Project Report

Music Generation with GAN and RL

Team: Ivan Anokhin, Egor Nuzhin

Introduction

In the recent year there are attempts to generate music using neural networks. Neural network models are particularly flexible because they can be trained based on the complex patterns in an existing musical dataset. One particularly interesting approach to music composition is training a probabilistic model of polyphonic music. Such an approach attempts to model music as a probability distribution, where individual sequences are assigned probabilities based on how likely they are to occur in a musical piece. Importantly, instead of specifying particular composition rules, we can train such a model based on a large corpus of music, and allow it to discover patterns from that dataset, similarly to someone learning to compose music by studying existing pieces. Once trained, the model can be used to generate new music based on the training dataset by sampling from the resulting probability distribution.

In last several year there was a lot of research work about reinforcement learning (RL) and generative adversarial networks. Authors of article [1a] drawing intuition from the Turing test, proposed open-domain dialogue generation, using GAN: the system was trained to produce sequences that are indistinguishable from human-generated dialogue utterances. But unlike other system, they casted the task as a reinforcement learning (RL) problem.

General purpose of our work was to adopt described RL and GAN methods to music generation task while improving based generative and discriminative models. Apply different learning techniques and study ability of their potential usage for music generation tasks.

1. Dataset description

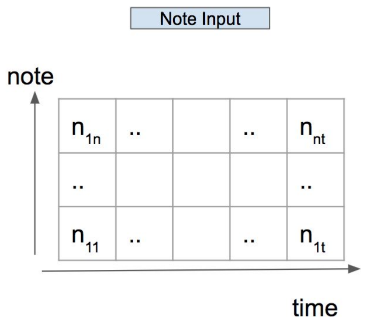
We used open source dataset: The Lakh MIDI Dataset

Transformed structure of midi file to use in model is:

- Play matrix - whether the particular note in particular time is played or not
- Replay matrix - -//- is replayed or not (i.e. key on piano is pressed)
- Volume matrix

2. Generator

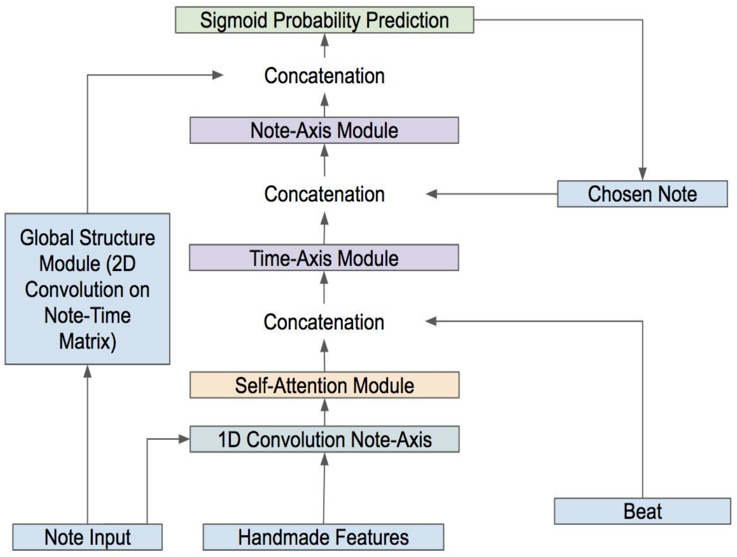
Input



The Generator model takes as input 3d tensors with dimensions that correspond to time, notes pitch and notes features (volume, play, replay features).

Handmade features are features for every note created by onehoting pitch position, onehoting position in gamma, and summing up of same notes in different octave. Beat input onehots the position of each note along time-axis.

Layers

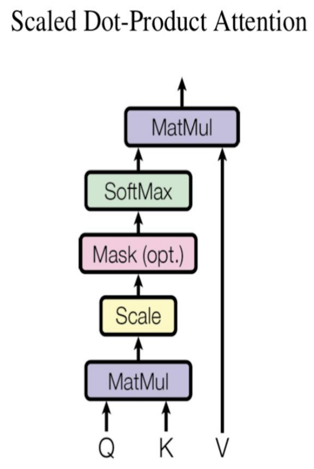


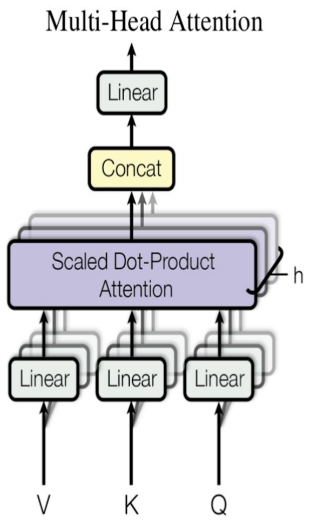
1D Convolution layers perform convolution with kernel = 24 (two times of octave) along the note axis and helps the model to deal with harmony.

Last added to the model Global Structure Module performs convolution on the whole input time-note matrix. Its output get concatenated with every notes feature before last dense layer. The model benefits from it by having more even and structured output.

Attention layer performs the multi-headed self-attention along the note axis as it is describe in the Attention is all you need ([1c]).

The authors described it in next words: The input consists of queries (Q) and keys (K) of dimension dk, and values (V) of dimension dv. We compute the dot products of the query with all keys and apply a softmax function to obtain the weights on the values.



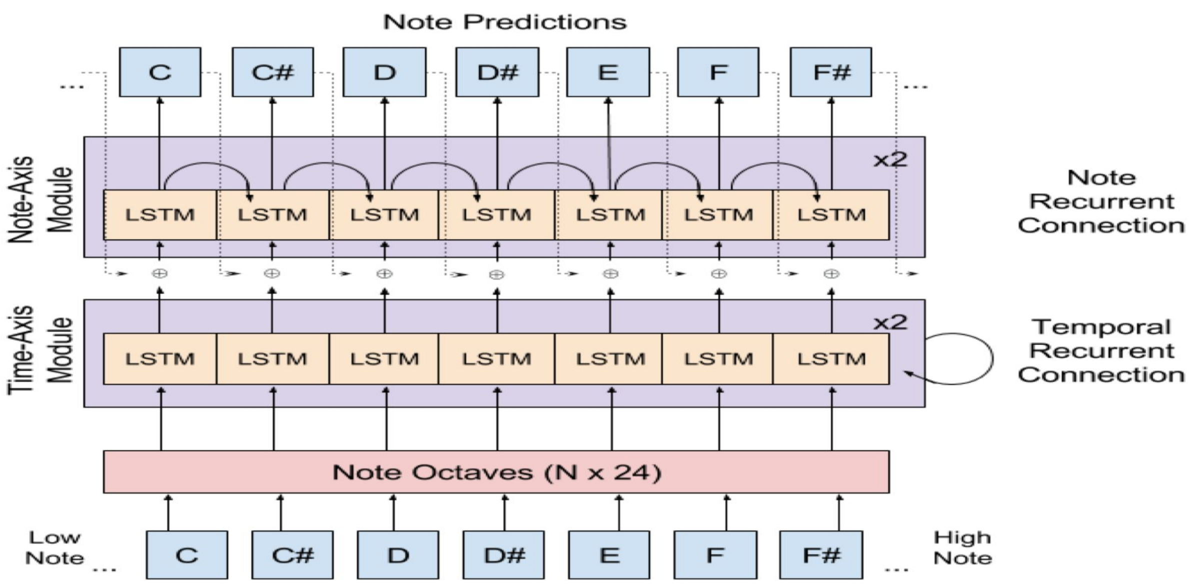


Instead of performing a single attention function the authors found it beneficial to linearly project the queries, keys and values several times with different, learned linear projections.

Self attention is the specific case of attention when Q=K=V. In our model we tackle with particular this mechanism.

In our case we find it beneficial to project the space to the same dimension and to perform Q by new weighted value (old Scaled Dot-Product Attention output) multiplication in the end of Scaled Dot-Product Attention output module.
We have used three heads. Self-Attention helps model to play cords.

Self-attention output is fed to time-axis module --- LSTM along time axis (and for each note we share weights of the LSTM - because of time invariance for each note). LSTM-time-output has the same dimension as input and it can be fed to the note-axis module --- note-LSTM along the note axis (the note-LSTM also shares weights for each time steps).



After that model makes predictions for the the next time-note matrix value (using dense layers). The model trying to predict next note not only using information about notes in previous time step but also about note that are already generated by model in a current time step as we generate note from the highest one to the lowest one.

3. Discriminator

The discriminative model is a binary classifier that takes as input a song part {x, y} and outputs a label indicating whether the input is generated by humans or machines. The input song is encoded into a vector representation using a hierarchical encoder [2a] which is then fed to a softmax function, returning the probability of the input song episode being a human-generated dialogue (denoted Q_+ ({x, y})).

Originally hierarchical encoder [2a] was applied to text input and consider as input utterances of sentences, but we adopted given architecture to music data.

To be specific, each cord of song part is mapped to a vector representation h_c using LSTM. And then another LSTM is put on time level, mapping cord representations to new vector representation h_t. Last hidden state h_t of the second LSTM is fed into liner layer and activated by sigmoid function to get probability.

Model architecture is described on figure 1.

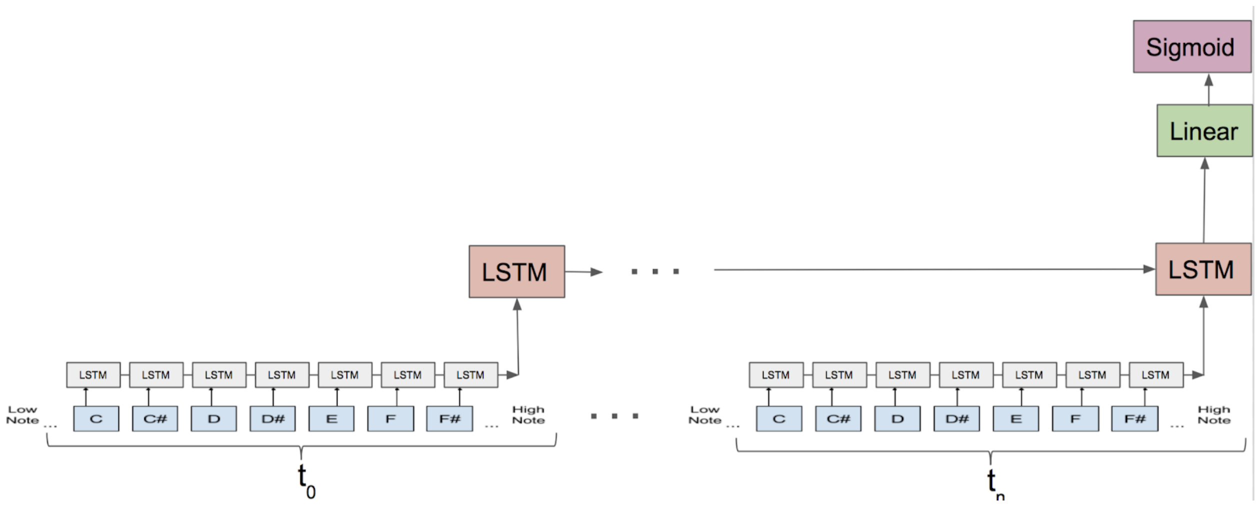


Figure 1 - Based hierarchical discriminator

In our experiments we modified based hierarchical discriminator shown above by note octave embedding (figure 2). It allow to discriminator to use more reliable self-learning features of music and use them while making decision about the data.

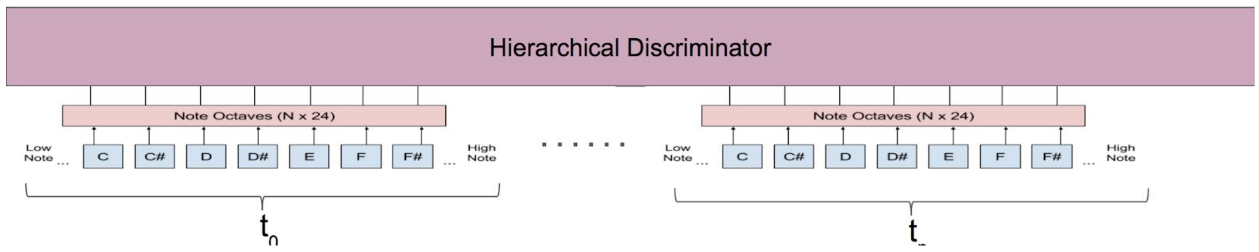


Figure 2 - Modified hierarchical discriminator

4. GAN models

There are many different approaches how to train generative adversarial networks and what type of networks to use. We tested two type of networks: condidianal GAN[3c] and usual unconditional GAN.

On figure 3 is shown the model interpretation for image generation task.

In case of music we feed into generator part of song and generate probability of the next song. Based on generated probabilities we sample an example and concatenating it with part of input sung we fed received data into discriminator. Based on this tuple of data discriminator is learnt to distinguish humans generated example from machine generated ones.

Described procedure teach discriminator to give feedback to generator of generating shifted by one time step output. Discriminator discover that all generated notes except the last should be equal to all input notes except the first. It's similar to classic MSE/NLL-loss between generated songs and real one as is done in arricale [1a].

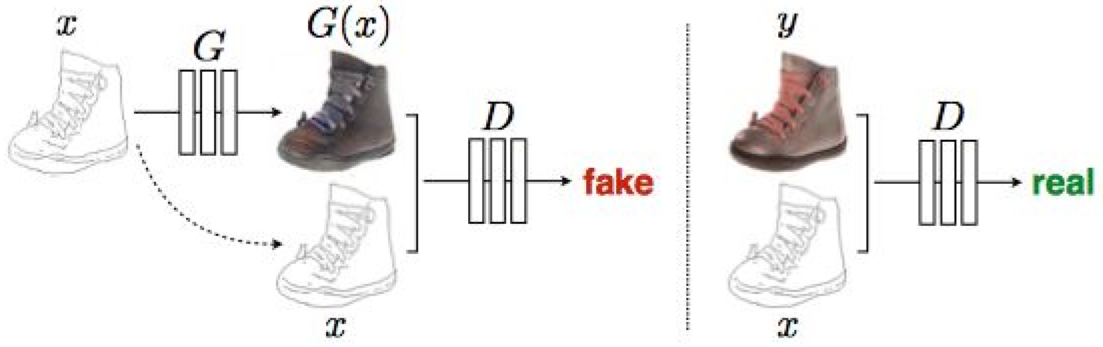


Figure 3 - Conditional GAN

In case of unconditional GAN (figure 4) we used some different method. We similarly generated music via generator, based on given probabilities, but instead of predicting notes only for the the next time step based on previous history we generated music as it happens on exploitation stage. We set as input full silence, generated the next cord, concatenated it to the input, then again, based on new input we generated next note and so on until we have full scale music sequence.

Generated song fed into discriminator for making decision and for computing reward the last probabilities are used.

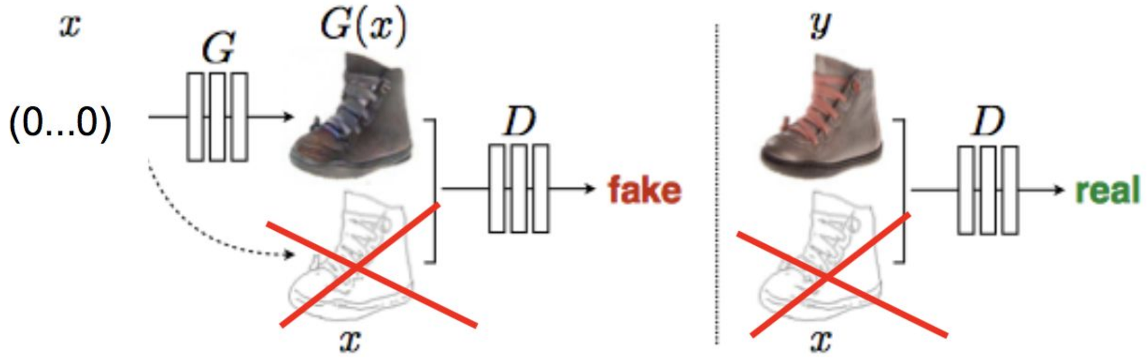


Figure 4 - Unconditional GAN

5. Learning technique

The key idea of the system is to encourage the generator to generate songs thatt are indistinguishable from human generated ones. We use policy gradient methods to achieve such a goal, in which the score of current song being human-generated ones assigned by the discriminator (i.e., $Q_+(\{x, y\})$) is used as a reward for the generator, which is trained to maximize the expected reward of generated utterance(s) using the REINFORCE algorithm[3b].

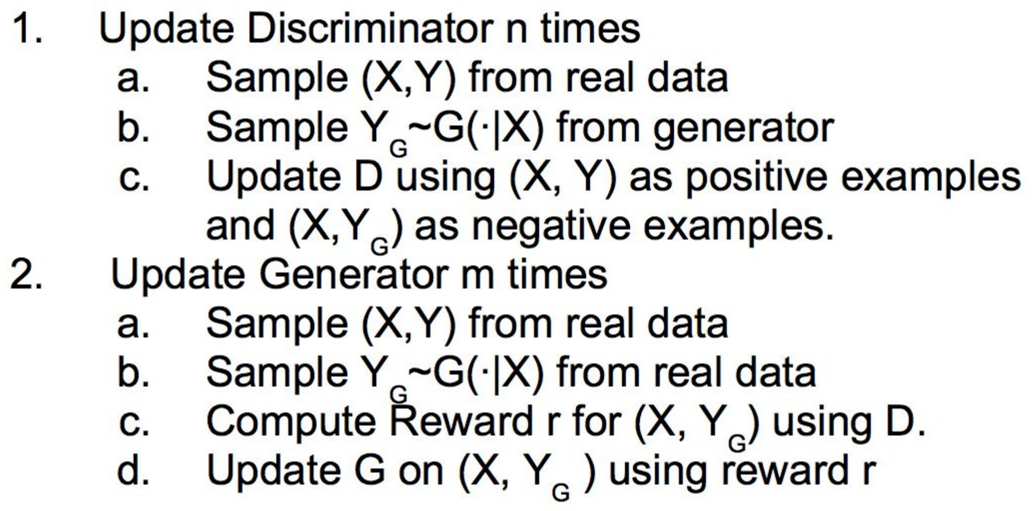
$$J(\theta) = \mathbb{E}_{y \sim p(y|x)}(Q_+(\{x, y\})|\theta)$$

Given the input song part, the model generates the shited by time notes y by sampling from the policy.. The gradient is approximated using the likelihood ratio trick:

$$\begin{aligned} \nabla J(\theta) &\approx [Q_+(\{x, y\}) - b(\{x, y\})] \\ &\qquad \nabla \log \pi(y|x) \\ &= [Q_+(\{x, y\}) - b(\{x, y\})] \\ &\qquad \nabla \sum_t \log p(y_t | x, y_{1:t-1}) \end{aligned}$$

where π denotes the probability of the generated song. b({x, y}) denotes the baseline value to reduce the variance of the estimate while keeping it unbiased. The discriminator is simultaneously updated with the human generated songs that contains song history x as a positive example and the machine-generated songs as a negative example.

Training algorithm is described on figure 5.



```
1. Update Discriminator n times
   a. Sample (X,Y) from real data
   b. Sample $Y_G$~G(·|X) from generator
   c. Update D using (X, Y) as positive examples
      and (X,$Y_G$) as negative examples.
2. Update Generator m times
   a. Sample (X,Y) from real data
   b. Sample $Y_G$~G(·|X) from real data
   c. Compute Reward r for (X, $Y_G$) using D.
   d. Update G on (X, $Y_G$ ) using reward r
```

Figure 5 - GAN-RL algorithm

As baseline value we trained the same based hierarchical discriminator (critic) to estimate the value (or future re-ward) of current state (i.e., the song history) under the current policy π. The critic network takes as input the song history, transforms it to a vector representation and maps the representation to a scalar. The network is optimized based on the mean squared loss between the estimated reward and the real reward.

6. Metrics

In order to estimate quality of generated music we implemented number of metrics. In accordance to article [4a] we used:

1. EB: ratio of empty bars (in %).
2. UPC: number of used pitch classes per bar (from 0 to 12).
3. QN: ratio of “qualified” notes (in %). We consider a note no shorter than three time steps as a qualified note. QN shows if the music is overly fragmented.

By comparing the values computed from the real and the fake data, we can get an idea of the performance of generators. The concept is similar to the one in GANs—the distributions (and thus the statistics) of the real and the fake data should become closer as the training process proceeds.

Additionally we watch on another metrics to estimate learning convergence. We watch on mean generated probability (generator confidence), to be sure that generator prediction is not too random. Watch on discrimination probabilities of true and false samples. They should be significantly different, but not too much, since discriminator overfitting may led to convergence to local minima. Baseline probability was watched too, that shouldn't variate a lot and all losses: discriminator, generator and baseline.

7. Training notes

During training we find it useful to

- Do Teacher Forcing
  - Helps to reach EB, UPC, QN metrics
- Use Baseline Value
  - Speeds up generator training
- Pretrain generator
- Pretrain discriminator
- Use noise as negative sample to pretrain discriminator
  - Speed up discriminator training
- Use classical cross entropy loss along with RL loss for generator
  - Generated music stay harmonic after a long training

We also have tried to do Monte Carlo sampling for conditional GAN but it seems to have no significant effect on the model.
There were two Teacher Forcing versions. One way was to give reward equals to 1 to Forcing examples and in the other we used discriminator probability as a reward. In our experience It is hard to say which version works the best.

Next strategies were used for GAN training:

- If mean nonnegative difference between human example and machine generated examples is small (0.1) we train discriminator, else train generator - the best example for conditional GAN was obtained by this strategy.
- One step for the generator, four for the discriminator - the best example for unconditional GAN.

It is interesting to note that during work on the project generator and discriminator were made more powerful several times. The improvement was caused by the fact that one of the model got stuck and can not win another one long time. In our last iteration of this process the discriminator model was stronger than the generator.

Contributions

Anokhin Ivan: rewrote generator architecture (from previous machine learning project) in Pytorch, improved generator model by adding Global Structure model, different attention module, implemented music sampling (with and without temperature) inside the Pytorch model, made generating music function compatible with new generator, implemented Monte Carlo sampling, trained basic generator model (without gan), Made experiments comparing conditional models obtained with different hyperparameters, generator architectures, learning strategies, losses.

Egor Nuzhin: developed various types of discriminators and baselines. Implemented GAN-RL algorithms for conditional and unconditional GAN cases. Implemented losses for discriminator, baseline and reword gradient for generator. Has made EB, UPC, QN metrics. Implemented teacher-forcing algorithm. Made experiments comparing models with different variants of loss function, training hyperparameters, model architectures, pretraining and combinations of classic and RL losses.

References

1. Generator Architecture:
    a. DeepJ: Style-Specific Music Generation (2018 arXiv:1801.00887)
    b. Generating Polyphonic Music Using Tied Parallel Networks (2016).
    c. Attention Is All You Need (2017 arXiv:1706.03762)
2. Discriminator Architecture:
    a. A Hierarchical Neural Autoencoder for Paragraphs and Documents (2015 arXiv:1506.01057v2)
3. Learning technique:

a. Adversarial Learning for Neural Dialogue Generation (arXiv:1701.06547)
b. Simple statistical gradient-following algorithms for connectionist reinforcement learning (1992)
c. Image-to-Image Translation with Conditional Adversarial Networks (2017 arXiv:1611.07004v2)
d. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient (arXiv:1609.05473)

4. Metrics:
   a. MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment (arXiv:1709.06298)

Git Repository: https://github.com/avecplezir/project_dnn2018
Music samples can be found in git repository in folder “Final Samples”.

The project was based on our previous machine learning project https://github.com/avecplezir/PopH, which itself was based on DeepJ project https://github.com/calclavia/DeepJ
The main difference is that previous projects are written in Keras and do not use GAN/Policy gradient to train the generator.