Finger Finagler: Automatic Piano Fingering

Macy Huang
University of Texas at Austin
Austin, TX 78705
macyh@utexas.edu

1 Introduction

1.1 Background

One of the earliest and most critical skills that is learned when playing an instrument is finger placement – which fingers should play which note within a sequence of notes. This problem is somewhat complex in that it requires an understanding of conventional fingering rules that develops as piano novices become experts based on musical context, dynamic expressiveness, and physical hand constraints [1]. In this project, musical context is defined as the notes (or rests, periods of silence) before and after a note is played, dynamic expressiveness refers to the force and speed with which a note is played, and physical hand constraints refer to each hand having five fingers (we designate them from 1-5 with the thumb on both hands being "1" and the pinky finger being "5"), the left and right hand being mirrors of each other, ability of the thumb to cross under the other fingers to play a sequence longer than five notes smoothly, and other such factors. A successful pianist modulates their fingerings based on all of these aspects and more. This makes fingering a somewhat subjective task as players have differing standards for comfort and musical interpretation. For pianists at a beginner to intermediate level however, fingering choices tend to favor stability and comfort above the other factors, making it possible–and helpful–for piano novices to standardize [4].

The problem of Automatic Piano Fingering (APF), as a challenge of combinatorial optimization, has been worked on extensively with statistic-based approaches and Hidden Markov Models but only recently had neural network models reach state-of-the-art performance [1]. This project seeks to implement an Automatic Piano Fingering scheme using a Recurrent Neural Network to generate fingerings. Accurate APF modeling has applications mainly for novice piano students but can also be applied in areas such as robotics (as a task for robot digital dexterity and manipulation), music information retrieval, and determining playability for machine-generated or transcribed music [2].

1.2 Related Works

The current state-of-the-art approach in the field of APF uses autoregressive encoding and decoding [1]. Ramoneda et al. use different variants of RNNs and GNNs throughout the process, producing excellent results using a bidirectional LSTM which is stronger in sequence-to-sequence learning and a directional Graph Gated Neural Network which can learn node-level note representations better. The authors take this approach in order to improve the performance of the model on polyphonic passages, where there are more than two melody lines taking place. Previous state-of-the-art approaches were based on statistics, and sequences of key presses were fed into a Gaussian Hidden Markov Model with Bayesian probability [2]. Viterbi search was then used to find the optimal next state. A similar work from 2020 also used the HMM approach but introduced two neural network-based approaches on the side, neither of which performed as well as the HMM predictor [3].

Other approaches to this problem include using dynamic programming to efficiently trawl the search space of possible fingerings to find an optimal fingering [4] and more recently, applying a transformer

Submitted to Dr. Alexander Huth as a project for the C S 342 Neural Networks course (Spring 2023). Do not distribute.

model architecture to the problem. Initial results of the transformer seem promising, though accuracy of predicted fingerings is only 70.9

2 Model and Methodology

Upon initial investigation of the problem, we considered using a transformer neural network as it would be able to address the contextual challenge of piano fingering across long passages of music. For various reasons that will be discussed in the ChatGPT Assistance section, we pivoted to the use of RNNs instead, specifically Long Short-Term Memory networks so as to avoid some of the issues of exploding and vanishing gradients. LSTM networks are easily trainable and are able to compute data efficiently, so for a task like APF which has traditionally been undertaken with statistics, we determined that note and fingering interdependencies within our training set were simple enough that an LSTM would suffice.

Additionally, given the relatively accurate statistical approaches to the APF problem, we considered integrating a rule-based deterministic algorithm to make our neural network faster to train and require less data to generalize well. This technique can be used very practically because it is faster and easier to train a network from some starting point, even if that point is sometimes wrong, compared to training it to understand something completely from scratch.

2.1 ChatGPT Investigation

As detailed in Assignment 4, we investigated the use of OpenAI's ChatGPT for assistance in researching information and code generation. Through interacting with ChatGPT, we decided that our dataset was not broad enough for us to use a transformer neural network to tackle the problem. We were hopeful that integrating a Gaussian Hidden Markov Model as a deterministic algorithm for our network to learn from would reduce the need for so much data but initial experiments with the transformer architecture produced extremely poor performance, with heavy overfitting on the training set.

ChatGPT instead recommended a Convolutional Neural Network and Recurrent Neural Network combination: a CNN to process input sheet music visually and an RNN to learn and predict fingerings for a piece of music. We determined that the RNN problem was more related to our APF challenge and the Optical Music Recognition problem that would arise out of training a CNN was out of scope, considering that recognizing and understanding musical notation has its own fussy complexities.

ChatGPT was consulted for code generation for the Gaussian HMM algorithm + LSTM approach we wanted to use as an experiment, but we found the code it produced to be vastly inferior to what we could synthesize with the Python hmmlearn library documentation. We hypothesize that this is because ChatGPT has limited experience with problems related to this and lacks experience with HMMs. It attempted to have us use Sci-Kit Learn's HMM library, which has been deprecated since 2016 and had to be corrected at multiple points in the process because it would hallucinate loosely-related lines of code even given the same prompt. The HMM + RNN approach was abandoned for this reason, in addition to the fact that our dataset was large enough that using a deterministic function to aid training was unnecessary.

2.2 Data Processing

The dataset chosen for this task is the PIano fingerinG (PIG) dataset compiled by authors of the former-SOTA HMM paper [2]. The dataset was generated for the Nakamura et al. paper and referenced in Ramoneda et al. for model fine-tuning. This dataset contains text files that provide a list of all of the musical notes in a particular piece and information relating to it like onset time (when the key is pressed), offset time (when the key is de-pressed), channel (0 for right hand and 1 for left hand) and finger number (the human-labeled fingering for a particular note).

For the purposes of my project, we converted the list of musical notes in each file to its corresponding key value in a piano. A full-size piano has 88 keys, black and white, and these keys were labeled as a spelled pitch in the dataset. Since the keys were provided as key tone (e.g. C, Abb, Bb) and octave number together. we processed these spelled pitches factoring in accidentals (sharps and flats) and

enharmonic tones (keys like B and Cb which are the same but may be written differently) so that the lowest note on a piano, A0, has a label of 1 and a key like C4 has a label of 63.

In terms of additional processing, since the dataset had left-hand fingerings labeled from -1 to -5, we had to convert all fingerings to be from 0 to 9 so that we could properly classify and predict finger classifications (as labels were not allowed to be negative for one-hot encoding). The dataset was processed as a custom PyTorch Dataset and split into a training, testing, and validation dataset to be put in a Dataloader object.

2.3 Architecture

The structure of our model is roughly similar to implementations of Long Short-Term Memory networks we implemented for homework assignments and were demonstrated in class. The model was initialized with an LSTM layer and a feedforward layer to transform the input. A hidden state and cell state were also passed in. The baseline model used an input size of 1 because the input data (key value from 1 to 88) has only that one feature. Hidden state size was initialized as 32 as a standard size. Since there are 10 possible fingers that could be selected, output size was set to 10.

We selected Mean Squared Error as the loss function and used Adam as an optimizer with a learning rate of 0.00001. The model was then trained over our training dataset, loss was backpropagated, and weights were updated. The baseline loss started at 0.12 and converged to around .0505 within 100 epochs with a similar value for validation loss.

3 Experimental Results

3.1 Computational Experiment 1: Hidden State Size

Motive This was one of Dr. Huth's suggested experiments, and considering our relative unfamiliarity with choosing good settings to initialize models to, we decided to investigate a few different hidden state sizes. This hidden state size parameter represents the number of LSTM units in a network.

Hypothesis We hypothesized that increasing the hidden state size would improve the performance of our model up to a certain point since adding more LSTM units provides increased ability to represent complicated patterns in the data. However, adding too many hidden units could result in overfitting which we would see manifested in the unchanged validation loss.

Results We experimented with setting the hidden state size to 64 (double our baseline of 32 units) and 256 (eight times the baseline). The 64-unit experiment resulted in a lower initial training loss and was slightly faster to arrive at the .05 range. After 100 epochs, the training loss converged to .0503 while the validation loss remained at 0.0507 which was slightly less than our average baseline validation loss. The 256-unit experiment had an initial training loss of .0539 which was much lower than the baseline loss at epoch 0. After 100 epochs, the training loss converged to

Discussion We decided that the result of doubling hidden state size was not significantly different from the baseline. We repeated our 64 and 256-unit experiments three times in total, and there was consistently lower loss as a result of our modification but the difference was extremely small. We interpret this result to mean that 32 units is a sufficient number for the complexity of the APF problem and increasing the hidden state size will not improve our model's ability to generalize a significant amount.

3.2 Computational Experiment 2: Adding Directionality

Motive One of the concepts introduced by Ramoneda et al. was that using a bidirectional LSTM encoder was the correct approach while building their network since music fingering is not only dependent on the states before a current state within a sequence of notes but also on the notes that come after it. This is also an important element in a pianist's comfort in playing and in a real-life fingering would be just as important of a consideration as the note immediately preceding our current state.

Hypothesis We expect that making our LSTM bidirectional will make our model more accurate. Based on our intuition about how sheet music is fingered, incorporating "backwards" information should result in a model with as good, or probably more accurate performance.

Results Upon making the LSTM layer bidirectional and adjusting the model input to reflect that (the data was literally reflected and concatenated) we found that the loss started out around baseline levels and after 100 epochs, converged to 0.0504 with a validation loss of 0.5069. These results are pretty close to the baseline results, like we found in the first experiment.

Discussion These results were very surprising to us, considering that musical context both before and after notes is something carefully appraised in real-world fingering. From further research, we learned that perhaps the nature of music being temporally reflexive might be relevant here, i.e. fingerings applied to a piece of music could be reversed and applied to the reversed piece of music, which is a property known as time inversion [6]. This property of music seems to make the bidirectionality of the Ramoneda et al. paper's model less strictly necessary and its removal could aid in increased computational efficiency.

4 Conclusion

Ultimately, our model was able to consistently achieve a mean squared error loss within around .05 of the expected value. The computational experiments we performed did not significantly improve our baseline model without trade=offs in computational time and model efficiency but we were able to get a lowest training loss of 0.05032 and a lowest validation loss of 0.05062. There were some limitations of our implementation of the model that prevented total accuracy and would limit its applicability in a real-world situation, such as the fact that the model does not account for fingering changes within a note (that is, when a finger is substituted for another finger on a note while the note is held). Also, the fact that all sheet music would have to be processed to fit the format of the dataset our model was trained on would prevent broad use.

In terms of future work, our model can be surely expanded to be more usable for a real piano student such that it handles more data processing and helpful music generating. Furthermore, since our LSTM model was trained mostly on classical music, we can make the model more robust by training it on contemporary forms of music with more complex polyphony and melody structures. Overall, the results of our experiments seem to suggest that boosting computational power and scaling our model up could result in some minor performance gains, which we would like to explore in the future.

References

[1] Ramoneda P, Jeong D, Nakamura E, Serra X, Miron M. Automatic piano fingering from partially annotated scores using autoregressive neural networks. In: 30th ACM International Conference on Multimedia (MM'22); 2022 Oct 10-14; Lisboa, Portugal. New York: ACM; 2022. 9 p. DOI: 10.1145/3503161

[2] Yonebayashi, Y., Kameoka, H., Sagayama, S. (2007). Automatic Decision of Piano Fingering Based on a Hidden Markov Models. *International Joint Conference on Artificial Intelligence.*

[3] Nakamura, E., Saito, Y., & Yoshii, K. (2020). Statistical learning and estimation of piano fingering. *Information Sciences*, 517, 68-85.

[4] Kasimi, A.A., Nichols, E., & Raphael, C. (2007). A Simple Algorithm for Automatic Generation of Polyphonic Piano Fingerings. *International Society for Music Information Retrieval Conference.*

[5] Suzuki, M. PIANO FINGERING ESTIMATION AND COMPLETION WITH TRANSFORMERS.

[6] Malwine Bree and Seymour Bernstein. 1997. *The Leschetizky method: A guide to fine and correct piano playing*. Courier Corporation