generation belongs to music generation which is a subbranch of music information retrieval (MIR). There are several terms either being used interchangeably or sharing very similar solutions to polyphonic music generation. Examples are composition style imitation, chord invention, and chorale reconstruction/completion etc. We list three main alternative topics of polyphonic music generation in figure 2.2.

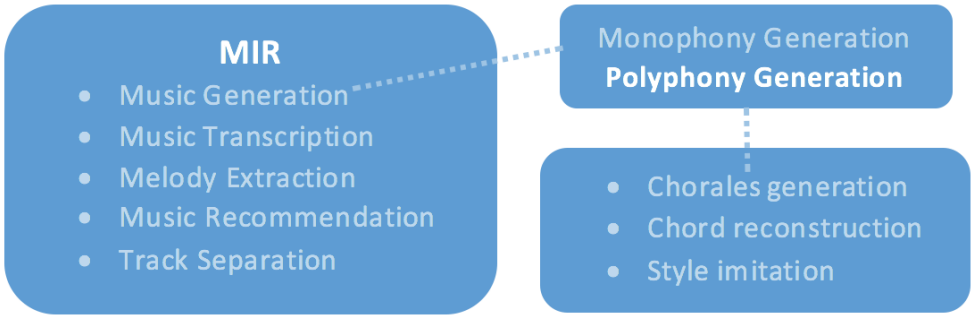


Figure 2.2 Categorization of MIR

The methods of polyphony generation can be simply divided into two paradigms: knowledge-based and non-knowledge-based(agnostic). Figure 2.3 shows a simple division of existing work, from which we know that not every NN-based approach is agnostic. Two dominant genres of cutting-edging representatives in polyphony generation are RNN-RBM(Boulanger-Lewandowski et al. [2012]) and RNN-DBMv(Goel et al. [2014]) as described in Liang [2016]. We will introduce and compare these contemporary generative approaches to polyphonic music generation by genre.

The scope of interest of this paper is within the range of deep agnostic generative models. These models are trained to predict the pattern of notes being played together at a certain time slice, given previous or surrounding notes information. Several rule-based methods are also listed here to compare with agnostic methods.

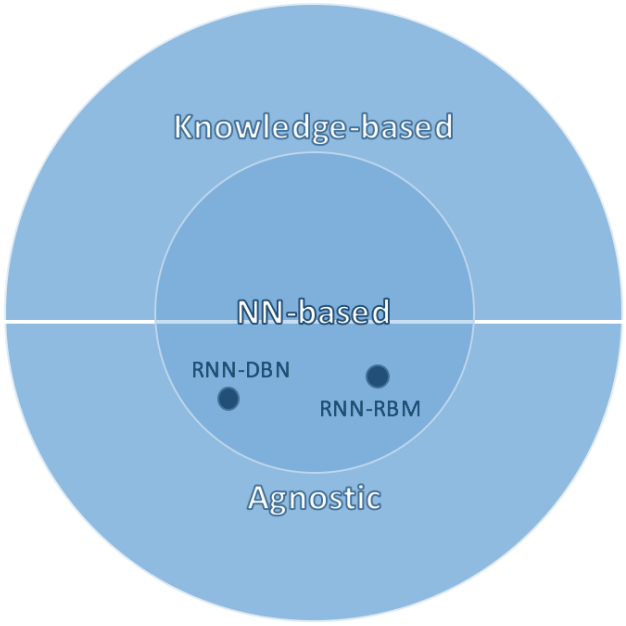


Figure 2.3 Classification of methods of polyphony generation

2.1.1 Knowledge-based Methods

Early polyphony generating methods were quite straightforward and require strong expert knowledge which relies severely on human involvements (Cruz-Alcázar and Vidal-Ruiz [1998],Ebcioğlu [1988],Quick [2014],Spangler et al. [1998],Tsang and Aitken [1991]). One of the most significant work at that time was called 'Experiments in Music intelligence' which was designed to produce music by recombining music pieces from a database.

The first mathematical optimization model for automatic polyphony generation is proposed by K. Ebcioğlu in 1998Ebcioğlu [1988], and named it as CHORAL. CHORAL is a knowledge-based method as it employed knowledge of harmonization and composition as the constraints in a optimization problem. The first well-acknowledged NN-based approach for Bach-like harmonization is presented by Hild in 1992 (Hild et al. [1992]) which is also rule-based. The authors trained multiple networks to tackle with different subtasks in Bach chorale respectively. The results were obtained by refining a predefined Bach-like skeleton.

Eppe et al. [2015], Hild et al. [1992] and Allan and Williams [2005] employs manual ontology for music composition. Another worth-mentioning workKaliakatsos-Papakostas and Cambouropoulos [2014] uses HMMs but only applicable for homo-rhythmic sequences. Polyphonic music generation can be also achieved by dynamic programming. Pachet and Roy [2014] solves a optimization problem with constrained Markov chains.

2.1.2 NN-based Agnostic Methods

However, since the rules and constraints mentioned above need to be imposed manually, it is inevitably to have biases in the generation process. Even with minor permutation in data, the same method may yield different results under different settings. Therefore, more and more attention has been put on the non-knowledge-based solutions to solve polyphonic music generation tasks.

As the development of deep generative models, more and more agnostic NN-based state-of-art generation approaches for Bach chorales are invented. The structures of NNs employed for generative tasks tend to be more and more complex yet still under computational budget due to the improvements in computational power.

However, computational power is not the only vital factor in polyphony generation. The architecture of the employed network also plays an important role in this game. The most significant workBoulanger-Lewandowski et al. [2012] in the development of NN-based agnostic methods is proposed by Boulanger-Lewandowski and Yoshua Bengio in 2012, which employs restricted Boltzmann machines(RBMs) and recurrent neural networks(RNNs) to model and train the distribution embedded in chorale chords. It presents an RNN-RBM model to learn the harmonic and rhythmic probabilistic rules from polyphonic music.

The major contribution of Boulanger-Lewandowski et al. [2012] is that this work overcomes the limitation of RNNs in polyphony generation by proposing to use energy-based model. Chorales are often high-dim sequences which causes the conditional distributions to be learned is often multi-modal. It is very complex and impractical to specify all possible configurations of the variables. Boulanger-Lewandowski et al. [2012] first points out the accurate requirements of automatic music modeling – for most of the time, we do not need to know the explicit probability distribution but we do need to measure the value of joint distributions. Negative log-likelihood is able to express the probability by defining an energy function. It is also the first study using inference in RBM that boosts the efficiency of chords generation. Variants of Boulanger-Lewandowski et al. [2012] are Liang [2016] and Chung et al. [2014] which employ long-short term memory(LSTMs) and gated recurrent unitS(GRUs) respectively. Liang [2016] focuses on optimizing system structure by finding the best combination of model parameters such as the size of units in each layer and the depth of the network etc. Stacked LSTMs are employed to capture temporal dependencies. Although the model is able to produce very convincing chorales, it has limitations in extendibility and might contain bias of the training data. The structure used in our work aims to pursue broader flexibility and versatility of the generative model.

References	**Rule-based**	**NN-Agnostic**	**Problem**	**Methods**
Sturm et al., 2003 Sturm et al. [2016]	X		music composition	LSTMs
Eppe et al., 2015 Eppe et al. [2015]	X		chord invention	algebraic specifications
Kaliakatsos et al., 2014 Kaliakatsos-Papakostas and Cambouropoulos [2014]	X		harmonization	HMMs
Pachet et al., 2014 Pachet and Roy [2014]	X		polyphony generation	constrained Markov chain
Allan et al., 2005 Allan and Williams [2005]	X		music composition	manual ontology
Lyu et al., 2015 Lyu et al. [2015]		X	chord invention	GRUs
Liang et al., 2016 Liang [2016]		X	chorale generation	LSTMs
Sakellariou et al., 2017 Sakellariou et al. [2017]		X	melodic modeling	max-entropy + Gibbs
Huang et al., 2016 Huang et al. [2016]		X	chord completion	NADE + Block-Gibbs
Hadjeres et al., 2016 Hadjeres and Pachet [2016]		X	chorale generation	RNNs + Gibbs
Boulanger et al., 2012 Boulanger-Lewandowski et al. [2012]		X	polyphony composition	RNN-RBM
Chung et al., 2014 Chung et al. [2014]		X	polyphony generation	GRUs
Hadjeres et al., 2016 Hadjeres et al. [2016]		X	polyphony generation	RNN-DBN

Table 2.1 Summarization of Bach chorale generation

Another milestone of study Hadjeres et al. [2016] achieves polyphonic music generation by imitating the style of Bach chorales. The authors construct a graphical model to learning the probability distribution as the style of authentic Bach chorales by using maximizing cross-entropy principle(Jaynes, [1957]). The cross-entropy is essentially the log-likelihood of the generations. The work is aimed to model the probability dependencies of a note in any Bach chorales given its neighboring information.

The most state-of-the-art Bach chorale generation DeepBachHadjeres and Pachet [2016] is developed upon the previous work. It uses an RNN-DBN network for the dependency learning and Gibbs sampling for chorale producing. Besides it provides an interactive interface for composers which enables further tuning of the generated chorals. In spite of these contributions, there exist several potential improvements yet to be full filled. For example, in the human evaluation process, instead of letting participants choose the most Bach-like chorale given several options, the authors of Hadjeres and Pachet [2016] have each piece of music listened by human and let them decide if it is either computer-composed or human-composed. Votes of DeepBach and other methods are collected and compared. The results show that the generated chorales via DeepBach still have not beat the original ones. Only less than a half participants think DeepBach chorales are the real ones.

Besides, although Gibbs sampling is believed to be an efficient sampling method as an MCMC method, the implemented version in Hadjeres and Pachet [2016] is pseudo-Gibbs. The authors use the approximations of the conditional probability distributions instead of the true conditionals of a probability distribution on whole sequences. Furthermore, Gibbs sampling is a greedy algorithm and does not guarantee to stop at global optimal.

Another recent work about Bach chorale modeling is COCONET Huang et al. [2016] that employ orderless NADEUria et al. [2014] for density estimation and blocked Gibbs sampling for Bach chords reconstruction. NADE is a deep net allowing direct ancestral sampling proposed by Uria and Benigno etc. It is the first application of NADE on polyphonic music completion. However, this paper does not address the convergence behavior of the employed Gibbs sampling and the result of human evaluation is very obscure and not supporting their conclusion. The readers also have no clues of how the process the training data. The degree of the sparseness of the neighboring notes to the generation performance is not mentioned either.

Having reviewed and compared with these remarkable studies of Bach chorales generation, the proposed SAM-Bach system has the following refinements:

1) We elaborate the process Bach chorales preprocessing and explain the reasons why we choose the data representation.

2) We implement RNN-DBN structured graphical models for inference training with better network architecture which considers the importance of notes on the same part.

3) For more efficient sampling, we replace the common practice – Gibbs sampling by the proposed SAM algorithm, which is an MCMC method combing simulated annealing and Metropolis sampler.

4) To address the obstacles in terms of the evaluation of Bach chorale generation, we use a modified KL-divergence distance as a measure to investigate the convergence performance.

2.2 Generation via Sampling

One may be curious about how did sampling become the mainstream method of polyphonic music generation. How does the development of generative models contribute to the study of music generation? What makes MCMC method stand out of all sampling algorithms? This section illustrates answers theses concerns by introducing different sampling approaches and how they relate to the music generation.

2.2.1 Sampling and Deep Generative Models

Reference Boulanger-Lewandowski et al. [2012] explains how to employ sampling methods to generate sequences such as notes in the music score, words in text and images in the video. The authors also emphasize the importance of the multi-modal nature of the conditional distribution embedded in the high-dim sequences in generative tasks.

In the content of Bach chorales, the variables are the notes being played at the current time slice. The condition contains the latent information of notes before or after current time step. It is not hard to know how costly it would be to enumerate all possible combinations of conditions.

To generate new samples based on the learned conditional distribution, there are in general two paradigms: 1) Exact sampling and 2) Approximate sampling. However, it is too expensive and unrealistic to conduct an exact sampling as we need to know the exact conditional information. Approximate sampling is more feasible in this case.

A new challenge of using approximate sampling would be the estimation of distribution which is crucial for the training procedure. Since we still need to know the likelihood of given configuration and adjust our model parameters based on it. This barrier motivates the favor of energy-based models to quantify the degree of likelihood. Restricted Boltzmann machineS(RBMS) and Deep Belief Networks(DBNs) became popular among all various energy functions, though DBNs are no longer widely used recently.

2.2.2 Sampling in Inference

This part will mainly discuss Monte Carlo Markov chain(MCMC) methods for sampling and why it is always selected for generative tasks.

MCMC was first proposed as a Monte Carlo methods to generate samples from certain distributions that approximate our target distribution. However, it has been widely and frequently used just for sample generation Givens and Hoeting [2012].

Unlike other closed-form methods, MCMC is an iterative algorithm and enjoys sufficient versatility to be applied to many complex tasks. The sampling principle of MCMC methods is to create an ergodic Markov chain with stationary distribution goes to the target distribution. The obtained the sequence of samples are the approximate realizations of marginal distributions. The most vital property of MCMC sampling methods is its limiting behaviorHastings [1970].

Nevertheless, classic MCMC methods neglect information in previous iterations. The proposed SAM algorithm incorporates previous information while generating the current proposal. Thus the old information interacts with the future ones which allows us to compare and update the samples. SAM is a train-based/interacting MCMC method. It has been mathematically and empirically proved that interacting MCMC method can accelerate the convergence rate compared with general MCMC approaches on sampling task Tahmasebi et al. [2016].

2.2.3 Simulated Annealing in Inference

Recall the graph of generative models provided in the last chapter. The underlying task of generative models is maximizing log-likelihood estimation(MLE), which is essentially an optimization problem. Simulated annealing(SA) is a meta-heuristic optimization method defined over discrete domains Kirkpatrick et al. [1983]. The mechanism behind SA is to draw samples from a sequence of distributions scaled with various temperatures. The only difference between SA and MCMC is that MCMC runs under a fixed temperature, while SA allows us to tune the value of temperature. Regular SA has its drawback as it is a stochastic approach and the convergence rate can be too slow for a practical annealing schedule.

Many variants of SA have been proposed to accelerate the searching processes, such as Cauchy annealing, simulated annealing, generalized SA, and SA with known global value. The stochastic approximation Monte Carlo (SAMC) algorithm Liang et al. [2007] is adaptable to adjust the accept rate of proposals depends on its rejection situations. The proposed SAM does not belong to SAMC, so we employ a different abbreviation to distinguish from SAMC.

The proposed SAM can be seen as an interacting simulated annealing methods Gall et al. [2008] that allows us to control the progress of sampling.

Chapter 3

Prerequisites

Polyphonic music generation as an interdisciplinary task involves knowledge from multiple fields including music, machine learning, statistical inference, graphical modeling etc.

This chapter is mainly deemed as a reference material aiming to help readers get familiar with the fundamentals of sequence generation via deep learning, especially polyphony generation. Readers may skip this chapter or any sections then return back as needed.

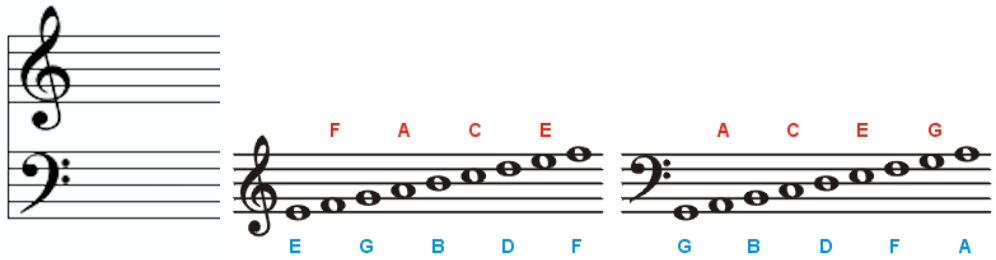
3.1 Basics of Music

3.1.1 Music Theory Essentials

This section covers several basic music notations appeared in the thesis. For more systematic knowledge regarding with music composing please refer to reference Surmani et al. [2004]. Although music generation is realized by agnostic approaches which require no knowledge of music in our work, knowing the essential knowledge of music theory would help read music sheets presented in this paper. Readers may skip and refer back to this section when needed.

Pitch and Step

In music notation system, a pitch represents the highness or the lowness of a sound. The range of pitches are naturally infinite, however, for simplicity, musicians employ merely seven basic pitches to cover the whole pitch range, representing by seven capital letters: A, B, C, D, E, F and G (more often as C, D, E, F, G, A and B) in ascending order by pitch from low to high respectively. This set of letters will be repeated in the same order to reach hight or lower pitch classes. Each repetition is called an octave. To specify a pitch class, one would use a number in subscript to indicate which version the pitch is. The distance between C and D is called a whole step. E and F are separated by a half step. These terms will be useful when we introduce accidentals later in this section.



(a) Staff and clefs (b) Treble staff placement (c) Bass staff placement

Figure 3.1 Pitch placement on different staffs

1) The Staff: It consists of five horizontal lines with four even spaces between them as shown in the figure(a)3.1. Each line/space has been assigned to a specific pitch. Notes will be represented above/on/between/under the lines of the staff depend on what they are.

2) Clef: Clefs are musical symbols, conventionally being put at the start of the staff, denoting the pitches of the written notes on the corresponding staff. There are mainly three types of clefs defined in the notation system: G-clef, C-clef, and F-clef. Only two of them are used in this paper, which can be seen in the figure 3.1 (a) (left side of the staff). The upper one is G-clef which is placed on the second line of the stave indicating the second line will always be F on this staff and the rest of the pitches can be therefore deduced upon this. As a convention, G-clef placed on the second line on the staff is often called the treble clef. The lower clef is F-clef being placed on the fourth line of the staff. Similarly, the line it being placed on determines where F pitch is. This clef is known as bass clef.

3) Pitch Placement: high pitches will be put higher on the staff, and the vice versa. The placement also depends on the type of the clef at the front. Figures 3.1 (b) and (c) show the scenarios of two different placements.

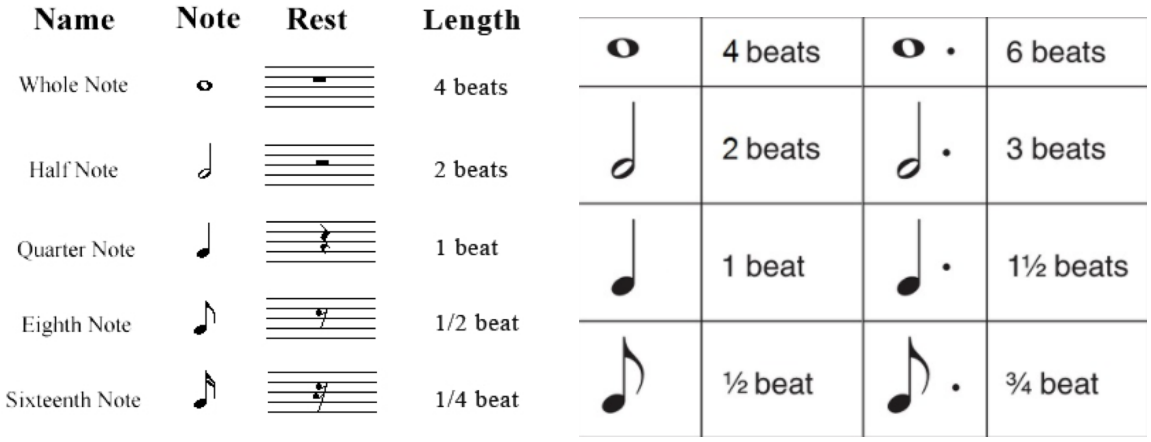
Notes and Rests

Notes are the musical symbols used to be placed on the staff to represent pitches. They seem to be very similar to pitches, so why introduce notes?

Figure 3.2 Note components

A note consists of three parts: note head, stem and flag as shown in figure 3.3. Different combinations of their variations denotes different notes values or say durations. Rests are the symbols indicating a certain duration of silence.Figure (a) 3.3 shows the definitions of basic notes and rests. Figure (b) 3.3 shows dotted notes and their durations. There are a few rules can not be seen from the lists:

- When the stem is above the note head, flags are always on the right side of the stem.



(a) Basic notes and rests (b) Dotted notes

Figure 3.3 Basic and dotted notes and rests

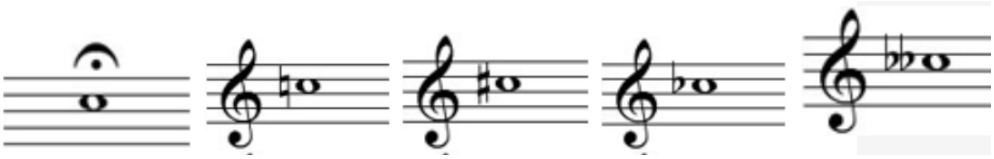
- A dot means adding half of the note value to itself, not exact a half beat. See figure 3.3 (b) for details.
- In British music notation system, notes have different names but same values. Here just list the American ones.
- Some notes may consist of two notes connected with a bar of both stems. This is called beaming which means that two or more flags are glued together. Beaming is just a compact form of flags.

Measure and Time Signatures

A measure in music refers to the space between two vertical bars on a staff. A measure contains beats and rhythms according to the time signature. Time signatures often appear in a pair of numbers written at the very beginning of the staff, symbolized as two numbers stacked together, indicating the type and the quantity of notes contained per measure. The upper number denotes the number of beats per measure. The lower number determines how small a beat is to be. The value per beat equals the value of a whole-note(of value 4)divided by the lower number in the time signatures. One often finds the term '**simple time signatures**', as opposed to other types of time signatures: compound, complex, mixed, etc. Simple time signatures are the easiest to count because a one-two pulse in a piece of music feels the most natural to a listener and a performer. Common examples are 4/4, 3/4, 2/4, 3/8, and 2/2. With simple time signature, beats can be divided into even divisions of two with 2, 3 and 4 as the top number (such as 2/2, 2/8, 3/2, 3/8, 4/2, and 4/8). By contrast, compound time signatures can be divided into three notes. Examples of simple signatures are shown as follows:

- 3/8 time means that the measures are in 3/8 time, i.e. there are three eighth-note beats in each measure/bar since $4/8 = 1/2$ which corresponds to the value of an eighth-note.
- 4/4 time means there are four quarter-note beats in each measure/bar since $4/4 = 1$ which corresponds to the value of a quarter-note. This kind of time signature is also called common time.

Fermata and Accidentals



(a) Fermata (b) Natural (c) Sharp (d) Flat (e) Double flat

Figure 3.4 Fermata and accidentals

- Fermata: is always placed above the note/rest to be held.

- Accidental: are symbols used to raise up or lower down a normal note pitch.

  1. Sharp: raises a note by a half step.
  2. Double sharp: raises a note by a whole step.
  3. Flat: lowers a note by a half step.
  4. Double flat: lowers a note by a whole step.
  5. Natural: cancels out any accidental of the same note set ahead.

  Therefore, C-sharp and D-flat would be the same note.

3.1.2 Music Representations

In most applications of music generation using either deep learning or other generative methods, music contents are often encoded into symbolics which is processable by computers. Symbolic representations define standard notations of audios which not only guarantees the better preservation of music, but an efficient interaction between musicians and music pieces. Symbolic music representation also enjoys its ease of incorporation with various deep learning models and transformations between different symbolic notations. Now we introduce three popular symbolic representations used in practice.

MIDI Representation

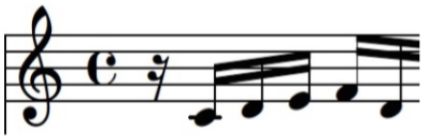
Musical Instrument Digital Interface(MIDI) is the most frequently used symbolic notation in music generation. MIDI is actually a technical standard which defines the digital interface and connectors between various software, electronic devices, and instrumentsRothstein [1995]. Information can be stored in MIDI range from note information(e.g. pitch) to control signals(e.g. volume). We will focus on the **Channel Voice** type of message in this section, yet there remain three other types of messages that can be stored as MIDI data.

(Note on or off, channel number, note's pitch, note's velocity). (3.1)

An example MIDI (Note on c, 2, 60, 50) indicates start playing a middle C with velocity 50 on channel 3. Note that the number is not directly translated to the real info, instead, there is a dictionary-like lookup table defined for these numerical notations. Every note event will be embedded into a **track chunk** which also includes timing information. Another example of MIDI file and its original score are shown in figure (a) and (b). Nevertheless,

```
2, 96, Note_on_c, 0, 60, 90
2, 192, Note_off_c, 0, 60, 0
2, 192, Note_on_c, 0, 62, 90
2, 288, Note_off_c, 0, 62, 0
2, 288, Note_on_c, 0, 64, 90
2, 384, Note_off_c, 0, 64, 0
2, 384, Note_on_c, 0, 65, 90
2, 480, Note_off_c, 0, 65, 0
2, 480, Note_on_c, 0, 62, 90
2, 576, Note_off_c, 0, 62, 0
```

(a) Readable MIDI file



(b) Original score

Figure 3.5 MIDI representation

there exists a salient disadvantage of using MIDI to store musical pieces. MIDI unrolls and concatenates track chunks end-to-end, it may not be possible to well record multiple notes being played simultaneously in different tracksHuang and Wu [2016]. Therefore, it is not easy for a model to learn this synchronized pattern if it is fed up with MIDI representations.

Piano-roll Representation

A picture of **piano roll** presentation is showed in figure 3.6, which is depicted in two dimensions and can be used to represent polyphonic music content. Unlike MIDI format, piano roll does not explicitly indicate the ending of a note. One may not able to tell if a bar stands for one or more notes with same pitch.

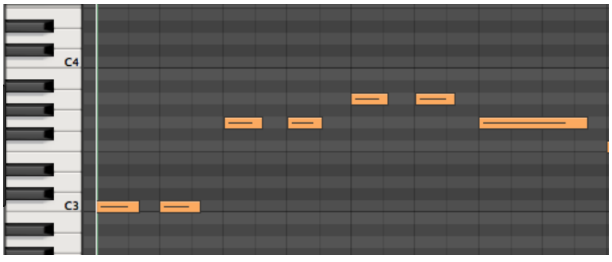


Figure 3.6 Piano roll representation

3.1.3 J.S. Bach Chorales

The purpose of this section is to help readers better understand the Bach chorale dataset used in the following chapters. Figure 3.7(From Hadjeres and Pachet [2016]) shows the original melody lyrics composed by Georg Neumark in 1641 (left) and its four-part Bach chorales (right) by reharmonization. One can easily tell the original melody is kept as the soprano part in the chorale. There are several other rules for Bach chorale harmonization. The four-part Bach chorales(soprano, alto, tenor and bass) is actually a harmonization of

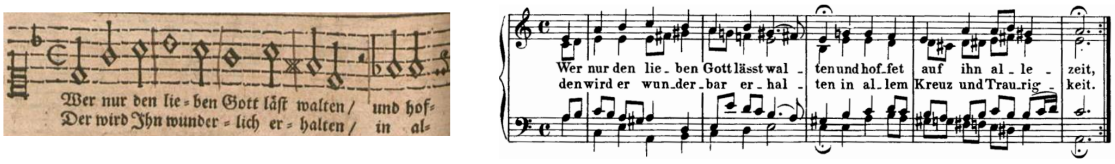


Figure 3.7 J.S. Bach chorales harmonized from a melody

the famous Lutheran hymn with Lutheran melody being the highest part, i.e., soprano. Music data used in this work is from a Python library music21[1] contributed by School of Humanities, Arts and Social Sciences at M.I.T. , which provides a corpus of 389 pieces of Johann Sebastian Bach chorales with approximately 400 minutes in totalBach [1985].

3.1.4 Requirements of Bach Chorale Representations

Having introduced the available music representation methods and the main features of Bach four-part chorales. Let's consider the selection of representations for our task. The data type need to be compatible and easy to work with learning models and maintaining the most vital factors of the original musical content. There are several critical issues need to be considered in the score transformation.

[1]Music21 is a Python-based toolkit for computer-aided musicology, enabling people to quickly and easily study the structure of certain music pieces. It also contains a corpus of music files that are available for downloading. The J.S. Bach chorales used in this thesis is from this corpus.

Temporal Scope

Timing is fundamentally important as it is directly related to the feeling expressed by a music piece. With the notation of time, one can better represent a music piece along with its timing information. The most common way is to set time-slice to the smallest duration of among all notes.

Quantization

Quantization is a term in digital music processing, which transforms musical notes to other representations. It is important since the criteria of quantization may vary from person to person, i.e. there is not a standard rule to do quantization and therefore the obtained representation may contain bias, imprecisions or expressions that may further affect the experiment performance. Quantization guarantees the beat-accurate timing of sounds. The note on/off messages in MIDI is an example of quantization in practice.

Note Duration

Note duration can be specified by a starting and an ending notation. Its usage lies in distinguishing two consecutive notes and a single notes that are of the same duration. Take piano roll representation as an example where there is not a specific ending of notes, for two consecutive quarter notes and a half note, the piano roll representations will be identical. One solution is to add a flag indicating the endings of notes. An alternative one is specifying the starting and the duration of a note.

Meta-data

Metadata refers to the data aside from music notes, such as the fermata, time signature, harmonics even the instrument played by each voice part. Proper employment of the metadata may improve the learning and generation performance.

Consider all these requirements and the existing methods for music representation, we employ a symbolic tuple-form representation that combines the advantages of several symbolic representations yet has its own strength in segmentation capture. We will introduce this representation in detail in chapter 4.

3.2 Probability Theory

This section will review some important and useful concepts in both probability theory and computational statistics. This section will only address the discrete case since the music pitches are naturally discrete. Without additional explanation, Ω denotes the sample space with values from set S. All random variables and events will be written in capital letters, and lowercases are used to denote realizations of random variables. $P(X)$ and $F(X)$ refer to the probability mass function and the distribution of X respectively.

3.2.1 Probability Measure

This section will clarify what components make a probability distribution that enables to quantify and analyze the randomness in the world. The definition of a probability measure, aka probability distribution for a random experiment, is a real value function that defined on a collection of events, and satisfies the following criteria:

1) $P(E) \geq 0$ for any event E.

2) $P(S) = 1$.

3) If the collection of events is countable, pairwise disjoint, then $P\{\cup_{i\in I} A_i\} = \sum_{i\in I} P(A_i)$

3.2.2 Distributions

Joint and marginal distribution

Suppose random variables X and Y take values in Q and T respectively. Now we bundle (X, Y) as a new random variable taking values in set $Q \times T$. The probability distribution of (X, Y) : $P(X, Y)$ is called **joint distribution**. The distributions $P(X)$ and $P(Y)$ are called **marginal distributions**.

Conditional probability distribution

The conditional distribution of X given Y is the probability distribution of X and based upon Y is a specific known value. Note that marginal distribution of Y: $P(Y)$ has to be non-negative as the definition of the conditional probability mass function is:

$$P(X = x|Y = y) = P(X = x|y) = \frac{P(x, y)}{P(y)} \tag{3.2}$$

Note that the condition remains a fixed realization. The conditional distribution is also a probability measure(can be verified by checking the three properties), therefore results that holds for probability measures can be generalized to conditional probabilities. There are several prominent results derived from conditional distribution:

1. Chain rule/ General product rule

$$P(A_1, ..., A_n) = P(A_1|A_2, ...A_n)P(A_2, ..., A_n) \tag{3.3}$$

   If we keep expanding the conditional probability according to equation (3.2), we obtain the following product form:

$$P(\cap_{i=1}^{n} A_i) = \prod_{k=1}^{n} P(A_k| \cap_{j=1}^{n-1} A_i) \tag{3.4}$$

   This equation 3.4 allows our to compute any joint distribution of a collection of random variables using conditional probabilities.

2. Bayes' theorem If we slightly manipulate with the definition of conditional probability (3.2) and just consider an event A_i and evidence B, then if we build an equation by leaving the joint distribution intact, it will yield a formula that calculates the probabilities of the occurrence of event A_i given evidence B:

$$P(A_i|B) = \frac{P(A_i)P(B|A_i)}{P(B)} \tag{3.5}$$

3. Law of total probability The law of total probability provides a way to obtain marginal probability by summing up conditional probabilities. Assume that event A has $A_1, ..., A_n$ partitions, then the marginal probability of event B can be expanded as following by the definition of conditional probability distribution:

$$P(B) = \sum_{i=1}^{n} P(A \cap B_i) = \sum_{i=1}^{n} P(A_i)P(B|A_i) \tag{3.6}$$

4. Bayes' theorem variant By using equation (3.6), the regular Bayes' theorem can be further written as:

$$P(A_i|B) = \frac{P(A_i)P(B|A_i)}{P(B)} = \frac{P(A_i)P(B|A_i)}{\sum_{i=1}^{n} P(A_i)P(B|A_i)} \tag{3.7}$$

3.2.3 Independences

The most exciting invention in probability theory is independence, as it provides access to people dealing with complex real-world problems by making independence assumptions.

Complete independence

It describes a pairwise independence between any two random variables. That is to say there's no association among any two random variables. This kind of model is often denoted as (X,Y,Z). Mathematically, if the joint probability of r.v. X and Y is equal to the product of their margin probabilities:

$$P(X = x_i, Y = y_j) = P(X = x_i)P(Y = y_j), \tag{3.8}$$

which let us to rewrite conditional probability into:

$$P(X = x_i|Y = y_j) = \frac{P(X = x_i, Y = y_j)}{P(Y = y_j)} = \frac{P(X = x_i)P(Y = y_j)}{P(Y = y_j)} = P(X = x_i). \tag{3.9}$$

When we say a sequence of random variables is independent and identically distributed(i.i.d.), it implies complete independence.

Joint independence

This concept is less frequently mentioned compared with other independence. Joint independence indicates that two random variables X and Y are jointly independent of a third one Z or more. This kind of independence may be useful when two random variables may be possibly related, yet both are independent of a third. Formally, the joint probability we can get from (XY,Z) model would be equivalent to the product of the marginal probabilities from XY margin and Z margin:

$$P(X = x_i, Y = y_j, Z = z_k) = P(X = x_i, Y = y_j)P(Z = z_k) \tag{3.10}$$

When the mutual independence (X, Y, Z) holds, then the models of joint independence: ((XY), Z), (X, (YZ)) and ((XZ), Y) will also be true as well. That is fundamentally why (X, Y, Y) is often employed to have a simplify the model.

Conditional independence

Formally, we say a random variable X is conditionally independent of another random variable Y given the value of the third random variable Z, if for every possible value for x_i, y_j and z_k, we have

$$P(X = x_i|Y = y_j, Z = Z_k) = P(X = x_i|Z = Z_k) \tag{3.11}$$

Given a specific value of Z, our belief in the value of X will not be altered even knowing the value of Y. The knowledge of Y doesn't give us additional information on the belief of X given the value of Z.

3.2.4 Convergence

This section distinguishes two major convergences in probability theory. The weaker type of the two is called **convergence in distribution**, the stronger one is called **convergence in probability**. However, the strength of convergence relates neither with its usefulness nor with the hardness of achievement.The central limit theorem is actually about the convergence in distribution but is still very powerful and widely used. Many methods are based upon these two convergences as such MCMC sampling method.

Convergence in distribution

We claim that the distribution of a random variable X_n converges to the distribution of X as $n \to \infty$ if

$$F_n(x) \to F(x) \text{ as } n \to \infty \tag{3.12}$$

Convergence in probability

1. Convergence in probability
   One may claim that $X_n \to X$ as $n \to \infty$ in probability if

$$P(|X_n - X| > \epsilon) \to 0 \text{ as } n \to \infty \text{ for } \forall \ \epsilon \in (0, \infty) \tag{3.13}$$

2. Converge with probability one
   We introduce a stronger type of probabilistic convergence which will be further addressed later this section.

$$P(x \in S : X_n(x) \to X(x)) = 1 \text{ as } n \to \infty \tag{3.14}$$

3.2.5 Two Fundamental Theorems in Probability Theory

Given a sequence of i.i.d. random variables real-valued $(X_1, X_2, ...)$ that sampled from the same distribution with mean $\mu \in (-\infty, \infty)$ and standard deviation $\sigma \in (0, \infty)$. Denote M_n as the average value of n sampled variables and S_n the standard deviation of these samples. Then we have the following two statements.

The laws of large numbers

1) The strong law of large numbers
   This theorem states that as the number of samples goes to infinity, the average of samples approximates to μ with probability one (see equation 3.14), i.e. :

$$P(M_n \to \mu) = 1 \text{ as } n \to \infty \tag{3.15}$$

2) The weak law of large numbers
On the contrast, the weak law of large numbers is about convergence in distribution (3.12) and in probability (3.13). Mathematically:

$$\lim_{n\to\infty} P(|M_n - \mu| < \epsilon) = 1 \text{ for } \forall \ \epsilon > 0 \tag{3.16}$$

We say a sequence of random variable $X_1, ..., X_n$ converges almost surely to the random variable X if we have:

$$P(\lim_{n\to\infty} |M_n - \mu| < \epsilon) = 1 \text{ for } \forall \ \epsilon > 0 \tag{3.17}$$

The central limit theorem

The Center Limit Theorem tells that the sum of n i.i.d distributed random variables $X_1, X_2, ..., X_n$ with mean $E(X_i) = \mu$ and $Var(X_i) < \infty$, then the random variable $Y_n = \frac{(\bar{X_n} - \mu)\cdot\sqrt{n}}{\sigma}$ will converge to the normal distribution $N(0, 1)$:

$$\sum_{i=1}^{n} Y_i \to N(0, 1) \ \ as \ \ n \to \infty. \tag{3.18}$$

The normalized sum of independent random variables will eventually approximate to a normal distribution even the original data are not normally distributed. Therefore many statistical methods designed for normal distribution can be applicable for certain other types of distributions. This theorem also enables us to approximate the desired distribution with very little knowledge about the unknown distribution that we sample from.

3.2.6 Maximum Likelihood Estimation and Maximum a Posteriori

Both of these two estimation methods are designed for variable estimations of statistical models. More frequently people would use them without realizing it as it seems to have been a ritual to use Maximum likelihood estimation(MLE) for the optimization task in machine learning without any thinking. This section aims to clarify why and when MLE and MAP are applicable given upon the problem to be solved.

Maximum likelihood estimation

Maximum likelihood estimation(MLE) is a method of parameters(hypothesis) estimation given certain independent evidence. The hypothesis here is called parameters as for the most machine learning tasks, we would like to estimation the parameters that best describe the model, and there might be multiple possible candidates before selecting out the best one. We call each of these candidates a hypothesis. By maximizing the likelihood function of parameters θ one could get the optimal hypothesis as the best estimation for the model.

Consider the discrete case, let $X = (X_1, X_2, ..., X_n)$ be a n-dimension discrete random variable with probability mass function p describing by parameter vector θ. Given a specific realization $x = (x_1, x_2, ..., x_n$ of the random variable X, in which x_i are independent and identically distributed(i.i.d.). The likelihood of θ can be written as $P(X = x|\theta) = p_\theta(x)$. Note that thought it is written as a conditional probability, it refers to the likelihood function of variable θ. Sometime one may see it being written as $\mathcal{L}(\theta|x)$ which may cause confusions. $\mathcal{L}(\theta|x)$ here represents the likelihood function and we have the following relation: $\mathcal{L}(\theta|x) = p_\theta(x) = P(X = x|\theta)$.
Our goal is to find the optimal parameter Θ^* that make the evidence is most probable to occur. Formally,

$$\Theta^*_{MLE} = \arg\max_\theta P(x|\theta) \tag{3.19}$$

$$= \arg\max_\theta \prod_{x_i \in x} P(x_i|\theta) \quad if \ \ x_i \ \ are \ \ i.i.d. \ \ i \in [1, n] \tag{3.20}$$

To solve this we take log as log function is monotonically increasing and gives us a close form for taking the derivatives,

$$\Theta^*_{MLE} = \arg\max_\theta \sum_{x_i \in x} \log P(x_i|\theta) \tag{3.21}$$

Maximum a posteriori

Similar to MLE, maximum a posteriori(MAP) is also used to estimate model parameters, however the objective is different than that of MLE. Instead of maximizing the likelihood, we maximizing the posterior probability, by Bayes' theorem we have:

$$\Theta^*_{MAP} = \arg\max_{\theta} \; P(\theta|x) \tag{3.22}$$

$$= \arg\max_{\theta} \; \frac{P(x|\theta)P(\theta)}{P(x)} \quad by \;\; Bayes' \;\; rule \tag{3.23}$$

$$\propto \arg\max_{\theta} \; P(x|\theta)P(\theta) \quad drop \;\; P(x) \tag{3.24}$$

$$= \arg\max_{\theta} \prod_{x_i \in x} P(x_i|\theta)P(\theta) \;\; \text{by chain rule (3.4) and if } x_i \text{ are i.i.d. } i \in [1, n] \tag{3.25}$$

By taking log, we have

$$\Theta^*_{MAP} = \arg\max_{\theta} \; (\sum_{x_i \in x} \log P(x_i|\theta) + \log P(\theta)), \tag{3.26}$$

which is very similar to the solution of MLE but adding a regularized term $\log P(\theta)$. Therefore MLE sometime can be seen as a special case of MAP. Note that MAP is a Bayesian approach yet MLE is basic of frequentist paradigm.

3.3 Bayesian Inference

The sampling process can be seen as a simulation procedure, which falls into the study of probabilistic inference. Probabilistic inference is a major task in **statistics inference** – a data analysis technique used to deduce properties/hypothesis of an underlying model, or probability distribution from given observations Dodge [2006]. It can be seen as a contrary concept of **descriptive statistics** that directly draw conclusions from observations and is not based on probability theory. Specifically, there are two different two major approaches of statistical inference: **Frequentism** and **Bayesianism**. This thesis is about applying an MCMC algorithm of random sampling process. MCMC is actually an implementation of Bayesian inference.

3.3.1 Bayesianism and Frequentism

This part will discuss two paradigms of statistical and probabilistic studies: Bayesianism and Frequentism. Frequentism uses deterministic parameters to describe the probabilistic model, bayesainism delivers in a probabilistic manner. In frequentism, model parameters are deterministic variables, however, parameters are random variables in bayesainism inference. Both will yield the optimal hypothesis, bayesians assign a probability to the hypothesis yet frequentists will not. Some salient aspects that differentiate these two concepts are as follows:

Definition of probability

- Bayesians advocate that probability describes our degrees of certainty about events, and assign probabilities to define the degree of certainty of the obtained model.
- Frequentists believe that probability is fundamentally related to frequencies of repeated events and use numbers of occurrences to describe each consequence among repeated experiments.

Here provide two examples of different interpretations of probabilities by Bayesians and frequentists. Example 1: Coin toss

- Bayesians: 50% to 50% to get a head/tail in a single trail.
- Frequentists: $\frac{500}{1000}$ trails will be heads/tails.

Example 2: Determining model parameters θ

- Bayesians: Given the existing data, there is 95% of chance that the true value of θ lies in the calculated credible region.
- Frequentists: As the same experiment is repeated, 95% of these experiments' confidence intervals will cover the true θ.

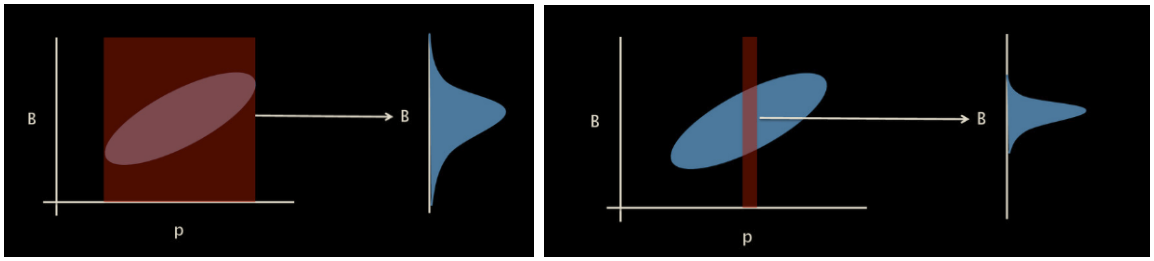
Content to be analyzed

- Bayesians work with the probabilities of parameters given fixed observations. Bayesians do not assign specific values to unknown parameters, but provide a probabilistic model of parameters and update it as observations being ongoing feed in.

- Frequentists update and derive static properties of data in terms of invariant model parameters and frequentists assume model parameters are fixed and estimate with confidence from observations.

Prior information

- Bayesians incorporate the prior information by doing marginalization(See figure3.8 (a)[2])
- Frequentists merely consider existing data therefore the derived information may only limited to the provided data.



(a) Bayesian methods (b) Frequentist methods

Figure 3.8 Comparison of Bayesian ans frequentist

Figure 3.8 B on y-axis represents the occurrence of a event, p on the x-axis corresponds to the model parameter related to event B, Bayesian and frequentist provides two ways to analyze the probability of the occurrence of B shown as (a), (b) respectively.

Therefore, frequentists and bayesians can be seen as conditioning vs marginalization. Bayesians inference incorporates prior information into the computation of posteriors, to alleviate the biases in model from the observed data. Frequentists draw conclusions directly from the given data without the consideration of any prior. Note that, although prior can be sometimes derived from data, it is a completely a different concept. Prior information might incorporate some information that come out of the collected data to help reduce over-fitting on the data.

[2]Image source: https://speakerdeck.com/jakevdp/frequentism-and-bayesianism-whats-the-big-deal-scipy-2014

3.3.2 Bayesian Inference

From Bayesians' perspective, the model should be updated as more data becomes available. Bayesian inference is a tool to update our beliefs in certain hypothesis when more data is observed. More concretely, Bayesians inference updates of the posterior probability distribution by using Bayes' theorem and is often implemented via Markov chain Monte Carlo methods to obtain desired limiting behavior with affordable computational time. Contrast to Bayesian inference, Frequentists employ a maximum likelihood estimate to find the model that best fit the data, which is often performed when a fixed model is needed. This chapter will focus on Bayesian inference.

The following list covers the conventional terms used in Bayesian inference:

1. $P(\theta|x)$: the **posterior probability**, indicates the probability of the hypothesis given evidence.

2. $P(x|\theta)$: the **likelihood**, denotes the likelihood of the occurrence of the evidence conditioning on certain hypothesis.

3. $P(\theta)$: the **prior** is the information known before inference which could be derived from the data or not. It also can be intentionally chosen to have certain impact on the final inference.

4. $P(x)$: the **marginal probability** can be computed by the law to total probability(see equation 3.6).

Bayesian inference updates model parameters according to Bayes' theorem. Rewrite Bayes' theorem for the ease of explanation.

$$P(\theta|x) = \frac{P(\theta)P(x|\theta)}{P(x)} = cP(\theta)P(x|\theta), \tag{3.27}$$

where θ and x stands for the model parameter and data respectively and $P(x)$ is often deemed as a constant in Bayesian inference and cannot be directly computed in most cases. That is why Bayesian inference is usually used by or with other methods to avoid this obstacle.

Convergence in Bayesian inference The posterior distribution of posterior mode will converge to $\mathcal{N}(\theta*, I(\theta*)^{-1})$ as $n \to \infty$, where $\theta*$ is the true value of model parameter θ. This convergence shares the same limiting distribution as for MLE, which implies that date should overwhelm prior information when the amount of data is considerably huge. Likewise, inference results may benefit from considering useful prior if the data is limited.

3.4 Markov Chains

Markov chains are excellent at tackling with challenging multi-fields real-world tasks merited by its limiting behavior. This limiting behavior is often called the stationarity or the convergence of a Markov chain which ensures that the generated objects are asymptotically uniformly distributed as the desired distribution. Another factor that contributes to the popularity of Markov chains is the employment of dependence. Creating a Markov chain is essentially a simulation of random walks on a set of states(states are finite in this paper). The transition probabilities between these states solely depend on the previous state and are time-invariant, due to which many computational difficulties are overcome.

A Markov chain is also a graph model, which is one kind of the graphical models. The definition and properties of Markov chains will be introduced after addressing several essential concepts in graphical models.

3.4.1 Graphical Models

A graphical model, or probabilistic graphical model, incorporates both graph theory and probability theory to compactly describe joint distributions of random variables by using conditional dependences between them.

Given different restrictions on the dependences between random variables, we can categorize graphical models into different genres accordingly. There are two major branches of graphical models: *Bayesian networks* and *Markov networks*, yet there exists another very popular one which can be deemed as a variance of both Bayesian and Markov network and is widely used in both statistics and machine learning applications. This kind of graph is called *dependency network* and firstly proposed by David Heckerman etc, 2000. Heckerman et al. [2000]. A dependency network is also implemented in the work of this thesis for the learning of underlying dependencies between different vocals and pitches of Bach chorales.

Graph Theory

The general idea of graph theory is extravagantly straightforward yet the derivations and applications of which make things intricate. Graphical models are the products of the combination of graph theory and statistical theory which enable us to analyse dependencies between variables from a probabilistic perspective.

1. Definitions This section serves as a prerequisite of the graphical model section. We will see how to use mathematical structures to build up pair-wise relations. The basic elements of a graph model are vertices and edges. Formally, a graph is an ordered pair set $G = (V, E)$ where $V = v_1, v_2, ..., v_n$ is the set of n vertices and $E = e_1, e_2, ...$ is the set of all edges. The connections between these edges and vertices depict the structure of a graph. Edges can be directed or undirected depends on settings(see figure 3.9 (a) and (b)). Next subsection will show how to use graphical models to visualize the dependencies between random variables.

2. Paths and cycles Several common used graphs will be frequently mentioned throughout the thesis, such as directed acyclic graph(DAG). Therefore it is necessary to know the meaning of these concepts. We say there is a **path** between two nodes if there exists at least one sequence of edges connecting these two nodes. The directions of each edge on the path should be identical. If there exists at least one path from a node to itself, we call this path a **cycle** in the graph. Take figure 3.9 (a) as an example, there is a path $(C - A - E)$ connecting node C and E but there is no path between nodes A and B. There is no cycle can be found in figure 3.9 (a), however, nodes (C, D, E) construct a cycle in figure 3.9 (c).

Bayesian networks

Bayesian networks are also known as Belief networks, which employ conditional probabilities to represent the dependencies of a set of random variables over **direct acyclic graphs** (DAGs) showed in figure 3.9 (a).
A Bayesian network for random variable $X = X_1, ..., X_n$ is a graphical model (G,P), where G is directed graph and $P = P_1, ..., P_n$ is a set of conditional probabilities of each variable. There are several takeaways for Bayesian networks that can help distinguish it from other graphical models:

Takeaways

1) There does not exist any cyclic dependencies between variables since Bayesian network is defined to have no cyclic path.

2) Every absence of a path between any two nodes indicates a conditional independence (definition 3.11). Therefore, the joint probability expressed by the chain rule (see equation (3.4) can be further simplified to the factorization shown in (3.29) if we take the Bayesian net setting in figure 3.9 (a) as an example.

$$P(A,B,C,D,E) = P(A|B,C,D,E)P(B|C,D,E)P(C|D,E)P(D|E)P(E) \tag{3.28}$$
$$= P(A|C)P(B|C,D)P(C)P(D)P(E) \tag{3.29}$$

Note that there are other ways to expand the joint distribution $P(A,B,C,D,andE)$, equation (3.28) only serves as an example to illustrate how to get a compact factorization of joint distribution by using conditional independence in Bayesian network.

3) It might neglect the potential dependencies between certain variables.

Markov networks

A Markov network is also called Markov random field which is implemented as undirected graph (U,ϕ). U represents an undirected graph and ϕ is a set of potential functions. The joint distribution can be written as followed:

$$P(X) = \Pi_{k=1}^{c}\phi_k(X^k), \tag{3.30}$$

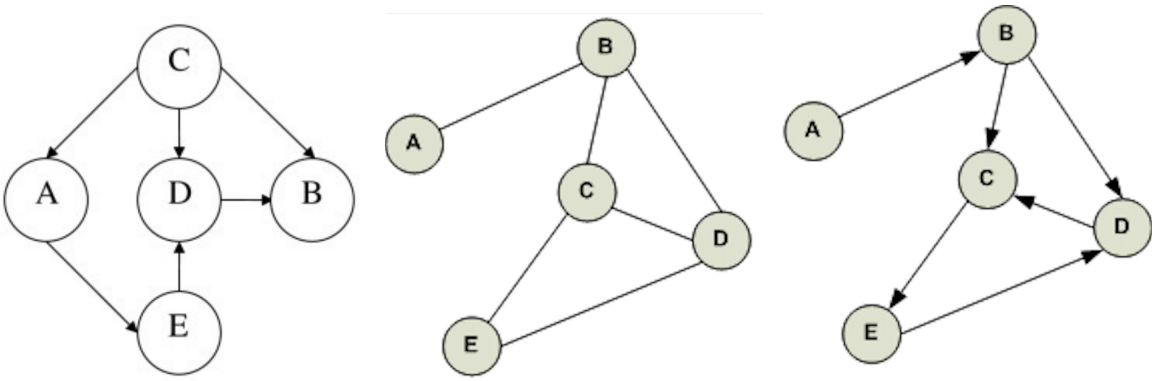
in which X_k are the random variables in clique k, $k = 1,...c$ (see Lauritzen [1996]). Figure 3.9 (b) shows a simple Markov network.

Takeaways

1) Since Markov networks are undirected, the edges in Markov networks indicate bi-directional dependencies between neighbors.

2) Similar to Bayesian networks, in Markov networks, one can declaim mutual conditional independence if there's no edge in between, which is known as the **pair-wise Markov property**.

3) A stronger version of conditional independence observed in Markov network is called **local Markov property**, which states that any random variable is conditionally independent of all the other variables when given its neighbors.



(a) Bayesian network (b) Markov network (c) Dependency network

Figure 3.9 Examples of three graphical models

Dependency Networks

1. Notations Provided that we have a finite set of random variables $X = (X_1, ..., X_n)$, and use the lower case x_i to denote a realization of variable X_i. Now we construct a dependency network (G, P) to represent these variables and their relations, where $G = (V, E)$ is a directed graph, potentially cyclic. Each vertice corresponds to a random variable in set X. $P = \{P_1(x_1|x\backslash x_1), P_2(x_2|x\backslash x_2), ..., P_n(x_n|x\backslash x_n)\}$ is the set of local conditional distributions, or say local probability distributions. For any random variable X_i we have,

$$P_i(x_i|x\backslash x_i) = P(x_i|x_1, ..., x_{i-1}, x_{i+1}, ..., x_n), \quad \forall i = 1, ..., n \tag{3.31}$$

Same to Bayesian and Markov networks, dependency network can be used to infer joint distribution of random variables. The authors proposed an ordered Gibbs sampler - a modified Markov chain Monte Carlo method described in the earlier section and proved that the Markov chain created by the ordered pseudo-Gibbs sampling process actually converges to a unique stationary joint distribution for random variables $X = (X_1, ..., X_n)$, no matter where we start from the initial state (see Theorem 3 in citation Heckerman et al. [2000]).

2. The consistent dependency network In practice, to make the joint distribution computable, we need to further expand equation (3.31) to get its factorization as equation (3.29). Since there could be enormous variants of expansions and one cannot guarantee any possible local probabilities is obtainable. To tackle this obstacle, David Heckerman etc. introduced the *consistent dependency network* and assumed all local distributions can be inferred from the joint distribution.

Takeaways

1) Dependency networks are very useful for *probabilistic inference* and preference prediction or in other words, *collaborative filtering*. As proved by the authors, a consistent dependency network can be seen as a Markov network in a certain senseHeckerman et al. [2000].

2) It is useful for constructing causal relationships and cooperates well with knowledge-based approaches.

3.4.2 Concepts in Markov Chains

Consider a finite state space set S indicating all possible states and a sequence of random variables: $X_0, X_1, X_2, ...$ with values $\in S$. We say a discrete-time Markov chain on S if satisfying the following *Markov property* which is nothing but a conditional independence (see equation (3.11):

$$P\{X_{t+1}|X_0 = s_0, ..., X_t = s_t\} = P\{X_{t+1}|X_t = s_t\} \ \forall i, i_0, ..., i_{t-1} \in S. \tag{3.32}$$

With assumption (3.32), or called 'one-step memory', the joint distribution of random variables $X_0, ..., X_n$ expanded by chain rule(see equation (3.4)) can be simplified as:

$$P\{X_0, ..., X_n\} = \Pi_{t=1}^{n-1} P\{X_{t+1}|X_t = s_t\} \cdot P\{X_0\}. \tag{3.33}$$

These one-step conditional probabilities is also call transition probabilities in Markov chains. A two dimensional **transition matrix** $P$is used to store these dependencies between random states. Each element in P satisfies:

$$p_{ij} = P\{X_{t+1} = j|X_t = i\} \ \ \forall i, j \in S. \tag{3.34}$$

All Markov chains mentioned in this thesis without additional explanation will be homogeneous, i.e., any transition probability is time-invariant(i.e. $p_{ij}^{(t)} = p_{ij}$ for all i,j $\in$ S) and merely depends on the last state if the dependency exists. Besides, p_{ij} must follows:

$$\sum_{j \in S} p_{ij} = 1, \ \ p_{ij} \geq 0 \ \ \forall i, j \in S \tag{3.35}$$

Irreducible Markov Chain

We say a Markov chain is irreducible if all states have paths to each other, that is to say, given initial state $X_0 = i$, we should have

$$p_{ij}^{(t_1)} = P\{X_{t_1} = j\} > 0 \ \ \exists \ \ t_1 > 0 \tag{3.36}$$

$$p_{ji}^{(t_2)} = P\{X_{t_2} = i\} > 0 \ \ \exists \ \ t_2 > 0 \tag{3.37}$$

Aperiodic states

Each state has an integer-valued period which is the great common divider(GCD) of a set of all possible time steps that this state is revisited, mathematically the period a state i is

$$GCD\{t : t \geq 1, p_{ii}^{(t)} > 0\}. \tag{3.38}$$

We say a state is aperiodic if and only if the period of this state is one, which indicates this state is accessible at any possible time step.

Ergodic Markov chain

An ergodic Markov chain is nothing but an irreducible finite Markov chain with all state being aperiodic. In practice, people usually aim to create an ergodic Markov chain as an ergodic chain has several nice limiting behaviors which will be discussed later in this section.

3.4.3 Convergence of Markov chains

We say a Markov chain converges to a stationary probability distribution if for all states, there exist a fixed probability distribution vector $\pi = (\pi_1, \pi_2..., \pi_n)$, such that

$$\lim_{t\to\infty} p_{ij}^{(t)} = \pi_j \ \ \forall i, j \tag{3.39}$$

The proof of the convergence of a Markov chain is omitted here, this section mainly focuses on the criteria of a converging chain. The reason we defined an ergodic Markov chain is that any ergodic chain has the following convergence properties:

1) There is a unique **stationary probability** vector $\pi = (\pi_1, \pi_2..., \pi_n)$ s.t.,

$$\pi_i = \sum_j \pi_j p_{ji} \ \ \forall i, \ \ or \ \ \pi = \pi P^t \ \ \forall t. \tag{3.40}$$

This property states that if we randomly generate a state X_0 according to π, then generate another state X_1 for the next time step, the ergodic Markov chain guarantees that X_0 and X_1 are asymptotically distributed. All generated states X_t actually share the same distribution defined by π.

2) Ergodic Markov chain will reach its stationarity regardless of its initial state:

$$\lim_{t\to\infty} p_{ij}^{(t)} = P\{X_t = j | X_0 = i\}\pi_j \ \ \forall i, j. \tag{3.41}$$

Property (3.41) shows that if a Markov chain is ergodic, it converges to a fix states distribution no matter where it starts as time goes to infinity. However, in practice, we cannot afford infinite time to see its convergence, so approximation will be usually employed in most cases.

Identify the stationary probability of a Markov chain

There exists a simple but sufficient verity if a π is a stationary probability vector. If a Markov chain satisfies the **time-reversible** property or **detailed balanced(DB)** property:

$$\pi_i p_{ij} = \pi_j p_{ji}, \quad \forall i, j \in S \tag{3.42}$$

then $\pi = (\pi_1, \pi_2..., \pi_n)$ is proportional to a stationary probability vector and one can just simply normalize the rows of π to obtain the solution(i.e. $\sum_j \pi_j = 1$).

DB criterion for checking reversibility is applicable when both stationary distribution and transition probabilities are known. Fortunately, there is another equation which allows us to check reversibility merely on the transition probabilities. Kolmogorov's criterion states that a Markov chain is time-reversible if and only if:

$$P_{i_1 i_2} P_{i_2 i_3} \cdots P_{i_{n-1} i_n} P_{i_n i_1} = P_{i_1 i_n} P_{i_n i_{n-1}} \cdots P_{i_3 i_2} P_{i_2 i_1} \tag{3.43}$$

for any finite sequence of states. In practice, people often intend to construct Markov chains that obey DB condition or Kolmogorov's criterion to ensure its convergence property. However, DB condition is not the only criteria that decide the convergence of a Markov chain.

3.5 Sequential Modeling in Deep Learning

Music as a kind of sequential data, has the same characteristic: time-dependent, just as other sequential data like text and audio. Howe to choose a suitable model for our sequential data is fundamentally vital for the generation tasks. This chapter will cover some essential concepts and techniques in machine learning then introduce sequential modeling. Specifically, we will focus on regular and advanced recurrent neural networks, which have been massively used in sequence modeling due to its inherent summarization ability of sequence history within an internal memoryBoulanger-Lewandowski et al. [2012].

3.5.1 Machine Learning Basics

Machine learning algorithms aim to automatically learn the underlying pattern from existing data and produce valuable prediction given new data. Table 3.1 lists the comparison of two main paradigms of machine learning. All concepts of machine learning are explained from the probabilistic perspective as the sequential models are used to approximate the probability distributions embedded in the Bach chorales in this paper.

Notations

- n: number of feature vector dimension.
- m: number of data.
- $x = (x_1, ...x_n)$: The input vector often called as the **explanatory vector** that represents n (latent) features of data.
- y: The output which is often called as a response variable with values from a finite set.
- $w = (w_1, ..., w_n)$: A set of weights indicating the relevance of features to the output.
- θ: Model parameter set, which may contain more than one variable. w belongs to θ.

Regression

The two most well-known methods for regression in machine learning are linear regression and logistic regression. The table 3.2 provides a comparison of these two methods which lists some parts that often cause confusions. We take supervised learning as an example for illustration.

1) **Linear regression**

By linear regression setting, we assume each element x_i can be interpreted as a factor that linearly contribute y, in other words, the value of y is a linear combination of $x_1, ...x_n$:

$$y(x) = \sum_{i=1}^{m} w_i \cdot x_i + b = w^T x + b, \tag{3.44}$$

Table 3.1 Supervised and unsupervised learning

	Supervised	**Unsupervised**
Input	Labeled	Unlabeled
Model	$P(y\|x,\theta)$ Conditional density estimation	$P(x\|\theta)$ Density estimation
Applications	Regression Classification	Latent feature learning Dependency learning Clustering
Output	y^* = the mode of the distributions	
	$argmax_i P(y_i\|x,\theta)$	$argmax_i P(x\|\theta_i)$

* Other paradigms of machine learning like reinforcement learning and mixtures are out of the scope of this thesis and therefore are omitted in the comparison.

Table 3.2 Linear, Logistic and Softmax Regressions

Item	**Task**	**Model**
Linear regression	Regression	$y \sim \mathcal{N}(w^T x, \sigma^2)$ $P(y\|x,\theta) = \mathcal{N}(w^T x, \sigma^2)$ $y^* = E(y) = w^T x$
Logistic regression	Binary classification	$y \sim Ber(\text{sigmod}(w^T x))$ $P(y\|x,\theta) = Ber(\text{sigmod}(w^T x))$ $y^* = 1$ if $sigmod(w^T x) > threshold$ $y^* = 0$ if $sigmod(w^T x) \leq threshold$
Softmax regression	Multi-class classification	See equation (3.50)

where b is the **residual error** or **bias** of the linear model as there remains a offset between the true value and out model prediction. Conventionally, we assume that the bias b follows a normal/Gaussian distribution with mean $w^T x$ and variance σ^2, denoted as $b \sim \mathcal{N}(0, \sigma^2)$. b is the randomness caused by the limitation of the computed linear model. Then we can rewrite the linear regression model as:

$$P(y|x, \theta) = \mathcal{N}(w^T x, \sigma^2) \tag{3.45}$$

By replacing x with: $\phi(x) = (x^0, x, x^2, ...x^d)$ in the model, linear regression can be used to model nonlinearity.

2) **Logistic regression**
The essential reason for the name logistic regression lies in the usage of the sigmod function which is the simplest logistic function.

- Sigmod function
  The usage of sigmoid function is the core reason making logistic regression such popular as it maps the value of linear combination of inputs to the range between 0 and 1. Therefore, it can be interpreted as the probability of the output being that specific value.
  Sigmod function is a special case of general logistic functions. The logistic function is shown below:

$$f(x)_{\text{logistic}} = \frac{L}{1 + e^{k(x_0 - x)}}, \tag{3.46}$$

  where L indicates the maximum value of $f(x)$, k denotes slope rate and x_0 is the mid-value of x domain. Sigmod function is just a special case of the logistic function as follows:

$$f(x)_{\text{sigmod}} = \frac{1}{1 + e^{-x}} \tag{3.47}$$

- Binary classification
  In binary classification setting, random variable y follows a Bernoulli distribution as there are only two possible values. Mapping the value of linear regression in to range of $[0, 1]$ which can be seen as the probability of a Bernoulli trial. Therefore the conditional density can be written as:

$$P(y|x, \theta) = Ber(p) = Ber(\text{sigmod}(w^T x)) \tag{3.48}$$

3) **Softmax regression**
For multi-classification tasks, we need to a generalized sigmod function: **softmax function**.

- Softmax function
  Softmax function is also called normalized exponential function as it has the following form:

$$f(x_j)_{\text{softmax}} = \frac{e^{x_j}}{\sum_{i=1}^{k} e^{x_i}} \quad \forall i, j \in [1, k] \tag{3.49}$$

  It is easy to verify that $\sum_{i=1}^{k} f(x_i) = 1$. Therefore, the set of $f(x)_i$ can be seen as a probability distribution over k classes.

- K-class classification
  Music reharmonization is a prediction task of k classes, finding the most likely note for a certain position. The output of each softmax function provides us the probability of the corresponding class. Stacking up the outputs forms a probability mass function that describes the distribution over all possible classes given observations. Softmax regression is a generalization of logistic regression, which is defined as below:

$$f(y = j|w, \theta)_{\text{softmax}} = \frac{e^{w_j^T x}}{\sum_{i=1}^{k} e^{w_i^T x}} \tag{3.50}$$

  When we need the output class to be singular, one can just choose the class with the highest probability among all classes. On the other hand, keeping the PMF form the output can be further used in the sampling process to bring randomness to the generation process.

$$p_1 = P(y_1|x,\theta) = Ber(w_1^T x) \tag{3.51}$$
$$p_2 = P(y_2|x,\theta) = Ber(\text{sigmod}(w_2^T x)) \tag{3.52}$$
$$\vdots \tag{3.53}$$
$$p_k = P(y_k|x,\theta) = Ber(\text{sigmod}(w_k^T x)), \tag{3.54}$$

where each w_i is a vector and $\sum_{i=1}^{k} p_i = 1$(Normalization).

$$P(y|x,\theta) = P(y_1,...,y_k|x,\theta) \tag{3.55}$$
$$= \Pi_{i=1}^{k} P_i(y_i|x,\theta) \quad y_i \text{ are independent as they consist the basis of } y \tag{3.56}$$
$$= \Pi_{i=1}^{k} Ber(\text{sigmod}(w_i^T x)) \tag{3.57}$$

Terminologies in machine learning

1) Generative Learning vs. Discriminative Learning

- **Generative learning** computes the conditional probability distribution by firstly obtaining the joint probability, then use the definition of conditional probability in equation (3.2).
- **Discriminative learning** directly works with the conditional probability model and tries to formalize it. Regression falls under the discriminative genre.

2) Parametric Learning vs. Non-parametric Learning

- **Parametric learning** has finite number of parameters which is invariant with respect to the size of data.
- In **Non-parametric learning**, the number of parameter grows as more data become available.

3) Multinomial vs. Categorical and Multinomial vs. Multivariate

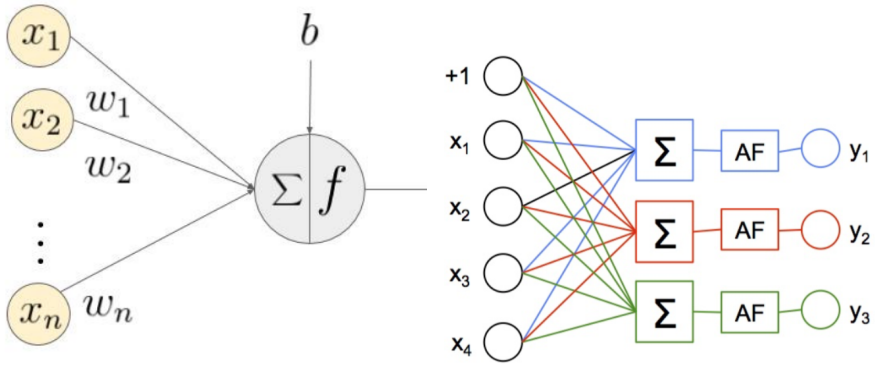
- Multinomial distribution describes that in the *multiple trials*, in which each trial has *multiple possible outcomes*.
- Categorical distribution does one single trail with *multiple possible outcomes*.
- Multivariate describes the number of random variables. Usually, there is only one as we hope to generate one prediction given existing data.

Therefore, most machine learning classification tasks are univariate as there is usually only one variable y to be estimated. If it a binary classification then y is a Bernoulli random variable. If there are multiple possible classes, y follows the categorical distribution.

3.5.2 Common Sequential Models

Neural networks

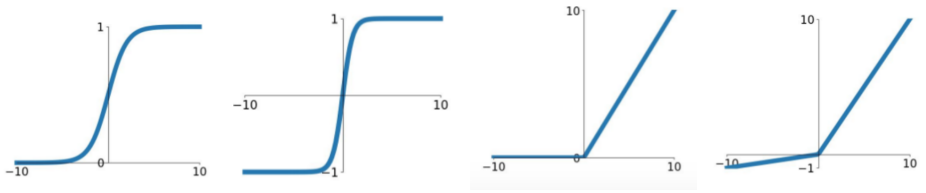
RNNs are invented based on the vanilla NNs which is modified from **perceptron**. The concept of NNs was first proposed by Rosenblatt in 1958Rosenblatt [1958]. It was inspired by the operation of synapses yet is more well-structured than our brain.



(a) Single layer NN with single output (b) Single layer NN with multiple outputs

Figure 3.10 Two basics architecture of NNs

There are several essential building blocks in NNs. A simple NN consists of at least three layers: input, hidden and output layers. The dimension of each layer relies on the number of units. Units of hidden layer are also called neurons. The connection between input and a neuron is depicted in figure 3.10 (a). Weights w on the edges refer the model parameters to be trained. There always are two tasks in each neural: linear regression and activation. The summation in 3.10 (a) represents $w^T x$ and f(AF) the activation function. Figure 3.11 lists four common activation functions. The role of hidden layers is to extract latent representations from the data. By further processing these extractions, one can be obtained a higher level of information embedded in the data. Figure 3.12 provides a NN architecture consisting of multiple hidden layers.



(a) Sigmod (b) Tanh (c) ReLU (d) Leaky ReLU

Figure 3.11 Four common activation functions

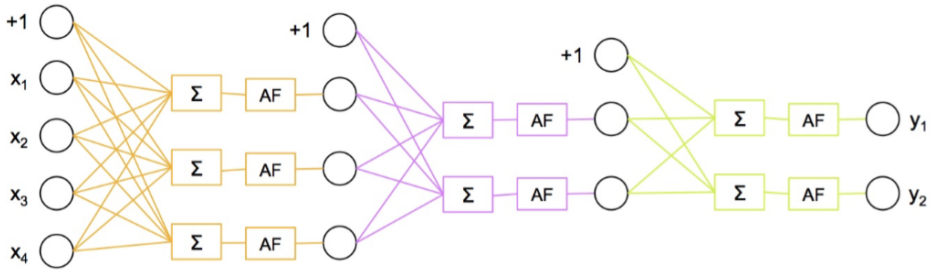


Figure 3.12 NN architecture with hidden layers

Recurrent Neural Networks

If use classic NNs to model sequential data, one might have to stack these time sequences making them into one vector for training regardless of how long it will be. It will to costly to afford a sophisticated weighted voting process. The architecture of your networks may not be good at capturing the pattern shared by each time step, also some experiments have shown that using regular NNs do not yield satisfying results no matter how dedicated selection is made on weights selection. And in terms of music scores, a note is often correlated with a sequence of leading notes.

Motivated by all these shortcomings and limitations of classic NNs, RNNs are invented for learning sequence structures Elman [1990] by adding time-dependent connections between hidden states. What an RNN does is to separate the data by their time step and feed them into the network sequentially. This not only reduces the input size of our system, but enables us to consider the prediction(could be a probability) of early time-stamped data for the current timestamp.

Figure 3.13 shows two ways of depicting RNNs structures. The left circuit graph can be unrolled into the diagram on the right. At each time step, input x will be incorporated with hidden state h to compute the hidden state of next time step. If we use θ to represent the model parameters, and consider this RNN as a probabilistic graphical model, then the update of hidden states can be written as:

$$P(h^t|h^{t-1}, x^{t-1}; \theta)$$

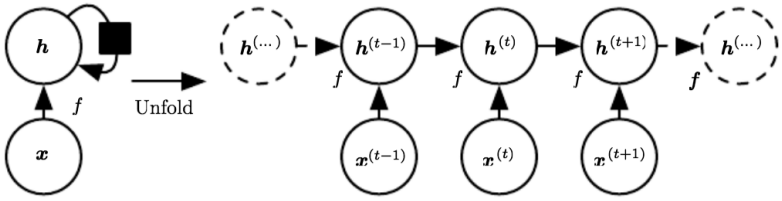


Figure 3.13 RNN architecture

The training process of typical RNNs for a classification task is illustrated in figure 3.14 which is an expansion of figure 3.13. We use softmax for the final output o and for middle layers we do not specify the activation functions δ. Loss L measures the deviation of o from its label y. Both W and V are weights matrices. b and c are biases. The update rules through training are as follows:

$$a^t = b + Wh^{t-1} + Ux^t$$

$$h^t = \delta(a^t)$$

$$o^t = c + Vh^t$$

$$y^t = f_{softmax}(x^t)$$

If we use log likelihood to measure the loss of each iteration given x labeled by y, we have:

$$L = \sum_t logP(y^t|x^1, ..., x^t; \theta)$$

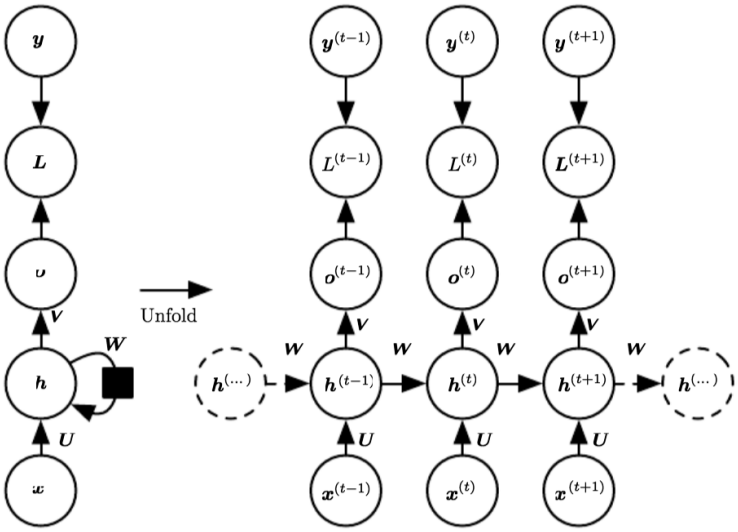


Figure 3.14 Training process of RNNs

Bidirectional RNNs

In regular RNNs, the models incorporate data of previous time steps to generate the prediction of the current time step. However, in many real-life applications, we would like to proceed with our data from both directions and consider both ends at the same time. The same situation applies to music composing. Composers have to take the whole sequence of notes into consideration to be able to avoid dissonances. Bidirectional RNNs(BRNNs) are proposed to address this request.

A BRNN consist of one regular RNN and another one moving from the end to start through time. Figure 3.15 depicts the updates in BRNNs. g and h stands for the backward and forward propagations of training flows respectively. As can be seen from the graph, the output of each time step incorporates the information from both past and future.

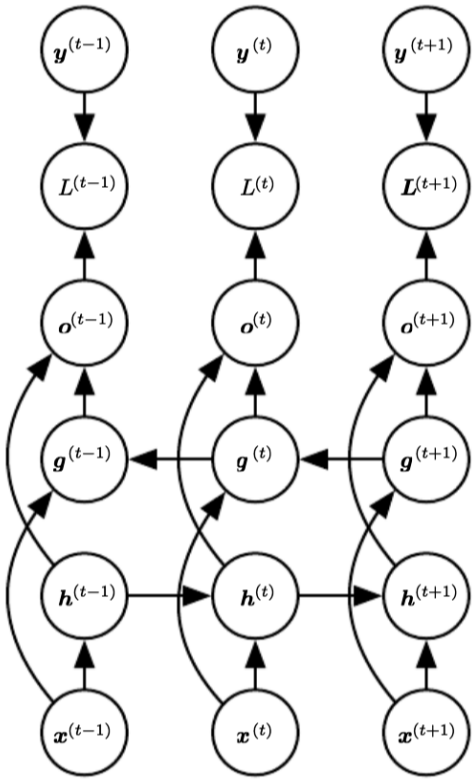


Figure 3.15 Training process of BRNNs

Gated Recurrent Neural Networks

However, it is not always beneficial learning long-term dependencies of data via RNNs. The major drawback is called **vanishing/exploding gradients problem** but more often vanishing. The underlying reason for this situation arises from the exponential decrease of model parameters as the computations of products of Jacobians. Long short-term memory(LSTM) networkHochreiter and Schmidhuber [1997] is the most popular modified RNNs to address this vanishing issue.

The core contribution of LSTM owns to the invention of self-loop gated of weights. Figure 3.16 shows three different gates(input, output and forget gates) replace the original hidden unit in regular RNNs. Even the model weights are fixed, one can modify the time scale of integration by changing the inputs.

Each LSTM cell shares the same inputs and outputs as original RNN cell, but with three gates deciding how much each state should participant in the current iteration. The outputs of gates have been normalized between zero and one representing the weights for each part. The gates are trained to be able to identify what information are valuable for the current setting and task.

The forget gate f_t decides what info is useful and should be stored in this cell for further reference. Then the input will be processed by a sigmoid and a tanh activations before incorporating the previous memory information by updating C_t. The final stage within each cell is generated by filtering the latent info h^{t-1} and current input x^t with what is chose to be remembered from history. The mathematical model of LSTM is as follows:

$$f_t = \sigma(W_f \cdot (h^{t-1}, x^t) + b_f)$$

$$i_t = \sigma(W_i \cdot (h^{t-1}, x^t) + b_i)$$

$$C_t = \tanh(W_c \cdot (h^{t-1}, x^t) + b_c)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot C_t$$

$$o_t = \sigma(W_o \cdot (h^{t-1}, x^t) + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

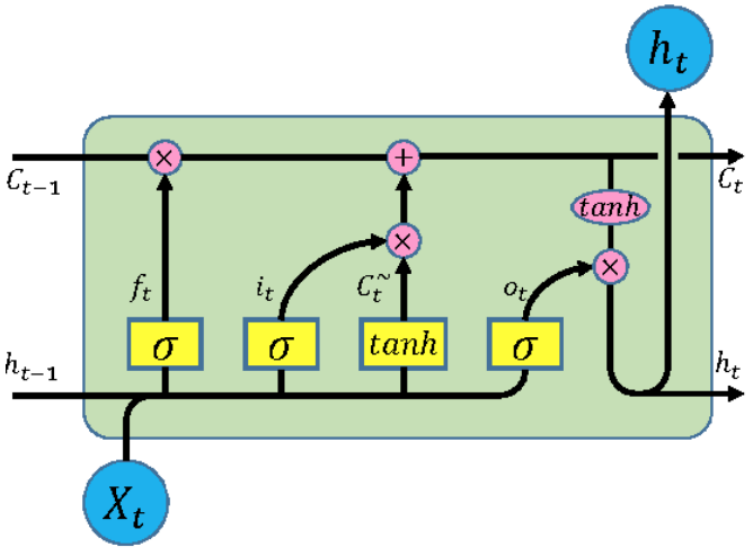


Figure 3.16 A LSTM unit

Likewise, LSTMs can be designed to be bidirectional as BRNNs. Figure 3.17 shows the structure of a bidirectional LSTM which is employed in SAM-Bach.

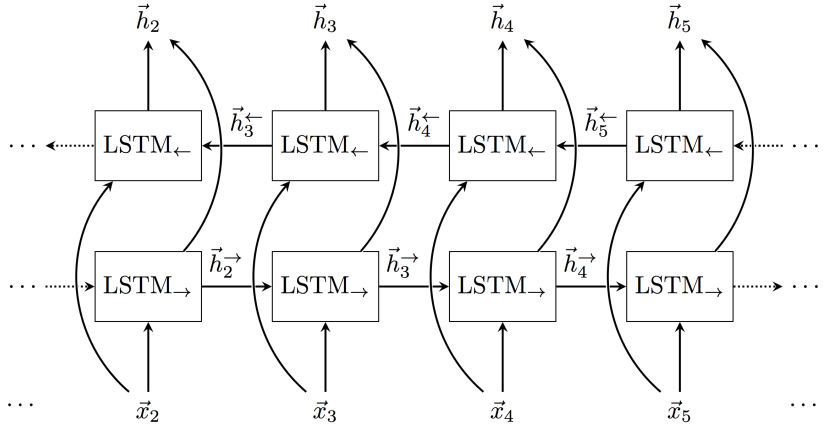


Figure 3.17 A BLSTM unit

3.6 Inference and Sampling in Deep Learning

In this paper, Bach reharmonization is deemed as a sample generation task. Sampling can be realized by **statistical inference** in two steps: 1) Learn the underlying probability distribution of given Bach chorales. 2) Generate samples that follow the desired distribution. However, in order to implement these two steps, knowledge from multiple fields are involved. The purpose of this whole section is to help readers understand the role of each part(either theoretical or technical) and how did they get to combined together to generate Bach-like chorales. The first subsection will provide a general knowledge structure graph covering all the techniques mentioned throughout the thesis. Then the middle five subsections will tackle with all the concepts and methods categorized by fields. The last subsection will introduce several classical MCMC sampling methods.

3.6.1 Overview of Statistical Inference

Figure 3.18 shows a structural relationship between different concepts covered in this work and what makes the SAM algorithm employed for the Bach chorale generation. Although it seems to contain various parts, the core lies in the **statistical inference**. More precisely is inferential statistical analysis. The intention is to infer underlying properties or patterns embedded in the observations. One popular way to handle this task is to study the expectations, i.e. $E(X)$ where X is a random variable. One way to estimate the expected value is through computing integral:

$$\int f(x)dx = E(X) \tag{3.58}$$

By the law of large numbers 3.15, we can approximate the expectation by the sample average of i.i.d. random samples: $X_1, X_2, ..., X_n$ drew from f:

$$E(X^*) = \frac{1}{n}\sum_{i=1}^{n} X_i \to E(X) \text{ as } n \to \infty \tag{3.59}$$

If the samples are available then one can directly perform numerical methods like Newton method, otherwise one may need to generate samples via simulation methods. **Monte Carlo simulation** is one of such methods that approximate integrals by simulating random realizations and taking the average of them. The challenges are how to draw such samples and how to guarantee these samples follow the target distribution. **Markov Chain Monte Carlo**(MCMC) method theoretically ensures that the generated samples will eventually converge to the target distribution by carefully construing **Markov Chains**. Metropolis and Gibbs samplings both are MCMC methods that yield chains with limiting behaviors.

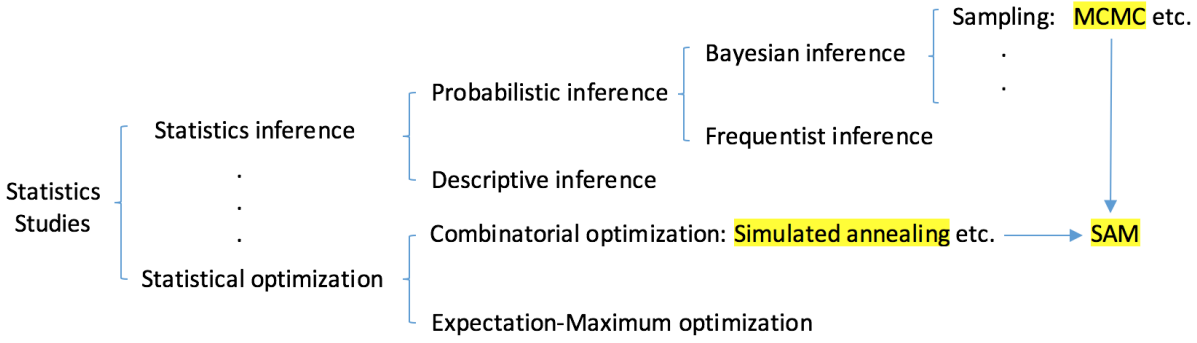


Figure 3.18 Overview of statistical inference

3.6.2 Simulated Annealing - Optimization in Inference

Annealing is a metallurgy term means heating up then cooling a solid material slowly. The internal energy of the material increases while it is heated and the internal molecules are moving randomly. Higher temperature leads to higher thermal energy and the vice versa. After being cooled down thoroughly, molecules of the solid should have the minimal thermal energy.

Algorithm 1 Simulated annealing

1: Initiate states $s = s_0$
2: **for** $k = 0 : K$ **do**
3: Update the temperature T
4: Random choose a neighbor of s as new proposal
5: Accept this proposal with probability $P = \exp -\Delta E/T$
6: **end for**
7: Output: Final state s_K

Inspired by this physical process, the SA algorithm tries to simulate the procedure and solve combinatorial minimization problem. It belongs to local search paradigm but is capable of approximating the global optimal even with a huge searching space in a meta-heuristic manner. To simulate the randomness in SA, probability techniques are required. Algorithm 1 provides a simple version of SA. Chapter 5 will apply an interacting SA to the task of music generation.

In the above SA algorithm, s is the state vector of all molecules. E represents energy value and the goal is to identify a state $s*$ with the lowest energy. Several rules are defined in order to find the minimal, which is called **Boltzmann's criterion** which is quantified as P in the above algorithm. **Boltzmann's criterion**

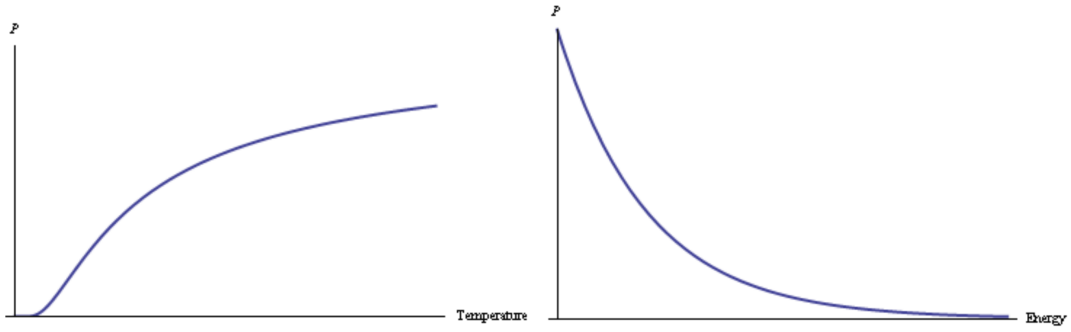
1) $P(T_s)$ must be non-negative even given worse proposal(i.e. $\Delta\mathcal{E}(t) > 0$) so as to prevent from being stuck at a local optimal.

2) When temperature T_s is very low, P should tend to 0 if the proposal is worse than the current state. If the proposal is better than the current state, P should be a (potentially unnormalized) positive value (In the presented algorithm, $P = 1$).

3) When temperature T_s is high, the state transitions are sensitive to the variation of energy.

For big T, we do not mind going uphills to explore more candidates in the space, while for sufficiently small T, we favor going down-hills to avoid the risk of leaving the optimal region. One should more careful and timid taking moves when the temperature is low. Figure 3.19 visually depicts these requirements.

When T is significantly low, no matter how good the proposal state is, the transition probability P will still tend to zero, which consequently force the Boltzmann's probability tends to zero. The temperature controls the moves going up or down and directly affects the transition decision. The **Boltzmann's probability** derives from Boltzmann's equation:



(a) Boltzmann P w.r.t. T (b) Boltzmann P w.r.t. E

Figure 3.19 Boltzmann probability

$$\frac{P(s_1)}{P(s_2)} = \frac{e^{-E_1/kT}}{e^{-E_2/kT}} = e^{-(E_1-E_2)/kT} = e^{-\Delta E/kT}, \tag{3.60}$$

which is probability ratio of being at state s_1 and s_2. k is Boltzmann's constant[3] which might be omitted in implementations.

3.6.3 Markov Chain Monte Carlo - Simulation in Inference

In statistics inference, given observations of variables of interests, one would like to learn and analyze from data to draw conclusions, predictions or for other usages that what people called 'inference'. One of the most popular studies would be the estimation of the expected value of variables or some functions of variables.

[3]Boltzmann's constant $\approx 1.3807x10^{-23}(J/K)$.

If we can rewrite the expectation in the form of an integration, it becomes a Monte Carlo integration problem. The above statement is based on the fact that the samples are available. When facing the absence of enough samples, one might need to generate some samples from a certain distribution then address the inference problems. Sampling methods are firstly designed to serve for inference, yet nowadays they are frequently used just in sample generation. Figure 3.20 shows the relationship among several concepts in inference.

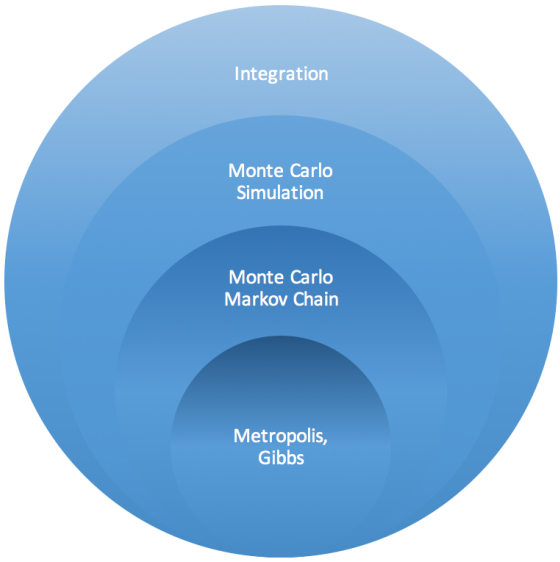


Figure 3.20 MCMC in Inference

Monte Carlo Simulation

Monte Carlo simulation was originally invented(Hammersley and Handscomb [1964]) for computing statistical estimation of the value of an integral by averaging a set of random points drawn from the distribution with support over the range of integration (see Metropolis and Ulam [1949]). Indeed, Monte Carlo simulation was intentionally proposed for computing integrals, however, it has been broadly used just for samples generation in many real-world tasks. Markov chain Monte Carlo method is an outstanding method that simulates random draws from a target distribution (Givens and Hoeting [2012]).

Markov chain Monte Carlo(MCMC)

MCMC sampling approaches aim to generate an ergodic Markov chain whose stationary probability $\pi(x)$ distribution is exact or approximate to the target distribution $P(x)$. MCMC sampling is an iterative method that generates an ergodic Markov chain whose stationary probability distribution is identical to the target distribution. More specifically, all MCMC algorithms are constructed in a way that the stationary probability distribution can be reached through updating rules. For a sufficient amount of iterations, the margin distribution of the realization of random states on the Markov chain will gradually approximate to the target distribution.

Two popular MCMC methods are Gibbs sampling and Metropolis-Hasting sampling. The proposed SAM algorithm is based on Metropolis-Hasting and compared with a pseudo-Gibbs which was used in DeepBach Hadjeres and Pachet [2016].

1) Gibbs sampling

   Gibbs sampling is frequently used in sampling tasks as its ease in implementation (no additional parameter needed) and nice approximation limiting behavior. The only downside requirement of this methods is one need to know full conditional distributions, which is not always accessible in real world. Therefore, people would use approximates to implement Gibbs sampling. That is essentially why the authors in DeepBach(Hadjeres and Pachet [2016]) named their sampling algorithm as 'pseudo-Gibbs'.

   The algorithm 3 provides the insights of this method. Note that Gibbs sampling accepts every new sample so it may stop at a local optimal state. In other words, the newly updated states may not be superior than the previous ones. This is fundamentally why SAM sampling has better convergence property compared with Gibbs sampling.

2) Metropolis-Hasting

   In sampling tasks, certain evaluation functions are often defined to quantify how good the sample is. However, it is not practical to record every sample and compare each of them. One needs to design a mechanism that ensures the convergence to the optimal sample region. Metropolis-Hasting(MH) is such a sampling scheme by comparing the current sample with the proposal one. The M-H ratio $\mathcal{R}(x'|x)$ is used for this comparison representing how probable to transit from x to x' given a proposal distribution $g(x'|x)$. Mathematically it can be written as: $P(x'|x) = g(x'|x)\mathcal{R}(x'|x)$.

Algorithm 2 Gibbs sampling

1: *Input*: full conditional distribution
2: Initialize $x^0 = (x_1^0, ..., x_k^0), t = 0$
3: **while** $t < T$ **do**
4: **for** i=1:n **do**
5: Sample x_i^{t+1} $p(x_i|x_1^{t+1}, ..., x_{i-1}^{t+1}, x_{i+1}^{t+1}, ..., x_k^{t+1})$
6: **end for**
7: $t = t + 1$
8: **end while**
9: Output: x^T

Recall when constructing a ergodic Markov chain, we claims it has stationary probability when the DB condition is meet: $\pi(x)P(x'|x) = \pi(x')P(x|x')$. Therefore, to preserve this stationarity, MH needs to consider this condition by setting the stationary probability $\pi(x)$to be our target distribution $P(x)$ in MH. Therefore, we have $P(x)P(x'|x) = P(x')P(x|x')$. Combine this equation with the one in the last paragraph, we have:

$$\frac{\mathcal{R}(x'|x)}{\mathcal{R}(x|x')} = \frac{P(x')P(x|x')}{P(x)P(x'|x)}$$

That explained how M-H ratio $\mathcal{R}$ is derived. Now introduce MH algorithm.

Algorithm 3 Metropolis-Hasting algorithm

1: Initiate states $s = s_0$
2: **for** $k = 0 : K$ **do**
3: Sample from the proposal distribution $s' \sim g(\cdot|s_k)$
4: Compute *M-H Ratio*: $\mathcal{R}(s'|s_k) = \frac{P(s')g(s_k|s')}{P(s_k)g(s'|s_k)}$
5: Accept the proposed sample with probability $P_{ac} = min(1, \mathcal{R})$
6: **end for**
7: Output: Final state s

Chapter 4

Methodology and Algorithms

This chapter presents the SAM-Bach architecture which consists of three main parts: 1) music preprocessing 2) estimation of probability distribution of Bach chorales via deep learning; 3) Bach chorale generation via MCMC sampling. This architecture is the combination of deep learning and sampling strategy. Figure 4.1 depicts the development cycle and process of the SAM-Bach system, which will be illustrated step by step in the following subsections.

4.1 Bach Chorales Preprocessing

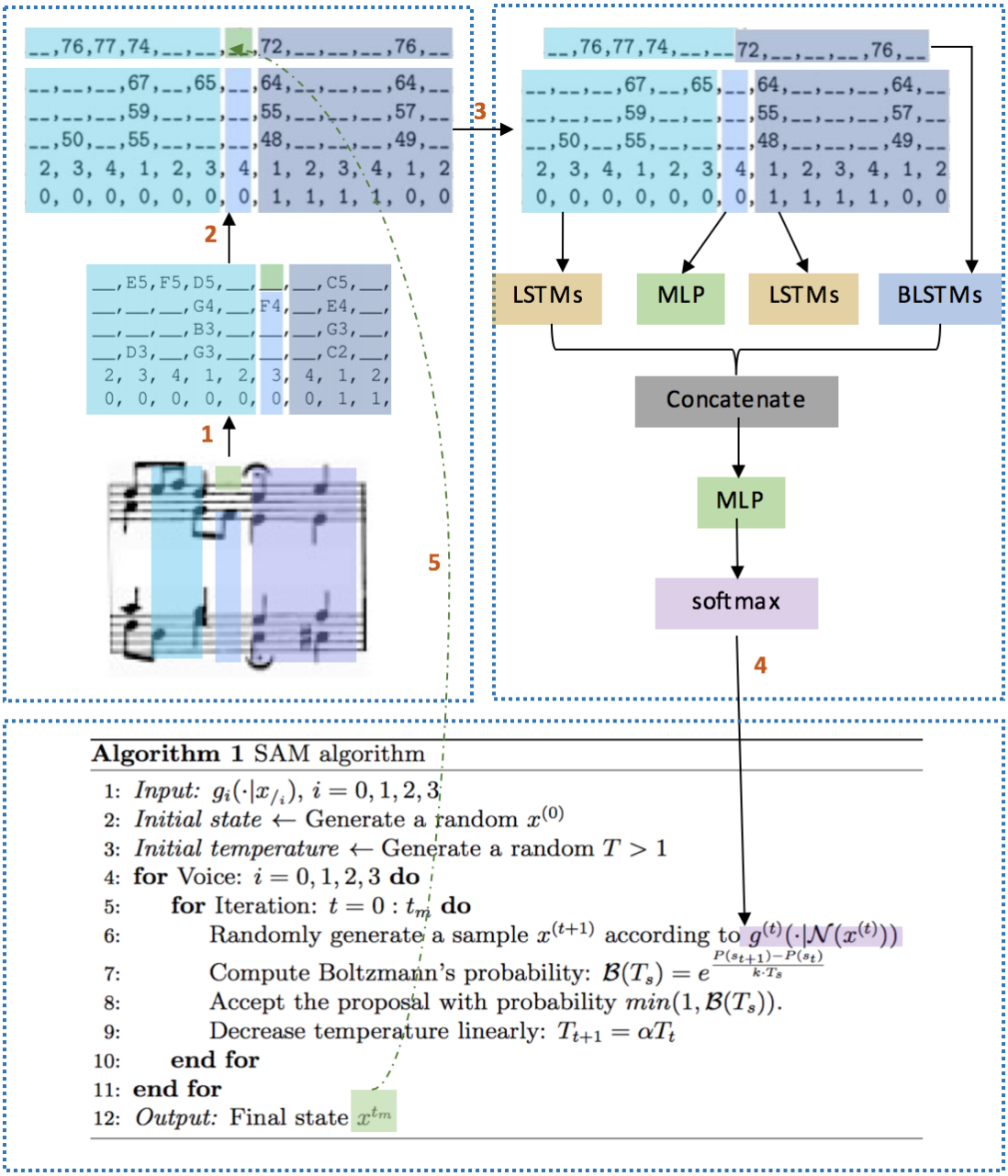
This part is corresponding to the left-up rectangle in Figure 4.1, which produces the proper music representations for the later learning and sampling processes.

4.1.1 Requirements of Bach Chorale Representation

The designed representation should not only preserve the necessary information of the original Bach chorales, but is compatible with the sampling process. Thus there are several aspects need to be considered when selecting the representation.

Informative requirements

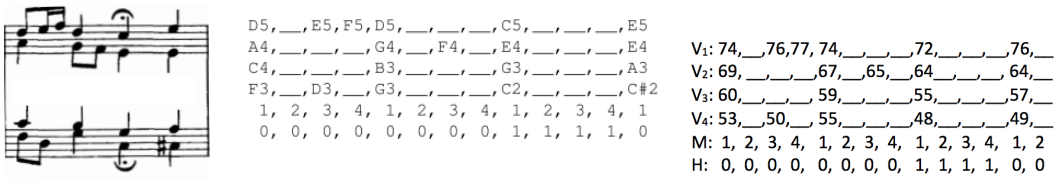
1) Consecutive notes with the same pitch have to be separable without changing or losing original musical information.



Algorithm 1 SAM algorithm

1: *Input:* $g_i(\cdot|x_{/_i})$, $i = 0, 1, 2, 3$
2: *Initial state* ← Generate a random $x^{(0)}$
3: *Initial temperature* ← Generate a random $T > 1$
4: **for** Voice: $i = 0, 1, 2, 3$ **do**
5: **for** Iteration: $t = 0 : t_m$ **do**
6: Randomly generate a sample $x^{(t+1)}$ according to $g^{(t)}(\cdot|\mathcal{N}(x^{(t)}))$
7: Compute Boltzmann's probability: $\mathcal{B}(T_s) = e^{\frac{P(s_{t+1}) - P(s_t)}{k \cdot T_s}}$
8: Accept the proposal with probability $min(1, \mathcal{B}(T_s))$.
9: Decrease temperature linearly: $T_{t+1} = \alpha T_t$
10: **end for**
11: **end for**
12: *Output:* Final state x^{t_m}

Figure 4.1 SAM-Bach System Architecture



(a) (b)

Figure 4.2 Music tuple representation

2) Temporal scope should be properly divided and flagged as Bach chorales strictly follow simple time signature.

3) Temporal and rhythm information ought to be also considered and can be easily modified.

Application requirements

1) The encoded representation should be compatible with both learning and sampling processes.

2) To be able to view and listen to the generated chorales, the encoded chorales should enjoy the ease of transitions between different formats, such as XML and MIDI etc. corresponding to music score and audio respectively.

4.1.2 Bach Chorale Representations

There are two genres of music data need to be encoded in Bach chorales: chorale score and rhythm information. The music sheets of Bach chorales are firstly transformed into notations defined in music theory (**step 1** in Figure 4.1). Metadata related with rhythm (that are embedded in the Bach time-signature) will also be appended after note notations as shown in the bottom two rows in **steps 1-3**.

In **step 2**, we employ the encoding method Hadjeres and Pachet [2016] and build a bag of notes for each voice (soprano, alto, tenor and bass). Notes are numbered in the order of their first appearances, thus the same number may refer to different notes in different voice parts.

Music data is a kind of sequential data, and we use a tuple (v_i^t, m_t, h_t) to horizontally encode Bach chorale at time step/slice t. v_i^t corresponds to the note being playing in voice i at time t. Consider that there might be empty slots because of the absences of notes. These absences are denoted as dashes. m_t indicates the number of subdivision of the note at current time t and h_t represents if notes at time t are held or not. If we stack these horizontal tuples by their timestamps, we obtain a tuple of matrices, which has the form as follows:

$$(V, M, H), \tag{4.1}$$

where V consists of notes of all four voices, M stands for metronome/subdivision indicating the order of current note in terms of the four subdivisions per beat and H represents the indicator of hold/fermata.

The empty slot and metronome perfectly ensure the simple time signature feature of Bach chorale. The combination of empty slot and holding flag enables the adjustment of duration without touching the voice pitches. Moreover, the blank pitch also allows the learning process to deem it as an element to be learned and leave spaces for the potential pitches to be generated in the sampling process.

The employed data representation is easy to encode notes but chords since there is only one slot assigned to each time slice per voice. Therefore, the data has to be in the format of all notes to use this representation. It might take some extra efforts if one wants to encode more generalized polyphonic music data. Additionally, it is neither feasible to encode holding symbols that are only applied to a certain part. If there is an indicator in the hold H, it means all notes in four parts will be held simultaneously. Therefore both training and generated data will not have these features.

After encoding procedure, data (V, M, H) except the current note will be further divided into five parts then fed into several different NNs. The training inputs include three types of data: 1) 16 neighbors of notes on the same voice(left and right are combined together); 2) notes at the same time slice 3) all the other notes excluded those in 1) and 2)(left and right are preserved separately). The visualized division is shown in the sub-picture after **step 3** in Figure 4.1.

4.2 Probability Distribution Estimation for Polyphonic Bach Chorale

The second component of SAM-Bach is the learning network, which outputs four conditional probability distributions of the four parts in chorales. The goal is to estimate any note of any voice given its neighbors' information. Each value of a final softmax output indicates the probability of the occurrence of a certain note given the embedding of its neighboring data.

The role of sequential models of this work is mainly about Bach chorale pattern learning. We assume there is a conditional distribution embedded in each part of the Bach music and use deep learning to summarize an approximation of it.

4.2.1 The Learning Scope of Training Process

Unlike common sequential learning models, which capture long-term dependencies of input data at each iteration, we only consider the notes within a sliding window of a predefined size. The motivations of this simplification are summarized as follows:

1. Bach chorales have a predefined temporal structure which uses simple time signature. The rhythm data of a certain note is mainly affected by its direct neighbors, instead of notes at far.

2. Music data needs more attention to the flow, instead of the semantic meaning like language words. The feeling of smoothness and the melodic pleasant are tended to be defined with a range of consecutive notes. Therefore, it allows us to focus on a local scale of data.

3. J.S.Bach chorales is composed by Johann Sebastian and Bach chorales are polyphonic which contains more information than monophonic music, so the perceptual range of human in the composing process is actually limited. Considering local notes might somewhat preserve the human composing behavior.

4.2.2 Training by Relevance

The division of input data in each sliding window is described in the last subsection, which actually relates with their relevances to the current note.

The notes at the same time step directly decide whether the four parts sound harmonic or not being played together. We use a stacked perceptron network (denoted as Multiple-layer Perceptron(MLP) in Figure 4.1 for simplicity) to extract the harmonic information at the same time slice.

Likewise, the nearby notes on the same part affect the melodious property of a chorale, especially for the soprano part which serves as the leading melody in a Bach chorale. We use a bidirectional LSTM network to learn the tuneful embeddings of its consecutive notes from two directions. The remaining notes positioned before or after current time slice are deemed as environmental inputs and trained by a stacked LSTM network of two layers. The furthest time steps of both sides will be initially fed into the corresponding networks.

4.2.3 Homogeneous Training Outputs

The underlying motivation of learning conditional probability distributions is to make the outputs compatible with the iterative sampling strategy. The learning process yields four dependency networks(See Chapter 3) of notes in different parts, which can be mathematically expressed by:

$$P_{i,t}(V_i^t|V_{/i,t}, M, H, \theta_i, t), \;\; i = 1, 2, 3, 4$$

Since SAM sampling is an MCMC method, we need to guarantee the input conditional distributions should be time-irrelevant so as to maintain the homogeneity property of the obtained Markov chain in the sampling process. Formally, we need to have:

$$\theta_i = \theta_{i,t}, \;\; p_i = p_{i,t} \forall t$$

We can rewrite the distributions as:

$$P_i(V_i^t|V_{/i}, M, H, \theta_i), \;\; i = 1, 2, 3, 4 \tag{4.2}$$

The general idea of this network is that given of information neighbors of a certain note in a certain voice, one can obtain the conditional probability of all candidate notes. The employed network contains four NNs dealing with three different tasks. Two stacked LSTMs are constructed to learn latent presentations of neighbors located in the former and latter time slices within the sliding window. All four latent presentations of four networks will be further merged as one vector of an MLP which is train by MLE:

$$\max_{\theta_i} logP_i(V_i^t|V_{/i}, S, F, \theta_i), \quad i = 1, 2, 3, 4 \tag{4.3}$$

The softmax layer of this MLP is the final output of the learning process. The same scanning and training procedures apply to all four parts. The specific details of implementations are provided in the next chapter.

A nicer practice would be using fixed window instead of sequence modeling, which would be helpful to test empirically. However, due to the time issue, we leave this as a future work.

4.3 SAM Sampling for Bach Chorale Generation

Having obtained the dependencies of four voices, we perform SAM sampling to generate Bach-like chorales. The proposed SAM sampling is essentially a marriage of the fast simulated annealing algorithm proposed by Dueck and Scheuer [1990] and a variant of Metropolis-Hasting Metropolis et al. [1953] in Bayesian inference. We name it as SAM to distinguish it from SAMC which stands for stochastic approximation in Monte Carlo. SAM is named after simulated annealing Metropolis sampler.

4.3.1 Incorporation of MCMC and SA

We have narrowed down the strategy space to sampling1, and discussed the drawbacks of both standard MCMC and SA in chapter 2. In this part, the focus will be the proposed SAM sampling algorithm and elaborate how MCMC and SA interact with each other.

Let's first look at the Metropolis algorithm to see the relation between MCMC and SA. Metropolis algorithm is a simple method for simulating the evolution to the thermal equilibrium of a solid for a given temperature Metropolis et al. [1953]. On the other hand, SA Kirkpatrick et al. [1983] can be deemed as a variant of the Metropolis algorithm with the temperature varying from high to low. SAM sampling creates a sequence of homogeneous Markov chains(one at each temperature) or in other words, a non-homogeneous Markov chain because that the stationary probability varies as the temperature changes. Therefore, the Metropolis algorithm is equivalent to a single run in SA.

The most salient difference between the simulated annealing and Metropolis samplers is the "cooling schedule" of SA. In fact, simulated annealing becomes fundamentally more "narrow-minded" as time goes by it finds a certain type of prospective proposal it likes, and thereafter goes searching only for close neighbors. With a very quick cooling schedule, this can result in a skinny and tall posterior; when applying SA for MCMC, we have to make sure to use a schedule that allows for sufficient exploration of the parameter space. Table 4.1 lists the comparisons of essential components in these algorithms.

	BI	**Metropolis sampler**	**SA**
Target	Bayesian posterior	$g^{(t)}(\cdot\|V^{(t)})$	State with minimal energy
Parameters	Model parameters	N/A	Temperature
Updating rule	Bayes theorem	MH/M Ratio	Boltzmann criterion

Table 4.1 Three prototype algorithm of SAM

One well-known application of the MCMC is performing Bayesian inference where we replace the target distribution with Bayesian posterior distribution. Bayesian inference aims to figure out the value of model parameters were employed in generating the given data. The comparison of Bayesian inference is also provided in Table 4.1 to assist the understanding of the incorporation of SA and the Metropolis sampler.

4.3.2 Notations in SAM Algorithm

Since SAM is derived based on MCMC and SA algorithms, there are several terms have been introduced in chapter 3.

1) $V^{(t)} = (v_1^{(t)}, ..., v_d^{(t)})$: the system state at time t, $v \in V$, where V is the our note space of current voice. We omit the voice index here for simplicity however runs of SAM on four voices are exactly the same, d is the size of the bag of notes of current voice.

2) $\mathcal{N}(V^{(t)})$: the set of potential states consisting of $V^{(t)}$'s neighbors, in our case, it refers to the notes around the target note.

3) $g^{(t)}(\cdot|V^{(t)})$: the proposal transition probability distribution conditioning on current state, which is obtained by training process.

4) V^*: the neighbor which is randomly selected as the proposal state $V^{(t)}$ at time t.

5) $\Delta\mathcal{E}(t) = \mathcal{E}(V^*) - \mathcal{E}(V^{(t)}) = P(V_{t+1}) - P(V_t)$: Energy gain of given proposal, in SAM it refers to the probability gain if moving to the proposal from current state.

6) $f(t, T_t)$: a function of the current temperature and total iteration times, which yields a scalar as a deduction factor for the tuning process of the temperature parameter.

7) k: Boltzmann's constant.

8) $\mathcal{B}(T_t) = e^{\frac{\Delta\mathcal{E}}{k \cdot T_t}} = e^{\frac{P(V_{t+1}) - P(V_t)}{k \cdot T_t}}$: the Boltzmann's probability threshold of accepting a worse proposal, where $P(V^t)$ is the probability of being at state V^t at time t.

Algorithm 4 SAM algorithm

Input: $g_i(\cdot|V_{/i})$, $i = 0, 1, 2, 3$
Initial state $\leftarrow$ Generate a random $V_i^{(0)}$, $i = 0, 1, 2, 3$
Initial temperature $\leftarrow$ Generate a random $T > 1$
for Voice: $i = 0, 1, 2, 3$ **do**
    for Iteration: $t = 0 : t_m$ **do**
        Randomly generate a sample V_i^* from $g^{(t)}(\cdot|\mathcal{N}(V_i^{(t)}))$.
        Compute Boltzmann's probability: $\mathcal{B}(T_t) = e^{\frac{P(V_i^{t+1}) - P(V_i^t)}{k \cdot T_t}}$.
        Accept the proposal V_i^* as V_i^{t+1} with probability $min(1, \mathcal{B}(T_t))$.
        Decrease temperature linearly: $T_{t+1} = f(t, T_t) \cdot T_t$.
    end for
end for
Output: Final chorale V

4.3.3 SAM Sampling

In the implementation, we run the SAM algorithm through batches as we train within a sliding window to learning the distribution. A major modification is to take out the inner loop and adjust the temperature accordingly. The reason for doing this is to improve the sampling efficiency. Since our initial score is generated randomly, if we run iterations and generate proposals based on these non-Bach neighboring notes, it will take too long to see a Bach-like chorale as found during the experiment. Putting the temperature outside of the voice loop ensuring that four voices enjoy the same level of accepting the ratio of a new proposal, because the value temperature directly affects the acceptance of a worse proposal. We would like every voice shares the same opportunity of jumping out of the local optima. The batch version of SAM sampling is provided as follows:

Algorithm 5 Batch version of SAM algorithm

Input: $g_i(\cdot|V_{/i})$, $i = 0, 1, 2, 3$
Initial state $\leftarrow$ Generate a random $V_i^{(0)}$, $i = 0, 1, 2, 3$
Initial temperature $\leftarrow$ Generate a random $T > 1$
for Iteration: $t = 0 : t_m$ **do**
    Decrease temperature linearly: $T_{t+1} = f(t, T_t) \cdot T_t$.
    for Voice: $i = 0, 1, 2, 3$ **do**
        for Batch: $b = 0, 1, 2, ...$ **do**
            Randomly generate a sample V_i^* from $g^{(t)}(\cdot|\mathcal{N}(V_i^{(t)}))$.
            ($\mathcal{N}$ is restricted to the sliding window size.)
            Compute Boltzmann's probability: $\mathcal{B}(T_t) = e^{\frac{P(V_i^{t+1}) - P(V_i^t)}{k \cdot T_t}}$.
            Accept the proposal V_i^* as V_i^{t+1} with probability $min(1, \mathcal{B}(T_t))$.
        end for
    end for
end for
Output: Final chorale V

Chapter 5

Experiment and Evaluation

5.1 Configuration and Platforms

5.1.1 Training Chorales

The dataset is a corpus of J.S. Bach chorales from a python library: music21 which is designed for digital music processing by M.I.T. Instrument information has been dropped from raw data and all data are transposed into C major. The total number of obtained chorales is 2503. The first 30 Bach chorales are kept alone from training and used in the final evaluation. 70% of the remaining is used for training and other 30% for validation.

Configurations of Training Nets for melody reharmonization

The training network is built upon Keras using Tensorflow backend. The parameters of SAM-Bach architecture are shown as follows:

1. Neighborhood range: $N = 16$ subdivisions(for both left and right). The range of notes in the condition is restricted within a length N. That is to say the neighborhood range of a certain note at current time slice t is $(t - N, t + N)$.

2. Bidirectional LSTM(BLSTM) for notes on the same voice part: 200 units.

3. Multilayer Perceptron(MLP) for neighbors of the same time slice: there are two MLPs in SAM-Bach, both are two layers of 500 units with ReLU being activation functions.

4. Left/right LSTMs for non-direct neighbors: both are two stacked-LSTM layers with 200 units in each layer with 0.3 input dropout and 0.4 recurrent dropouts.

5. Output layer: a softmax layer with x_i-dim output corresponding to the size of bag of notes of part i. $X = [55, 57, 57, 76]$.

5.1.2 Configurations of Sampling Process

The values of parameters used in SAM sampling are:

1) $V^{(t)} = (v_0^{(t)}, v_1^{(t)}, v_2^{(t)}, v_3^{(t)})$: corresponds to four voices in Bach chorale.

2) $\mathcal{N}(V^{(t)})$: refers to notes in the region defined by N.

3) $g_i^{(t)}(\cdot|v^{(t)})$: four prediction models learned via deep nets, $i = 0 : 3$.

4) V^*: the neighbor with highest proposal transition probability of state $V^{(t)}$ at time t.

5) $\Delta\mathcal{E}(t) = P(V_{t+1}) - P(V_t)$: given prediction probability distribution, the probability gain of accepting a new move.

6) T: the temperature is initially set to 1.3 and is update via linear deduction rule: $T = 0.9923 * T$.

7) k: Boltzmann's constant is omitted here($K = 1$).

5.1.3 Platforms

Aside from python libraries like music21, keras and tensorflow, we use several other software or plug-ins to complete the generation task. Figure 5.1 visualizes the transformation of data and interactions between these platforms. The left-hand side depicts the training process, the right side corresponds to the sampling implementation. Descriptions of components are provided below:

1) **Musecore**: a open-source music composition and notation software which enjoys the ease of transformation between XML, MIDI and .wav etc.

2) **Musecore plug-in**: enables users to add customized features to Musecore by adding its configuration file under a certain Musecore directory. The one used to implement SAM is written in .qml by the authors of DeepBach.

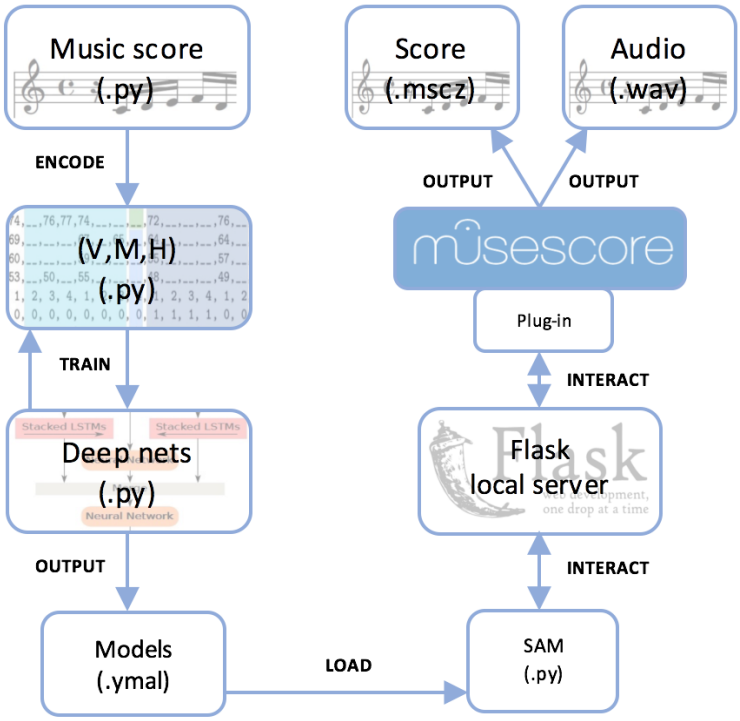


Figure 5.1 Implementations with platforms

3) **Flask**: a micro web development/application framework for Python.

4) **.ymal**: stands for 'Yet Another Make-up Language'. It can be seen as a superset of .json.

5) **.mscz**: the Musecore format to store music score sheet.

6) **SAM**: is written in a .py file serving as the flask server file. Methods in this file can be triggered by an operation in Musescore transmitted via customized Musescore plug-in.

Run flask server before composing in Musescore. The click on the compose button will trigger the SAM algorithm on the current score in Musescore. The score could be a randomly generated four parts chorales, or pre-defined melody to be reharmonized. The SAM sampling algorithm will scan the whole score to generate a refined score and present it in a new tab in Musescore.

5.2 Experiments and Results

We conduct two types of experiments with different inputs and generating strategies. The first experiment is given leading melodies, to generate corresponding Bach-style chorales. Another experiment mainly serves for the comparison of convergence rates of two sampling algorithms.

5.2.1 Melody Reharmonization

The SAM-Bach system is designed for Bach-like chorales generation given leading melody. The first 30 training chorales of the original dataset are not used for training and we randomly select five chorales from these 30 pieces for evaluation. The chorale is obtained in two steps: 1)load the melody as the soprano part and fill the remaining three parts with random notes 2) perform SAM sampling iteratively with the soprano part fixed. The aim of doing this experiment is to compare the generated chorales with the real Bach chorale and recent advance results. To compare with real Bach, we use the sopranos of preserved pieces as leading melody, To bes fairly able to compare with peer work, we choose the same instruments and speed to play our generated chorales as in the related work. Figure 5.2 shows an example of the SAM-Bach reharmonized chorale used to compare with original J.S. Bach chorale.

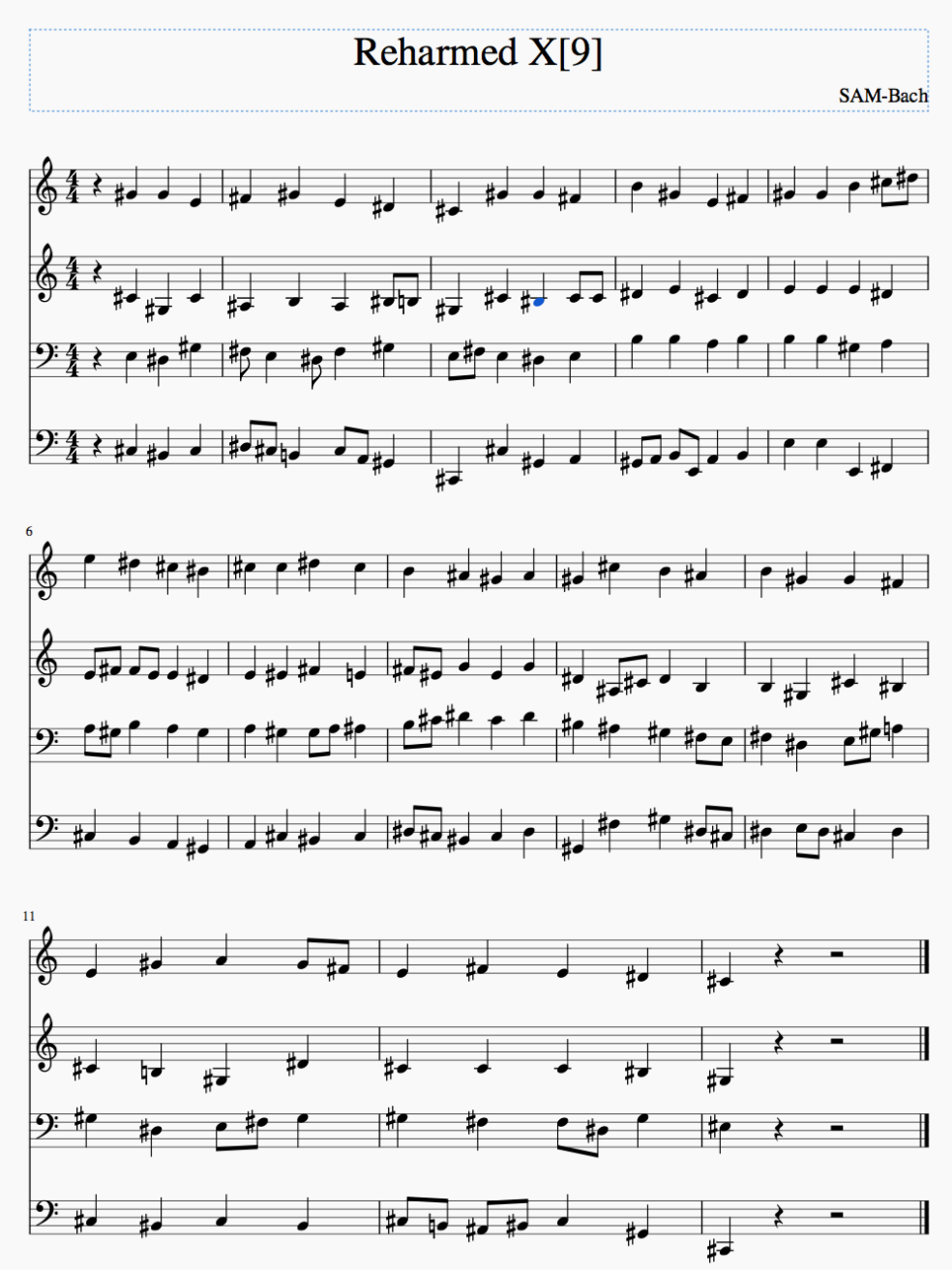


Figure 5.2 Reharmonized Bach-like chorales

5.2.2 Chorale Generation

This experiment is mainly conducted for two purposes: 1) to collect the data needed for convergence comparison; 2) to generate Bach-like chorales via different sampling methods and have them evaluated by human.

The proposed SAM sampling can be performed either in parallel or non-parallel manner. Both parallel and non-parallel SAM are compared with the most popular sampling method used for Bach chorale generation – Gibbs sampling. Since the random nature of sampling procedure, we repeatedly perform experiments for 10 times for each algorithm per group. There are totally four groups of experiments.

Parallel sampling means that four voices are updated using same unchanged neighbor information. Non-parallel sampling requires to update neighbor information and get new proposal every time a note in the current time-slice is updated. Figures 5.3 and 5.4 shows two examples of generated chorale sampled by parallel and nonparallel SAM algorithms.

5.3 Evaluation

We evaluated the generated chorales from three different perspectives: 1) preference by the human with different familiarities with Bach chorales. 2) Opinions given by Bach experts 3) Algorithm convergence analysis. Statistical significant test with random generation is also provided at the beginning of the evaluation.

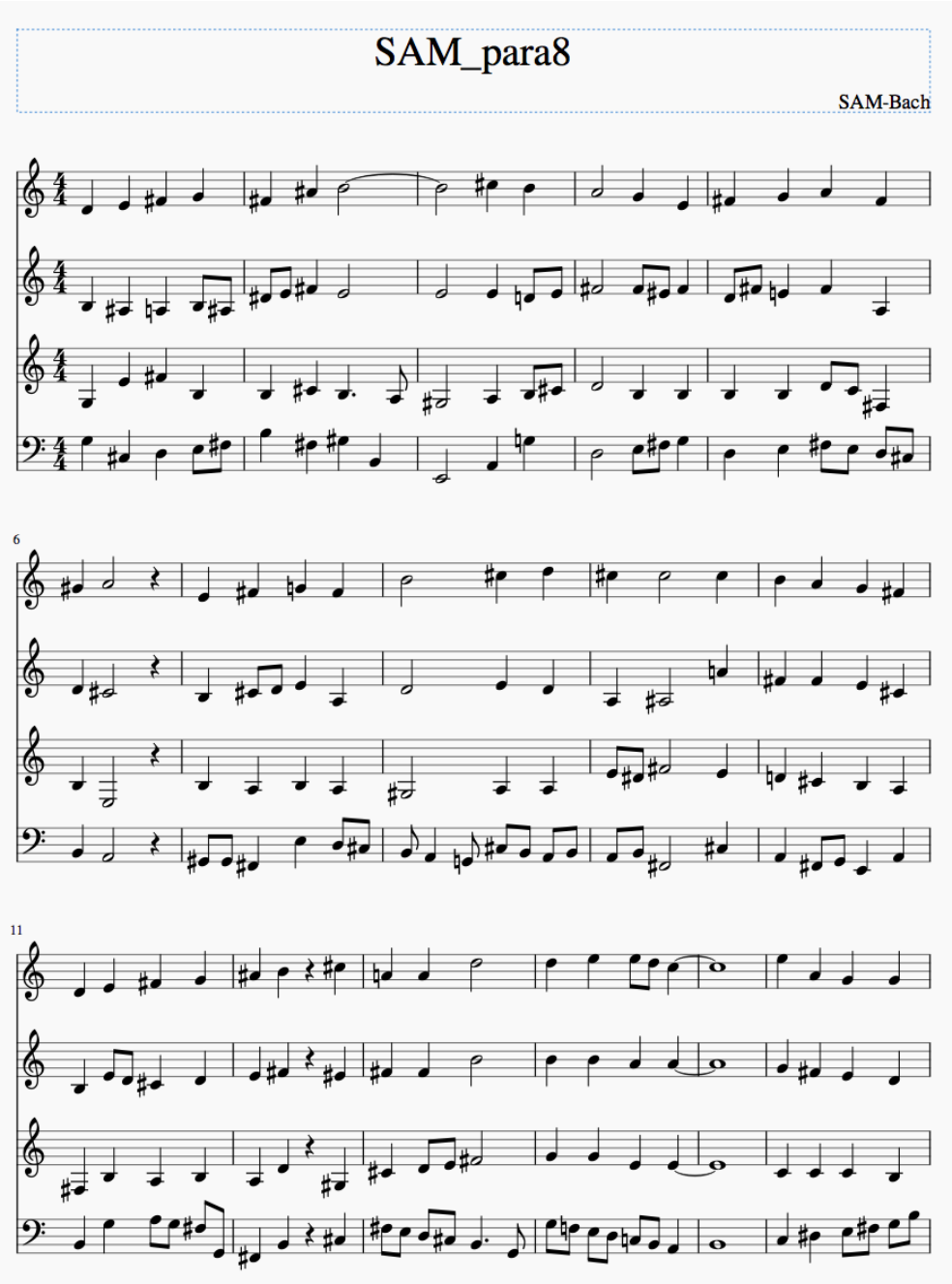


Figure 5.3 A sample of generated chorale by parallel SAM

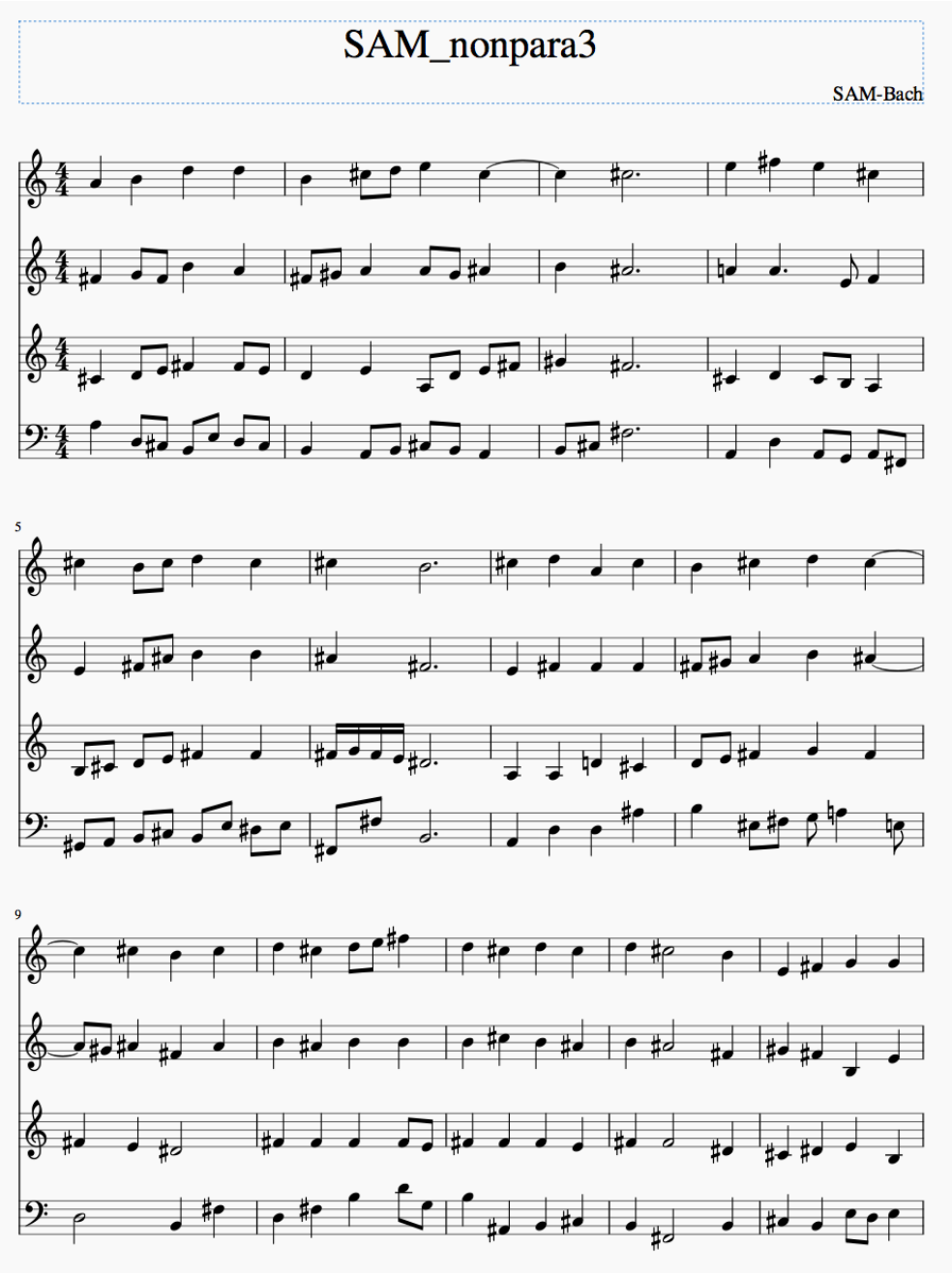


Figure 5.4 A sample of generated chorale by non-parallel SAM

5.3.1 Human Evaluation

Random Generation

To better prove the statistical significance of our work, we added human evaluation with random generation. Due to the significant difference among these two results, we only invite 27 participants in this simple sign test. We select three different SAM-generated chorale pieces to compare with random generations. All samples are remixed with the same instrument – church organ and played at half speed. As shown in fig 5.6 from the survey feedback, 100% of invited people prefer SAM-generated chorales to random generations. This figure only shows one comparison group, the remaining two share the same result and are omitted here. Figure 5.5 shows an example of a randomly generated chorale score. [1]

Reharmonization

To investigate the quality of the chorales generated by SAM-Bach, we compare our generated Bach-like chorales with the compositions by J.S. Bach, as well as with two state-of-the-art results of same task by an online survey. The survey is submitted to Amazon Mturk and receives 248 responses in total from participants with different familiarities with Bach music. Figures 5.7 summaries the preference of the subjects[2]. the oranges bars represent the number of votes for chorales generated by SAM and the table below the bar chart shows the percentage of voting results.

The results are summarized in three groups, representing the comparison between SAM-Bach and J.S., BachBotLiang [2016], DeepBachHadjeres and Pachet [2016] respectively. The cyan-yellow bars indicate the votes by participants with different levels of knowledge in Bach music. The gray-orange bars sums up the results of three kinds of participants in each group. The explanations of notations used in indexes of x-axis are: The first capital letter N, I and E refer to Novice, Intermediate, and Expert respectively. The letters: B, BB and DB represent J.S.Bach, BachBot and DeepBach accordingly.

[1] Link to the online evaluation page: https://sites.google.com/view/mc-bach/random-vs-sam?authuser=0

[2] The bar chart and the table is plotted via Matplotlib

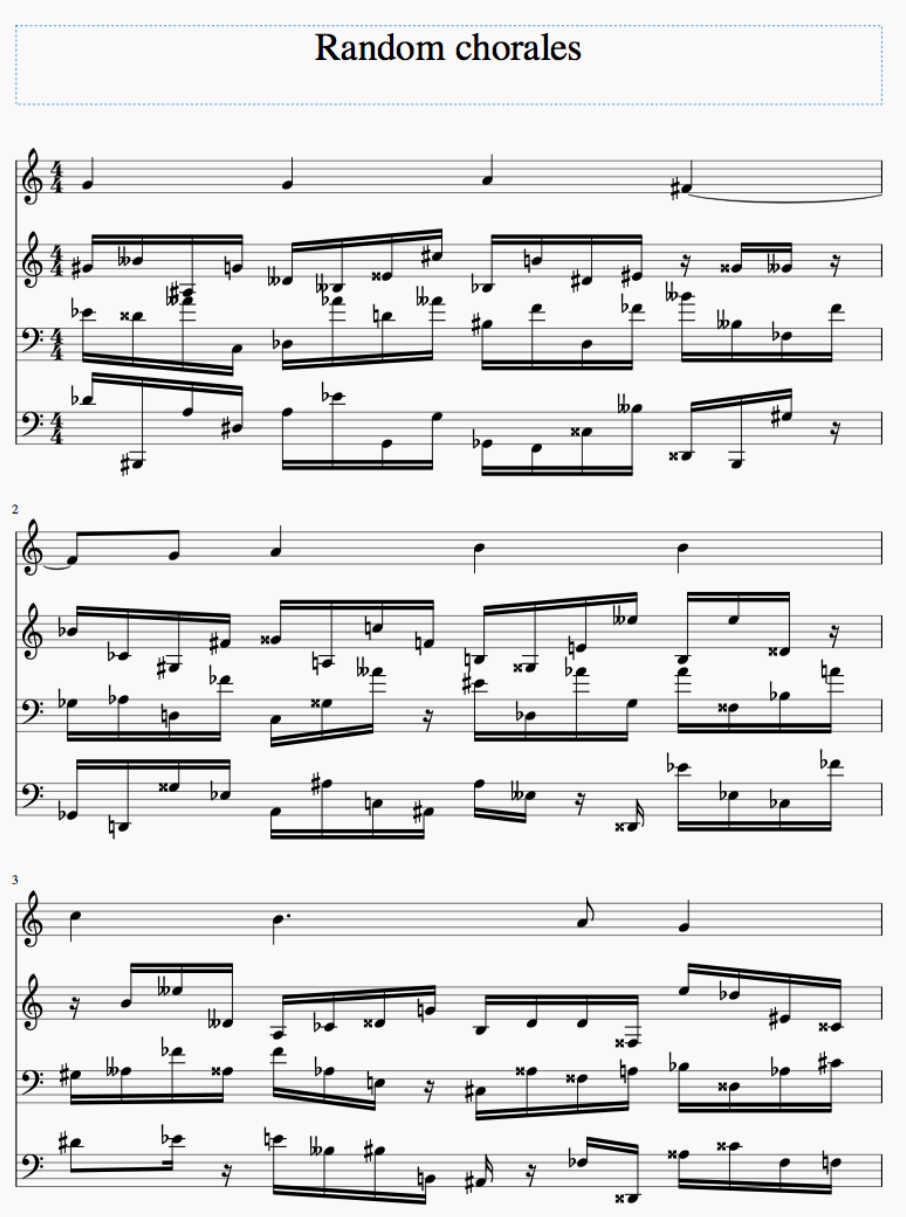


Figure 5.5 Music score of random generation

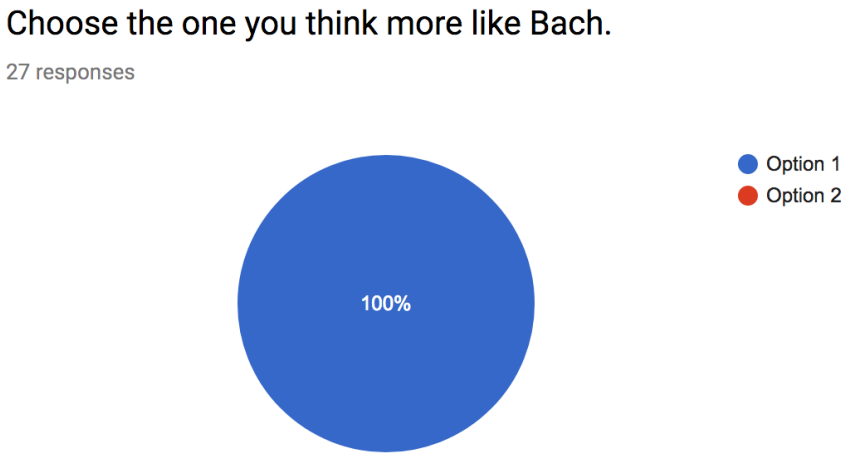
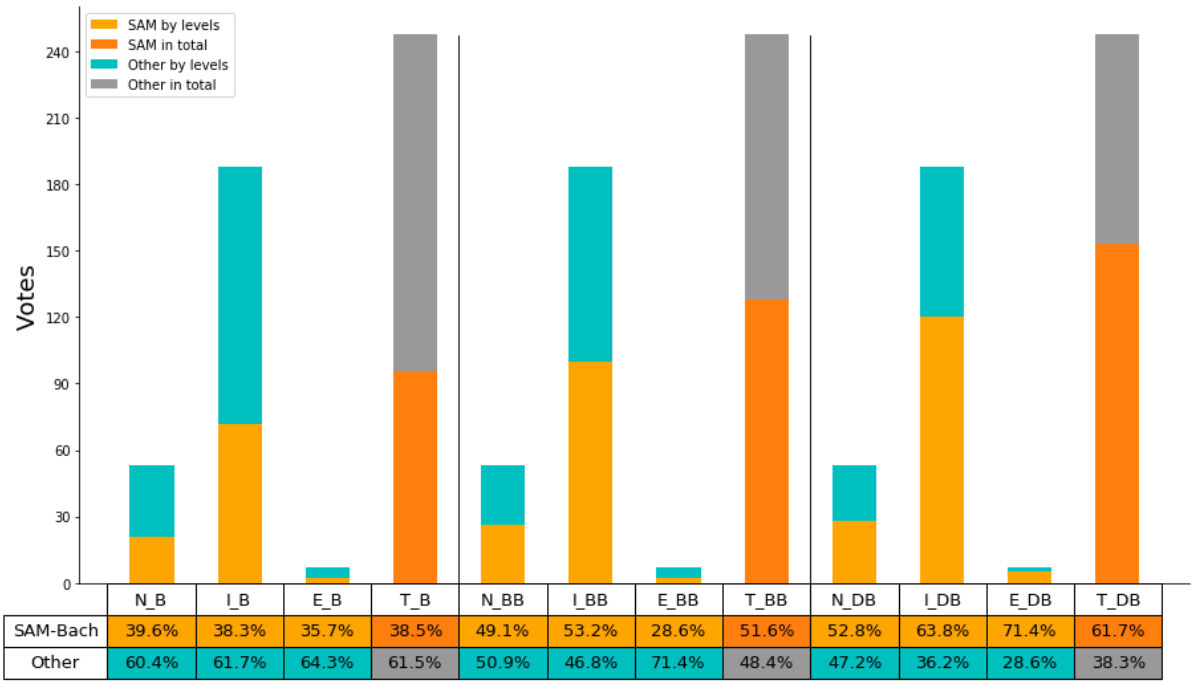


Figure 5.6 Comparison with random generation

It is easy to tell that no matter what their familiarity levels are with Bach music, the majority of our participants choose SAM-Bach chorales over other generative methods. There is not a significant gap between the preference of SAM-Bach and BachBot. BachBot explores all search space for model structures and hyper-parameters to obtain the most suitable architecture for Bach-like chorale generation. On the other hand, in the competition between SAM-Bach and DeepBach, more than 70% of people favor SAM-Bach generations.

In terms of the comparison of SAM-Bach and the real J.S.Bach chorales, although SAM-Bach won the 40% of votes, it proves that even for people with Bach music knowledge(I and E groups), the chorales generated by SAM-Bach is even compatible with the real ones.



	N_B	I_B	E_B	T_B	N_BB	I_BB	E_BB	T_BB	N_DB	I_DB	E_DB	T_DB
SAM-Bach	39.6%	38.3%	35.7%	38.5%	49.1%	53.2%	28.6%	51.6%	52.8%	63.8%	71.4%	61.7%
Other	60.4%	61.7%	64.3%	61.5%	50.9%	46.8%	71.4%	48.4%	47.2%	36.2%	28.6%	38.3%

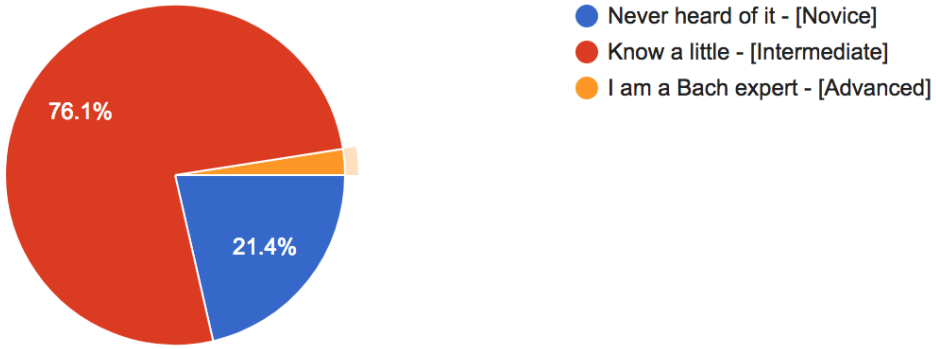


Figure 5.7 Human evaluation of reharmonization

	Soprano	tenor	Alto	Bass
1)	Church organ	Ahh choir	Violin	Ahh choir
2)	Ahh choir	Ahh choir	Ahh choir	Ahh choir

Table 5.1 Sounds played by each part

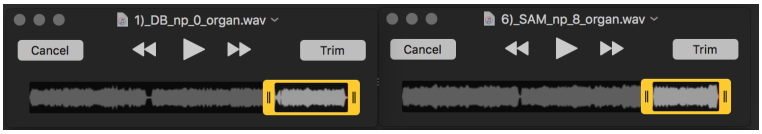


Figure 5.8 A screen-shot of trimming selection

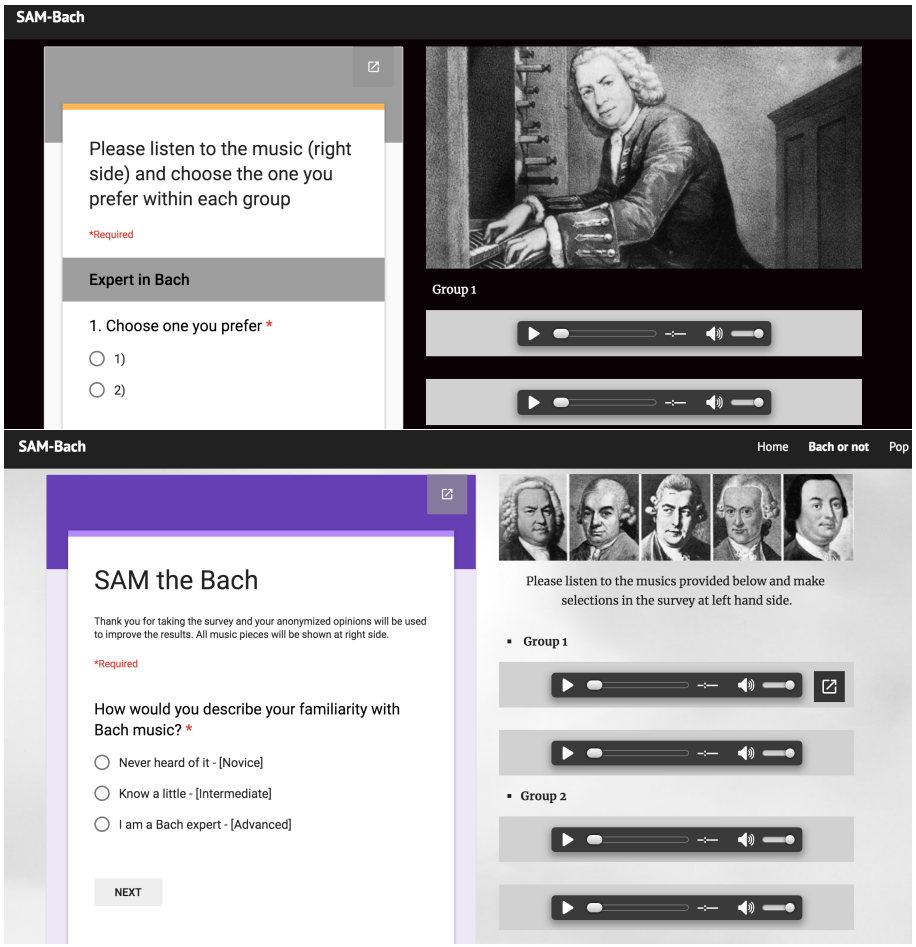


Figure 5.9 Screen shots of online evaluation survey

Chorale Generation

For the evaluation of the generated chorales, we use Google forms embedded in a Google site[3]. The audio files are uploaded and placed corresponding to the questions. All labels are removed and the order of these two methods are randomly shuffled. To make them sound more like Bach chorale, we use half speed for each part(soprano tenor alto bass) and use different sounds without any refinement on the generated score. There are total two settings as shown in table 5.1.

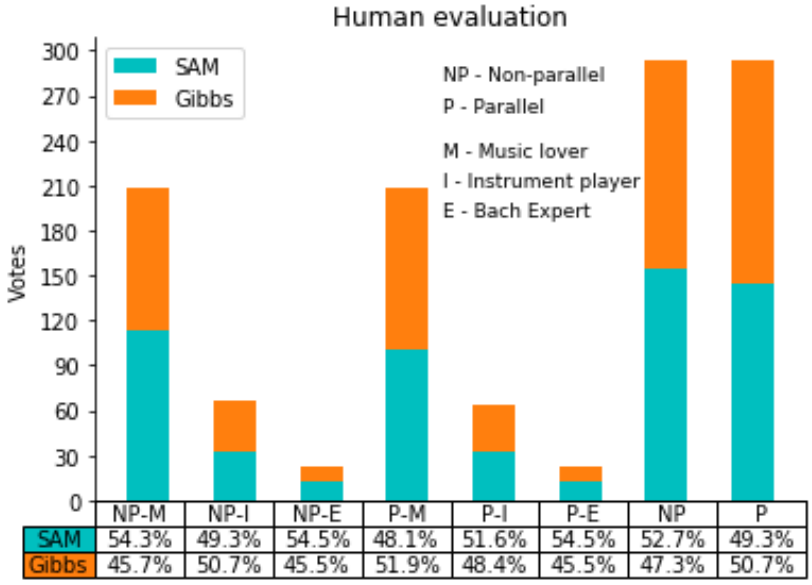
Additionally, to make sure the total time of the survey can be done with acceptable time, the generated chorale are trimmed into clips of 20 seconds. To maintain the fairness, the clips are selected from the same range of the generated piece from SAM and Gibbs as shown in figure 5.8. Every participant listens to eight clips generated by four algorithms described above and with two of each method. Therefore, each candidate has two votes. The result of voting is summarized in figure 5.10 (a) and (b). The vertical bars in (b) visualize the number of votes for preferred samples. There are total 147 participants have taken this survey by far. Among all participants, 70% of participants are music lovers, around 20% are instrument players, the remaining 8% are very familiar with Bach music. Questions are exactly the same no matter which genre a participant selects.

The last two bars are the aggregation preference of all participants. As can be seen in the table below bar chart, there is no salient preference of the chorale generated by either SAM or Gibbs.

5.3.2 Expert Comment

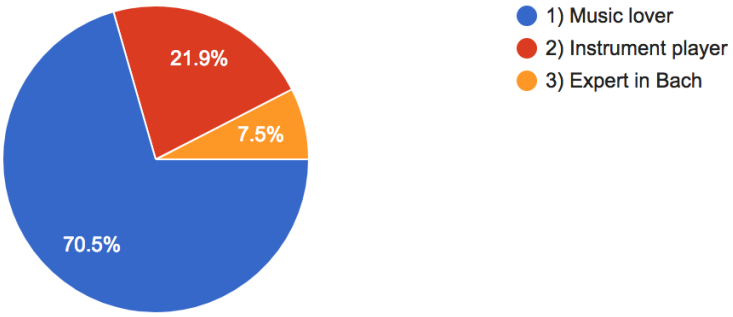
Although the voting result can indirectly reflect the harmonization level of the generated music, it does not give any concrete evaluation of the results of SAM. Therefore, we ask expert opinions from a musician Edward Wingell(double bassist) and his colleagues at Montreal symphony orchestra. Edward has been working in the music community for over three decades and is knowledgeable in harmonization as well as Bach music.

[3]https://sites.google.com/view/mc-bach/home



	NP-M	NP-I	NP-E	P-M	P-I	P-E	NP	P
SAM	54.3%	49.3%	54.5%	48.1%	51.6%	54.5%	52.7%	49.3%
Gibbs	45.7%	50.7%	45.5%	51.9%	48.4%	45.5%	47.3%	50.7%

(a) Preference results



(b) Subjects levels

Figure 5.10 Preference experiment

They commented that the generated chorale has the structure of a Bach chorale in terms of voice leading and vertical sonority yet enharmonic in its melodic progression. Each voice is in the appropriate part of the musical staff (soprano, tenor, alto, and bass) and follows a general melodic progression consistent with the style of a chorale (moves within the voice range with leading tones). However, the four parts do not always line up in a harmonic key, leading to dissonances. By saying 'enharmonic', it means two notes that are the same but are notated differently, but not indicating 'dissonance' which refers to a harmony that is inconsistent with the key signature and creates a clash.

Usually, Bach chorales are lead by a melody as the soprano, and the rest parts are composed to be harmonic with soprano, the samples generated by SAM sounds to have no melody or very noticeable melody.

5.3.3 Convergence Analysis

A common obstacle to evaluating sampling performance is how to define the gap between the probability distribution inherited in the generated samples and the distribution one tries to approximate. Due to the limiting property of MCMC methods, we can evaluate the convergence performance by measuring the distance between the proposed distribution P and the target one Q. Kullback–Leibler(KL) divergence is deemed as a common practice to quantify how one probability distribution diverges from another. It is actually the expectation of the logarithmic difference between two distributions:

$$D_{KL}(P||Q) = \sum_i P(i) log \frac{P_i}{Q_i} \tag{5.1}$$

However, this formula is not applicable in our case for the following reasons.

1. The ratio of $\frac{P_i}{Q_i}$ is within the range of $(0, 1]$. The value of $log \frac{P_i}{Q_i}$ could be from negative infinity to zero. If we use this metric and draw the convergence curves, we have to draw the y-axis long enough to capture its convergence behavior. Besides, another potential and fetal problem is overflow in the computation.

2. The distribution to be approximated here is a conditional one, which means that the distribution varies as condition changes. The target distribution is actually dynamic as to the sampling process. This adds additional computational cost to update our target as we need to go through the whole learning network to get a new inference.

Theoretically, the proposed SAM algorithm should have better convergence property than that of Gibbs as the former is a meta-heuristic global optimization approach, while Gibbs sampling is just a greedy algorithm, which is very likely to get stuck in the local optima. Given these observations and considerations, an element-wise convergence analysis is derived based upon KL divergence. We employ the inner ratio of KL divergence as the measurement(CP-ratio) which is defined as follows:

$$CP_{ratio} = P(\text{current note})/P(\text{proposal note}) \tag{5.2}$$

where the current and the proposal notes refer to the note to be reharmonized and the one proposed respectively at each iteration which corresponds to the energy concepts in the SAM algorithm.

The underlying reason why these metric works is that SAM is essentially an MCMC method, which yields a Markov chain with limiting behavior. As the states in chain approximate to the target distribution, one should be able to observe each state(element-wise) is converging to its target stationary probability accordingly. Therefore the probability of being at the current state and the proposal one should become gradually close to each other. In a fraction expression, the ratio should progressively climb to one. That is the motivation behind equation (5.2).

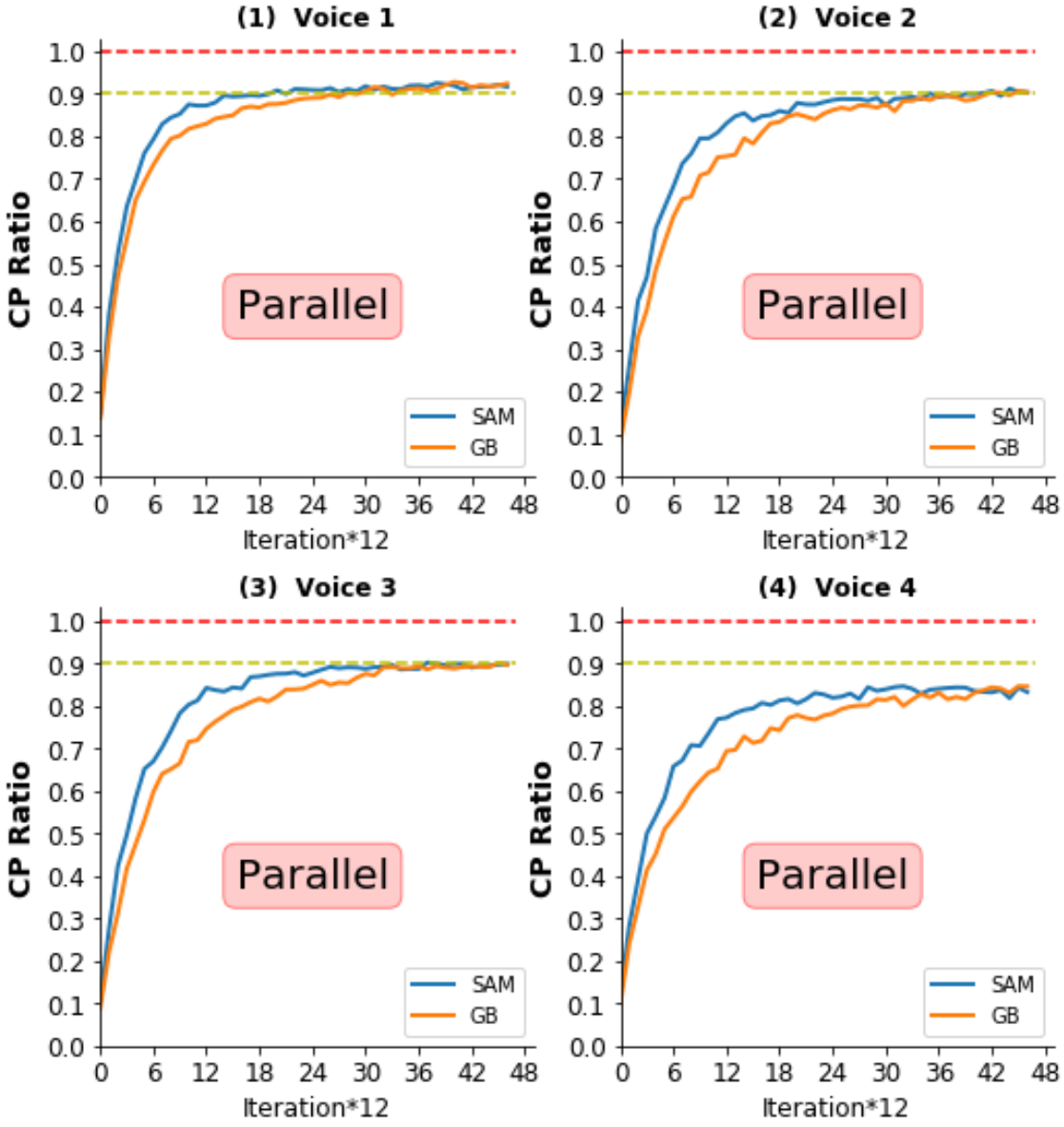


Figure 5.11 Parallel reharmonization convergence comparison

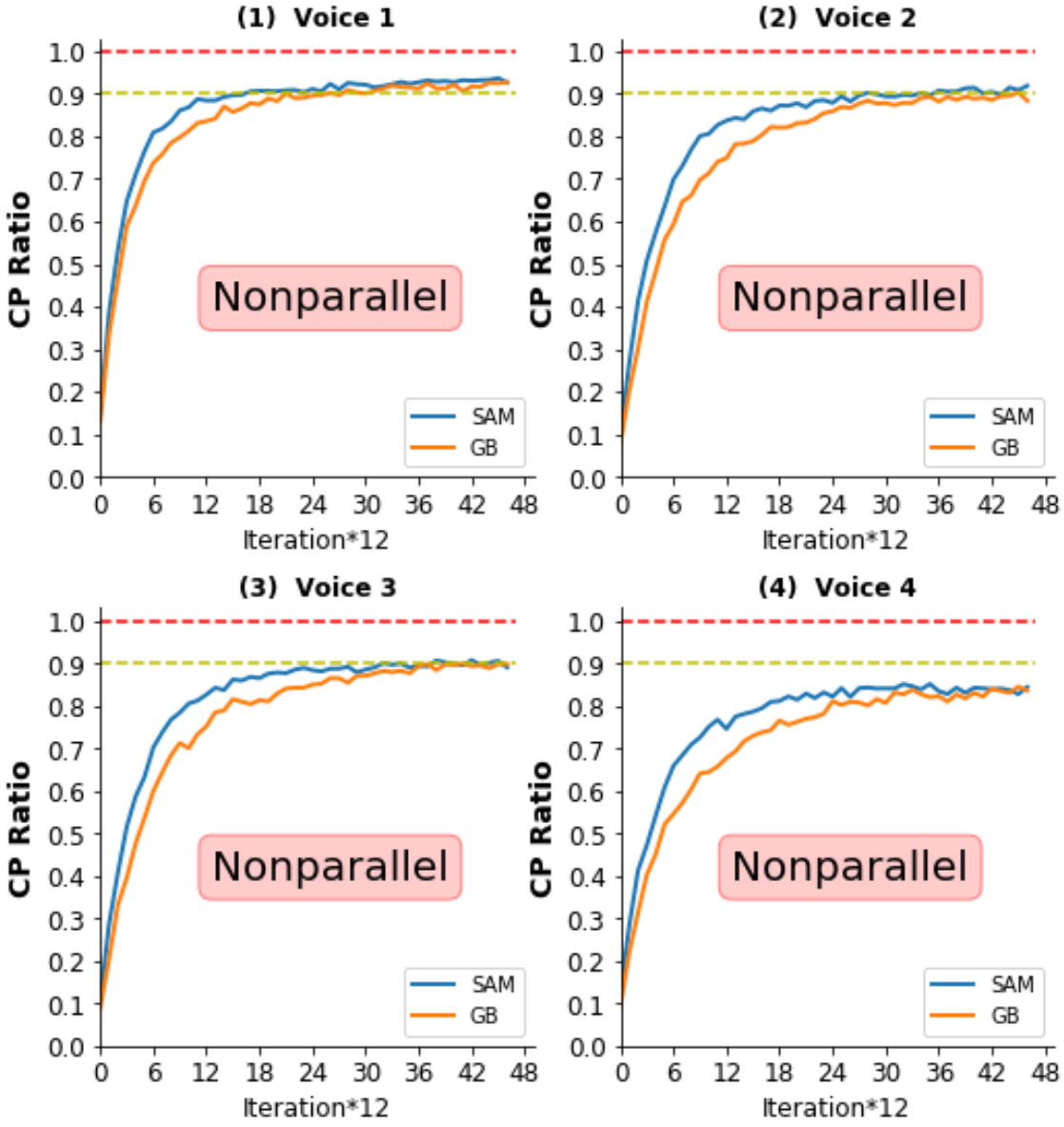


Figure 5.12 Nonparallel reharmonization convergence comparison

However, the ratio is not ideally and steadily approximate to one, but with visible fluctuations. Therefore to make the curves more presentable, points are grouped together and averaged as shown in figures 5.11 and 5.12. The analysis is conducted by voice and the numbers 1,2,3,4 are corresponding to soprano, alto, tenor, bass parts respectively in Bach chorale. Each voice is firstly averaged by the sliding window size(16), i.e. a CP ratio of voice i is computed by averaging 16 notes in the current window current iteration. There are total 564 iterations in the experiment and the curve is obtained by further taking the average of 12 iterations(i.e. each point in the curve represents $12 * 16$ original data points). To reduce the biases of randomness, all sfour experiments were conducted 10 times each and the results are then averaged. The same computation applies to figure 5.13, with a minor difference of taking another average over four voices. As shown in figures 5.11 and 5.12, it is obvious to see SAM sampling always outperforms Gibbs sampling in terms of CP curve. This observation verifies that SAM indeed has faster convergence property than that of Gibbs in terms of this setting. Additionally, another interesting fact is that the former voices tend to converge faster and better compared with the later ones. That is to say, the curves of voice 1,2,3 are closer to the ideal limit 1 (above 0.9) yet the curve of voice 4 hardly reaches 0.9.

We compare the quality of generated chorales using SAM and Gibbs via an online human survey. Aside from the music quality, another concern would be the time cost as SAM seems to take longer time to get a good proposal in the searching space. However, we compare the runtime of both sampling methods by running each sampling for 10 times and taking the average. Table 5.2 shows that the slightly more complex sampling process of SAM does not significantly increase its runtime.

	Parallel(s)	**Non-Parallel**(s)
SAM	233.0	233.8
Gibbs	229.26	240.4

Table 5.2 Runtime comparison of SAM and Gibbs sampling

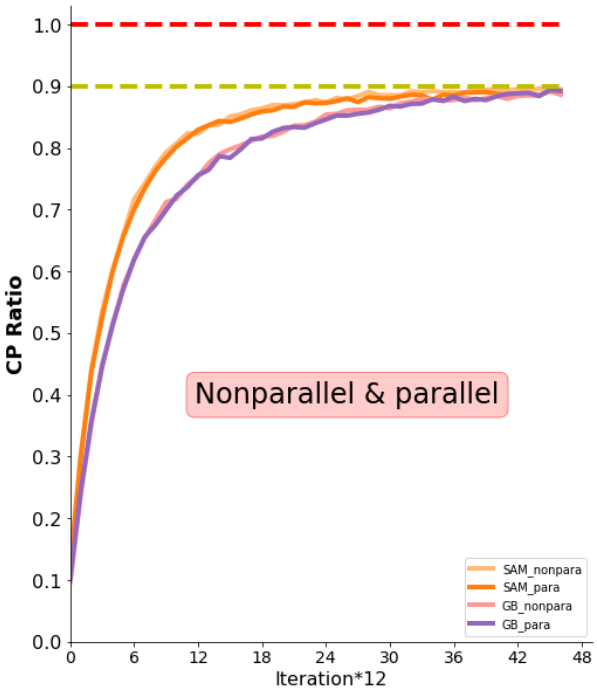


Figure 5.13 Nonparallel and parallel comparison

Chapter 6

Discussion and Future Work

6.1 Discussion

6.1.1 Discussion on the Results

Convergence of Algorithm

For most methods related to Markov chains, a common practice is to construct reversible Markov chains to ensure the convergence. On the other hand, no study claims that non-reversible MC will not converge. We just have not discovered the rules of constructing a converged non-reversible Markov chain yet. In the original Metropolis or simulated annealing algorithms, conditional distributions are derived from the same joint distribution. However, in term of the Bach chorale content, it is hard to represent its composing patterns in one particular probabilistic distribution. Therefore the learned conditional distributions are not compatible in this sense. It also has been experimentally proved that SAM sampling gives us a chain that is non-reversible but with certain limiting behavior by checking Kolmogorov's criterion. There might be two major steps bring chances to escape from being trapped in local optimal in SAM sampling. One is a random proposal selection process, the other is the acceptances of worse proposals with certain probabilities, which corresponds to the random movements of molecules and Boltzmann criterion respectively in the thermal annealing process. This fact further proves that non-reversible Markov chains can converge under some settings which are just unclear and undefined right now. The authors reference Turitsyn et al. [2011] and Vucelja [2014] proposed a method to construct a non-reversible Markov chain with better convergence property. However, this method requires to construct a reversible chain first, then modify it into a duo chain so as to make it non-reversible. For Bach chorales, it is very challenging to summarize the composing rules into one transition matrix for the corresponding ergodic Markov chain.

Take a review on the results of our work, admittedly, there is not much significant systematic improvement in terms of the quality of the generated Bach-like chorales by implementing SAM sampling algorithm compare with other existing work. However, it is trivial to see that the convergence speed of SAM sampling is visibly faster than that of Gibbs.

CP ratio limit

By observing results provided in the last chapter, one may notice that both algorithms' CP ratios converge to approximate 0.9 and keep oscillating around it. We analyze the potential reasons and summary them as follows:

1. The sampling target conditional distributions cannot be derived from the same joint distribution.

2. The parameters (such as temperature and Boltzmann probability) are not optimal.

3. The bass part seems to be less controllable compared with other parts(see the fourth subgraph in the CP ratio graphs). This might due to the nature of learning data, that is to say, the harmonization rules for bass part may less strict compared with other voices.

6.1.2 Discussion on AI-composed Music

Although music has its own theory system, it is still deemed as a very abstract subject. What it tries to convey is usually very hard to define. It could be a culture, a feeling or a story etc. Slightly different combinations of notes and tempos could produce completely different styles of musics, just like the changes of speaking tone in understanding a speech. Music is a very subtle and emotional. To capture and regenerate the pattern of a music is not easy for a human, as it may take years or even decades to grasp the skills. However, that is where the maze of AI lies in – with hours of training, the algorithm is able to generate somewhat Bach-like chorales which even could compete the original ones. The purpose of our study is not to have work done by AI, but to use its great potential and ability to better understand the world we live in and make improvements to our life.

In terms of the evaluation of AI-composed music, people are calling for more intrinsic performance measures rather than human judgments due to cost concern. However, it is very hard to make a direct relation between the measures to the quality of the produced Bach chorales as the whole process is random. What we can do is to mimic the way of human evaluation. However, the dilemma of doing this lies in that if we set a specific measure for the model and the music it produces, the measure itself is actually equivalent to a human with certain judging criteria. No matter how objective the measure is set to be, it still is an absolute evaluation instead of opinions from hundreds of human beings with individual thinkings. More and deeper investigations should be put into the measures of AI-evaluation.

6.2 Future Work

1. In the work of this thesis, we implement a semi-GSN structure deep network. There are yet many other architectures might be more suitable for the pattern learning of Bach chorales as shown in the summarization graph by Ian Goodfellow.

2. To further improve the quality of the generated chorales by refining the SAM sampling algorithm, we would like to investigate the underlying nature of the convergence criteria of non-reversible Markov chains. The four learned probability distributions are feed into the SAM sampling process as the probability equilibrium target. However, the continuously-changing temperature in the SAM sampling constructs a series of Markov chains with different stationary probabilities. To obtain better convergence performance, it would be a nice move to increase the inner loops within each temperature iteration, as currently we only employ one run to reduce computational cost.

3. The distribution is trained via LSTMs and DNNs structures. LSTMs are known for its good performance in long-term dependency sequence modeling, but Bach chorale itself does not have very long dependencies between notes of the same voice. It is possible to obtain comparable learning results by using simpler network units.

4. The generative model of this thesis is versatile and therefore could be extended to other data types like text, speech, and images etc. which are all worth trying.

Bibliography

M. Allan and C. Williams. Harmonising chorales by probabilistic inference. In *Advances in neural information processing systems*, pages 25–32, 2005.

J. Bach. 389 chorales (choral-gesange): Satb (german language edition). kalmus classic edition, 1985.

N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

J.-P. Briot, G. Hadjeres, and F. Pachet. Deep learning techniques for music generation-a survey. *arXiv preprint arXiv:1709.01620*, 2017.

S. Chib and E. Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

D. Cope. *Computers and musical style*, volume 6. Oxford University Press Oxford, 1991.

P. P. Cruz-Alcázar and E. Vidal-Ruiz. Learning regular grammars to model musical style: Comparing different coding schemes. In *International Colloquium on Grammatical Inference*, pages 211–222. Springer, 1998.

M. S. Cuthbert and C. Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. 2010.

L. Deng, D. Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

Y. Dodge. *The Oxford dictionary of statistical terms*. Oxford University Press on Demand, 2006.

G. Dueck and T. Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, 90(1): 161–175, 1990.

K. Ebcioğlu. An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3):43–51, 1988.

J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

M. Eppe, R. Confalonieri, E. Maclean, M. Kaliakatsos, E. Cambouropoulos, M. Schorlemmer, M. Codescu, and K. Kühnberger. Computational invention of cadences and chord progressions by conceptual chord-blending. AAAI Press; International Joint Conferences on Artificial Intelligence, 2015.

J. Gall, B. Rosenhahn, and H.-P. Seidel. An introduction to interacting simulated annealing. In *Human Motion*, pages 319–345. Springer, 2008.

A. Ghosh, V. Kulharia, V. Namboodiri, P. H. Torr, and P. K. Dokania. Multi-agent diverse generative adversarial networks. *arXiv preprint arXiv:1704.02906*, 2017.

W. R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

G. H. Givens and J. A. Hoeting. *Computational statistics*, volume 710. John Wiley & Sons, 2012.

K. Goel, R. Vohra, and J. Sahoo. Polyphonic music generation by modeling temporal dependencies using a rnn-dbn. In *International Conference on Artificial Neural Networks*, pages 217–224. Springer, 2014.

I. Goodfellow, M. Mirza, A. Courville, and Y. Bengio. Multi-prediction deep boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 548–556, 2013.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

P. J. Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

G. Hadjeres and F. Pachet. Deepbach: a steerable model for bach chorales generation. *arXiv preprint arXiv:1612.01010*, 2016.

G. Hadjeres, J. Sakellariou, and F. Pachet. Style imitation and chord invention in polyphonic music with exponential families. *arXiv preprint arXiv:1609.05152*, 2016.

J. M. Hammersley and D. C. Handscomb. General principles of the monte carlo method. In *Monte Carlo Methods*, pages 50–75. Springer, 1964.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1(Oct):49–75, 2000.

H. Hild, J. Feulner, and W. Menzel. Harmonet: A neural net for harmonizing chorales in the style of js bach. In *Advances in neural information processing systems*, pages 267–274, 1992.

G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

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

A. Huang and R. Wu. Deep learning for music. *arXiv preprint arXiv:1606.04930*, 2016.

C.-Z. A. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck. Counterpoint by convolution. 2016.

G. L. Jones et al. On the markov chain central limit theorem. *Probability surveys*, 1 (299-320):5–1, 2004.

M. Kaliakatsos-Papakostas and E. Cambouropoulos. Probabilistic harmonization with fixed intermediate chord constraints. In *ICMC*, 2014.

F. P. Kelly. *Reversibility and stochastic networks*. Cambridge University Press, 2011.

N. Killoran, L. J. Lee, A. Delong, D. Duvenaud, and B. J. Frey. Generating and designing dna with deep generative models. *arXiv preprint arXiv:1712.06148*, 2017.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

W. Krauth. *Statistical mechanics: algorithms and computations*, volume 13. OUP Oxford, 2006.

H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011.

S. L. Lauritzen. *Graphical models*, volume 17. Clarendon Press, 1996.

F. Liang. *BachBot: Automatic composition in the style of Bach chorales*. PhD thesis, Masters thesis, University of Cambridge, 2016.

F. Liang, C. Liu, and R. J. Carroll. Stochastic approximation in monte carlo computation. *Journal of the American Statistical Association*, 102(477):305–320, 2007.

Q. Lyu, Z. Wu, J. Zhu, and H. Meng. Modelling high-dimensional sequences with lstm-rtrbm: Application to polyphonic music generation. In *IJCAI*, pages 4138–4139, 2015.

R. A. McIntyre. Bach in a box: The evolution of four part baroque harmony using the genetic algorithm. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 852–857. IEEE, 1994.

N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21 (6):1087–1092, 1953.

T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*, 2014.

R. Paap. What are the advantages of mcmc based inference in latent variable models? *Statistica Neerlandica*, 56(1):2–22, 2002.

F. Pachet and P. Roy. Non-conformant harmonization: the real book in the style of take 6. In *ICCC*, pages 100–107, 2014.

Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

D. Quick. *Kulitta: A framework for automated music composition*. Yale University, 2014.

S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

J. Richardt, F. Karl, and C. Müller. Connections between fuzzy theory, simulated annealing, and convex duality. *Fuzzy sets and systems*, 96(3):307–334, 1998.

C. P. Robert and G. Casella. The metropolis—hastings algorithm. In *Monte Carlo Statistical Methods*, pages 231–283. Springer, 1999.

C. P. Robert, G. Casella, and G. Casella. *Introducing monte carlo methods with r*, volume 18. Springer, 2010.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

J. Rothstein. *MIDI: A comprehensive introduction*, volume 7. AR Editions, Inc., 1995.

P. Roy, A. Papadopoulos, and F. Pachet. Sampling variations of lead sheets. *arXiv preprint arXiv:1703.00760*, 2017.

J. Sakellariou, F. Tria, V. Loreto, and F. Pachet. Maximum entropy model for melodic patterns. In *ICML Workshop on Constructive Machine Learning*, 2015.

J. Sakellariou, F. Tria, V. Loreto, and F. Pachet. Maximum entropy models capture melodic styles. *Scientific Reports*, 7(1):9172, 2017.

R. R. Spangler, R. M. Goodman, and J. Hawkins. Bach in a box-real-time harmony. In *Advances in Neural Information Processing Systems*, pages 957–963, 1998.

B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova. Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*, 2016.

A. Surmani, K. F. Surmani, and M. Manus. *Alfred's Essentials of Music Theory: A Complete Self-study Course for All Musicians*. Alfred Music Publishing, 2004.

P. Tahmasebi, F. Javadpour, and M. Sahimi. Stochastic shale permeability matching: Three-dimensional characterization and modeling. *International Journal of Coal Geology*, 165:231–242, 2016.

F. Tibaldi, G. Molenberghs, T. Burzykowski, and H. Geys. Pseudo-likelihood estimation for a marginal multivariate survival model. *Statistics in medicine*, 23(6):947–963, 2004.

P. Todd. A sequential network design for musical applications. In *Proceedings of the 1988 connectionist models summer school*, pages 76–84, 1988.

P. M. Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.

C. P. Tsang and M. Aitken. Harmonizing music as a discipline in contraint logic programming. In *Proceedings of the International Computer Music Conference*, pages 61–61. INTERNATIONAL COMPUTER MUSIC ACCOCIATION, 1991.

K. S. Turitsyn, M. Chertkov, and M. Vucelja. Irreversible monte carlo algorithms for efficient sampling. *Physica D: Nonlinear Phenomena*, 240(4-5):410–414, 2011.

B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning*, pages 467–475, 2014.

M. Vucelja. Lifting–a nonreversible markov chain monte carlo algorithm. *arXiv preprint arXiv:1412.8762*, 2014.

H. Wang, B. Raj, and E. P. Xing. On the origin of deep learning. *arXiv preprint arXiv:1702.07800*, 2017.