Computational Harmonic Analysis and Prediction in the Bach Chorales

by Hugh P. Zabriskie ’16

submitted in partial fulfillment of the requirements for an AB degree with honors in Computer Science and Music

Departments of Computer Science and Music
Harvard College

March 7, 2016

Abstract

The Baroque composer J.S. Bach harmonized over 300 chorale melodies over his lifetime, which today remain a pivotal body of music in the history of Western music and exemplify Bach's groundbreaking compositional techniques. Bach's harmonizations establish a series of compositional conventions that musicians today continue to emulate in order to learn 4-part counterpoint. By transforming the Chorales into a numerical dataset, this thesis examines the ability of a variety of machine learning models to learn perform harmonic analysis over the chorale melodies as well as generate new and stylistically appropriate harmonizations.

An introduction to the core theoretical concepts is provided in Chapter 1 as a foundation for the interdisciplinary research conducted in this thesis. Chapters 2 and 3 explains the methods and results for the non-neural and neural models, respectively, demonstrating the ability of both model types to effectively analyze and generate chorale harmonizations. Chapter 4 seeks to generalize these findings and explore the task of contrapuntal melodic generation, focusing on the two-voice counterpoint in Bach's Inventions.

Acknowledgements

To be completed.

Contents

Chapter 1

A Theoretical Overview

This thesis examines the potential for algorithmic models to learn the processes and conventions involved in harmonizing a chorale melody that Bach pioneered centuries ago. One objective of this work is to improve upon previous computational attempts at chorale harmonization by consistently incorporating musical knowledge about the Chorales and the general approach to harmonizing a melody into the major design decisions of the research. By thoughtfully extracting features and selecting model architectures that are well-suited to the format of the musical dataset, we hope to advance computational knowledge related to the Chorales and the ability of various models to approximate the complex tasks of harmonic analysis and melodic harmonization.

The purpose of this chapter is to provide a high-level introduction to the topics in the music and computer science fields that are relevant to this thesis. The highly interdisciplinary nature of this research requires a certain level of familiarity with both the fundamentals of music theory and counterpoint, as well as the mathematical mechanisms that drive the machine learning models used in later chapters.

1.1 An introduction for the musician

Supervised machine learning

Supervised learning is a general task in machine learning concerned with learning outcomes based on observations from a dataset. Machine learning can be broadly divided into two categories, supervised and unsupervised learning, which are defined by their different learning objectives. Unsupervised learning involves examining datasets for pattern or statistical structures that define the dataset. There are no explicit outcomes specified, so an unsupervised learning algorithm has no sense of the ”correct” answer. In contrast, a supervised learning model is trained to correlate observations from a data set with a corresponding set of outcomes. Given a dataset of observations and outcomes, the model can learn to predict future outcomes. Consider the task of learning to predict the price of a car given a set of information that describes the car's features (i.e. color, year manufactured, etc.). In supervised learning, the model observes several cars and their known prices, and then updates it parameters in order to more accurately predict other car prices based on the data it has observed. Therefore, the primary task is to optimize the model's *parameters*, denoted by the symbol θ, in order to improve accuracy of predictions. In algebraic terms, θ is often a n-dimensional array of parameters such that our prediction is modeled as $h(x) \sim \theta^T x$. During training, the model updates its parameters based on a cost function - a metric for calculating the error between the predicted outcome and the known outcome - in order to reduce error for future predictions.

Distributions and classification

The models used in this thesis are classifiers, meaning that they accept a vector of input features - or a series of input vectors - and for each input, it outputs a corresponding *distribution*. In logistic regression and other classification algorithms,

the possible outcomes or values of y represent a discrete set of k classes, and the distribution produced by the algorithm represents the probability that $\boldsymbol{x}$ belongs to each class. These distributions are initially unknown, but the objective is to learn to estimate these distributions given a dataset of observations and known outcomes. Mathematically, the objective is to learn the mapping $f : \mathcal{X} \to \mathcal{Y}$, given a set of n examples, such that $h(\boldsymbol{x}) \approx \boldsymbol{y}$, where $\boldsymbol{x}$ is an individual input vector. The training data is the subset of the original data on which the model learns $h(\boldsymbol{x})$ by updating its parameters, and the model's ability to predict future outcomes is measured based on a test set.

Logistic regression

Logistic regression is a binary (2-class, $\{0, 1\}$) classification algorithm. Given an observation $\boldsymbol{x}$, logistic regression predicts a binary outcome, where $\boldsymbol{x}$ is classified as $\boldsymbol{y} = 1$ with probability p, and $\boldsymbol{y} = 0$ with probability $1 - p$ (this is also known as a Bernoulli distribution). More simply, logistic regression models $h : \mathcal{X} \to \mathcal{Y} \in \{0, 1\}$ (Murphy, 2012, p. 21). A real-life example of logistic regression might be predicting a binary outcome of a patient (i.e. they have a disease or not) given an input vector that describes a patient's symptoms.

$$P(\boldsymbol{y} = 1|\boldsymbol{x}, \theta) = \text{Ber}(\boldsymbol{y} = 1|h(\boldsymbol{x}))$$

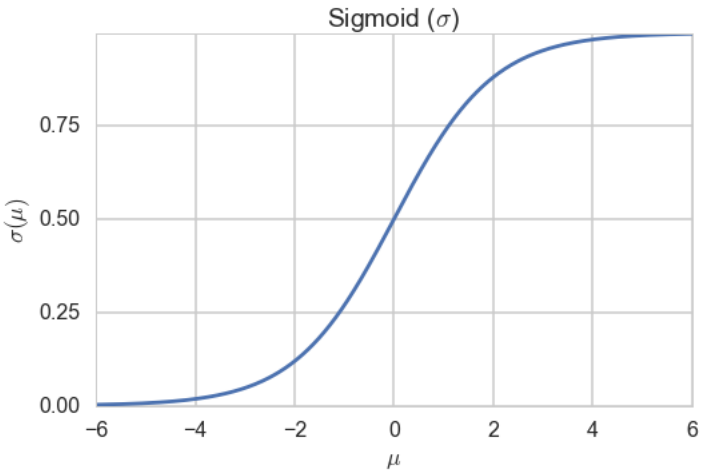
The output distribution is therefore a function (specifically, a linear combination) of the input $\boldsymbol{x}$ and the model parameters θ.

$$h(\boldsymbol{x}) \sim \theta^T \boldsymbol{x} = \theta_1 \boldsymbol{x}_1 + \theta_2 \boldsymbol{x}_2 + \ldots + \theta_m \boldsymbol{x}_m$$

where m is the number of features, and $\theta^T \boldsymbol{x}$ representing a activation value. Different parameters for θ will alter the activation for the given input, and therefore update the model's prediction for $\boldsymbol{x}$. In order to obtain a probability distribution from

$\theta^T \boldsymbol{x}$, the sigmoid function $\sigma(\mu)$, or "squashing function" is then applied, which maps any real value in $\mathbb{R}$ to the range $[0, 1]$.

Figure 1.1: The sigmoid function: $\sigma : \theta^T \boldsymbol{x} \in \mathbb{R} \rightarrow [0, 1]$



The *hypothesis*, $h_\theta(\boldsymbol{x})$ given the weights θ, is defined as

$$h_\theta(\boldsymbol{x}) = \sigma(\theta^T \boldsymbol{x}) = \sigma\Big(\sum_{i=1}^{n} \theta_i \boldsymbol{x}_i\Big)$$

$$\Pr(y|\boldsymbol{x}, \theta) = \text{Bern}(y|h_\theta(\boldsymbol{x}))$$

Finally, y is mapped to the discrete binary set $\{0, 1\}$ using a decision boundary d, where $0 \leq d \leq 1$.

$$h_\theta(\boldsymbol{x}) \geq d \rightarrow y = 1$$

$$h_\theta(\boldsymbol{x}) < d \rightarrow y = 0$$

Based on training data, a logistic regression model learns to optimize its predictions for newly observed data by updating the weights θ. The weights in θ can be thought of as the control gates for the flow of information, and increasing the value of weight represents an increase in the importance of that information. In order to

make the parameter update, a cost function J is used to generate an error metric for the predicted outcome $h_\theta(\boldsymbol{x})$ based on the "correct" observed outcome $\boldsymbol{y}$ for each observation-outcome pair in the training data. θ is then updated based on $J(\theta)$ by a method known as stochastic gradient descent (SGD), which is not discussed further here. The ultimate objective, using SGD, is to minimize the cost $J(\theta)$ over the training data.

$$J(\theta) = \frac{1}{m} \sum_{x} Cost(h_\theta(\boldsymbol{x}), \boldsymbol{y})$$

So the optimal parameters $\hat{\theta}$ are

$$\hat{\theta} = \arg\min_{\theta} J(\theta | \mathcal{X}, \mathcal{Y})$$

Multinomial logistic regression is a generalization of logistic regression to the case where we want to classify data into K classes, not just two. The objective is to develop a hypothesis to estimate $\Pr(\boldsymbol{y} = k|\boldsymbol{x})$ for $k \in 1, \ldots, K$. This is useful in handwritten digit recognition, for example, where $\boldsymbol{x}$ is a numerical representation of an image, and $h_\theta(\boldsymbol{x})$ describes the probability that the image represents a specific digit for all digits 0-9 ($K = 10$). $\arg\max_k h_\theta(\boldsymbol{x})$ therefore tells us which digit is most likely represented in the image.

In order to represent K classes, θ is now a *matrix* of weights, making $\theta^T \boldsymbol{x}$ an activation *vector*. The sigmoid function is replaced with the softmax, which generalizes the sigmoid function and normalizes the activation vector such that the resulting vector represents a probability distribution over the K classes. Since $h_\theta(x)$ is a distribution, its elements must sum to 1.

$$h_\theta(\boldsymbol{x}) = \begin{bmatrix} P(\boldsymbol{y}=1|\boldsymbol{x},\theta) \\ P(\boldsymbol{y}=2|\boldsymbol{x},\theta) \\ \vdots \\ P(\boldsymbol{y}=K|\boldsymbol{x},\theta) \end{bmatrix} = \frac{1}{\sum_{i=1}^{K} \exp(\theta^{(i)T}\boldsymbol{x})} \cdot \begin{bmatrix} \exp(\theta^{(1)T}\boldsymbol{x}) \\ \exp(\theta^{(2)T}\boldsymbol{x}) \\ \vdots \\ \exp(\theta^{(K)T}\boldsymbol{x}) \end{bmatrix}$$

where θ is represented as

$$\theta = \begin{bmatrix} | & | & | & | \\ \theta_1 & \theta_2 & \dots & \theta_K \\ | & | & | & | \end{bmatrix}$$

The most likely classification is therefore

$$\arg\max_k Pr(\boldsymbol{y}=k|\boldsymbol{x},\theta)$$

Neural Networks

In this section, we introduce *artificial neural networks*, a family of machine learning models that are the focus of this thesis. Neural networks give us a way to model deeper interactions, and to generate predictions based on a combination of many non-linear functions - a series of cascading smaller decisions to determine a larger decision. They are exceptionally powerful, and by the Universal Approximation Theorem (Cybenko et. al. 1989), a feed-forward 3-layer neural network of finite size is proven to approximate any continuous function bounded by n dimensions with any desired non-zero error (Goldberg, 2015). As in multinomial logistic regression, the objective is K-class classification. Neural networks are loosely inspired by the architecture of biological neural networks, where the complex decisions computed

by our brains is a function of the many, small computations made by individual neurons. In an artificial network, the "neuron" is a computational unit that accepts a vector input and returns a scalar output. Neurons are organized as a series of layers, where the first is referred to as the "input" layer, the intermediary layers being the "hidden" layer, and the final layer being the "output" layer. In order to obtain the K-class distribution, $\boldsymbol{x}$ is "fed forward" through the network by a process known as forward propagation. The process is shown below for a 3-layer network, where σ is the sigmoid function (applied element wise). $\theta^{(i)}$ is the matrix of parameters that control the mapping of input units to hidden units for the ith layers. $\boldsymbol{b}^{(i)}$ is the bias neuron for the ith layer. L_i is the size of the ith layer.

$$
\begin{aligned}
\boldsymbol{z}^{(1)} &= \theta^{(1)T}\boldsymbol{x} + \boldsymbol{b}^{(1)} \\
\boldsymbol{a}^{(1)} &= \sigma(\boldsymbol{z}^{(1)}) \\
\boldsymbol{z}^{(2)} &= \theta^{(2)T}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)} \\
\boldsymbol{a}^{(2)} &= \sigma(\boldsymbol{z}^{(2)}) \\
\boldsymbol{z}^{(2)} &= \theta^{(3)T}\boldsymbol{a}^{(2)} \\
h_\theta(\boldsymbol{x}) &= \sigma(\boldsymbol{z}^{(3)})
\end{aligned}
$$

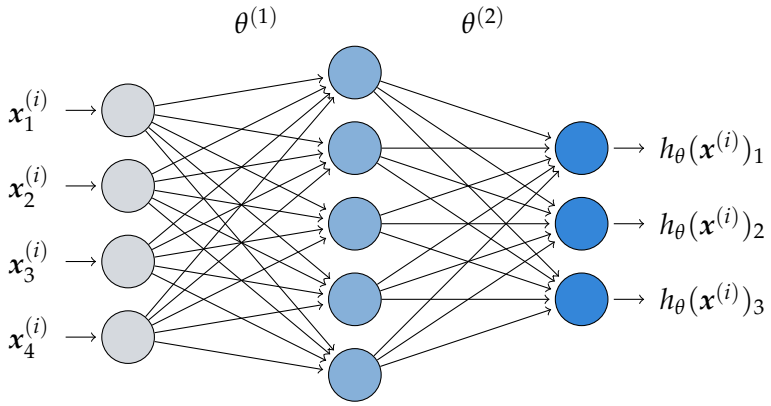
where

$$x \in \mathbb{R}^m, h_\theta(\boldsymbol{x}) \in \mathbb{R}^K, \theta^{(i)} \in \mathbb{R}^{L_i \times L_{i+1}}, b^{(i)} \in \mathbb{R}^{L_i}$$

When a layer is reached, each neuron in the layer performs a linear combination of the input and its parameters, applies a non-linear function (i.e. sigmoid σ), and then passes the result forward to each neuron in the next layer. Each connection between two neurons also carries a separate, adjustable parameter. This continues until the final layer, where the output layer returns a vector that is forwarded through a softmax function to obtain the K-dimensional distribution $h_\theta(\boldsymbol{x})$.

For all neural models used in this thesis, the objective is to minimize *negative*

Figure 1.2: Abstract representation of a three-layer neural network architecture, where each circle represents a neuron. θ controls the flow of information between activation layers.



log likelihood. Likelihood ($\mathcal{L}$) is a function of the parameters θ given a set of observations $\mathcal{X}$, and it is in fact equivalent to the probability of those observations given the parameters.

$$\mathcal{L}(\theta|\mathcal{X}) = P(\mathcal{X}|\theta)$$

For mathematical ease, log likelihood is used.

$$\log \mathcal{L} = \log \prod_{i=1}^{m} h(\boldsymbol{x}_i|\theta) = \sum_{i=1}^{m} h(\boldsymbol{x}_i|\theta)$$

And maximizing log likelihood is equivalent to minimizing negative likelihood.

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{m} h(\boldsymbol{x}_i|\theta) = \arg\min_{\theta} \Big[-\sum_{i=1}^{m} h(\boldsymbol{x}_i|\theta) \Big]$$

The parameters θ are updated by calculating $J(\theta)$ and computing the partial derivatives (the "gradients") of $J(\theta)$ with respect to each of the parameters.

$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

The partial derivatives of J, or the gradient vectors, describe the direction of *steep-*

est ascent of $J(\theta)$. So by subtracting these gradients at each update, the network descends towards a minimization of J. The learning rate, α, adds a multiplicative factor that controls the strength of each gradient update. This process of passing gradients through the network and updating θ is known as backwards propagation.

Recurrent neural networks (RNNs) and sequential data

An important limitation of "vanilla" neural networks, like the ones described in the previous section, is that they treat each input independently of all other inputs. In supervised learning tasks where the input data is in the form of *sequences*, recurrent neural networks (RNNs) are an effective model because of their ability to "remember" features of recent computations. Formally, a sequence of data is defined as a series of input vectors over a discrete set of time-steps, such that $\boldsymbol{x}_i$ is the input vector at time step i. RNNs can train very effectively on datasets even when the inputs lack a naturally sequential order (Karpathy, 2015).

At each time step, the recurrent model receives two inputs, the feature vector x_t as well as the output of the hidden layer from the previous time step s_t. During forward propagation, an RNN internally stores the output of each hidden layer and returns a distribution for the next time step, h_{t+1}. The most basic RNN architecture, known as the Elman network (Goldberg, 2015, p. 56), can be modeled as follows:

$$\boldsymbol{a}_i(\boldsymbol{x}) = \sigma(\theta^{(1)T}\boldsymbol{x}_i + \theta^{(2)T}\boldsymbol{y}(\boldsymbol{x}_{i-1}))$$
$$\boldsymbol{y}(\boldsymbol{x}_i) = g(\boldsymbol{a}_i(\boldsymbol{x}))$$

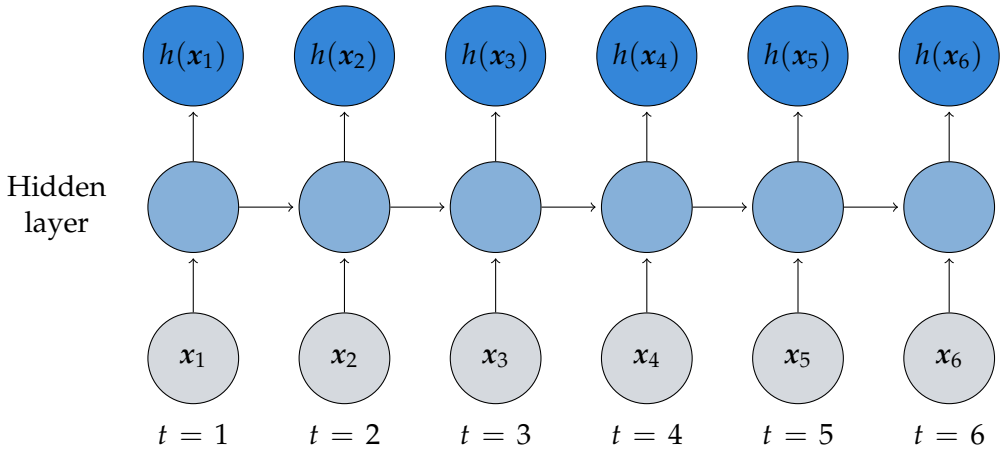
where

$$\theta^{(1)} \in \mathbb{R}^{|\boldsymbol{x}|\times h}, \theta^{(2)} \in \mathbb{R}^{h\times h}$$

and g is a non-linear transformation, such sigmoid (σ) or tanh. Note that the pa-

rameters θ can be updated in the middle of forward propagation over a sequence, or after the entire sequence has been fed through. Training an RNN follows the same procedure as the non-recurrent model - create a computation graph over time, calculate error for the most recent prediction, and then back-propagate the error across the unfolded network, generating gradients and updating each weight in θ (Goldberg, 2015, p. 63).

Figure 1.3: Abstraction of an RNN over a time series. The activation of the neurons in the hidden layer is a function from the previous layer (in this architecture, this is always the input layer) and the hidden layer output from the previous time step.



The recurrent model of incorporating feedback from previous decisions is a promising approach to the harmonization task, where the chorale can be represented as a sequence of melody notes with corresponding, interrelated harmonizations. Karpathy, 2015 famously cited the "unreasonable effectiveness" of recurrent neural networks in learning sequential data for a variety of tasks, including speech recognition, language translation, and image captioning. As a result, we hypothesized that recurrent models would perform particularly well on musical tasks that require contextualized decision-making and benefit from correlating temporally distant inputs.

Improving the recurrent model: Long Short-Term Memory networks

Long Short-Term Memory networks (LSTMs) are a variant of RNNs that replace the hidden layer neurons with specialized cells for enhanced memory capabilities, first introduced by Schmidhuber and Hochreiter in 1997. In the original RNN architecture, the feedback is "short-term" in that s_t is only a function of the input at time $t-1$ (and much more weakly for previous time steps). Therefore as the sequence is fed through the network, the signal from earlier time-steps is gradually lost. This phenomenon is known as the "vanishing gradients problem" (Goldberg, 2015, p. 56) and it constitutes a major drawback to using the original RNN architecture. LSTMs solve this long-term dependency issue by substituting the regular neuron with a *memory cell* that can retain much more distant signals from previous inputs. Information is stored and segregated within the cell by use of multiplicative gate units, such as input, forget, and output gates. These gates allow information to flow through the cell without affecting other memory contents, while also deciding what cell information should be kept versus overwritten by newer signals. A gate $\boldsymbol{g}$ is represented as a n-dimensional one-hot vector, and the values of $\boldsymbol{g}$ are considered parameters of the model. A value of 1 in a gate has the effect of retaining its corresponding signal element, while a value of 0 effectively eliminates that element. The deeper mathematical foundations for LSTMs are not necessary to understand, but they have proven to be exceptionally effective models, designed specifically for retaining information over long periods of time. When applied to music, LSTMs have proven effective at learning global musical structures and generating melodies (Eck and Schmidhuber, 2002).

1.2 An introduction to tonal harmony and the chorale

The primary objective of this thesis is to train a variety of models on the four-voice chorales composed by J.S. Bach in order to learn the task of chorale harmonization.

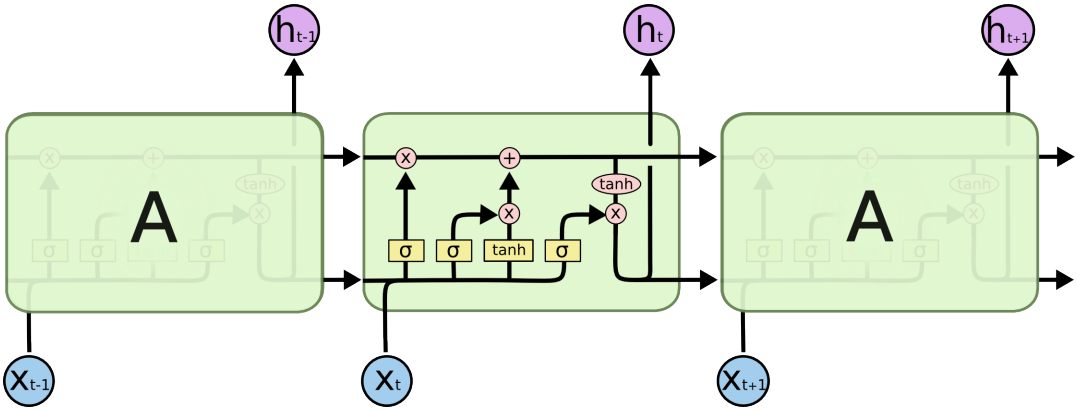


Figure 1.4: The chained structure of an LSTM memory cell. Each cell contains multiple network layers, where the RNN presented previously has only one (the sigmoid layer). Diagram created by Olah (2015).

However, each model is not given an actual musical score as input, but rather a numerical representation of "features" extracted from the score. Features are variables that describe some aspect of the data, and a thoughtful selection of features is one of the largest factors in having the model learn effectively. The features selected from the chorales include both score-wide features, such as the key signature, as well as local features that describe the melody at a specific moment in numerical terms, such as the beat strength of the melody note. In order to understand the meaning of the musical features selected from the chorales and their significance in learning the task of harmonization, a tailored overview of tonal harmony and the properties of the chorales is provided here. Musical properties are described as computational objects to illustrate the ability to translate between these properties and numerical representation.

Pitch class, pitch, note

The fundamental musical object in Western tonal music is the *pitch class*. The tonal system operates over a series of 12 *pitch classes*, and some pitch classes (i.e. C♯/D♭) are referred to by different names depending on the current context. A pitch class

is defined by its unique index - a integer between 1 and 12. It is common to assign the tonic pitch class as 1, the pitch class above it as 2, and so on.

C, C♯/D♭, D, D♯/E♭, E, F, F♯/G♭, G, G♯/A♭, A, A♯/B♭, B

A *pitch* is an object defined by a pitch class and an octave, identifying a unique frequency. A common representation of pitch is MIDI (Musical Instrument Digital Interface) notation, which today remains the most widely used protocol for communication of musical information between electronic instruments. In MIDI, a pitch is identified by a unique integer between 21 and 108 (inclusive). However, a disadvantage to MIDI is that it conflates enharmonic pitches (i.e. C♯5 has the same MIDI value as D♭5), so information about the function of a pitch within a key signature is lost with this representation.

We then define a *note* as a pitch with the additional feature of duration.

Scale, chord, key

The musical objects defined here are fundamental indicators of harmonic information. A *scale* is an ordered collection of pitch classes defined by an initial pitch class and a quality - major or minor - that defines the intervals between each note in the collection. Each pitch class in the scale is given an indexed scale degree and a name. The two most important elements in the scale are the "tonic" ($\hat{1}$) and "dominant" ($\hat{5}$). But more generally, a melody over a scale is guided by the characteristic tension or stability of the scale's pitch classes. The "tonic" represents the ultimate point of stability, whereas the leading tone ($\hat{7}$) creates a sense of motion towards the tonic. Therefore, pitch is a crucial feature in predicting the continuation of a melody or the harmonization of an existing melody.

The strong parallel between scales and chords can be defined as a *chord-scale*

duality. A *chord* is a collection of three or more notes sounded together. The pitches that comprise a chord imply a scale that contains those pitches; and similarly, a scale implies the set of *triads* that can be constructed starting from each note of the scale.

The *key* of a piece is a combination of two pieces of harmonic information - the tonic pitch class, and the chord (or scale) that represents full harmonic resolution. In major and minor keys, this chord is a *triad*, which in the key of C major is the pitch class set {C, E, G}. The triad alone is able to define the diatonic scale for the piece, which is represented symbolically as a key signature that specifies the pitch classes of the diatonic scale.

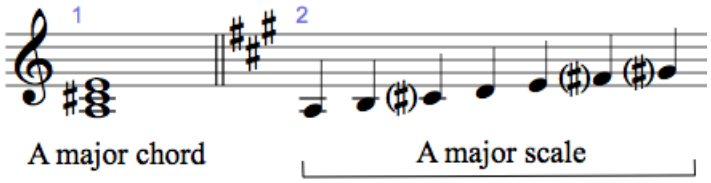


Figure 1.5: chord-scale duality.

Harmonic analysis and motion

When harmonizing a chorale, an important first step is to analyze the chorale melody and determine the chord progression that the harmonizing voices will imply. A standard approach for notating chord progressions in classical harmony is Roman numeral analysis. Roman numeral analysis describes a harmony by assigning a Roman numeral based on the chord root and a uppercase or lowercase version (i.e. II vs. ii) based on the chord quality. Superscripts are used to denote chord inversions, indicating which pitch class in the chord appears in the lowest voice (Laitz, 2012, pg. 68-9). For example, in the key of A major, the A major triad {A, C♯, E} is assigned I and an E dominant 7th chord {E, G♯, B, D} is assigned V^7 based on the E major triad, notated as V. Roman numeral analysis is powerful be-

cause it focuses on the harmonic information stored in triads. The motion between tonic and dominant triads alone can firmly establish a key, as well as a sense of tension and resolution that is generated by harmonic motion away from and towards the tonic harmony (ibid., pg. 106).

In performing Roman numeral analysis, context is a crucial factor because each harmony is analyzed with respect to the key of the chorale as well as the harmonies that preceded it. This is why harmonies are often described as a progression, or *sequence*. The ordering of harmonies - like the ordering of words in a sentence - is a essential feature of a harmonic progression, and an important feature to consider when constructing models to generate new progressions.

Cadences are the components of a harmonic progression that conclude a musical phrase or section. They are important harmonic indicators that lead to a point of a resolution or heightened tension, depending on the type of cadence, and as a result they control the flow of the music. Statistically, the ability of a model to accurately predict the location and quality of a cadence can serve a useful qualitative indicator of a model's performance because cadences are the most defining feature of a harmonic progression.

Bach's settings of the chorales

A *chorale* is a congregational hymn that first came into use during the early decades of the German Protestant Reformation, under Martin Luther. These hymns were originally composed as a one-voice melody with lyrical text, and composers of the time drew heavily (and sometimes borrowed verbatim) from existing secular songs, medieval Gregorian chant, and other sacred works when writing new chorales. In the Baroque era, the composer Johann Sebastian Bach revived the

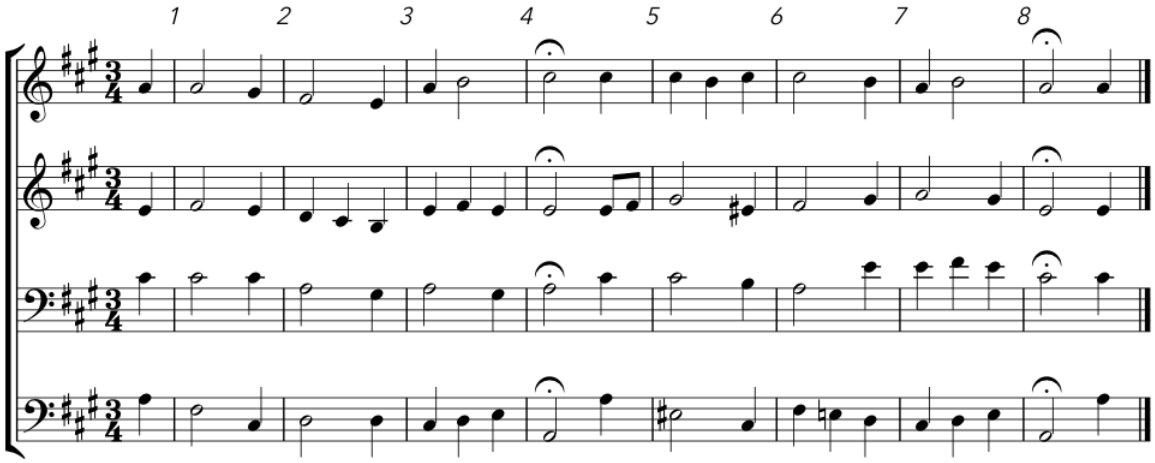
chorale tradition and composed several new chorale melodies. However, his most lasting contribution to the chorale form remains his harmonizations of hundreds of chorales, which were inserted into many of his larger vocal and instrumental compositions, including the St. Matthew Passion and the cantatas (Leaver and Marshall, 2015). His harmonization of four-voice chorales - of which 341 exist in the Riemenschneider collection - are masterful studies in four-voice counterpoint, and they remain a guide for modern church musicians, jazz writers, arrangers and students alike. This corpus established fundamental conventions for tonal harmony with respect to voice leading, cadential movement, and intervallic relationships between voices.

Structurally, the chorale is written for the four standard voice ranges: soprano, alto, tenor, and bass. The original chorale melody is sung by the soprano, while the lower voices collectively embody the *harmonization* of the melody. It closely resembles the chordal motion of the modern sacred hymn, and the rhythmic complexity of the chorales is intentionally minimal in order to draw focus to the harmonic motion taking place on each beat. The entire chorale is segmented into shorter phrases by a series of *fermatas* that indicate a pause and point of emphasis. The fermata denotes a point of cadence where a point of tension or resolution is firmly established.

The chorale should be viewed in multiple dimensions. In the vertical dimension, the notes of each voice at time t can be heard simultaneously as a chord; and therefore, the chorale can be abstracted to a four-voice chordal progression. Learning to recognize and generate choral progressions is the primary task of models of this thesis. The progression is guided by the cadences that structure each phrase, and each cadence is defined by the relationship between the melody and the bass line. The inner voices (alto and tenor) function as supportive voices that ”fill out” the harmonies. However, in the linear dimension, the chorale represents a combi-

nation of four independent but complimentary melodic lines, and the contour of each line is governed by the conventions of *voice leading*. Voice leading embodies a broad set of characteristics, but they include rules about preferred and undesired intervals, parallel and contrary motion, and methods for rhythmically embellishing a chord progression (i.e. passing tones). The conventions of voice leading the acceptable motion in each voice as the harmony shifts in the vertical dimension.

Figure 1.6: The opening of the chorale *Nun lob mein' Seel', den Herren*

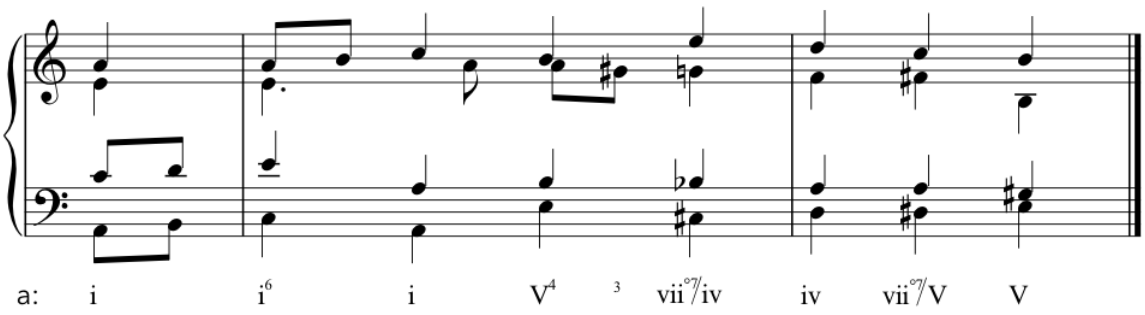


The opening section of the chorale *Nun lob mein' Seel', den Herren* (Figure 1.6) illustrates the importance of *context* in chorale harmonization. Bach's choice of harmonization for each beat is not an independent decision, but rather a choice guided by the harmonic destination of each phrase, which the composer almost certainly decided in advance of writing out the other voices. The harmony at time t constrains the set of harmonies that it can smoothly transition to at time $t + 1$, and therefore if we decide to assign a certain harmony at time t, the preceding harmonies must be chosen with the constraint that they provide a satisfying towards that harmony. Note, for example, the ii^6 - V - I cadences that Bach constructs in measures 3-4 and 7-8. Were we harmonizing this chorale ourselves and decided to assign a ii^6 harmony to beat 2 of measure 3, we would constrain which harmonies

we could convincingly transition to on beat 3. The fermata on the downbeat of measure adds the additional constraint that a cadence conclude there - like an authentic (I) or half (V) cadence. The harmonization of beat 3 must therefore bridge its surrounding harmonies to satisfy the expression ii^6 - * - {I, V}.

The $C\sharp^7$ harmony in measure 5 of *Nun lob mein' Seel', den Herren* further illustrates the need for careful planning when constructing a harmonic progression because the harmony is not to the key of A-major. Non-diatonic harmonies most commonly occur in areas of tonicization - when the harmony briefly establishes a new tonic - or when the harmony is functioning a secondary dominant. In this instance, the $C\sharp^7$ is functioning as a secondary dominant, since it acts a dominant harmony with respect to *F*♯-minor in measure 6. Secondary dominants are an effective tool for prolonging the resolution towards a certain harmony, and Bach uses them frequently to expand his harmonic palette. The chorale *Warum betrbst du dich, mein Herz* (Figure 1.7) also contains non-diatonic harmonies functioning as secondary dominants, which facilitate the chromatically ascending bass line in measures 2 and 3.

Figure 1.7: First phrase of *Warum betrbst du dich, mein Herz*



In summary, the process of harmonization requires consideration of the larger harmonic structure and how each chosen harmony will contribute to the harmonic progression of the phrase. Characteristics of the melody note - such as pitch, beat

strength, or the presence of a fermata - provide information about what harmonies might compliment it, while knowing the location of the next fermata or the preceding harmony inform how future harmonies should be chosen to create a satisfying progression.

Bibliography

Eck, Douglas and Jurgen Schmidhuber (2002). "Finding temporal structure in music: Blues improvisation with LSTM recurrent networks". In: *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*. IEEE, pp. 747–756.

Franklin, Judy A (2006). "Jazz melody generation using recurrent networks and reinforcement learning". In: *International Journal on Artificial Intelligence Tools* 15.04, pp. 623–650.

Goel, Kratarth, Raunaq Vohra, and JK Sahoo (2014). "Polyphonic Music Generation by Modeling Temporal Dependencies Using a RNN-DBN". In: *Artificial Neural Networks and Machine Learning–ICANN 2014*. Springer, pp. 217–224.

Goldberg, Yoav (2015). *A Primer on Neural Network Models for Natural Language Processing*. Tech. rep. Bar-Ilan University.

Greff, Klaus et al. (2015). "LSTM: A Search Space Odyssey". In: *arXiv preprint arXiv:1503.04069*. URL: http://adsabs.harvard.edu/abs/2015arXiv150304069G.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural Computation* 9.8. Ed. by MIT Press, pp. 1735–1780. URL: http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf.

Karpathy, Andrej (2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*. Blog. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Laitz, Steven Geoffrey (2012). *The complete musician: An integrated approach to tonal theory, analysis, and listening*. 3rd. Vol. 1. New York: Oxford University Press, USA.

Leaver, Robin A. and Robert L. Marshall (2015). "Chorale". In: *Grove Music Online. Oxford Music Online*. URL: http://www.oxfordmusiconline.com.ezp-prod1.hul.harvard.edu/subscriber/article/grove/music/05652.

Liu, I-Ting and Bhiksha Ramakrishnan (2014). "Bach in 2014: Music Composition with Recurrent Neural Network". In: eprint: 1412.3191. URL: http://arxiv.org/abs/1412.3191.

Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*. Cambridge, Massachusetts: MIT press.

Olah, Christopher (2015). *Understanding LSTM Networks*. Blog. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Rohrmeier, Martin and Ian Cross (2008). "Statistical properties of tonal harmony in Bach's chorales". In: *Proceedings of the 10th international conference on music perception and cognition*. Hokkaido University Sapporo, Japan, pp. 619–627.