CS230

PianoNet: A Hyperparameter Study of a Deep Neural Network for Automatic Music Transcription

Maddie Wang
Department of Computer Science
Stanford University
maddiew@stanford.edu

Susan Chang
Department of Computer Science
Stanford University
schang92@stanford.edu

Abstract

In music transcription, musicians typically listen to a recording then transcribe what they hear, note by note. This process can take hours, depending on the complexity of the music. It is important for composers and musicians to make scores without needing to manually write each note out. Despite research showing that convolutional neural networks (CNN) has led to the most accurate results in computer vision these past couple of years, we decided to do a hyperparameter study on a fully connected Deep Neural Network (DNN) to see if DNNs can outperform CNNs in polyphonic music pitch prediction. At a high level, our DNN takes in Constant Q Transforms of raw audio files, then outputs the predicted pitches present in the main melody of the audio files.

1 Introduction

Transcribing music is a difficult task even for human experts. As of now, musicians, who have extensive knowledge in music, must manually compose music online or on paper. To make music composition less time-consuming and more accessible to those not familiar with music theory, we introduce a deep-learning application that can automatically transform piano recordings into digital notes. Currently, automatic music transcription (AMT), which attempts to simplify the transcription process through detecting rhythm, pitch, and chord patterns, plays a significant role in music transcription. However, it is fails to be comparably accurate with human performance because of its sensitivity to instrument timbre and harmonic overlap [1]. In recent years, convolutional neural networks have led to great improvements in acoustic modeling and is currently the leading approach in automatic music transcription [8]. Though CNN currently outperforms all other approaches in AMT, we noticed that there is a lack of research in optimizing DNN hyperparameters. We thus created a deep learning application that allows musicians to transcribe piano recordings using a deep neural network (DNN) approach and tuned its hyperparameters to see if DNNs can outperform the leading approach in automatic music transcription. Specifically, we experimented with various optimization algorithms including Adam, Adagrad, AdaMax and RMSProp, tuned the dropout rate, incorporated isolated notes in training, and increased the number of hidden layers in our hyperparameter study.

On the whole, our algorithm takes in an audio recording of piano music in .WAV format as its input. After pre-processing our .WAV file through Constant Q Transform, we can then use our deep neural network to output a predicted array of pitches that correspond to the piano notes played in the audio file. The output of our DNN can potentially be further processed as a MIDI (Musical Instrument Digital Interface) file which can then be converted to a music score.

CS230: Deep Learning, Spring 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

2 Related work

Our project seeks to extend the approach done by Diego Morin, where the author implemented a Deep Neural Network following the recommendations proposed by Sigtia, Benetos, and Dixon in their 2016 paper "An End-to-End Neural Network for Polyphonic Piano Music Transcription" [8]. Due to time constraints, the author did not tune their model's hyperparmeters. We thus build upon their research by optimizing performance through tuning the model's hyperparameters.

This task of applying deep learning methods to music transcription is interesting, because the research of Sigtia et al., and replicated efforts by Bereket and Shi, Stanford authors of the project entitled "An AI Approach to Automatic Natural Music Transcription", and Li, Ni, and Yang, Stanford authors of the project entitled "Music Transcription Using Deep Learning" have consistently shown that ConvNets consistently perform better than the unsupervised methods most commonly used in current AMT programs.

Following the approach taken by Sigtia et al, we are breaking down our AMT system into two parts, acoustic modeling that identifies pitches in polyphonic audio, and score generation that converts the MIDI file into music notation.

3 Dataset and Features

The dataset we used in our project was the MIDI Aligned Piano Sounds (MAPS) dataset, which is a database of MIDI-annotated piano recordings. The MAPS dataset is divided into 9 subsets, each of which contain 30 audio recordings in wav format. In each subset, for each wav file there is a corresponding ground truth provided both as an aligned MIDI file and a text file. The text file contains the pitches of all the notes contained in the recording, as well as the onset and offset times of each note. Seven of the nine subsets contain recordings generated by high quality synthesis software and in various recording conditions. Furthermore, each of the nine subsets of audio recordings were subdivided into four categories of recordings: isolated notes and monophonic excerpts (ISOL set), chords with random pitch notes (RAND set), usual chords that are standard to Western music such as classical or jazz music (UCHO), and pieces of piano music (MUS set). For the purposes of our project, we used the recordings from the MUS set. Thus, the MUS sets from the seven software-generated recordings comprised our initial training set, totaling 210 pieces of music. The two remaining subsets of the MAPS dataset consist of recordings done on a real piano, so the MUS set from these real piano subsets comprised our validation and training sets, with 30 pieces of music for each of the validation and training sets.

Raw audio files are represented as byte arrays that contain information about the samples, or amplitude of the audio signals, in that audio file. We preprocessed each wav file in our training and test sets to extract the main features from audio files. The sample rate and raw data from each wav file is read, and the audio files are transformed from stereo (audio using multiple channels or signals) to mono (audio using one channel) by taking the mean of the samples from the two channels. Then the time series representation of the audio in mono is then transformed to a time-frequency representation using Constant Q Transform (CQT). CQT is like the Fourier transform except with geometrically spaced center frequencies. The frequency of musical notes are in geometric progression, so this makes makes extracting CQT features more suitable for automatic music transcription. As shown in the below figures, the CQT produces frequency-time plots that best resembles the ground truth pitch-frame plots that are to be predicted.

Figure 1: Pitch-frame Plot of ground truth label (x-axis is frame, y-axis is pitch)

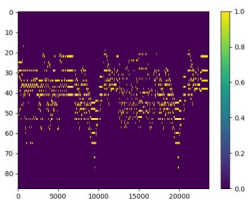
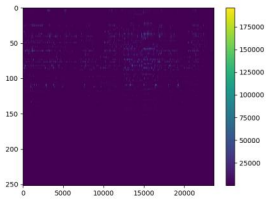


Figure 2: Time-Frequency Plot of CQ Transform of a WAV input (x-axis is time, y-axis is frequency)



The CQT performed on the raw data from our input wav files was computed over 7 octaves with 36 frequency bins, or frequency intervals between samples, per octave. Thus, our features are the 252 total frequency bins per frame.

The ground truth labels for the training audio were created from the text file that corresponds to the audio file. More specifically, the output label is a binary value array with height that corresponds to the number of frames in the audio file, and width of 88, to correspond to the 88 notes of a piano keyboard, and the onset and offset times and the pitch values from the text files were used to create the ground truth labels for the training audio. What we get out of the preprocessing are two arrays for each audio file, an array representing the constant-Q transform of the audio file, and an array representing the ground truth labels for the audio file.

Then, each of the 252 features across all frames are normalized by subtracting the minimum value across all features, then dividing by the range of the features, and then subtracting the mean.

4 Methods

Our project is a hyperparameter study of the deep neural network architecture built by Diego González Morín [3]. The DNN architecture Morín built was based on the architecture recommended by Sigtia et. al. [8]. The architecture is a 3-layer DNN with 256 units in each hidden layer, with each layer using ReLU activation. The DNN uses Adam optimization, and the output layer uses sigmoid activation. The output layer has a size of 88 units, to represent the 88 possible pitches of a piano keyboard. To prevent overfitting, a dropout rate of 0.2 and early stopping were used. The loss function used in Morín's architecture is the mean squared error between the ground truth labels vector and the predicted labels for each frame.

5 Experiments/Results/Discussion

Following the recommendations for a DNN model in [8], our model uses a mini-batch size of 100 for the stochastic gradient descent updates. Due to time constraints, hyperparameter tuning was beyond the scope of the project of the architecture's original author. DNN's have been shown by Sigtia et al [8] to have poorer performance than their proposed convolutional neural network (CNN). Our project seeks to study the effects of different hyperparameters as well as the augmentation of data, in isolation, on the effectiveness of the model to predict the pitches of a test audio set, to see if we can get performance comparable to that of Sigtia et al's CNN.

The output is a binary-valued vector that represents which pitches were predicted at each frame. The predictions were rounded to 0 if the output values were lower than 0.5 and rounded up to 1 if values were greater or equal to 0.5. The evaluation metrics used were accuracy, precision, recall, and

F-measures. Below are how these metrics are calculated:

$$\text{Precision (P)} = \sum_{t=0}^{N} \frac{\text{True positives(t)}}{\text{True positives(t)} + \text{False positives(t)}} \tag{1}$$

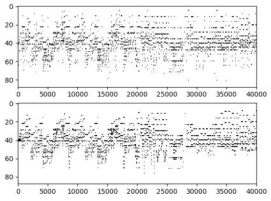
$$\text{Recall (R)} = \sum_{t=0}^{N} \frac{\text{True positives(t)}}{\text{True positives(t)} + \text{False negatives(t)}} \tag{2}$$

$$\text{Accuracy (A)} = \sum_{t=0}^{N} \frac{\text{True positives(t)}}{\text{True positives(t)} + \text{False positives(t)} + \text{False negatives(t)}} \tag{3}$$

$$\text{F-measure (F)} = \frac{2PR}{P+R} \tag{4}$$

where t is the tth frame in the data input.

Figure 3: Top: Plot of Predictions for one test sample, Bottom: Plot of Ground-Truth Label (x-axis: frame, y-axis: pitch)



For our parameter study, first we increased dropout from 0.2 to see its effect on the evaluation on the test set:

Model	Layers	Units	Optimization	Dropout	F-measure	Accuracy
DNN	3	256	Adam	0.2	67.1	50.5
DNN	3	256	Adam	0.3	65.5	48.7
DNN	3	256	Adam	0.4	65.6	48.8

Another of our experiments was seeing the effect of different optimizers on the evaluation of the test set:

Model	Layers	Units	Optimization	Dropout	F-measure	Accuracy
DNN	3	256	Adam	0.2	69.1	52.7
DNN	3	256	Adagrad	0.2	64.5	47.6
DNN	3	256	AdaMax	0.2	69.2	52.9
DNN	3	256	RMSProp	0.2	66.8	50.1

Morin's dataset consisted only of the MUS subset of the MAPS dataset. We wanted to see the effect on accuracy rating of adding audio recordings of isolated monophonic notes (ISOL set) from the MAPS dataset:

Model	Layers	Units	Optimization	Dropout	F-measure	Accuracy
DNN	3	256	Adam	0.2	67.1	50.5
DNN	3	256	Adam	0.2	0	0

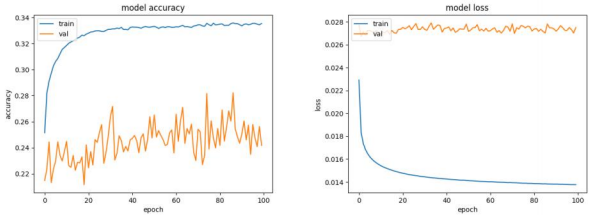
We also increased the number of fully connected layers to see if we can increase performance:

Model	Layers	Units	Optimization	Dropout	F-measure	Accuracy
DNN	3	256	Adam	0.2	67.1	50.5
DNN	4	256	Adam	0.2	68.2	53.7
DNN	5	256	Adam	0.2	68.3	54.3
DNN	6	256	Adam	0.2	68.1	55.1
DNN	7	256	Adam	0.2	65.6	60.2

Lastly, we experimented with different learning rates on the Adam optimizer:

Model	Layers	Units	Optimization	Dropout	learning rate	F-measure	Accuracy
DNN	3	256	Adam	0.2	0.001	67.1	50.5
DNN	4	256	Adam	0.2	0.0005	68.1	52.2
DNN	5	256	Adam	0.2	0.0004	68.4	53.6
DNN	6	256	Adam	0.2	0.0002	69.1	55.1
DNN	7	256	Adam	0.2	0.0001	69.2	60.2

We received the following accuracy and loss curves for both training and validation sets on the baseline model, which is the 3-layer, 256-unit DNN with Adam optimization and 0.2 dropout:



6 Conclusion/Future Work

We found that the Adamax optimization got us better accuracy and a higher f-1 score. We know that AdaMax optimization is best used for sparsely updated paramaters, such as in word embeddings. Since our model is a DNN, and word embeddings are generally trained using LSTM, we think that the AdaMax optimizer would work best in Morin's LSTM implementation for AMT. This leads us to possible hyperparameter tuning in Morin's LSTM architecture as future work.

As would be expected, accuracy improved with an increase in the number of hidden layers in our network.

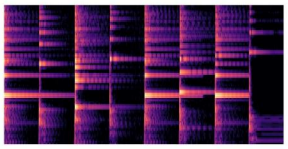
Since Morin's work showed that accuracy for his baseline model plateaus around 48% accuracy, our accuracy curve, which given the limited epochs over which the model was trained, seems to portend that our training and validation accuracy will not converge over time. This suggests underfitting. This could be because that our training set consists solely of software-generated piano music pieces, whereas our validation and test sets consisted of only recordings of real piano-played music pieces. Thus, our training and validation/test sets are not coming from the same distribution. In future work, we could improve performance by adding more real piano music pieces into the training set.

Along a similar note, adding the ISOL set to our training set dramatically decreased the accuracy of our predictions on both the validation and test sets. This can best be explained by the fact that our validation and training sets did not include any pieces from the ISOL set.

Our loss curve for the base line model suggests that for the Adam optimizer, we have a high learning rate, and not a good learning rate. As our learning rate study showed, the default learning rate for the Adam optimizer is too high, but for lower learning rates, we get improvements to our prediction accuracy.

Sigtia's research in [8] showed that DNNs are good classifiers for stationary data such as images, but are not designed for sequential data like audio. This is why in the future, we would like to try to improve the performance of our DNN by using spectrograms of the Constant Q Transform, an example of which is shown in Figure 4, of a raw audio file as input to the neural network rather than using matrix representation of the CQT as we did in our project. If with this approach, we still don't get performance close to that of CNNs, we would like to run a parameter study on the CNN architecture to improve accuracy.

Figure 4: Time-Frequency Plot of CQ Transform of a WAV input (x-axis is time, y-axis is frequency)



7 Contributions

Maddie Wang contributed to the editing of the training part of the python scripts originally written by Morin [3]. Susan Chang contributed to the editing of the preprocessing part of the python scripts originally written by Morin [3].

References

[1] Bereket, M. & Shi, K. (2017) An AI Approach to Automatic Natural Music Transcription. Stanford, CA: Stanford University.

[2] Chollet, F. & et. al. (2016) Keras. `https://keras.io`.

[3] González Morín, D. (2017) Deep Neural Networks for Piano Music Transcription. *GitHub repository* `https://github.com/diegomorin8/Deep-Neural-Networks-for-Piano-Music-Transcription`

[4] Li, L & Ni, I. & Yang, L. (2017) Music Transcription Using Deep Learning. Stanford, CA: Stanford University.

[5] Liu, B. & Shao, L. & Wu, X. (2017) Automatic Melody Transcription. Stanford, CA: Stanford University

[6] R. Badeau, V. Emiya & David, B. (to be published) Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle. *IEEE Transactions on Audio, Speech and Language Processing*

[7] Schorkhuber, C. & Klapuri1, A. (2010) Constant-Q Transform Toolbox for Music Processing.

[8] Sigtia, S. & Benetos, E., and Dixon, S. (2016) An End-to-End Neural Network for Polyphonic Piano Music Transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* **24**(5):927–939.

[9] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.