TOWARDS DNN-HUMAN REAL-TIME MUSIC IMPROVISATION

Christodoulos Benetatos
University of Rochester
c.benetatos@rochester.edu

ABSTRACT

The purpose of this project, is to explore the idea of human performers interacting in real-time with a neural network, to produce music. For this purpose, we created a dataset of two-part midi musical pieces (duets) from Bachs music, and trained two different neural network architectures, to predict the next musical note, given the past notes of both parts. The two different approaches that were used, are a) a RNN architecture using LSTM cells, and b) a casual dilated convolutional network. Even though the traditional way for modeling sequences is using recurrent networks, the latter and other non recurrent approaches, seem to give similar and better results, with the additional advantage of significant decrease in computation time.

1. INTRODUCTION

The last years we have witnessed a large amount of research on using neural networks for generation purposes and not just classification. Creating algorithms that can generate original content and create art, is considered a very difficult task, and in the future it may work as an important tool to enhance human creativity. In the field of visual arts we have some very impressive examples of creativity. In [4], they used CNNs to change the style and texture of an image, while maintaining the original content (style transferring). Another example are the generative adversarial networks GAN [5] which achieved of generating human faces that do not exist in reality [7].

Regarding music now, there is a long history of attempts of generative systems, using many techniques such as HMMs, or neural networks. Recently music generation has become is a very hot topic due to the applications it can be used. For example, a lot of companies want to generate copyright free music to reduce their costs. Many researchers tried to address this problem, and the most common approach is work on a semantics level (notes) and treat music generation as a natural language processing problem. In this way, character level recurrent neural networks (RNNs) can be used directly, by considering notes as characters. Some of the most impressive models of this kind come from Google's Magenta project which was created in 2016 [1]. However most of them, either generate music in offline fashion (i.e bidirectional RNNs), or in online, but in solo configuration, without incorporating humans in the generation chain. In similar systems to ours, like AI Duets [1], or the *Continuator* [8], a human performer and a computer are interacting, however not at the same time, but in a call and response configuration, and this is the basic difference with our approach.

This paper is organized as follows: In section 1 we present the dataset we created, in section 3 we focus on describing the architecture of the system, while in section 4 we present our results. Finally in section 5, we make our conclusions and defining the future directions.

2. DATASET

2.1 Data collection

For the purposes of the project we needed duet pieces, in musicXML or MIDI form, where each of the parts is monophonic. The last requirement is used to simplify the problem, it however limits our options regarding the dataset. For example we could consider piano pieces as duets (right and left hand), however rarely these parts are monophonic. An idea was to use Bachs Chorales, which are 371 short 4- part choral compositions (soprano, alto, tenor, bass), and they are characterized by some useful properties. Each part is monophonic, and each possible pair of parts sounds good, independently of the others. In this way, as shown in Figure 1, we can extract 12 different duets, from each of the chorales. Using the same rationale, we used also one other kind of short compositions by Bach, Inventions, which are 15 short two-part keyboard pieces, where each part again is monophonic.

For the chorales, we used the corpus in music21 library, while for inventions we downloaded the musicXML files from MuseScore . We had to parse them, organize them in duets, convert the notes in the proper representation for the neural networks. We only parsed 346 from the 371 chorales of the music21 corpus, and all of the inventions. For each of the duets we transposed them in all the 12 musical keys. This helps the system to be key invariant. Another way to achieve that is by transposing all the pieces in the same key, i.e C major. While this solution works, it limits the musician to play melodies only in that key.

The result is that for each of the four-part chorales, there are 12 different permutations of two-part pieces, and 12 transpositions so the total number of duets is $12*12*371 = 49824$. For the experiments on the paper we did not use inventions, and from chorales we used only the permutations

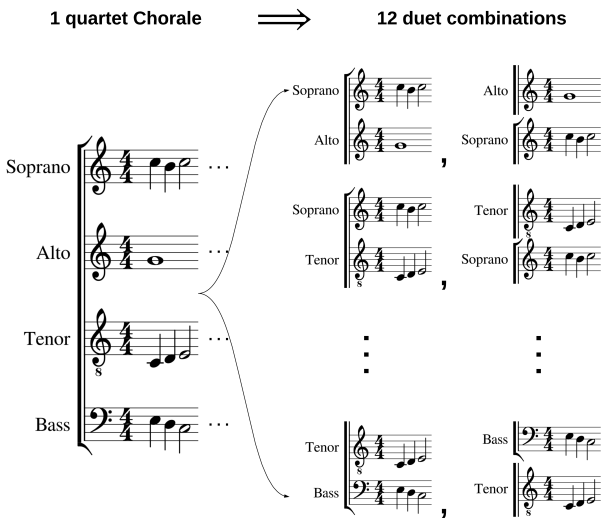

Figure 1: A chorale example consisting of four monophonic parts. Taking all possible combinations of 2 parts we can create 12 duet pieces.

of the two more musical interesting parts (soprano, bass). The reason we reduced the dataset is due to the limited time to train the models.

2.2 Data Representation

2.2.1 Pitch, Duration, Articulation

Each of the notes in a melody, has different relative time duration, indicated by specific symbol. Omitting this information and encoding only the pitch of the note, the result will not be able to model rhythm structures. Additionally, we want the DNN to calculate the next note on constant time rate. We define the duration of a sixteenth note, as the minimum time step. This means that notes with relative duration less than a sixteenth should be removed. However this is not a problem in our dataset since in chorales the durations never go beyond that limit. For the the notes of duration larger than that, one common approach is to replace the remaining sixteenths of a note, with a hold/tie symbol, i.e "1". Trying this approach we were not able to train the DNN. The reason is that, since in chorales, the most common note duration is either quarter or eighth , the sequence of notes contained mostly 1 symbols, and some sparse MIDI numbers for the pitch (fig2 top). This had as a result, the DNN to learn to predict mostly 1 symbols without any information about the pitch. The solution that worked in this project, was to augment the MIDI symbol of each note with the information about the articulation i.e hit 1, or hold 0 (fig2 bottom). In this way, we reduce the imbalance of the classes, and also even if the DNN starts predicting hold symbols without having any hit before, we still have the information of the pitch.

2.2.2 Embeddings

After defining the representation of the notes, the next step is to create the embeddings for the DNNs. We use an embedding layer to enforce our system to encode the relations

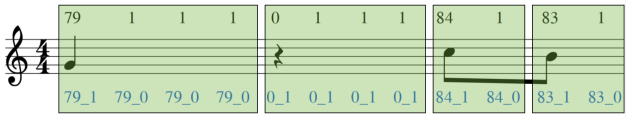

Figure 2: Representation of the notes, containing pitch (MIDI) and articulation information (hit/hold). Instead of using the same "hold" symbol for all pitches (top line), we use different for each note (bottom line).

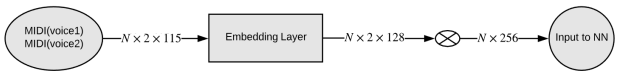

Figure 3: The embedding layer was jointly trained with the neural network. N is the note sequence length, 2 is the number of parts, 115 is our note vocabulary size and 128 the embeddings size.

between each note, in a way similar to how embeddings are used for word encoding. "Athens" is more similar to "Greece" than to "China", and in the same way, note C1 more related to C2 than A#1. After examining the dataset, we observed that only 57 MIDI numbers were used (30 - 86) so the total note classes are $57 * 2 + 1 = 115$ (each note has the hit and hold version, plus one symbol for rest). We used embedding size of 128, and concatenation of the embeddings for each of the two parts (Figure 3. At each timestep the output of the DNNs will be a vector of size 1x115 with the probability distribution over each class (note).

3. MODEL ARCHITECTURE

3.1 Overall Structure

We can see the overall structure of the system in Figure 4. In real life music improvisations, each performer takes actions based not only on his previous actions but also on the past actions of the other performers. In a similar way, the DNN predicts the next note based on both its own and human's previous generated notes. We use a sliding window over the previous notes of both parts and vary the size of the sliding window from 16 to 64 depending on the type of DNN

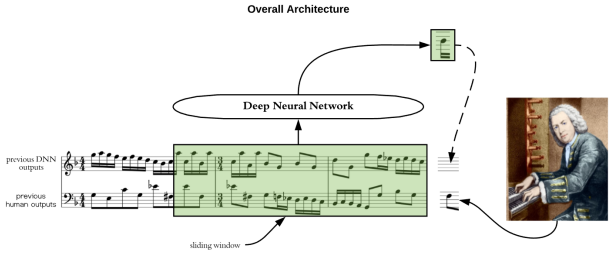

Figure 4: A human performer and a DNN interacting to generate a two-part music melody. We train the DNN to predict the next note of its corresponding part, based on the past generated notes of both parts.

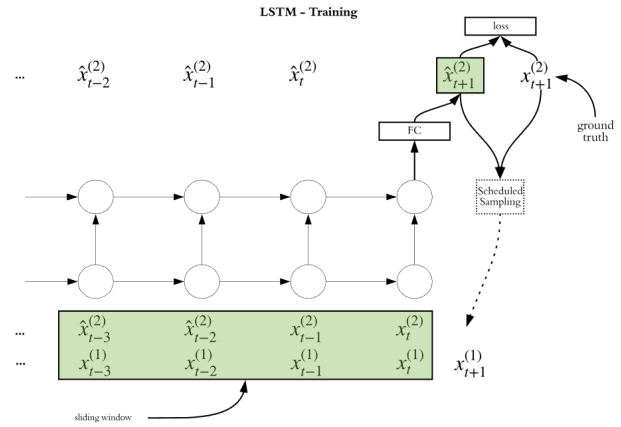

Figure 5: The architecture of the LSTM network, and the scheduled sampling schema.

More specifically, we are using two different models. First a classic NLP approach for sequence modeling, recurrent neural networks (RNN) with long short term memory (LSTM) cells, and a causal dilated convolutional neural network (cd-CNN). Suppose that $\boldsymbol{X^{(1)}} = \left[x_1^{(1)}, x_2^{(1)}, \ldots, x_t^{(1)}\right]^T$ and $\boldsymbol{X^{(2)}} = \left[x_1^{(2)}, x_2^{(2)}, \ldots, x_t^{(2)}\right]^T$, are the first and second parts of a sliding window of size t from a training sample, and lets consider that the DNN is responsible for predicting the note of the second part. The ground truth next note is $x_{t+1}^{(2)}$, while the predicted by the model is $\hat{x}_{t+1}^{(2)}$. We use cross categorical entropy loss function, since we treat the note prediction model, as a classification one, where classes are each of the possible notes. Finally, during, training, instead of just using *teacher forcing*, we experiment with a technique called *scheduled sampling* [3]. In *teacher forcing*, at each timestep, we feed back the ground truth $x_{t+1}^{(2)}$, to be conditioned for the next prediction, while in scheduled sampling we feed back the predicted output $\hat{x}_{t+1}^{(2)}$, with a probability that changes through training time (curriculum learning).

3.2 LSTM

This model of recurrent NNs [6], is used to solve the problem of vanishing and exploding gradients that simple RNNs have, and they are capable of learning long-term dependencies. They can be used in many schemas, depending on the problem, such as one-to-many (image captioning), or many-to-many (sequence translation). In our model we use the many-to-one architecture 5, and we train using truncated backpropagation through time with $k_1 = 1$ and $k_2 = 16$ [9], meaning that, for every 1 timestep we do backpropagation through time for 16 steps (size of the sliding window). The parameters we tried are 2 stacked layers and 512 hidden units. A fully connected layer is used to convert the lstm output size from 512 to 115.

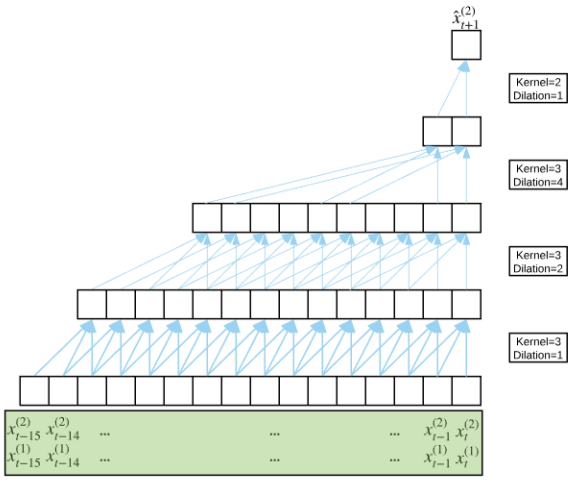

Figure 6: A dilated causual CNN, for predicting the next note based of a sequence. The number of layers depend on the size of the window, and we gradually increase the dilation factor exponentially, except for the last layer.

3.3 cd-CNN

As we mentioned before, there is a trend in replacing RNNs with CNNs for sequence modeling [2]. Even though CNNs are feed forward models without internal memory cells to capture temporal information, we can take advantage of their computational efficiency and do calculations using long history windows. For example, cd-CNNs using a sliding window of 64 timesteps, train at the same time as RNNs when using window of 16 timesteps. The term causual means that the prediction at each timestep t is the convolutional result of past only elements, while the dilation is a technique to increase the receptive field of each stacked CNN layers, by using a kernel that skips timesteps 6. If dilation = 1 then we have the classic convolution.

These cd-CNN blocks are used in the core of a very popular architecture, WaveNet [10], as well as in temporal convolutional networks (TCN) [2]. In our implementation we used a simplified version of the above, without gated convolutions and residual blocks. We do not use padding at each layer, so the sequence size decreases layer by layer, until it becomes one. Also we used $stride = 1$ instead of 2 in the TCN paper. We assume that the input window size will always be a power of 2, such as $N = 2^i$. In this case, for $kernel = 3$, the total number of layers will be $i-1$. We use exponential increased dilation factor for the first $i - 2$ layers, while for the last, we always use $dilation = 1$ and $kernel = 2$.

We built this network in a similar way with the LSTM approach, so it would be easy to treat them the same way. For example, we can easily apply as before, teacher forcing, and schedule sampling schemes, even if these are techniques developed for recurrent networks.

4. RESULTS

Due to time limitations, we did not let the DNNs to train for many epochs, and we did not fine tune the model parameters, however we were able to get some very interesting results with both the architectures. First of all, we split our dataset in training (80%), validation (10%) and testing (10%) and we were able to to achieve 91.3% accuracy in the testing dataset, using with cd-CNN with $dropout = 0.2$ and $N = 32$. In Figure 7a, we can see the loss values during the training of cd-CNN.

About LSTM, the best accuracy score that we achieved with LSTM was 88.9%, using $dropout = 0.2$ and $N = 16$, however we had some difficulties during training due to overfitting. In Figure 7b, we can see the validation loss increasing, without affecting the accuracy. However we expect to see accuracy decreasing when training for more epochs. However, classification accuracy is not a very meaningful metric for music generation, since the wrongly predicted notes may still make sense in a musical way.

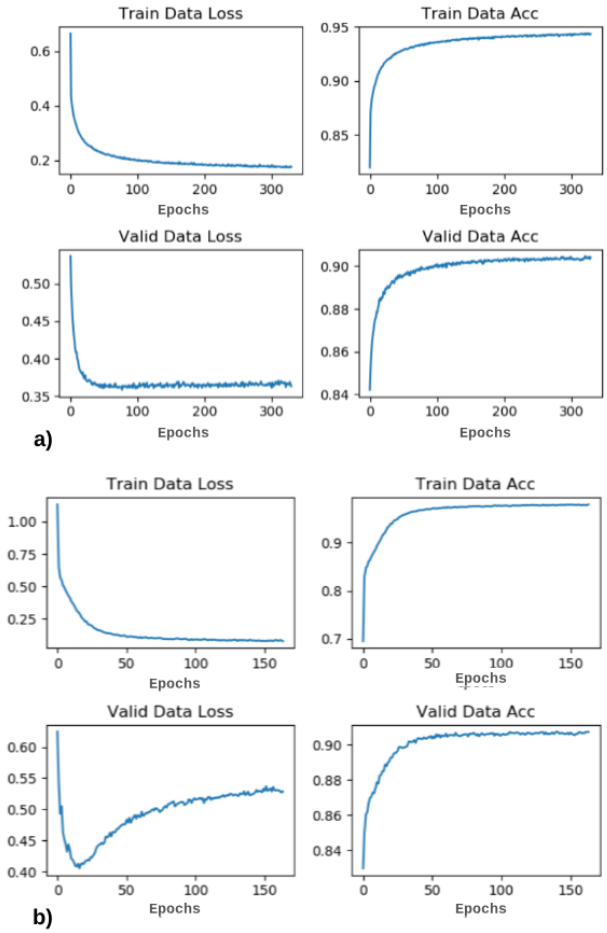

Figure 7: Loss and accuracy on training and validation dataset for a) cd-CNN and b) LSTM.

We should mention that we did the above evaluation using teacher forcing. Meaning that we did not let the DNNs generate music freely, but instead, after each prediction, we force feed the ground truth note instead of the predicted. In a real-life scenario, DNNs condition their output on their previous predictions. To test them in this way, first we initialize the sliding window with the beginning of chorales from the testing dataset, and then let them freely generate music.

In all the presented examples, the generated part is the upper one (blue notes), and also the reference melody part is from the testing dataset. First we select a chorale and use the LSTM network to generate the Soprano part given the Bass line (Figure 8a and then the opposite in 8b. Then, in Figure 8c, we can see the generation result of the cd-CNN network. In the case of cd-CNN,it is interesting to notice the very nice major cadence in a D minor key, the rests, and how nicely it starts again as a response to the theme played by the other voice.

Also, we tried as the reference part, a melody of different style from Chorales, the theme from the Bach's fugue No.2. The cd-CNN generated a response very pleasant to listen Figure (9), with minor mistakes. On the other hand, the LSTM did not give good results as it could not even follow the key.

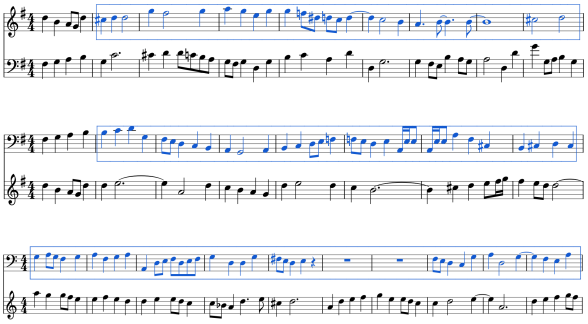

Figure 8: a), b) Generation example of the LSTM and c) cd-CNN networks. The first bars are the initialization, and after that, the upper part (blue) is generated. In a) the bottom (reference) part is a basso voice, and in b) is the soprano voice from the same chorale. For c) we used the Soprano of another chorale as reference melody.

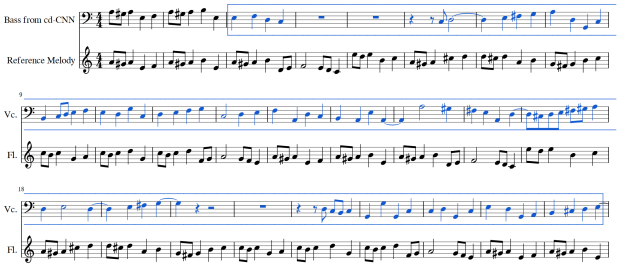

Figure 9: The cd-CNN model was able to generate meaningful melodies even when we tried a reference melody in a style different than Chorales.

Listening to these and many more results lead us to the following observations:

- Most of the time, both DNNs were choosing notes belonging in the right key (the key defined by the hu-

man reference melody, but cd-CNN was more stable in this aspect.

- Both of them were able to select the right range for the melody. When the reference melody was in the bass range, then the predicted output was in the soprano, and vice versa.
- The parts generated from cd-CNN where better harmonically tied with the reference melody than the LSTM, but the melodies from LSTM, where more interesting and harmonically rich.
- Changing the sampling temperature, both models could not follow the harmony of the reference melody, but again the generated melodies where very interesting, with many key modulations and more complex rhythmic patterns.

Finally, the running time for the prediction of one note (sixteenth) was less than $10ms$ for both DNNs, using an average CPU. This means that designing an application for real-time interaction with a human performer is feasible, and we can achieve high BPM speeds.

5. CONCLUSION - FUTURE WORK

In this project we designed an music generation system that is able to interact with a reference melody and generate musically pleasant two-part melodies. In our tests the reference part was predefined, however in the future we plan to implement a real-time application where this part will be created instantly by a human performer using a MIDI keyboard. Of course, the dynamics of this system will be a bit different from our offline experiments, since we expect the human to be affected by the DNN's output, which is a kind of acoustic feedback that we did not considered. As for the model architectures, the cd-CNN performed clearly better in both objective (loss, accuracy) and subjective (music quality) results. However, maybe a reason for that was that we trained the cd-CNN for more epochs, and also we did not experiment with a lot of parameters and configurations during training, which is something that we have to do in the future. Finally, more experiments should be done using scheduled sampling, since training models with that, resulted an unstable loss function and slow convergence.

6. REFERENCES

[1] Magenta Project. `https://magenta.tensorflow.org/`. Accessed: November 2018.

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[3] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

[4] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[8] Francois Pachet. The continuator: Musical interaction with style. *Journal of New Music Research*, 32(3):333–341, 2003.

[9] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, Ontario, Canada, 2013.

[10] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.