MUSIC AND MACHINE LEARNING

A Study of Data Representation in Piano Music

Andrew Leathwick

Supervised by Mark Gahegan

A report for COMPSCI 380, Undergraduate Project in Computer Science

The University of Auckland

Introduction

In George Orwell's famous dystopian novel, *1984*, the totalitarian ruling body creates a language, 'Newspeak', designed to constrain expression within politically acceptable bounds. Syme, a co-worker of Winston, says to him, "Don't you see that the whole aim of Newspeak is to narrow the range of thought? In the end we shall make thought-crime literally impossible, because there will be no words in which to express it."

Orwell's point was that there is a relationship between the grammatical structures and vocabulary of a language, and the ease with which certain concepts can be expressed; perhaps by controlling language, one can even control thought. This idea is relevant to the field of deep learning. In particular, when considering the application of deep learning to various problems, we must recognize that in choosing the language of the data, we constrain the ease with which we can express different concepts. The language, here, is the choice of how to represent the data.

Given a task consisting of input information and a target (in a supervised context), there are two basic questions the deep learning practitioner must address: first, what should the architecture look like, and second, how should the inputs and outputs be represented? It is the latter question that this project is devoted to, which I explore specifically in the context of classical piano music.

This project has three main objectives:

1. Compare two different methods of representing piano music for deep learning, and their effects on a model's ability to find signal in the data;
2. Train a model that successfully generates new piano note sequences, complete with expressive timing and velocity information;
3. Experiment with some simple feature engineering, and evaluate the effects of adding these features to some models.

This project report is structured as follows.

1. Discuss what information is needed to encode a piano performance, and why data from piano performances is so suited to machine learning;
2. Introduce midi and the MAESTRO dataset;
3. Introduce two encodings for piano music from the literature;
4. Discuss my results from training models using these two encodings;
5. Examine the effects of adding simple derived features to one of these encodings.

Code and models may be found here: https://github.com/aleathwick/380-music-representation

When a trained model is identified by name (mainly in plot titles), in most cases the training history, weights, and model description are available in the GitHub repository.

Realized versions of generated sequences referred to by name in this report can be found here: https://www.youtube.com/playlist?list=PLCO5IgjyszQvCVXG4f_JiPaQwcCvoQCpN

1) Pianists, the piano, musical expression, and machine learning

What information is needed to store a piano performance, and why is piano music particularly suitable for deep learning?

The piano's mechanism of sound production is not complex. In exchange for having such immediate and free access to the piano's notes, a cost is paid in the pianist's ability to influence timbre. Once a note has started, a pianist can do almost nothing to change its sound, except to decide when it ends. This makes the piano perfect for machine learning, because all of the mechanisms a pianist uses to produce and control sound can be easily and precisely measured. In this section I spend some time examining these mechanisms.

When controlling the sound of the piano, the pianist may play notes, in which case the only determining factor of sound is the speed at which the hammers hit the string (which is controlled by how fast the keys are depressed). Additionally, the pianist may depress pedals with the feet, most importantly the sustain pedal, whose job it is to prolong notes and add resonance by unmuting all the strings of the piano. The unmuted strings then sympathetically vibrate with those that are struck by hammers. It is a trivial task to measure the speed of the hammers for each note, the positions of the pedals (how far they are depressed), and the duration for which each key is held down; capturing this information enables accurate reproduction of a piano performance, and the storage requirements for such information are small.

Because the speed of the hammer is all the pianist has to work with (aside from the pedals), timbre (the tone quality of a note) and volume have an unusually close relationship; they do not vary independently of one another, and measuring the velocity of the hammer is enough to determine both. By contrast, in most other instruments this is not so. For example, to accurately reproduce the timing and tone colours of the performance of a violinist, one would need to capture the position of the bow relative to the bridge, the pressure of the bow on the string, the angle of the bow relative to the string around its vertical and horizontal axes, the speed of the bow relative to the string, and the position of the fingers on the fingerboard; furthermore, all of this would need to be captured continuously, i.e. not just one measurement per note, but a continuous stream of information.

As a consequence, while software abounds for digitally recreating the audio of a piano performance from information about hammer speed and pedal positions, the same is not true for solo violin, because there is no way of succinctly representing performances such that they can be reproduced. Indeed, the piano is unique in having a long history of 'player pianos' which rely on this captured information to perform pieces without the need for a pianist. In the 19th and 20th centuries, this information was stored in paper 'piano rolls', encoded in punched holes. This has since been replaced by digital files, and the aforementioned computer programs that, given this information, can recreate the audio of a piano performance with a level of realism indistinguishable from the audio of a live performance.

It is not obvious to many pianists that the piano's mechanism of sound production is so simple. Many pianists are convinced that a single note on the piano is capable of a world of colour and expression, even when it is played with no variation in volume. I dedicate a paragraph to rebutting their ideas, and in so doing, deal exclusively with the fingers, forgetting the pedals. What is commonly asserted is this: that it is possible to change the tone colour of a note without any change in volume, and that this is achieved by changing the weight or relaxedness of the hand and arm. This would render hammer speed an inadequate measure of the tone and volume of a note. One who inspects the mechanism of the piano with a more objective eye will see that hammer speed really is sufficient; the key is depressed by the finger, accelerating a hammer towards the string. Before this hammer reaches the string, and before the key has been depressed to its maximum depth, the hammer is released from its mechanical connection with the key, thus entering free flight. During this time the finger is not capable of accelerating or otherwise influencing the flight of the hammer. The hammer strikes the string, and the sound produced by the string is entirely determined by the speed at which the hammer strikes. Ah, but the hammer can be accelerated to the same speed using different portions of the travel of the key, says the pianist. Irrelevant, says physics; hammers have momentum, acceleration does not. Arm weight and other physiological factors are relevant to the sound produced only in how they affect the final velocity of the hammer at the point at which the mechanical connection to the finger is lost. If a sound is 'harsh', it is because the speed of the hammer was too great. The connection between volume and tone colour is total, and measuring hammer speed is enough to capture both these attributes of a note.

That piano performances should be so simple to capture is a fantastic attribute of the piano. And to think that such simplicity is found in an instrument so expressive that many of its practitioners refuse to believe it could be so simple! It makes piano the perfect instrument for investigating the application of machine learning to human musical expression, and as a result, much of the work to date has indeed focused on piano music. The few attributes of each note make the data amenable to use as inputs or outputs of a model; the mountain of continuous data that would come from a violin performance would require far more wrangling and thought, not to mention computational power and memory. The work with piano data to date has included the generation of performance from score, and the generation of performance *and* score (i.e. composing and performing simultaneously).

2) MIDI and the Maestro dataset

An introduction to MIDI and the MAESTRO dataset

The most ubiquitous format used for sending and receiving musical notes and messages in digital form is described by the MIDI technical standard (*The Complete MIDI 1.0 Detailed Specification*, n.d.). In this format, the pitch and velocity of a note are each expressed using 7 bits, giving 128 possible pitches (with 0 being the lowest and 127 the highest), and 128 possible velocities (with 0 being the softest and 127 the loudest). When a note is played, a MIDI message containing this information indicates that a note of a particular pitch has been activated at a certain velocity (which indicates intensity); no duration information is sent yet. When that note is released, a separate midi message is sent later, a 'note-off' event, to indicate that a note of that pitch has been released. The separation of note-on and note-off events allows midi to work in live situations. Other messages, such as those regarding sustain pedal, are similarly sent as 7-bit messages, giving the sustain pedal

128 gradations of depth. Although MIDI breaks up values of a continuous nature, such as velocity and pedal, into 128 discrete bins, in practice this is more than enough bins for the discretization to be unnoticeable.

One of the largest available collections of MIDI files of piano performances is the MAESTRO dataset. A team from Google partnered with the International Piano-e-Competition, a competition in which the pianos used are fitted with a system for recording and playback of MIDI. The dataset contains audio and MIDI recordings of performances, with audio and MIDI aligned with precision of around 3ms. The dataset contains data across many years of the competition, encompassing over 1000 performances, with a total of 6.18 million notes (Hawthorne et al., 2019)[1]. This dataset has been used for tasks such as training models to transcribe a piano score from the audio of a performance (Hawthorne et al., 2019), generate piano audio from the MIDI of a performance (Hawthorne et al., 2019), and generate sequences of piano notes complete with human-like expression (Oore et al., 2018) (Huang et al., 2018)[2].

3) Representations for Machine Learning

Two musical encodings from the deep learning literature

When seeking to use MIDI data from piano performances in deep learning problems, careful consideration needs to be given to the issue of representation. The temporal nature of music data alone has implications for how we may wish to represent notes; other aspects that may come into consideration are the well-established harmonic relationships between notes, and the choice of how to incorporate sustain pedal. Any model that uses information from a live performance will also need a representation that can represent time on a much finer scale than what is present in the unperformed score version of a piece; representing a *performance* is a very different task to representing a *score*. Scores are mostly comprised of notes arranged on a grid divided recursively in halves (hence the American naming convention for the different kinds of notes: whole note, half note, quarter note, eighth note etc.). Grid like representations, in which the presence or absence of a note are indicated at every step of the grid, are therefore commonly used with tasks in which the score needs to be represented[3]. But for a live performance, expressive timing, including speed ups, slow downs, and the expressive delay of notes, results in the need to represent timing at a millisecond scale.

Performance Representation

One of the primary objectives of this project is to experiment with and compare two representations of piano music used for the purpose of note sequence generation. The first of these representations emerged in one of the first papers examining the issue of generating both score and performance

[1] This project used version 2.0.0 of the MAESTRO dataset.

[2] The papers by Oore et al. and Huang et al. were released before the MAESTRO dataset, at which time this data was referred to as the Piano-e-Competition dataset.

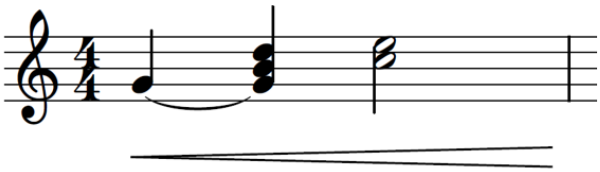
[3] For an example of this sort of representation, see the paper *Learning to Groove with Inverse Sequence Transformations*, in which the authors used a matrix to represent drumbeats. Each row of the matrix corresponds to a sixteenth note slice of a bar of music, and each of the nine columns corresponds to nine different drums, indicating the presence or absence of a hit (Gillick et al., 2019).

simultaneously. Until recently, most work in music generation has focused on generating musical scores, rather than musical performances (Oore et al., 2017). That is, the data used has not represented timing and velocity on the fine scale needed to capture the human expressive elements that are part of a performance of a work; instead the focus has been on producing a piece that might later be performed by a human. Oore and colleagues from Google Brain and DeepMind, in their 2017 paper *Learning to Create Piano Performances*, devised an event-based encoding scheme for piano performances, and trained a three layer long short-term memory (LSTM)[4] model to generate novel musical sequences that "…while lacking high-level structure … sound very much like human performance" (Oore et al., 2017).

Oore and colleagues represented sequences of notes using events that operate at a sub note level; that is, notes and their attributes are determined by multiple such events. There were 333 different events, and their model generated a discrete probability distribution across these. The events were as follows:

- 88 note-on events, each corresponding to the start of a note, for 88 different pitches;
- 88 note-off events, each corresponding to the end of a note, for 88 different pitches;
- 125 time shift events, ranging from 8ms to 1000ms in increments of 8ms, indicating a shift forward in time;
- 32 velocity change events, for 32 gradations of velocity from soft to loud; a velocity change event determines the velocity of every note that follows it, until the next velocity change event occurs.

The number of gradations of velocity, and the size of the smallest time shift event (8ms) were chosen so that the discretization of time and velocity would not be perceptible. We will refer to this representation as 'Performance Representation'.



[305, 46, 298, 312, 50, 53, 298, 134, 138, 141, 319, 51, 55, 300, 296, 139, 143]

*Figure 1: An example of a musical fragment encoded using Performance Representation, with colours indicating **note-ons**, **note-offs**, **velocity changes**, and **time shifts**.*

[4] Long short-term memory is a technique in which the hidden state of a layer at a given time step is a combination of 1. the hidden state from the previous time step and 2. the activation in response to the input at the current time step. How much of the old hidden state to carry across to the current time step is regulated by a 'forget gate', which controls the flow of information in response to the current input and the hidden state from the previous time step. Thus, an 'LSTM cell' can store information for use later in a sequence. A forget gate has weights that must be learned in the course of training, as do an output and an update gate – for full technical details, see the original paper by Hochreiter and Schmidhuber (1997).

In a later paper by (mostly) the same authors elaborating upon their experiments, the authors explained that they dealt with the sustain pedal in one of two ways: firstly, they tried ignoring it entirely, and secondly, they lengthened notes to emulate the effect of the pedal, taking any notes still sounding when the sustain pedal was depressed, and delaying their corresponding note off events until the release of the sustain pedal (Oore et al., 2018). The latter method achieves a sustain pedal like effect, but doesn't perfectly replicate the effect of the sustain pedal; in addition to extending notes as if they were held down for longer, the sustain pedal adds resonance and affects a note's ability to continue sounding. However, the difference is small enough that a non-pianist would not notice the difference unless they had it pointed out to them (and even then might still be unable to discern the difference).

This sort of representation, in which each of the 333 bins is treated as its own separate category, requires the model to learn nearly equivalent musical relationships independently of one another. There are 32 bins for velocity; if a model predicts that the 17th bin should be next in a sequence, but the ground truth has the 18th velocity bin next, then according to a standard loss function like categorical cross entropy, the only important fact is that the model predicted the wrong category; that it predicted a bin that would be highly correlated with the correct bin is irrelevant. We should be perfectly happy for the model to predict the 17th velocity bin when it should predict the 18th, because this is nearly equivalent, and the difference would likely be imperceptible to the listener. These relationships are captured if velocity is expressed as a continuous value but are lost when velocities are expressed as separate unrelated events. Even if we think of bins having an order, the model is agnostic to this. The same can be said for time; the relationship between bins 10 and 20 (80ms and 160ms, respectively) is similar in some sense to that between bins 11 and 22 (88ms and 172ms, respectively), but the model must learn these related relationships independently of one another, with no hints from the representation of their equivalency.

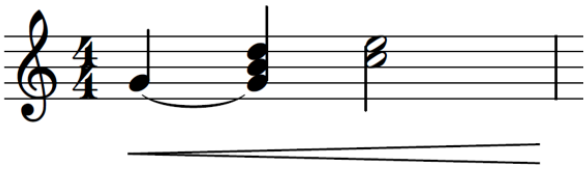
Similar is true of harmony, but perhaps to a greater degree, due to the many complex relationships across many levels of abstraction that could be hinted at by a different representation chosen to accentuate these relationships; the most fundamental of these relationships is what results in only 12 classes of pitch, from A to G#, so that multiple notes carry the same name. Performance Representation hints at none of these relationships. Neither does the next representation we examine, NoteTuple, that similarly discretizes time, velocity, and pitch attributes.

NoteTuple Representation

In 2018, Performance Representation was used again by a group from Google Brain, this time in conjunction with a transformer architecture. The improvement in the samples produced was immense; the original three layer LSTM model had difficulty capturing long term structure, but the relative self-attention mechanism of the transformer was able to remedy this issue, making possible minute long sequences with "coherent structure" and elaboration upon musical motifs (Huang et al., 2018). Later that year, a group from Google Brain mostly comprised of the same people investigated a different representation, also in conjunction with a transformer model. Instead of splitting up notes into sequences of events that together define and modify characteristics of notes, the modified representation grouped all the characteristics of a note into a tuple of attributes that are predicted together as a single event. They named this representation 'NoteTuple' (Hawthorne et al.,

2018). They found that using this representation reduced the number of parameters needed in the model and reduced the time it took to generate sequences.

Each note of a NoteTuple has four attributes: pitch, time shift since the previous note, duration, and velocity. Like Performance Representation, each of these attributes is divided up into discrete bins. Pitch and velocity are divided up into 88 and 32 bins, respectively. Using Performance Representation, a note could last for an arbitrarily long amount of time, as could silence; although the largest time shift event was 1000ms, time shifts longer that this could be represented using multiple time shift events. However, using NoteTuple, there is one time shift and one duration per note, so that if the largest time shift was 1000ms, then sequences with gaps between notes of more than 1000ms could not be represented. The question that naturally arises from this is: how do we represent sufficiently long periods of time, at fine enough resolution, without needing an excessively large number of bins? To represent durations and shifts up to 10 seconds long at 8ms resolution, each would need 1250 bins! To resolve this problem, the authors divided time shifts into two sub-attributes. One had a set of values incrementing in large time steps, and the other incremented in small time steps. They did the same for duration. Specifically, "time shift has 13 major ticks and 77 minor ticks, representing 0 through 10 seconds. Duration has 25 major tick values and 40 minor ticks" (Hawthorne et al., 2018).



[[46, 0, 0, 2, 19, 4], [50, 1, 22, 2, 19, 11], [53, 0, 0, 2, 19, 11], [51, 1, 22, 4, 38, 18], [55, 0, 0, 4, 38, 18]]

*Figure 2: An example of a musical fragment encoded using NoteTuple, with colours indicating **pitch**, **velocity**, **time shift**, and **duration**.*

NoteTuple is a more constrained language than Performance Representation. Using the latter, it is possible to generate sequences that are meaningless. For example, if two velocity change events have no note-on events between them, the first is rendered redundant; similarly, a note-off event is meaningless if there is no note of that pitch currently sounding. A string of velocity changes, note-off events, and time shift events would have no musical meaning whatsoever. Using NoteTuple, it is only possible to represent musically possible sequences; even an untrained model will at least generate coherent sequences of notes. Could this difference contribute to the ease with which a model using NoteTuple learns to generate sequences? One could argue it has a head start, because it is using a language in which spelling errors are impossible; it doesn't have to learn how to spell, so to speak. But this constraint comes at a cost: because of the hard upper limit on time shift and duration values, there are sequences of notes that are impossible to represent.

The authors found that NoteTuple sequences were about a third or a quarter the length of the equivalent Performance Representation sequence; this is due to the combining of attributes into a

single timestep. When working with transformer models, this results in large savings of memory (Hawthorne et al., 2018). The biggest drawback of NoteTuple noted by the authors is that modelling note lengths is more difficult. In piano music, it is very common for a large number of notes to be released simultaneously, either by the hands releasing multiple notes at once, or by the foot releasing the sustain pedal. The latter is pervasive in piano music; sustain pedal is integral to the piano, constantly lengthening large numbers of notes, and then releasing them simultaneously when it is released. Using Performance Representation, a large number of notes released together can be represented by a large number of contiguous note off events. But using NoteTuple, in which each note has a duration attribute instead of a corresponding note-off event later in the sequence, releasing multiple notes at the same time is much more difficult; if they have different start times, then later starting notes must be correspondingly shorter, and this arithmetic is not trivial for a model to learn. Their model using NoteTuple therefore struggled with overlapping notes causing unsavoury dissonances (Hawthorne et al., 2018).

4) Building Models

Building models and generating sequences using NoteTuple and Performance Representation

Building a simple model using NoteTuple

The second aim of this project was to build a model that could generate musical sequences with human characteristics, thus repeating the work originally carried out by Oore and colleagues in 2017 that showed a musical score and its performance could be generated simultaneously by training the model on data from piano performances. For this task, I used the same model architecture described in the original work, namely, a three-layer LSTM recurrent model with hidden state size of 512, trained with teacher forcing. However, instead of using Performance Representation, I used the NoteTuple representation from later experiments at Google Brain.

I used the full MAESTRO dataset, dividing each MIDI file into segments 128 notes in length. This choice of sequence length was arbitrary; later I would deal with sequence length in a more sensible fashion. These sequences were then converted into NoteTuple representation. For all of my experiments, I dealt with the sustain pedal by extending the lengths of notes and removing sustain pedal. I disregarded any data from the remaining two pedals; the soft pedal has a mild timbral effect and is less frequently used, and the sostenuto pedal is used very rarely indeed. Code for processing data was written using Python.

The authors of the paper in which NoteTuple representation is introduced state that the maximum time shift or duration that they allow for is 10 seconds (Hawthorne et al., 2018). They also state the number of major and minor time ticks that are used for each, but they never explicitly state the size of these major and minor ticks. However, given that they used 13 major ticks and 77 minor ticks for time shift, and 25 major ticks and 40 minor ticks for duration, it seems almost certain that they represented events to a resolution of 10ms. If minor ticks increment in 10ms steps, the largest minor tick time shift would be 760ms. Then, for 770ms to be represented, the smallest major tick needs to be 770ms. If major ticks increment in 770ms steps, then using 13 major ticks and 77 minor ticks, any time from 0 to 10 seconds can be represented in 10ms resolution, with only one possible major and minor bin combination for each time increment. 25 major ticks and 40 minor ticks works out

similarly to a maximum of 10 seconds if 10ms resolution is used. Time shift and duration seem to have identical resolution and range of times, but using different numbers of major and minor ticks; the authors don't give any justification for these choices, they merely state what they did.

In piano music, notes are constantly overlapping, especially when lengthening notes to imitate the effect of the sustain pedal. It follows that the average time shift between two notes is far shorter than the average duration of a note, and I found that representing durations of up to 10 seconds in length, and time shifts of up to 6 seconds in length was enough for representing most examples in the dataset.

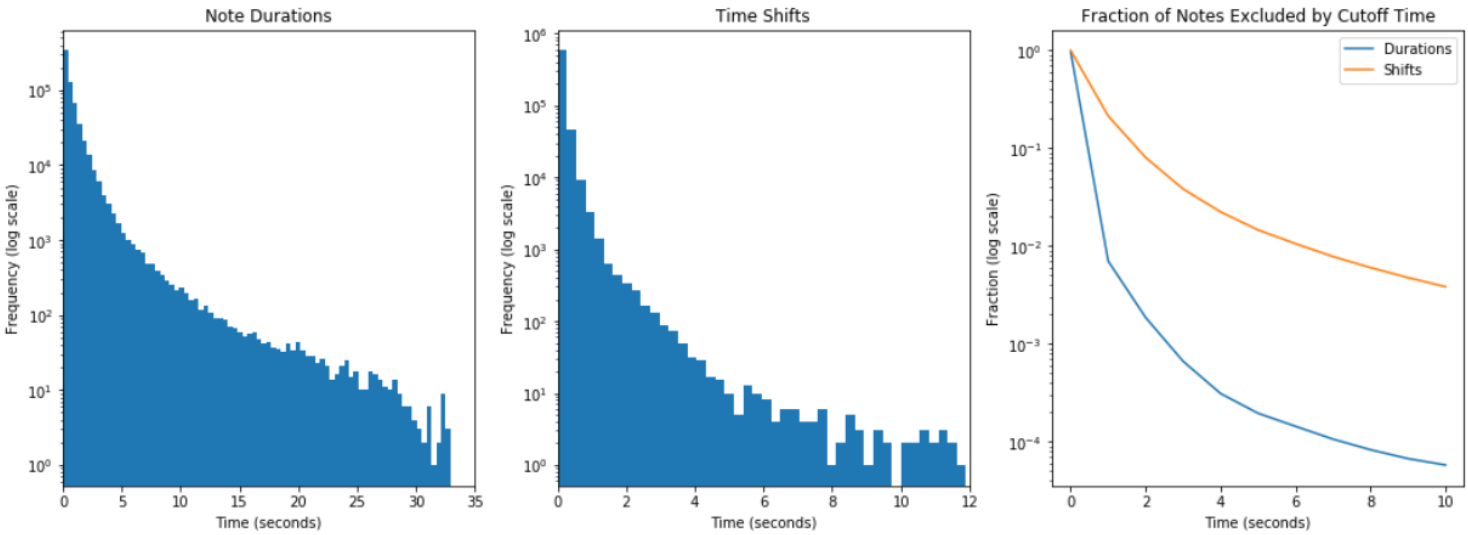


Figure 3: When working with NoteTuple, we must decide upon the maximum duration and time shift between two notes. The left and middle panels show the frequencies of different durations and time shifts in the validation data. The right-hand panel shows the fraction of notes that can't be represented as we vary the maximum duration and time shift allowed. A threshold of 10 seconds for durations with no upper limit on time shifts renders NoteTuple unable to represent 0.39% of notes. The reverse situation, with up to 10 seconds for time shifts and no upper limit on durations, has NoteTuple unable to represent 0.000058% of notes. This is because time shifts tend to be much shorter than durations (observe the differing time scales on the left and middle plots), which provides motivation for shortening the maximum allowable time shift from that originally used by the inventors of NoteTuple. Note that the Y axes are in log scale for all three plots.

The other aspect of piano music I took into consideration when implementing NoteTuple is that small differences in the time delay between two notes is much more noticeable than small differences in the note-off time of two notes. The attack portion of a piano note is sudden, percussive, and loud when compared to the decay portion that immediately follows; it is able to generate orders of magnitude more rhythmic impetus than can the silencing of a note. It follows that duration need not be represented at such fine resolution, and I therefore used a resolution of 10ms for time delay, and 20ms for the duration. I kept the ratio of major ticks to minor ticks roughly consistent with that used by Hawthorne and colleagues; the configuration I used for my first experiments with NoteTuple were 10 major ticks and 60 minor ticks for time shift, and 18 major ticks and 30 minor ticks for duration.

Models were written using the Keras functional API, a high-level API for TensorFlow. The targets for NoteTuple examples are a multilabel prediction problem in which each note must have a label assigned for every attribute, given the notes that have come previously. The loss function used was categorical cross entropy applied to each of the six attributes, and summed. Storing the training examples in RAM using one hot encoding was an issue; each attribute has many categories and storing them using one hot encoding takes up a lot more space than storing them as integer class numbers. Keras has a version of categorical cross entropy that doesn't need a one hot encoded target, instead implicitly comparing an integer target with the output of a softmax layer, avoiding the need to store the target as a one hot vector. I also used a simple custom layer in the model for mapping integer input to a one hot encoded vector, thus avoiding the need to store any data in one hot encoded format. I trained the model using a batch size of 128, and the ADAM[5] optimizer with the default values for learning rate, beta 1, and beta 2 (Kingma & Ba, 2014).

Initial training on a laptop CPU suggested these parameters worked well; the loss rapidly declined, and I was able to generate some initial samples showing the model had learnt *something* about music, even if this was only the ability to regulate dynamics by way of crescendos and diminuendos, and a consistency of velocity across nearby notes. Any harmonic relationships this poorly trained model captured were very short term and spoiled by the constant dissonance of many overly long notes overlapping.

Sampling a well-trained model

After gaining access to a Tesla K40 GPU, which reduced training time to around the tenth of what it had been on my laptop's CPU, I was able to train a model for 60 epochs and generate some more samples. Generating samples involves priming the hidden states of the LSTMs by feeding the model notes from a starter sequence, and then predicting what the next note will be. This note is then fed back into the model to update the hidden states of the LSTMs, and another note is predicted. To start with, I fed the model a single note to prime the states of the LSTMs.

When predicting a note, the final softmax layer of the model outputs a discrete probability distribution for each attribute of the note. How one acquires a prediction from these probability distributions has a dramatic effect on the quality of the sequences generated. Taking the value with the highest probability from each distribution is the simplest approach that makes sense when making a prediction for a single timestep, but fails to find the most likely sequence when the prediction is itself a determining factor for future predictions in a sequence. Taking the most likely event early on may result in less likely events later.

One possible remedy is to sample from each distribution. This frees the model from greedily taking the most likely solution at each step, which will generally not produce likely sequences; to put it intuitively, it is very unlikely that an unlikely event should *never* happen, but this is what taking the greedy solution results in. Sampling from the distribution causes the model to sometimes choose

[5] ADAM is an efficient algorithm for optimizing the model weights with respect to the loss function. It makes use of adaptive learning rates and momentum; for full technical details, see *Adam: A Method for Stochastic Optimization* by Diederik Kingma and Jimmy Ba (2014).

notes that are less likely but more interesting. The level of 'interest', or frequency with which unlikely events are chosen, can be controlled with the temperature of the softmax distribution; higher temperatures even out the resulting distribution, causing less predictable sequences in which less likely events happen just as often as likely ones, whereas lower temperatures increase the differences between high and low probability events, generating more conservative sequences.

Choosing an appropriate value for temperature proved crucial to generating sensible sequences. Notes generated by sampling from the softmax distribution with a temperature of 1 were a mess, and sounded rather like a group of toddlers bashing the piano keys while a fellow toddler controlled the sustain pedal. Temperatures that were too low resulted in the model quickly latching onto a chord and repeating it over and over again. Remember that lowering the temperature corresponds to sampling that is closer to taking the maximum probability; repeated chords are common in piano music, and if a chord is played once, it is not a bad guess that the same chord will be repeated a short time later. Given the same chord played a few times in a row, it is highly likely that the same chord will come again the same time interval later, so the safest, most conservative bet is always to predict that chord again.

In between these two temperature extremes, the model was capable of producing sequences with human expressivity, but this was local in scope; most lengthy sequences that had any consistency in mood and style were characterized by the model collapsing into repetition of a limited range of notes with little regard for long term structure or phrasing. Occasionally, coherent fragments a few bars long would emerge, but these would collapse into repetition of a few notes, or become chaotic and jarring, with rapid staccato chords no human pianist could play.

Some comments on specific samples[6]:

- *nb2[7] 0.03 chordy* – the start of this sample shows the model's tendency to find a chord and repeat it. This is a good greedy strategy for maximizing the probability of getting the next prediction correct, but a poor strategy for producing sequences that are likely overall. A-major chords rule the first six seconds before the model breaks free, moving through a few G-diminished chords in various inversions, then to Db7 (at first with a stray D-natural, which adds some colour!). This resolves, just as you would expect of a Db7 chord, to Gb-major. The sequence moves through other chords before finally alternating between Ab-minor and its dominant chord, Eb-major, the latter being repeated many times at the very end – one gets the feeling that by the final Eb-major chords, the model has forgotten Eb-major was ever the dominant of Ab-minor, that is, the harmonic context and *function* of Eb-major has been forgotten. Throughout this sequence, although there are a few 'spicy' chords, the local progressions of two or three chords for the most part make perfect harmonic sense. But there is no consideration for longer term structure and direction – these progressions aren't going anywhere.

[6] Musical samples referred to by name can be found on YouTube in the following playlist: https://www.youtube.com/playlist?list=PLCO5IgjyszQvCVXG4f_JiPaQwcCvoQCpN

[7] The first part of a sample name refers to the model that generated the sample; these models can be found in the GitHub repository.

- *nb2 0.021 snakey* – here is a more melodic sample, of which there were few. This one sounds rather like someone doodling over a simple, repetitive left hand. Again, harmony and melody make sense locally, but don't make sense long term. Notes make bars, and notes and bars make phrases which should breathe, tensing and relaxing, but here time seems amorphous. The model can effectively capture tension and release on a local scale (notice the passing note dissonances in the melody that clash briefly with left hand harmonies) but misses tension and release longer term. To put it in human language terms, it can pair words together well, but cannot write beautiful sentences. From around the 7 second mark, there is a subtle but rather gratifying crescendo, showing that the resolution of 32 gradations of velocity is more than enough to make the discretization of velocity unnoticeable.
- *nb2 0.029 chordy slightly crazy* – this sample shows what happens when the model progressively 'loses control'. The process starts when the model predicts a series of unlikely events not like anything seen before in the training data. This can cause the LSTMs to have hidden states unlike any produced by a normal training sample, which in turn produces wilder predictions – the situation spirals, the sequence degenerates, and we get rapid, crazy chords and notes that are a lot of fun.

In an attempt to generate sequences with more coherent structure, I tried using different length input sequences for priming the model. The initial generated sequences had only a single note to prime the hidden states of the LSTMs. I used three different musical fragments: the first seven and a half bars of a Chopin Nocturne, the first fifteen bars of a boisterous baroque fugue, and a soft melancholic ten bar sequence. I split these into a variety of sequence lengths, but using different length sequences to prime the model had little or no effect on the quality of the generated sequences; the generated sequence would quickly leave the stylistic realm of the input sequence, and either collapse into repetition, or pass fragmentedly pass through different musical textures.

Overfitting

Up to this point, no consideration had been given to a training/validation split for the purpose of assessing the ability of the model to generalize to unseen sequences. It seemed possible that the poor quality of generated sequences could be due, at least in part, to the model being unable to generalize well. The MAESTRO dataset comes with a suggested train/validation/test split, which I used to split up the data. At this point I also lengthened the example length to 256 notes (a little under 30 seconds long on average), which did not significantly influence the performance of the model. Training a model on the training partition and validating on the validation partition revealed that the model was very overfitted, with validation loss reaching its minimum early on in training before climbing steadily, and training loss declining for the whole training run. I tried various methods to remedy this.

None of the methods I tried were able to fully address overfitting. One that proved somewhat effective was augmenting the data. Music is perfect for augmentation because the rhythmic relationships hold true for a piece played at different tempi, and the harmonic relationships are equivalent regardless of which of the twelve possible keys a piece is played in. New training examples can be made by taking existing ones and modifying their speed or transposing them to different keys. When using a representation in which timing and pitch information is discretized into

bins, this sort of augmentation has the effect of taking a musical sequence that originally gave the model information about a particular collection of bins, and using it to give information about another collection of bins that have similar mutual relationships.

The harmonic augmentation was implemented by randomly transposing samples within five semitones (a perfect fourth) of the original key, and rhythmic augmentation was implemented by generating two extra copies of the training data at 0.9X speed and 1.1X speed of the original data, respectively. Although this failed to significantly reduce the minimum recorded validation loss, at the end of training the model with data augmentation had significantly lower validation error at a small cost to training error.

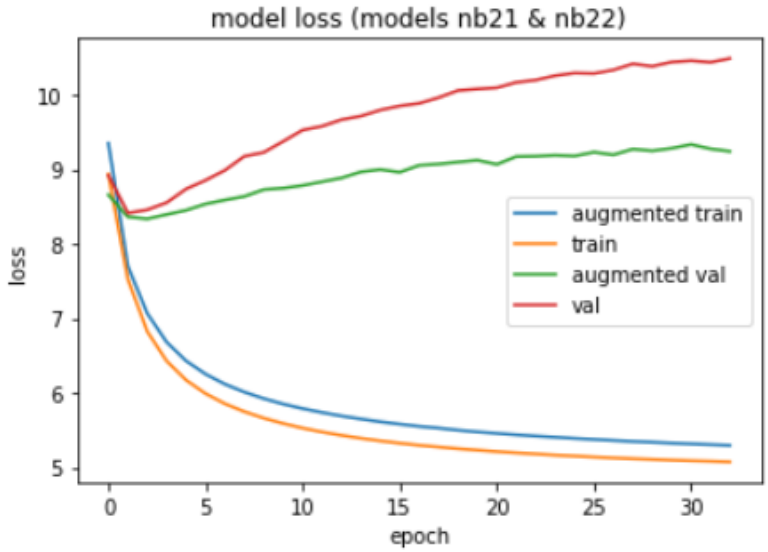


Figure 4: Training and validation loss compared with and without data augmentation. Training loss without augmentation is only a little worse than with augmentation, but the difference in validation loss is large, showing that the addition of data augmentation enables the model to generalize to new data much better. Here, one epoch is measured as seeing all the data three times at 1X speed, or once at the three different speeds.

Before trying data augmentation, I also tried some other regularizing methods. However, when trying these, I was using early stopping in my experiments, to prevent unnecessarily long training runs if validation loss stopped improving early on. As Figure 4 shows, it is possible that later stages of training can still show improvement in overfitting, even if the minimum validation loss isn't significantly impacted, and therefore the effectiveness of these techniques warrants further investigation. They include:

- Reducing the hidden state size of the LSTM from 512 to values as low as 200 – this hyperparameter had surprisingly little effect on the performance of the model, worsening training loss a little, but having little impact on validation loss. This echoes the findings of Oore and colleagues who, on the effect of this hyperparameter on the performance of their RNN, wrote, "The model consisted of three layers of 512 cells each, although the network did not seem particularly sensitive to this hyperparameter." (Oore et al., 2018)
- Adjusting the learning rate – higher learning rates result in the optimizer taking bigger steps when it updates the weights according the derivatives of the loss function with respect to

the weights. These derivatives give the slope at a single point of the loss surface, and a bigger step can result in updates that are sub-optimal short term. Raising the learning rate can thus have the effect of introducing noise into the training process, forcing the model to explore a wider area of the loss surface. In my experiments, this made it possible to train models with higher training loss, but unchanged validation loss, bringing the validation and training loss closer together. I did not try varying the learning rate during a single training run, which would be another possible approach.

- Recurrent dropout – I briefly tried adding dropout to the connections between LSTMs across timesteps, but quickly abandoned this due to lack of out of the box GPU support in TensorFlow for this option.

Comparing NoteTuple with Performance Representation

For comparing the quality of sequences generated using NoteTuple and Performance Representation, I reduced the temporal resolution of each, in order to reduce the fineness of the relationships the model has to learn; re-encoding the data like this makes the data 'go further'. In the original papers, Performance Representation represented events to a resolution of 8ms (Oore et al., 2017), and NoteTuple to a resolution of 10ms (implied but not stated by the authors) (Hawthorne et al., 2018). Using a performance of Liszt's first transcendental etude taken from the dataset, I conducted some informal listening tests at different temporal representations to see at what point a lack of resolution becomes noticeable.

Choosing temporal resolution is akin to snapping all notes to a grid whose resolution must be decided. When the average time between two notes in a fast passage approaches the minimum distance between two slots on the grid, the temporal relationships between notes become more defined by the grid resolution rather than by how the pianist plays them, even resulting in neighboring notes of fast scales being placed together in the same slot on the grid. Up to about 25ms resolution, the fast passagework of the etude survived; by 50ms, it had become a fast sequence of two and three note chords. I therefore reduced the temporal resolution of each representation by a factor of 2.5, using 25ms for time shifts in NoteTuple, and 20ms for time shifts in Performance Representation. For NoteTuple, I also reduced the resolution of note durations from 20ms, which I had used earlier, to 50ms.

For the NoteTuple model, aside from the change in resolution and the corresponding reduction in the number of possible bins for each attribute of each note, all other aspects of the architecture were left unchanged. For Performance Representation, I built a model identical in architecture except for the output, which needed to be a discrete probability distribution over 242 possible events, instead of five discrete probability distributions over six note attributes. Both were trained with data augmentation as described earlier.

The reduction in resolution, the use of data augmentation, and a longer training time had a noticeable impact on the performance of the NoteTuple model, which now achieved better consistency in its samples. The samples would still morph stylistically, wandering through a variety of

moods and textures, but each of these styles could be sustained for longer periods. Some comments on specific samples:

- *nb21 fug2-0.028 two voices* – The first two bars of this sample are from a fugue fed to the model to prime the hidden states of the LSTMs. The model continues in a quasi-fugal style for the rest of the fifty second sample, including the later entrance of a second independent line – multiple independent voices is what makes a fugue, so it was encouraging that the model learnt this, even if harmonic and rhythmic structure left a lot to be desired, being very short term still. When I had tried priming the earlier NoteTuple model with the opening bars of the same fugue, it ignored the input, quickly going its own way stylistically.
- *nb21 noc7.5-0.028 two voice ending* – The model had less success in maintaining long term stylistic consistency when given the opening bars of a Chopin Nocturne; this isn't surprising, given the density of information is so much greater in a piece with accompaniment. The opening of a fugue, being a single melody line with no accompaniment, only has a single note occurring at a time, meaning the model has to predict less notes for the same number of bars; furthermore, these notes all have the same function, whereas in a Chopin Nocturne some notes serve as a bass line, some as accompanying chords, and some as melody; the model must propagate the information necessary to model all of these elements through time which is much more costly than a passage with melody only. In this sample, after continuing in very roughly the same style for a few bars (not that rhythmic structure is clear enough to demarcate bars!), the model 'gets distracted' by a fast run that is the catalyst for stylistic change, moving from a romantic texture into a more fugal style. Any such stylistic steppingstones are difficult for the model to handle, revealing the local scope of its memory.

Generating musical sequences with Performance Representation proved much less successful. The sequences the model generated were very poor. Five or six notes would sometimes make sense together, but generated sequences were very disjointed. Generation was handled in the same way as with NoteTuple, using the temperature of the softmax distribution to control the predictability of sequences. Temperatures too low resulted in only time shift events being generated, equivalent to silence. As the temperature was raised, short unpredictable sequences of notes unrelated to the notes the model was primed on were introduced. Sounding notes would occasionally be 'forgotten' – that is, no corresponding note off event would be produced to release a note after it had been played, meaning it would remain depressed for the remainder of the sequence. I provide one sample to illustrate this:

- *oore8 0.053 strange fragments* – The model is primed with two bars in A-minor, and the first note generated is an E. So far, so good – E fits into A-minor well – but then comes a long silence, and then a mess. The most noticeable feature of this sample is the complete lack of control over timing the model has. Even though the NoteTuple models struggled with modelling phrasal structure on the order of bars, at least they captured rhythm on a local scale. Here, even that is lacking. Strange fragments are short and sporadic. Harmonic relationships are there, but only faintly. Around five seconds in, there is a G-major chord that resolves to a C-minor chord, which becomes C7; such moments of harmonic clarity exist only occasionally. Early on, there are a high F# and A that are left sounding. One gets the feeling the model has forgotten about these; it is only because these notes are played later in the sequence that the corresponding note-off signal is sent to release them.

One of the aims of this project was to compare note representation with NoteTuple. The possible advantages and shortfalls of each have been compared on paper, but ideally generated samples would provide more insights into their comparative strengths and weaknesses. However, due to the unexpectedly poor sequences generated using Performance Representation, it is difficult to come to any conclusions; the Performance Representation samples are much worse than those provided in the material extra to the original paper.[8]

Possible reasons for this poor performance include:

- Poor choice of hyperparameters – It is possible the model using Performance Representation needed further optimization to achieve the performance noted in the original paper.
- Poor method of sampling – Even if the choice of learning rate and other hyperparameters were appropriate, the method used to sample sequences from the model was inferior to the original paper. Sampling one event at each step of the sequence does not guarantee a likely sequence overall. Beam search was used in the original paper, which involves trying different choices of event at each time step, and observing what impact each of these choices has on the total probability of the sequences generated from them. In an ideal world with limitless computational power, the full space of all possible sequences would be evaluated, but this is impossible, and beam search provides a computationally cheap way of exploring a very small part of this space.

Possible reasons why we would expect the NoteTuple model to perform better than Performance Representation include:

- It could be argued that the model using NoteTuple, by using the same hidden state to model the probability of six attributes of a note, is provided with the opportunity to model the probability of these six attributes jointly. This may be roughly equivalent to implementing a short-term version of beam search.
- NoteTuple, by predicting tuples of attributes, condenses sequences so that they are shorter; as far as the LSTMs are concerned, this reduces the timesteps over which they have to propagate memories to inform future predictions.
- Because the language of NoteTuple is more constrained, the model already had a head start on the Performance Representation model; using such a language, it is only possible to make notes with a complete set of attributes, even if these attributes are not musically meaningful.

Possible reasons why both models might be performing sub-optimally:

- Training examples begin and finish in random places; most of the time, the first and last parts of samples will contain notes that only make sense in the context of the larger phrase.
- Although the MAESTRO dataset contains classical piano music only, the genre of ‘classical music’ spans hundreds of years and countless sub genres, each of which constitutes a very different prediction task.

[8] For generated sequences from the work by Oore and colleagues, see: https://clyp.it/user/3mdslat4

- What the model learns to do during training doesn't always help at prediction time: in particular, once the model has generated some musically crazy outputs, these are fed back into the model, so that it is now working with a sequence and hidden state unlike anything it has seen in the training examples.

The problem of comparison

As noted above, if the NoteTuple model is doing something similar to jointly modelling the probability of the six attributes of a note, then it is unfair to compare a Performance Representation model without beam search directly with a NoteTuple model, and conclude the latter performed better in identical circumstances. Introducing beam search to only the Performance Representation model also seems 'unfair', because beam search introduces even longer-range joint modelling of events. Although one solution would be to use beam search with both, the use of beam search is not straightforward with NoteTuple, due to the distributions across multiple attributes produced at each timestep, making it unclear how to rank the top k most probable events.

It therefore appears that the differences of these two representations are too great, such that they cannot be benchmarked against each other through use of a common architecture and sampling method. Different choices of architecture may be more appropriate for each representation; this was commented on by the authors of the paper in which NoteTuple was introduced, who noted that their representation did not require such large models. When two methods are not directly comparable, the most reliable comparison seems to be to optimize the models in every possible way, and compare the best possible performance achieved by each. This, after all, is the final benchmark that matters.

5) Chroma

The representations explored so far are very low level and do not hint at the wealth of harmonic relationships that exist in music. In this section I explore the effect of adding chroma, a condensed representation of sounding notes, as a supplemental input to a NoteTuple model.

The Western classical music tradition has made use of twelve pitch classes, which range from A to G#. The frequencies of notes common to a pitch class have a particularly simple relationship, causing them to share tonal characteristics to the point that we have come to regard them as equivalent in some sense. Pitch classes alone can be enough to harmonically summarize a selection of notes; for example, an A, C# and E form an A major chord, regardless of which A, which C#, and which E are chosen.

Chroma is the collection of all pitch classes currently containing sounding notes, and can be represented as twelve indicator variables indicating the presence ('1') or absence ('0') of each pitch class. One paper, which tackled the task of chord recognition in audio samples, describes the link between chroma and harmony and the relevance of chroma to chord recognition as follows:

> *Chromagrams are concise descriptors of harmony because they encode tone quality and neglect tone height. In theory, this limits their representational power: without octave information, one cannot distinguish e.g. chords that comprise the same pitch classes, but have a different bass note (like G vs. G/5, or A:sus2 vs. E:sus4). In practice, we can show that given chromagrams derived from ground truth annotations, using logistic regression we can recognise 97% of chords (reduced to major/minor) in the Beatles dataset. This result encourages us to create chroma features that contain harmony information, but are robust to spectral content that is harmonically irrelevant (Korzeniowski & Widmer, 2016).*

Providing a NoteTuple model with chroma could strengthen the signal relating to how bins are harmonically related. Using 88 discrete bins to represent the 88 notes of the piano, the common harmonic relationship across every B and E needs to be learnt by the model independently. Chroma describes this commonality succinctly. As well as helping elucidate relationships common across pitch classes, chroma can also provide the model with information about what sort of sonic and harmonic environment currently exists due to all sounding notes. A pianist, via the ears, has real time information regarding the intensities of all sounding notes and the harmonic color generated as a result. Without any feature indicating what notes are currently sounding, the model must infer this environment by remembering the durations and timings of previous notes.

Chroma is a good harmonic descriptor of music by the Beatles, but probably not such a good descriptor of harmony in classical piano music. In classical piano music, during the course of a single harmony, many expressive melodic notes that are in conflict with the prevailing harmony may be played. If these notes are relatively high, then these 'passing notes' that briefly elevate the tension of a passage can have their sound carried on in the sustain pedal to no detrimental effect because, despite still ringing, they are forgotten once the next melody note is played. The high notes of the piano fade very quickly, and the attack of a note is so much louder than the sustain portion, so that it is common practice to keep the sustain pedal depressed, relying on the greater sustaining power of the bass notes and the loud attack of subsequent melodic notes to maintain harmonic clarity in the face of many still-ringing passing notes. Any high passing notes are lost to the perception of the listener when the attack of the next melody note draws all attention to it. But they are still present, and will have just as strong an effect on chroma as will a strongly resounding bass note. There is the possibility that using chroma to capture all sounding notes without regard to tone height could pollute the harmonic signal carried by the chroma; this provides justification for testing chroma in which tone height information is provided in some way.

I incorporated these ideas into two variations on chroma, resulting in four experimental setups to compare:

1. No chroma.
2. Normal chroma, using 12 indicator variables, each indicating the presence ('1') or absence ('0') of a pitch class.
3. Chroma with a simple weighting scheme, giving more weight to pitch classes with lower notes. The 12 indicator variables are replaced with values between 0 and 1, with the value indicating the height of the lowest note of that pitch class. Values closer to 1 indicate lower notes, and values closer to 0 indicate higher notes; 0 indicates absence.

4. Chroma in which the only pitch class indicated is that of the lowest note. Of any sounding note, the most important harmonically at any time is the lowest, and providing the lowest note only ensures that chroma has the highest possible bass note signal to noise ratio, at the cost of stripping the chroma of any information regarding how this lowest note relates harmonically to other sounding notes.

The model without chroma was given the same number of inputs as the models with chroma, with the twelve inputs normally used for the twelve pitch classes set to zero.

All four models were trained for 75 epochs, and without any data augmentation. In each case, the presence of chroma provided a speed boost to learning, accelerating the initial drop in both training and validation loss.

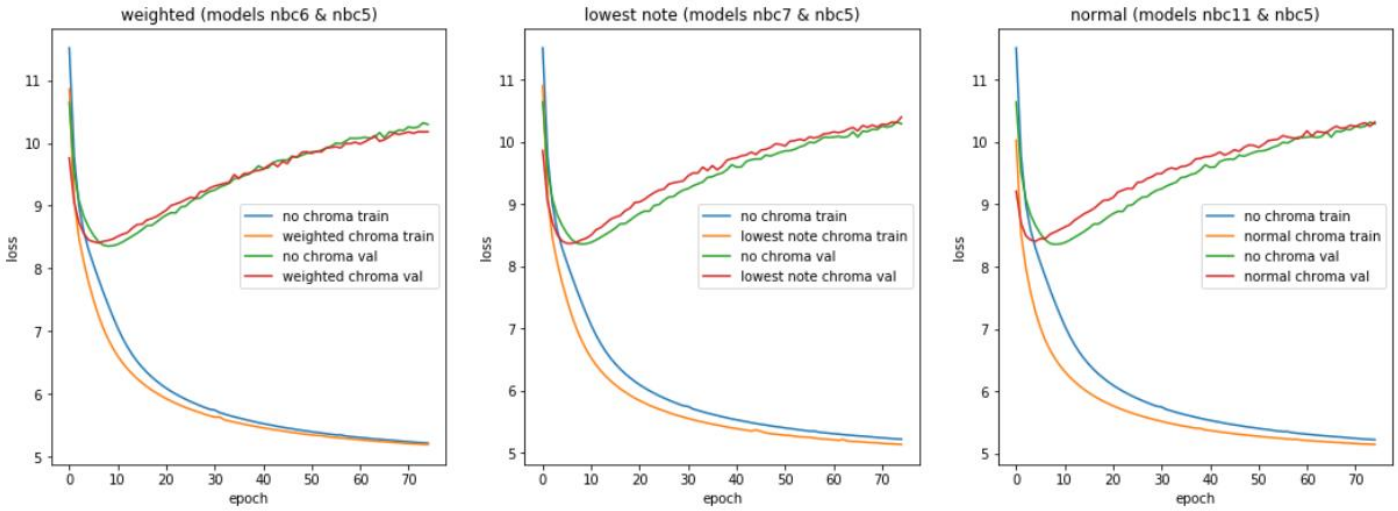


Figure 5: Each plot contains the loss of a model with chroma and the loss of the baseline model without chroma for comparison. In each case the model with chroma trains faster early on.

The model without chroma more or less 'caught up' to the models with chroma by the end of 75 epochs, and final training and validation loss were similar between all four models. Initially, I thought that the presence of normal chroma was significantly lowering the final validation loss, without a corresponding raise in training loss. This would have indicated chroma was enabling the model to generalize better to unseen sequences while still performing just as well on sequences from the training set – but this turned out to be an error on my part. All four models were trained without data augmentation, but for the model with normal chroma I accidentally referred to metrics from a separate training run in which random transposition was applied to the training sequences. At least this bears witness to the effectiveness of random transposition at regularizing a model.

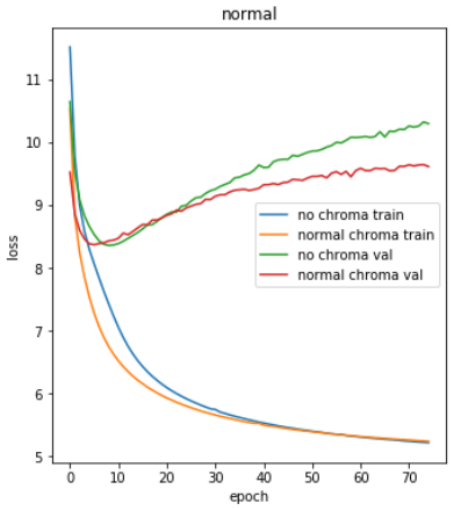


Figure 6: The offending plot. I initially thought the improvement was due to the inclusion of normal chroma, but it turned out I was mistakenly referring to metrics from a training run that included random transposition.

It is difficult without further experiments to say exactly what it is about chroma that contributes to the early speed up of model fitting. It is probably a combination of the two factors described above, namely,

1) Providing the model with succinct harmonic signal, so that the model is saved from needing to learn particular harmonic abstractions itself;
2) Providing a harmonic/sonic picture of what notes are currently sounding, 'giving the model ears'.

Given that the final loss was similar for models with and without chroma, it seems that whatever abstractions chroma saves the model from making during the early stages of training are abstractions that the model is capable of making itself, given enough time. It is also interesting to note that the biggest speed up came from introducing normal chroma. This indicates the high notes of the piano contain useful signal that is more helpful than the noise they introduce is harmful.

Possible future directions

The results of the experiment with different forms of chroma raises some interesting possibilities for further investigation into such derived features. These include:

- Testing chroma within specific genres of classical music – testing chroma across different genres could give insight into how it is able to affect model performance. Piano music differs greatly across different genres, and different versions of chroma may be more suitable for music of different periods. In this project, training data was never filtered to specific sub genres of classical music.
- Deriving features to give a more complete picture of the sonic environment – this could include things like using a separate model to estimate the current volume of each sounding

note, or total volume of all sounding notes. Very accurate virtual piano models exist that could be used for this.

- Deriving features that reveal the course of the future harmonic/sonic environment – chroma, as it has been applied here, can inform the NoteTuple model about the harmonic and sonic environment at the time the last note prediction was made, but the model must then predict a note with a time shift attribute, so that the note occurs *after* this harmonic/sonic snapshot (if the time shift is non zero).
- Investigating derived features as a possible remedy to the pitfalls of NoteTuple – the absence of a separate note off event makes remembering currently sounding notes at each timestep a more complex task for NoteTuple; this was noted as one of the biggest drawbacks of NoteTuple by its inventors, who found it resulted in dissonant overlapping notes (Hawthorne et al., 2018), and it may be that derived features that reveal the current or future harmonic/sonic environment ('giving the model ears') could be used to address this.
- Testing chroma with Performance Representation – Performance Representation, by having separate time shift events, can utilize chroma as a snapshot of the *current* harmonic/sonic environment. Chroma might affect Performance Representation models quite differently.
- Testing such derived features with different quantities of training data – less data is a simple matter of restricting the training set, and it is possible to acquire orders of magnitude more training data than used in this set of experiments, for example, when models are trained on automatically transcribed recordings from YouTube (Simon et al., 2019). It may be that features that improve performance can 'make data go further', but do not contribute to a model's performance when there is sufficient data to learn equivalent or better higher-level features from the data alone. It may also be that certain derived features or representations that introduce helpful signal or bias when data is limited instead introduce harmful signal or bias when the training data is of sufficient volume.

Conclusion

One of the clearest conclusions to emerge from this set of experiments is that *representation and encoding matter a great deal in deep learning*. Just as some languages are more polite, succinct, or prone to particular sorts of miscommunication, the encodings explored in this project each have unique attributes that cause models using them to vary widely in their ability to capture particular signals in the data. In the case of NoteTuple, this includes things like constraining the model to output musically possible sequences, and the possibility of using a single LSTM hidden state to model all the attributes of a note at a single time step. By contrast, Performance Representation was less constrained, and had much better modelling of groups of note offs, due to their representation as contiguous events.

The experiments with chroma show that there is the potential to find succinct, higher level musical descriptors that can aid the training of models using low-level representations such as NoteTuple. In the realm of human language, this is akin to enriching a language with vocabulary that encapsulates higher level concepts, so that discussions about those concepts are not awkwardly verbose, even making possible further abstractions to even higher-level concepts. Succinct musical descriptors can capture musical concepts on a level closer to that which humans use to describe or perceive music, or perhaps act as sonic descriptors that 'give the model ears', saving the model the need to learn

such abstractions itself. It is possible too that these features could harm a model's performance; as humans, our high-level vocabulary constrains or biases how we conceive of abstract concepts, and there is the possibility that higher level musical features could bias a model, leading it away from more helpful representations that it might otherwise find. More experimentation would be required to ascertain exactly how chroma is aiding NoteTuple, and to determine in what circumstances such a biasing effect might occur.

It can be tempting to think that given enough data and learning capacity, representation doesn't really matter, so long as it is low level and doesn't assume too much. This project used a large dataset with more than one thousand performances and more than six million notes; combined with data augmentation, this provided a wealth of information with which to inform a model about musical relationships. But the large dataset was no silver bullet, and neither was deep learning, and careful thought had to be given to the encoding of the data at every stage of this project; just as human vocabulary and language are essential to and even constrain high level discussions in humans, so too is data representation vitally important to the task of building deep learning models.

Bibliography

Hawthorne, C., Huang, A., Ippolito, D., & Eck, D. (2018). Transformer-NADE for Piano Performances. *Submission, NIPS Second Workshop on Machine Learning for Creativity and Design*.

Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.-Z. A., Dieleman, S., Elsen, E., Engel, J., & Eck, D. (2019). Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset. *International Conference on Learning Representations*.

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

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Simon, I., Hawthorne, C., Shazeer, N., Dai, A. M., Hoffman, M. D., Dinculescu, M., & Eck, D. (2018). *Music transformer: Generating music with long-term structure*.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*.

Korzeniowski, F., & Widmer, G. (2016). Feature learning for chord recognition: The deep chroma extractor. *ArXiv Preprint ArXiv:1612.05065*.

Oore, S., Simon, I., Dieleman, S., & Eck, D. (2017). *Learning to create piano performances*. NIPS Workshop on Machine Learning for Creativity and Design.

Oore, S., Simon, I., Dieleman, S., Eck, D., & Simonyan, K. (2018). This time with feeling: Learning expressive musical performance. *Neural Computing and Applications*, 1–13.

Simon, I., Huang, A., Engel, J., Hawthorne, C., & Dinculescu, M. (2019, September 16). *Generating Piano Music with Transformer*. https://magenta.tensorflow.org/piano-transformer

The Complete MIDI 1.0 Detailed Specification. (n.d.). Retrieved 15 February 2020, from https://www.midi.org/specifications-old/item/the-midi-1-0-specification