Noise (Music) Composition Using Classification Algorithms

Peter Wang (pwang01)

December 15, 2017

Background

Abstract

I attempted a solution at using machine learning to compose music given a large corpus of human-generated music. I made an assumption that the next note or group of notes in a work of music was dependent solely on the past k seconds of music. For the most part I framed this as a multi-class classification problem, where the preceding music would be used as features to predict the "label" which was the next note.

Introduction

As with most things in life, I am terrible at producing music. Many people are much the same. In fact, only a select group of people can produce music enjoyable to listen to, and even then, they have as many critics as fans. Machine learning offers a remedy: have a computer generate large quantities of cheap, yet original, music. Obviously, this will appease no connoisseur of music, but it's far better than playing the same tired songs repeatedly to stimulate our audio inputs. Music is merely sound, and sound is merely a time-varying signal. However, signal processing is expensive and difficult, and humans do not generate the raw audio signals, instead generating sequences of notes. This makes it easy to obtain a dataset for training a model (or several models). I experimented with several models, including linear regression, an SVM, two iterations of a standard multilayer perceptron, an LSTM model, and two iterations of a GAN. The inputs to the linear regression, SVM, both multilayer perceptrons, and one of the GANs were the 400 playing notes (given in MIDI number, which is a linear function of the logarithm of the fundamental frequency) from the past 100 samples (sampled at 10 samples / second). The goal for these models, excepting the GAN, was to predict any of the maximum of four notes from the actual music samples. The other models were tasked with generating full collections of notes to be heard by the user.

Related work

Research on this problem has explored several different techniques. Without using deep learning, some researchers have incorporated the Markov assumption (that I have also made) into their model by using a hidden Markov model. (Van Der Merwe and Schulze 2011) A hidden Markov model is similar to a normal Markov chain, except that the transition probabilities are not directly observed and must be indirectly observed after being passed through a confusion matrix. In this case, the researchers used several HMMs to generate various aspects of the music, such as the harmony, melody, and rhythm. Indeed, in constructing the model, there is a lot of human intervention and domain-specific knowledge applied. However, the results were quite impressive – out of 263 participants, around 38% misidentified the music piece they were given. However, there were quite a few restrictions on the human composers so as not to give the answer away (such as notes not carrying between measures, a consistent time signature, and no arpeggiation). Douglas Eck uses LSTMs to generate blues music; the input data is a series of binary vectors with "1" representing a note as being played. However, it's worthy to note that he leaves it up to the network to discover that harmony is desirable. (Eck and Schmidhuber 2002) Additionally, this network makes no distinction between holding a note and playing it, something that is instrumental (no pun intended) in today's music. Additionally, he correctly points out that LSTMs have difficulty composing music with global structure, something that I decided not to tackle. Daniel Johnson attempts to correct for the lack of local structure assumptions (large ones being that music is transposable and that harmonies sound good together) (Johnson 2015). He uses a convolution layer along with an array of connected RNNs as his network architecture. (To be honest, this is my first foray into deep learning, so I have little knowledge here.) However, his elaborate methods produce fairly good results; the music sounds fairly respectable; however given the number of features other than just the previous notes and the complexity of the neural network architecture, this is not surprising.

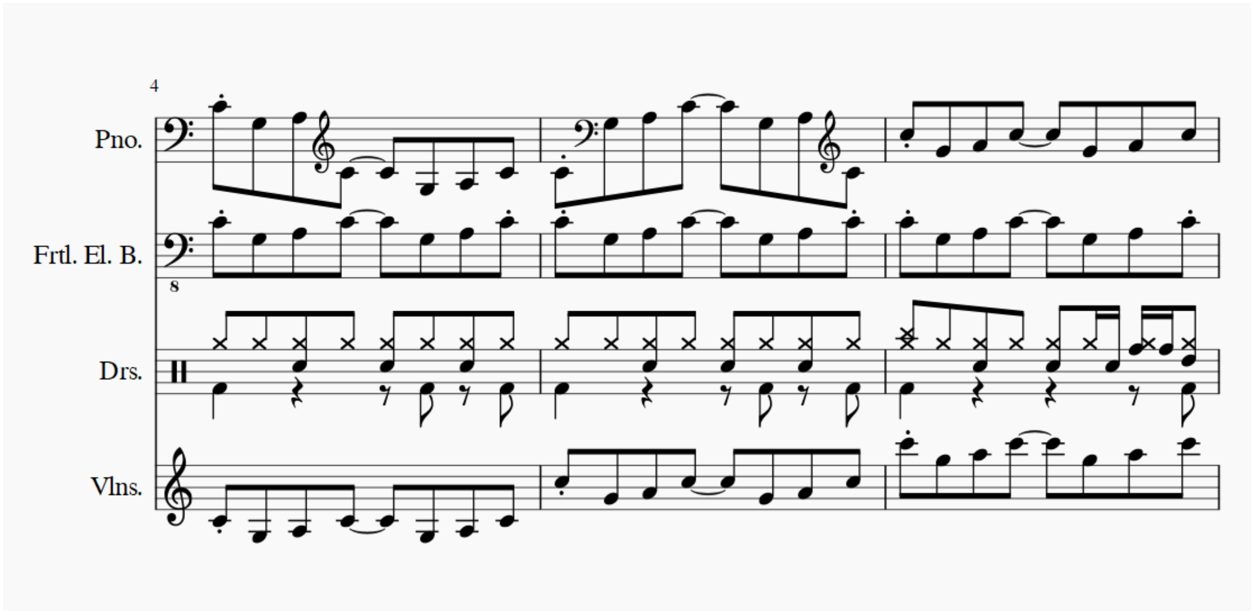
Another interesting concept is that of a "word embedding". If one thinks of music as a series of chords and a melody, then it's easy to see how it is like writing sentences. There are subtle interconnections between words in prose and poetry, just as there are in musical notes; often multiple notes will be used to represent the same concept. In this vein, somebody at Machine Learning Mastery ("How to Develop a Word Embedding Model for Predicting Movie Review Sentiment" 2017) attempted to use this before an LSTM to predict movie review sentiment. This helps by decreasing the dimensionality of the input data, increasing training performance as well as making optimization more effective.

Finally, an interesting development is the use of "chaotic inspiration" to help an LSTM avoid overfitting and also make it produce more appealing music (Coca, Corrêa, and Zhao 2013). While one set of inputs to the LSTM are melodies to train it, there is another "chaotic" inspiration input that harmonizes with the melodies and helps give depth to the output music. However, this model again requires a lot of human intervention, as the various structures in music are given their own models and later put together.

Experimentation

Dataset and Features

My dataset consisted of over 40000 MIDI files from the Lakh MIDI Dataset (http://colinraffel.com/projects/lmd/). However, I was unable to use the vast majority of these files due to computing and memory constraints. I applied a sliding window of 101 samples over each of the files, taking the first 100 samples (each consisting of a maximum of four allowed simultaneous notes = 400 features) to be the features, and the last sample to be the expected outputs of my models. For my linear regression, SVM, and one of my multilayer perceptron models, I had them predict only one of the four output notes – if they got any of the output notes, it was considered a correct answer, and the maximum score was awarded. Even with only about 150 MIDI files, I had over 300,000 examples to contend with, each consisting of a massive feature vector (which sometimes got converted into binary vectors, straining the memory of my system). I processed this dataset using a converter I wrote into a $34168 \times 101 \times 128$ numpy array. For my dev set, I used just 5 MIDI files, and the same was used for my testing set. While these may seem small compared to the training set, I made this choice for ease of development, since I quickly realized that my models performed abysmally poorly on both of these sets (and therefore probably wouldn't get to an acceptable level of performance no matter how big I made the test set). As my dataset is audio, it's hard to visualize it. Nevertheless, here is a picture of the sheet music of one of the MIDIs.



Methods

Linear Regression

I first used linear regression on the features to provide a baseline. I rounded the outputs and used those as the output notes of the model. This was the only model to get precisely 0% accuracy on both the test and training sets. This led me to conclude that the problem as I posed it is definitely a classification problem and that it is highly non-linear.

Support Vector Machine

Because of the extreme underfitting of the linear model, I opted for a supervised learning model that I knew could fit and overfit the data. I used an RBF kernel $K(x, x') = \exp(\frac{|x-x'|^2}{2\sigma^2})$ since it appeared popular (I'm not too well-versed in this). This fared slightly (or infinitely) better than the linear regression, coming in at 1.4% test accuracy. Of course, this approach probably suffered from the One-vs-One approach to the translation of the multiclass classification problem into binary classification.

Multilayer Perceptron 1

After realizing that the standard machine learning tools had completely failed, I decided to try to use the magical deep learning that I had no experience in (that pretty much explains the low quality of the results in this project). My neural network consisted of an input layer with 400 neurons, one for each feature, and then 3 and later 5 hidden layers of 2048 neurons with ReLU activation ($\max(x, 0)$), and finally an output layer of 4 neurons with ReLU activation. Unfortunately, this original neural network did even worse than the Linear Regression in terms of mean squared error (probably because the optimization hit a local minimum). Of course, it also got a 0% test accuracy. However, I changed the output to 128 neurons with the softmax activation function and considered these outputs as probabilities that a particular note was one of the four in the label. This fared slightly better, with a 5.1% test accuracy. However, this neural network was a complete failure, as it simply repeated four numbers no matter its input (the four numbers actually varied depending on how the network was trained, but they were the same regardless of the input neurons).

Multilayer Perceptron 2

After the failure of my initial network, I attempted to feed in the notes to the network more intelligently. By feeding in vectors of MIDI numbers, I was inadvertently confusing the neural network, as I used 0 values to represent silence. However if a note went off before another note, then the remaining note's position would shift, confusing the network and leading to a regression to the mean. I changed the input to the network to 12800 neurons, conceptually grouped as 100 samples of 128 binary neurons that represented depressed notes. The output of the network was 128 neurons. I shifted the problem from a multiclass single-label problem to a multiclass multilabel problem. This network could output arbitrary numbers of depressed notes (up to the theoretical maximum of 128). The hidden layer structure remained the same, and the output activation function was changed to the sigmoid function to allow multiple outputs. Unfortunately, this network fared poorly, as multilayer perceptrons are not good at classifying sequences of data, since the old data dies rather rapidly. I used a binary cross-entropy loss here, as this has worked well on binary outputs. This achieved an accuracy of a 7.7%, a modest improvement over the previous network.

Generative Adversarial Network 1

I decided to augment my second multilayer perceptron by turning it into the generator for a GAN. I changed the inputs so they would take in 32 numbers distributed about the standard normal distribution, and the output to be 12928 neurons (128 neurons per time slice with 101 time slices). My discriminator was similar; its input was 12928 numbers and the output was a single sigmoid-activated neuron that determined whether the generator output was "fake" or "real". Unfortunately, the discriminator performed with almost perfect accuracy (99.4%), while the generator was completely lost, again outputting the same vector repeatedly without any hope of changing.

LSTM

After the failures of standard multilayer perceptrons, I caved in and sought the LSTM for help, an architecture that has been proven to work in other research. The input to my LSTM is a time series of 128-dimensional binary vectors that describe the notes being held in a particular time slice. The output is another time series of 128-dimensional binary vectors. Accuracy numbers are reported in the "96.5%" range, but this includes the copious numbers of zeros that are in the output. This method, so far, has been the most successful, yielding the only valid, playable music (although it sounds like a three-year-old banging on the piano). Sample audio files, as well as the zipped dataset are available with the code.

LSTM at a loss

I realized that perhaps the binary cross-entropy loss was not the best loss function to describe music. So I decided to write my own loss function. Assume without loss of generality that the output y_p of the LSTM always has a higher frequency than the true value y_t (I just take the maximum in my code). Let T_R be a vector describing the frequency ratio of the two vectors. I defined my loss function to be $L = |2 \times T_R - \text{round}(2 \times T_R)|^2 + 0.1 \times |2 \times T_R|$. The first term allows notes to easily be whole-number multiples of others' frequencies. This is because multiples of the fundanmental frequency sound consonant to human ears, while those in the middle sound very badly dissonant. The second term ensures that the notes don't drift too far in the octave range. However, even with this improvement, the LSTM fared little better, sounding now like two three-year-olds banging on two pianos.

```python
def musical_loss(y_true, y_pred):
    desired_sz = K.tf.cast(K.max(K.maximum(K.tf.count_nonzero(y_true, 2), K.tf.count_nonzero(y_pred, 2)
    y_true_d, _ = multihot_tensor_to_normal(y_true, desired_sz)
    y_pred_d, confidence = multihot_tensor_to_normal(y_pred, desired_sz)
    true_frequency = K.clip(K.exp(((y_true_d+1)*64 - 69.0)/12.0 * LOG_2) * 440, 10, 15000)
    pred_frequency = K.clip(K.exp(((y_pred_d+1)*64 - 69.0)/12.0 * LOG_2) * 440, 10, 15000)
    tensor_ratio = K.maximum(true_frequency, pred_frequency) / K.minimum(true_frequency, pred_frequency

    # Compute difference between tone and harmonic of the true value
    tensor_difference = (2*tensor_ratio - round_tensor(2*tensor_ratio))/2

    # weight tensor_difference with the respective outputs so unconfident (=0.5) outputs mean little lo
    # weight K.log(tensor_ratio) the same way

    return K.sum(K.square(tensor_difference)) + 0.1 * K.sqrt(K.sum(K.square(K.log(tensor_ratio))))
```

GAN Take 2

I then tried to apply my LSTM to a GAN. Unfortunately, this took forever to train and yielded NaN values for all predictions.

Hyperparameter Choice

I'm not well-versed in deep learning so I cannot say much about my hyperparameter choice. I have read about the no free lunch theorem (https://en.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization) which ensures that optimizers will always have their strengths and weaknesses. A good heuristic is to with the default options on Keras or what's popular. For that reason, I chose RMSProp and Adam as my optimizers. The number of neurons was determined by the input shape, the output shape, and the maximum number that would allow a reasonable training time. Clearly, I have not overfit my training set as all of my models have currently failed to predict the training set to a reasonable degree.

Future Work

In short, the models that I have tried have all done rather poorly. However, the LSTM shows promise. Unfortunately, I was unable to train it for a large number of epochs and I have also formulated the problem rather poorly. I believe that by incorporating some of the ideas in other research papers (of human intervention being used to create the general structure of the process of music composition, then letting the machines decide on the actual values) and using several LSTMs, each for a different part of music composition, I can do far better than the ad-hoc approach of sampling and then asking a single neural network to predict the output over a relatively arbitrary timescale. Additionally, with the exception of the crude loss function I defined, I did not consider the invariance of transposition in my model, which inevitably led to difficulties generalizing.

In conclusion, the formulation of music generation as just simply a multiclass, multilabel classification problem is rather poor for current algorithms to tackle. While a neural network may be a universal function approximator, the curernt formulation was essentially trying to predict an almost completely random variable (creative license ensures that even humans can do no better than perhaps 25% for the next note) and fared poorly. Adding in transposition invariance and more sophisticated music generation will probably make the networks less confused and more focused on more tractable problems.

References

Coca, Andres E, Débora C Corrêa, and Liang Zhao. 2013. "Computer-Aided Music Composition with Lstm Neural Network and Chaotic Inspiration." In *Neural Networks (Ijcnn), the 2013 International Joint*

Conference on, 1–7. IEEE.

Eck, Douglas, and Juergen Schmidhuber. 2002. "A First Look at Music Composition Using Lstm Recurrent Neural Networks." *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale* 103.

"How to Develop a Word Embedding Model for Predicting Movie Review Sentiment." 2017. *Machine Learning Mastery*. https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/.

Johnson, Daniel. 2015. "Composing Music with Recurrent Neural Networks." *Hexahedria*. Daniel Johnson. http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/.

Van Der Merwe, Andries, and Walter Schulze. 2011. "Music Generation with Markov Models." *IEEE MultiMedia* 18 (3). IEEE: 78–85.