L45 - Mozart was Secretly a Neural Algorithmic Reasoner: Music Generation with Graph Neural Networks

Michitatsu Sato[*]
University of Cambridge
ms2899@cam.ac.uk

Bruce Cheung[*]
University of Cambridge
kcc41@cam.ac.uk

Ali Keramatipour[*]
University of Cambridge
ak2427@cam.ac.uk

Abstract

We propose a novel approach of generating music by creating representations of Tonnetz with Graph Neural Networks (GNN). Tonnetz is a lattice graph that captures the structural information of music through mapping tonal relations between notes. We evaluate several model architectures based on the accuracy of a regression task and the quality of generated music. Rather than just considering music as a sequence of sounds, we show that GNNs encapsulate both temporal and structural information to emulate musical patterns when generating music. This helps using an encoder-decoder structure of Graph U-Net to learn musical patterns inspired by Mozart.

Statement of contribution

We, Michitatsu Sato, Bruce Cheung, and Ali Keramatipour of the University of Cambridge, jointly declare that our work towards this project has been executed as follows:

- **Michitatsu Sato** contributed by implementing and experimenting with Baseline models, RNN+GCN based models, and ST-GNN model. In addition, he implemented the piano roll visualizer and wrote the corresponding parts of the report, including the Introduction section.
- **Bruce Cheung** contributed by creating the data processing pipeline, converting the Tonnetz structure into a graph adjacency matrix, testing baseline models, implementing the MPNN model, and writing the music generation functions. For all of the above tasks, he wrote the corresponding parts of the report.
- **Ali Keramatipour** contributed by implementing the pipelines to convert MIDI to Tensor and Tensor to MIDI, and a piano key visualiser to track the movements of hands. Implemented the Graph U-net encoder-decoder model for encoding the initial keys and a generalised distance-based loss function.

We independently submit identical copies of this paper, certifying this statement to be correct.

GitHub repository with commit log

The companion source code for our project may be found at https://github.com/Vinnesta/l45-music-gnn.

1 Introduction

The generation of music using machine learning and deep learning is a popular research topic [1]. One of the most well-known ways of formulating the task of music generation is with autoregression.

[*]Equal contribution.

Michitatsu Sato & Bruce Cheung & Ali Keramatipour, L45 - Mozart was Secretly a Neural Algorithmic Reasoner: Music Generation with Graph Neural Networks. *Representation Learning on Graphs and Networks (L45), University of Cambridge*.

By regarding music as a temporal sequence of data, it is possible to apply sequence-related methods like Hidden Markov Models (HMM) [2], Recurrent Neural Networks (RNN) [3], and Transformers [4].

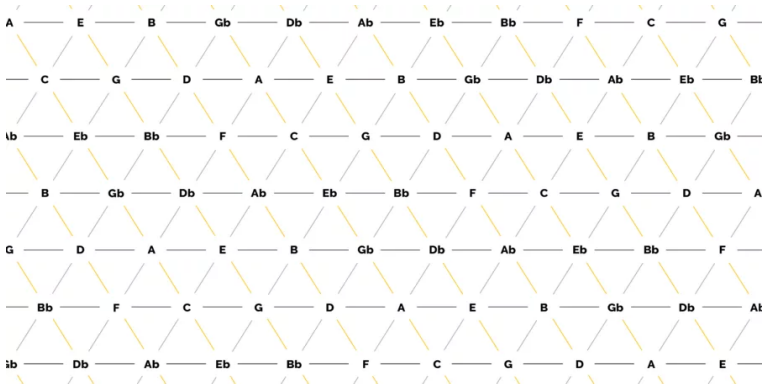


Figure 1: Tonnetz graph structure [5]

A unique approach proposed by Chuan and Herremans in [6] attempted to capture and use not only temporal (i.e. sequential) information, but also the structural information of music. Their research encoded the spatial relationship of each musical note using Tonnetz (Fig. 1), a graph representation that maps the tonal landscape of music [5]. Musical compositions were first converted to Tonnetz images and encoded using Convolutional Neural Networks (CNN), then the representations were fed into an LSTM sequence data processing encoder (Fig. 2).

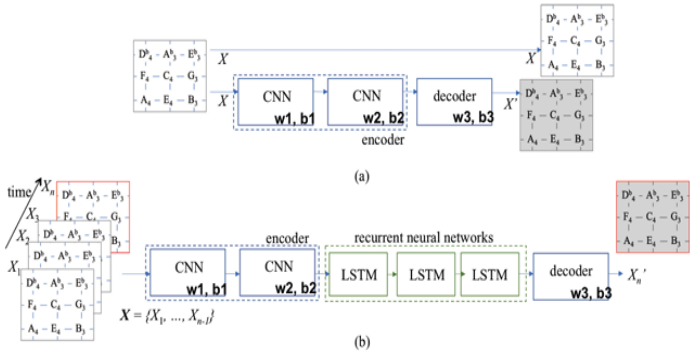


Figure 2: The pipeline of the CNN-LSTM-based music generation from [6]

This research was our inspiration for using Tonnetz as the graph structure for generating music. By encoding spatial information contained in the Tonnetz structure, we expect that the generation of the music will be more effective compared to just treating music as a sequence of sounds. At the same time, the graph structure of Tonnetz lends itself to a GNN representation more neatly than using a CNN encoder. Our research aims to investigate whether GNNs are effective at representing music, and if so, to construct a music generation deep learning model based on a GNN approach.

We find that a Message Passing Neural Network (MPNN) [7] architecture has the highest accuracy in predicting the next timestep of a song, and we use the trained model to generate piano pieces that contain some elements of musical patterns.

2 Methodology

In this section, we provide detailed information about our experiment setting. First, we conduct experiments on simple models to determine a baseline and inform us of our next approaches. Then, we run several experiments on model architectures designed based on the observations we obtained from

the baseline experiments. Before explaining the experiment setups, we first describe the common processes to all the experiments.

2.1 Music datasets

The datasets we use for our experiments consist of the MAESTRO collection of modern piano performances [8, 9] as well as the repository of Mozart piano sonatas from piano-midi.de [10], henceforth referred to as the "Mozart dataset".

The MAESTRO collection contains 1,276 performances with a total length of 198.7 hours. Due to computational limitations, a subset of 10 songs is used in our preliminary experiments. A larger subset of 50 songs is used in training the most performant architecture, the MPNN model (see Section 3.4). In contrast, the Mozart dataset consists of 21 pieces that have a total length of 2.4 hours, therefore the entire set is used in the relevant experiments.

In the experiments comparing the performance of various model architectures, only the MAESTRO dataset is used for training and evaluation. The Mozart dataset was introduced for the later experiments involving music generation in order to generate songs that sound "less busy", i.e. have a lower number of notes per second. We note that the MAESTRO dataset is comprised primarily of postmodern piano pieces which are busier than classical music, such as those composed by Mozart, with an average of 9.6 notes per second as compared to 6.2 notes per second in the Mozart dataset.

2.2 Data conversion and processing

In both the MAESTRO and Mozart datasets, songs are provided in MIDI format, a technical standard that contains messages about how an instrument should be used. The main type of message we need is of type *note_on*, which contains three important detail that can be encoded into a tuple, *(note, velocity, time)*. The type *note_off* can be represented with a *note_on* with no velocity. We use two approaches to convert data.

The first approach converts data into tensors of shape $(num\_notes, time\_steps)$, where num_notes is 128 according to the MIDI specification, and $time\_steps$ is the total number of "ticks" in the MIDI files. Each note is mapped to a graph node, with timestep values (velocities) as node features. A tick is the smallest unit of time in the MIDI format, typically corresponding to about 1-2 milliseconds of a song. This granularity resulted in enormous tensors that contained mostly repeated information; therefore, we compressed the tensors by merging multiple ticks into one. The compression ratios are chosen so that short notes can be represented in the tensors. After compression, each tick corresponds to approximately 50-150 milliseconds, depending on the tempo of the song, which shrank the data sizes considerably.

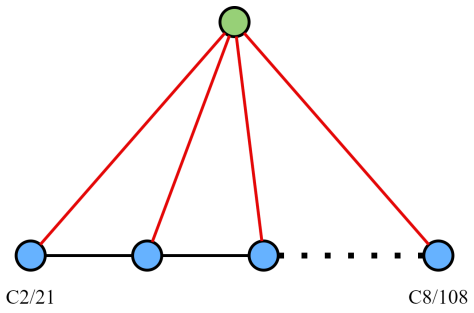


Figure 3: A universal node (vertex) connected to all vertices of a path

In the second approach 3, which we will call the UN (universal-node) approach, alongside notes' nodes, we consider a universal node such that its features represent the timesteps. Using this approach, we completely encode the data without any compression. However, since this universal node has a different purpose, it is connected to others using a different edge type (feature). The notes are represented as a path grid from the lowest to the highest note.

In addition, the range of notes used on pianos is only between 22 and 108. In addition, considering only notes 33 to 96 removes just 0.02% of all used notes. By only considering these notes, we have halved the number of graph nodes.

2.2.1 Tensor representation

When a piano key is pressed, a hammer strikes the piano wires to make a sound which reverberates until the key is released (unless the sustain pedal is activated - we ignore this possibility for simplicity of explanation). MIDI files record the entire duration in which a key is held down, so in our conversion to tensors of binary values, we insert 1s into the timesteps for these durations, and 0 when the key is not pressed.

	Timestep (t)									
Representation	0	1	2	3	4	5	6	7	8	9
MIDI	0	1	1	0	0	1	1	1	1	0
OPO	0	1	0	0	0	1	0	0	0	0

Figure 4: Example of the same sequence of music represented in MIDI and On-Press-Only (OPO) formats: a key is pressed at t=1 and t=5, and is held down for 2 and 4 timesteps respectively.

However, this conversion results in long sequences of 1s for keys which are held down over long periods of time. In our initial experiments, we find that the models learn to simply repeat the very last timestep of the input, as this degenerate strategy is easy to learn. As such, we devise an alternative representation of the data which we call the On-Press-Only (OPO) format. In the OPO format, a 1 value is inserted only at the timestep which the key is pressed. Fig. 4 shows an example comparing the two formats. While the OPO format drops information about note sustain, which is musically important, it simplifies the learning process and results in much better model performance.

In the UN approach, the universal node's features contain timesteps in which the pianist's hands move. The i-th feature of a node represents the velocity of its note during the i-th feature of the universal node. This approach also prevents too many consecutive non-zero values from appearing.

2.3 Representing Tonnetz as a graph

As shown in Fig. 1, Tonnetz is a triangular grid graph where every node represents a musical note on the twelve-tone chromatic scale, and each edge represents the musical relationship between two notes, such as major thirds and perfect fifths. By assigning a pitch register to every note, as was proposed in [5], the graph is rolled up into a cylindrical lattice where every node has a degree of six, aside from the nodes at the ends of the cylinder. As described in the previous section, we choose to use the part of the Tonnetz corresponding to the notes from A1 to C7, which we convert to a graph adjacency matrix. The edges are encoded as attributes using one-hot vectors for each of the six types of relationships (i.e. six edges) between notes.

For the notes in our chosen range, it takes up to 11 hops to connect any two nodes, therefore our model architectures require multiple message passing layers to disseminate information across the whole graph. However, the high number of layers increases the risk of oversmoothing, where node representations become too similar to each other. For this reason, we briefly experimented with using a fully connected graph, but adding connections between all 64 nodes results in more than 10 times the number of edges compared to the original graph, which led to impractical inference times given available compute resources. As such, we use the sparse graph for our experiments but we note that a dense graph may provide better theoretical results.

2.4 Experimental setup

Our experiments are formulated as an autoregression problem, where the model is provided inputs of timesteps $t_i, \ldots, t_{i+k}$ and predicts the subsequent timestep t_{i+k+1}. By taking a sliding window of size k across the processed data, we get input tensors that has shape (num_notes, k) and target labels which are vectors of dimension num_notes. k is a hyperparameter that determines the length of history shown to a model, and is set to 32 for all experiments except where noted.

The model prediction is passed through a sigmoid function to determine whether the next state of a specific note is pressed (1) or released (0). Accuracy is then measured by the number of timesteps where all predicted notes match the target label. The loss function of BCEWithLogitsLoss [11] is used in all experiments.

2.5 Baseline experiments

In this section, we describe the baseline experiments that we conduct to observe how the simple architecture performs on a music generation task. For the baseline models, we use a simple Multi-layer Perceptron (MLP), RNN [12], and Graph Convolution Network (GCN) [13]. The models are each trained for 10 epochs using the Adam optimiser. For the activation functions, ReLU is used.

1. **Simple MLP**: A three-layer MLP with one input layer, one hidden layer, and one output layer.
2. **Simple RNN**: Two RNN layers followed by three linear layers, with activations between the linear layers. Note that an embedding layer is not applied since we consider the binary data representation to already be the simplest representation.
3. **Simple GCN**: A linear input layer, a GCN layer, followed by two linear layers, with activations applied after input and hidden layers. Unlike the above two models, the GCN requires the Tonnetz structural information, therefore we provide the Tonnetz adjacency matrix. In this model architecture, the temporal information (i.e. fixed sized key-pressed history) for each note is used as the initial node representation of the Tonnetz graph.

2.6 GCN with RNN

As extensions of the baseline models, we construct models which combine the GCN and RNN layers introduced in the above subsection. We expect that the GCN part of the model will capture structural information about the input music sequence data, while the RNN part will capture the temporal information of the data. The following are the proposed models.

1. **2RNN + 1GCN**: This model stacks a GCN layer after two RNN layers. We expect that the RNN layers will first encode the temporal information about each musical note.
2. **1GCN + 2RNN**: This model is the opposite of the model above, with the first layer being a GCN which then passes the output to two RNN layers. By considering the music generation process a traditional autoregression task, it is more natural to put RNN layers after the GCN as the GCN can act as an information enrichment layer, similar to the role of an embedding layer in natural language processing, and RNN layers as sequence processors.

2.7 Spatio Temporal Graph Neural Networks

Since it is important for a model to capture both the temporal information and structural (spatial) information of the input music data, the "Spatio Temporal Graph Neural Network" (ST-GNN) [14] architecture is a natural fit for our task. At its core, the ST-GNN can be considered as a function or layer which deals with time-varying node features [15], which are the note state histories in our GNN models. For the implementation, we choose "GConvLSTM" [16] (Figure 6) as the ST-GNN layer of the model. The model used in this experiment is constructed with a single GConvLSTM layer followed by linear layers. The Chebyshev filter size K is set to 2.

2.8 Message Passing Neural Network (MPNN)

In the above models, the aggregation of neighbourhood feature information is done through convolutions, which assumes that the information from each neighbour is equally important. Moreover, the models are not equivariant to musical transpositions, i.e. where every note is shifted by the same number of semitones. Since musical patterns are independent of the actual note pitches, e.g. a song is recognisable even if it is transposed, tranposition equivariance is an important property to have.

The Message Passing Neural Network (MPNN) [7] architecture overcomes these limitations by using a learnt, shared function that passes messages between nodes. The learnt function allows the network to differentiate and weigh the importance of neighbouring nodes, while the sharing of the function across a regular graph makes the model equivariant to transpositions (except at the ends of the lattice where nodes have fewer neighbours). The MPNN layer we implement can be formulated as follows:

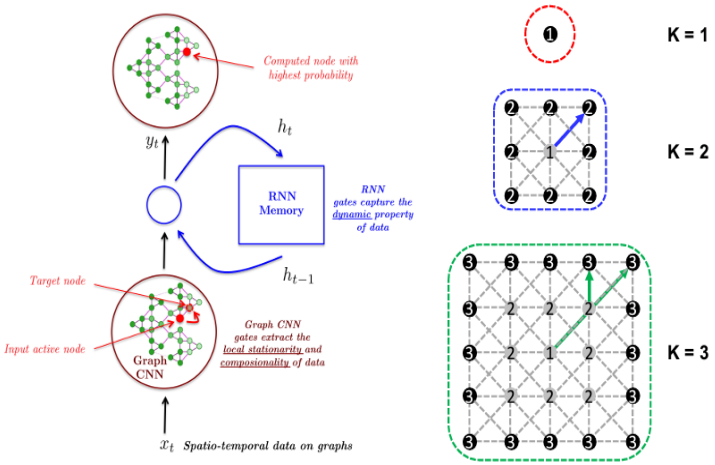


Figure 5: The architecture of the GConvLSTM and Chebyshev filter[16]

$$h_i^{\ell+1} = \phi\left(h_i^{\ell}, \bigoplus_{j \in N_i}\left(\psi\left(h_i^{\ell}, h_j^{\ell}, e_{ij}\right)\right)\right),$$

where ψ a two-layer MLP operating on the concatenation of the node and edge features h and e, $\bigoplus$ is the summation aggregation function, and ϕ is another two-layer MLP.

The full MPNN model consists of a two-layer MLP to encode the input, four MPNN layers, and a two-layer MLP to decode the graph representations. All of the hidden and embedding dimensions are set to 48.

2.9 Graph U-Net

To capture the pianist's musical pattern, we can use an encoder-decoder approach, which in this case is the Graph U-Net approach [17]. Similar approaches [18] have been used to generate music. Using the encoder-decoder idea, we can generate a 10-key piano, where the user is able to generate Mozart-like music. The decoder GNN is able to capture musical patterns inspired by Mozart. The encoder can be combined with training large amounts of data as it enables a way of effective compression.

Within this approach, a smaller graph with 88 + 1 nodes is considered alongside the UN-modelled graph with 10 + 1 nodes. The model's input is a window of size $k = 20$, which will be encoded into the smaller graph with the same window size. The encoder's and decoder's model is MPNN, as the UN graph contains different features for its edges.

3 Results and Evaluation

3.1 Experiment 1: Baseline models

In this section, we present the results of the baseline models. The final accuracy achieved by each model is presented in the following table (Table 1).

Model	Accuracy (%)
Simple MLP	44.80
Simple RNN	44.09
Simple GCN	15.25

Table 1: Next state prediction performance achieved by baseline models

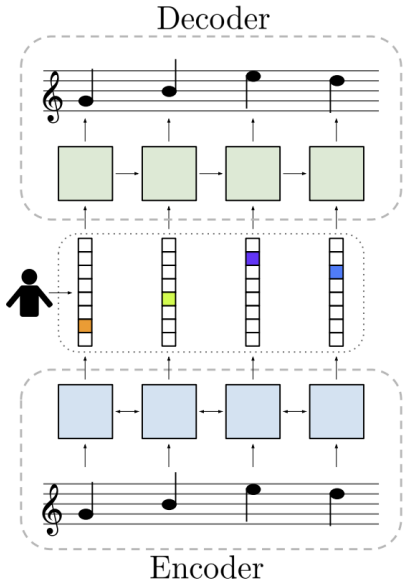


Figure 6: Encoder-decoder architecture for piano notes[18]

From the results, we observe that the simple MLP and simple RNN baseline models have adequate performance. While we expect the simple RNN model, which is usually used for sequence data processing and autoregression, to achieve decent results, it is surprising to observe that even the simplest architecture MLP can achieve similar performance. When comparing the simple MLP and simple GCN models, given that they have similar architectures aside from the input layer being replaced by a GCN layer, the results suggest that the GCN layer is less effective than an input linear layer. From this, we infer that the GCN layer without any modification is detrimental to this task.

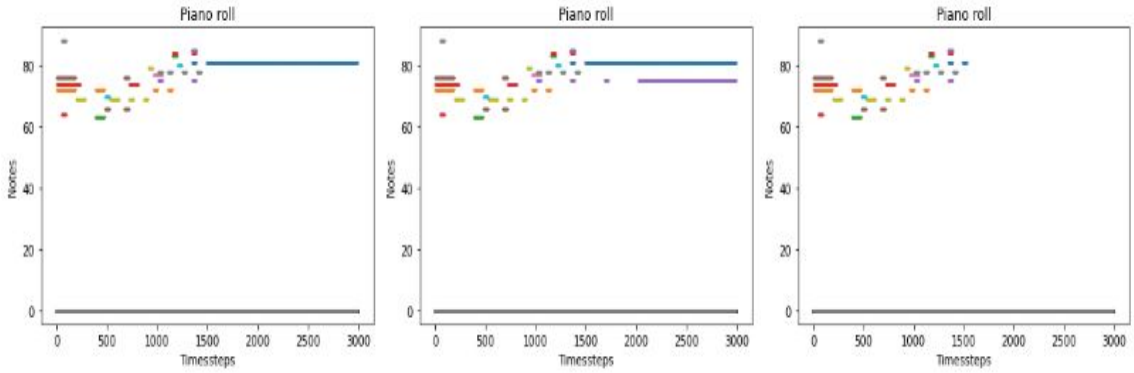


Figure 7: Music generated by each model with same seeds (first few steps of the music) in Piano roll view: From left, simple MLP, simple RNN, and simple GCN

However, by observing samples of generated music by 3 baseline models (Fig. 7), we cannot conclude that the model with the highest accuracy performance on the dataset is the best model for the music generation task. For example, looking at the generated music by simple MLP, it is clear that the model is simply extending the last note of the seed music until the end. Whereas, the simple RNN model is able to generate new sound (e.g. the purple line in the plot), and is not just extending the last note. Unfortunately, the simple GCN did not generate any music, as expected from its low performance which suggests that modification to the model architecture is needed in order to utilise the Tonnetz graph structure.

Based on these observations, we identified that the simple MLP is resorting to a degenerate strategy of repeating the values in the last timestep of the input data. Also, we note that accuracy of the task is just one of the factors in deciding a model's performance.

3.2 Experiment 2: GCN + RNN models

In this section, we evaluate the results of the models with a combination of GCN and RNN layers. The accuracy of each model on the given test dataset is given below (Table 2).

Model	Accuracy (%)
2RNN+1GCN	15.61
1GCN+2RNN	39.18

Table 2: Next state prediction performance achieved by GCN+RNN models

Based on the performance of the models, it is clear that the model which put the RNN layers at the end of the model architecture has better accuracy. As discussed in the methodology section 2, this implementation is more natural from a sequence data processing point of view. By comparing the results of the 1GCN+2RNN model with the simple GCN baseline, it is reasonable to infer that using a GCN layer for node information enrichment of the Tonnetz structure is a more effective way of applying graph convolutions.

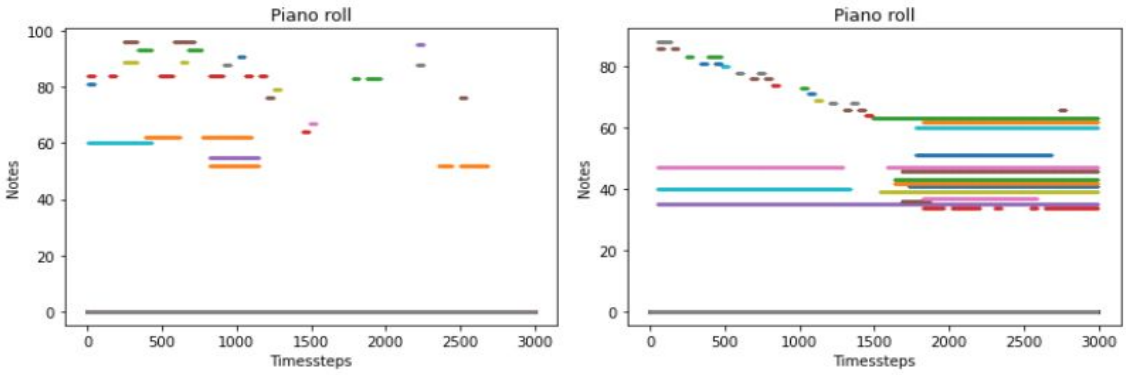


Figure 8: Music generated by each model in Piano roll view: From left, 2RNN+1GCN and 1GCN+2RNN

Looking at the generated music by each model, we can observe that the first model is generating very sparse and short sounds while the other model produced more dense and long sounds. We find that the second model is better as an autoregression model based on performance accuracy. The sparse and shorter generation means the model is biased toward outputting zeros, while dense and long sounds can be interpreted as the model biased towards the non-zeros output.

3.3 Experiment 3: ST-GNN model

Despite the complex architecture of the GConvLSTM layer which should be able to capture both temporal and structural information in the music data, the performance of the model is not as good as expected. The accuracy of the model on the test dataset is **39.63%** which is very similar to the accuracy obtained by the *1GCN+2RNN* model.

Observing the generated music (Figure 9), it looks to be an improvement over previous models. The produced sounds are not too dense or sparse compared to the given seeds (i.e. the first few timesteps). However, from the plot, we can observe that the model is very sensitive to the previous history of the music since the model only generates notes which had been played, e.g. the four split lines at the top of the plot. This suggests that the model struggles to generate new sounds which had not already been played in previous steps.

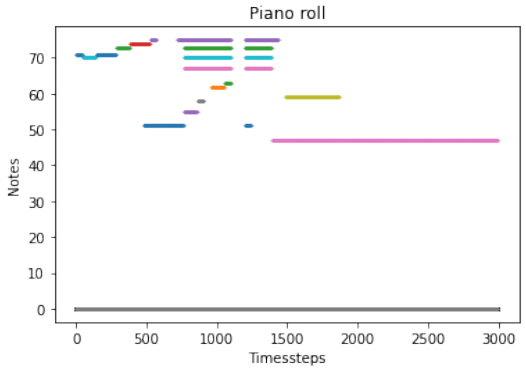


Figure 9: Music generated by ST-GNN model in Piano roll view

3.4 Experiment 4: MPNN model

The MPNN model achieved a test accuracy of **40.58%**, which exceeds the performance of the other models. Although the accuracy does not match the baseline's strategy of repeating the last timestep, the relatively higher accuracy and decreasing loss show that the model is learning to an extent. Since this model was the most performant, we continue with the MPNN architecture for the following experiments involving the proposed OPO data representation format.

3.5 Experiment 5: UN MPNN

In the UN approach 10, the MPNN model was used to predict the timestep. This yielded good results in predicting the time when the musician is pressing or releasing the next key. This provides us with a good method to simulate actual hand's movements without breaking time into different chunks.

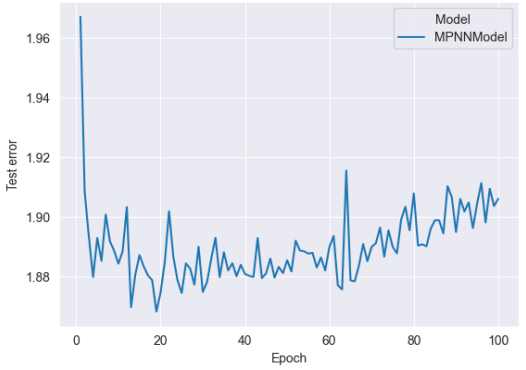


Figure 10: Test error on timestep prediction

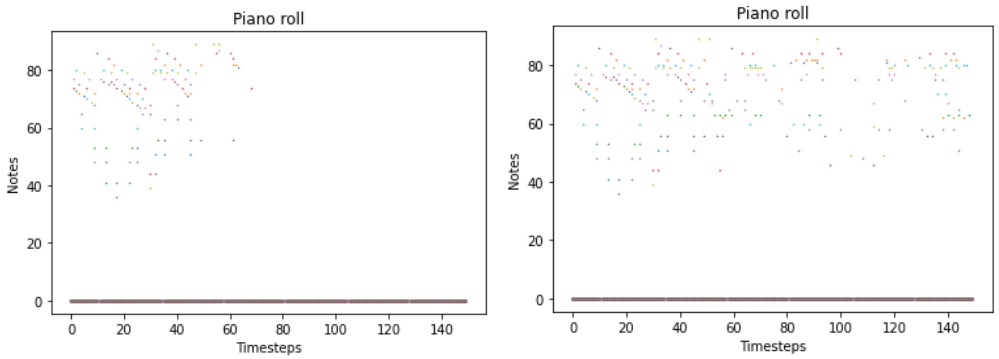
3.6 Experiment 6: OPO tensors

Due to the nature of the format, the OPO tensors contain many timesteps comprised entirely of zeros. Similar to the previous experiments, the baseline models devolved into a degenerate strategy of outputting vectors with all zeros, giving a baseline accuracy of 41.4% for the MAESTRO dataset and 31.7% for the Mozart dataset.

In contrast, the MPNN model achieved test accuracy of **42.9%** and **47.5%** for the MAESTRO and Mozart datasets. As discussed in Section 2.1, the Mozart dataset is far less busy and is likely a much easier dataset to train on, resulting in the MPNN model significantly outperforming the baseline. Note that the hyperparameter k, which is the number of timesteps in the input, was tuned and set to 48 for this experiment.

OPO Tensors	MAESTRO	Mozart
Baseline	41.4%	31.7%
MPNN Model	**42.9%**	**47.5%**

Table 3: Test accuracies of the baseline strategy versus the MPNN model predictions on the MAESTRO and Mozart datasets after converting to OPO tensors



(a) Greedy generation **(b)** Sampled generation

Figure 11: Music generated using different methods by the same MPNN model on the same seed. The smaller dots compared to previous piano rolls are a result of the OPO data representation

Using the trained models, we attempted to generate new music in an autoregressive manner as with the previous models. However, despite this model's relatively strong performance, the generated notes gradually reduced to silence, as shown in Fig. 11a, suggesting that there is still a bias towards predicting zero vectors.

3.7 Experiment 7: Graph U-Net

As we can see from the plot 12, the results of the decoded data have a low loss. We can infer that the model is able to learn and compress the patterns into a smaller piano with fewer sets of notes. By considering a larger dataset and models we can make our models more solid. The current output generated, as can be seen in 13, shows us patterns similar to the actual music.

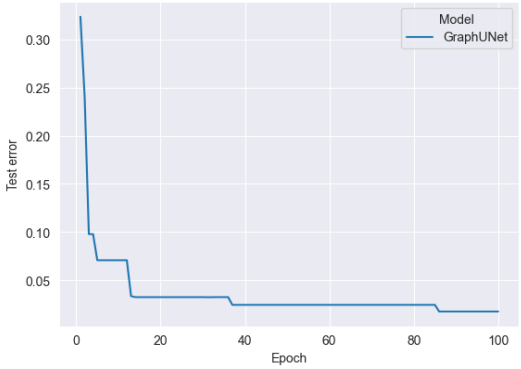


Figure 12: Graph U-Net test loss over Epoch, which is a combination of timestep and keys' velocity loss.

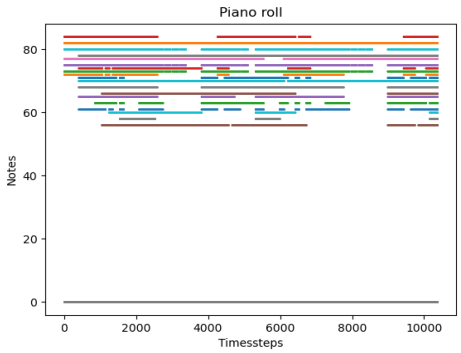


Figure 13: Piano roll view for U-Net UN model

4 Discussion

4.1 Music generation

In all of the music generation attempts described above, the outputs were selected greedily by filtering for notes that had predicted probabilities of over 0.5. Due to the models' biases towards degenerate strategies, such as repeating the last time step or only outputting silence, this greedy approach of generation resulted in unusable output.

Instead, we experimented with a sampled-based generation approach, i.e. notes are picked randomly with weights based on the probabilities predicted by the model. This approach resulted in significantly better output, as shown in Fig. 11b. While the judgement of musicality is partly subjective, we note that the generated pieces lack structure but exhibit some musical patterns, such as ascending scales and repeated chords. Samples of generated music can be found in the project repository[2] - the first six seconds of each piece is the seed music.

4.2 Data representation

The conversion of music into structured data is a complex process, as described in Section 2.2 about data conversion. While the OPO format enabled the models to learn more easily from the training data, it loses sustain information, which is an important element of musical mood and style. Furthermore, this representation is only valid for percussive instruments that are struck, including the piano, but will not work for instruments where notes are sustained, such as wind and string instruments. As such, we recognise that our data representation simplifies music to an extent.

5 Conclusion

In our initial experiments, we found that simply using a GCN does not lead to better results for music autoregression. Follow-up experiments indicated that using a GCN as an information-enriching embedding layer to sequence processing layers like an RNN improves performance. This suggests that the model needs to be designed to capture both temporal and structural information in a principled manner.

We investigated the more powerful MPNN model architecture and found that it addresses the limitations of convolution-based models. However, the representation of data as MIDI tensors allowed simple models to learn spurious correlations and apply degenerate strategies, making it difficult to evaluate model performance. We devised the alternative OPO format, which allowed the MPNN model to better learn from the training data, enabling it to beat the baselines. The Graph U-Net approach was able to encode and decode and generate audio reach in patterns.

Finally, we used the MPNN model to generate music autoregressively. By taking a sampling approach, we were able to create pieces that contain some semblance of musicality, including segments of scales and chords, which we make available in our repository.

[2] https://github.com/Vinnesta/l45-music-gnn/tree/main/generated_music/MPNN

Acknowledgements

We would like to thank our supervisor Lorenzo Giusti for all of the guidance and recommendations he provided throughout the course of this project.

References

[1] Filippo Carnovalini and Antonio Roda. Computational creativity and music generation systems: An introduction to the state of the art. *Frontiers in Artificial Intelligence*, 3, 04 2020. doi: 10.3389/frai.2020.00014. 1

[2] Anna K. Yanchenko. Classical music composition using hidden markov models. 2017. 2

[3] Alexander Gunawan, Ananda Iman, and Derwin Suhartono. Automatic music generator using recurrent neural network. *International Journal of Computational Intelligence Systems*, 13:645, 06 2020. doi: 10.2991/ijcis.d.200519.001. 2

[4] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam M. Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer: Generating music with long-term structure. In *International Conference on Learning Representations*, 2019. 2

[5] Josh Walsh. Using tonnetz tone mesh to understand jazz harmony. https://jazz-library.com/articles/tonnetz/, 2020. [Online; accessed 23-March-2023]. 2, 4

[6] Ching-Hua Chuan and Dorien Herremans. Modeling temporal tonal relations in polyphonic music through deep networks with a novel image-based representation. AAAI'18/IAAI'18/EAAI'18, page 8. AAAI Press, 2018. ISBN 978-1-57735-800-8. 2

[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017. 2, 5

[8] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1lYRjC9F7. 3

[9] magenta. The MAESTRO dataset. https://magenta.tensorflow.org/datasets/maestro, Oct 29, 2018. [Online; accessed 22-March-2023]. 3

[10] Bernd Krueger. The Mozart dataset from piano-midi.de. http://www.piano-midi.de/mozart.htm. [Online; Licensed under the cc-by-sa Germany License; accessed 22-March-2023]. 3

[11] PyTorch. Bcewithlogitloss-pytorch 2.0 documentation. https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html, 2023. [Online; accessed 22-March-2023]. 5

[12] PyTorch. Rnn-pytorch 2.0 documentation. https://pytorch.org/docs/stable/generated/torch.nn.RNN.html, 2023. [Online; accessed 22-March-2023]. 5

[13] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. 09 2016. 5

[14] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzmán López, Nicolas Collignon, and Rik Sarkar. Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. CIKM '21, page 4564–4573, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384469. doi: 10.1145/3459637.3482014. URL https://doi.org/10.1145/3459637.3482014. 5

[15] Yugesh Verma. A beginner's guide to spatio-temporal graph neural networks. https://analyticsindiamag.com/a-beginners-guide-to-spatio-temporal-graph-neural-networks/, Jan 26, 2022. [Online; accessed 22-March-2023]. 5

[16] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In Long Cheng, Andrew Chi Sing Leung, and Seiichi Ozawa, editors, *Neural Information Processing*, pages 362–373, Cham, 2018. Springer International Publishing. ISBN 978-3-030-04167-0. 5, 6

[17] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019. 6

[18] Chris Donahue, Ian Simon, and Sander Dieleman. Piano genie. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 160–164, 2019. 6, 7