Machine Learning on Music Files

Classification and generation of music.

Julie Løgstrup Magnusson, Kasper Munch, Siri Hjermind and Søren Jefsen

Applied Machine Learning Exam

11th of June 2025

UNIVERSITY OF COPENHAGEN

Goal:

Given some music as input, make an algorithm that continues playing

Which leads to sub-goals:

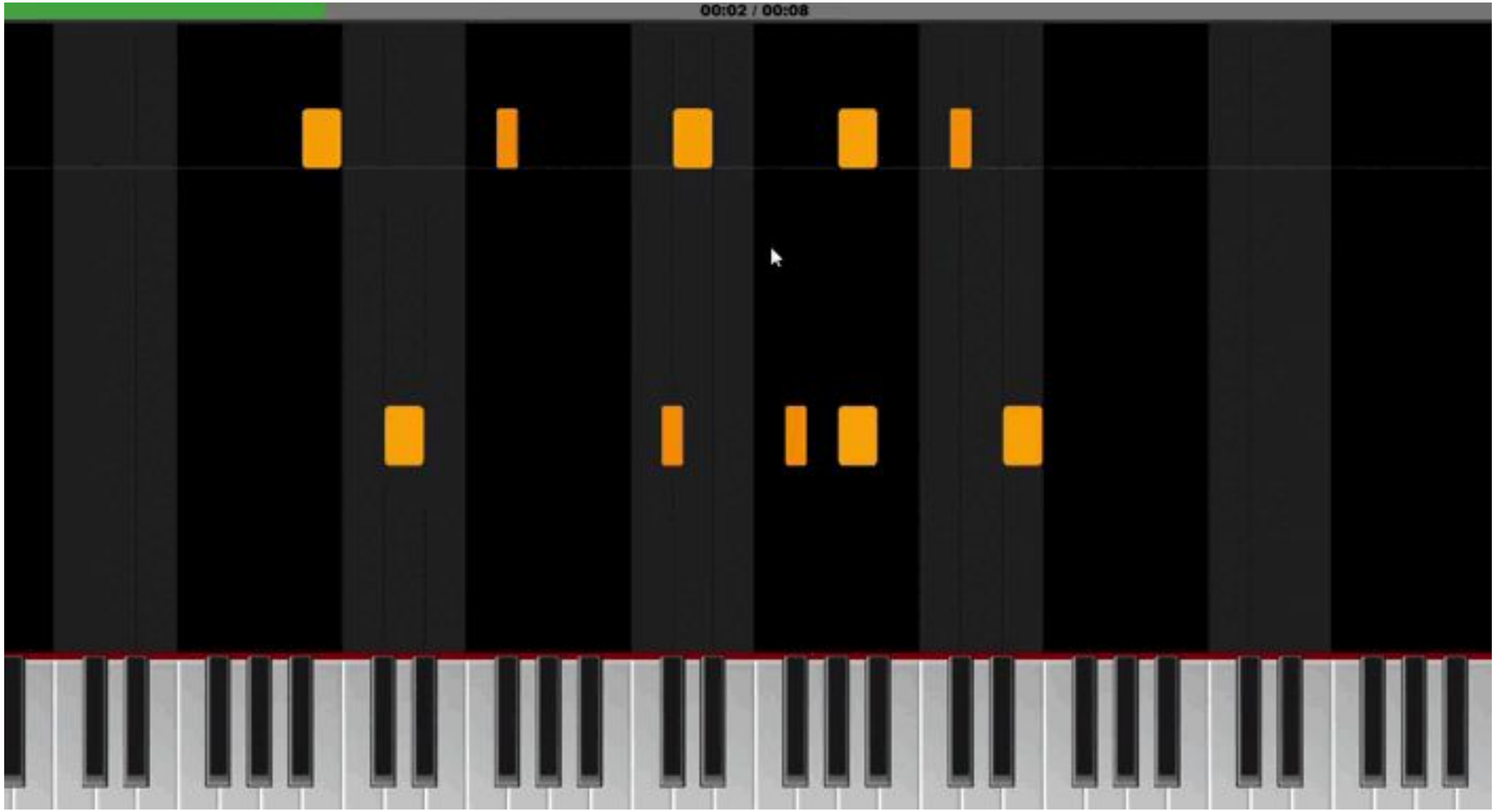
- Can we represent the features of music ?
- Can we classify music ?
- Can we generate music ?

Introduction to MIDI files

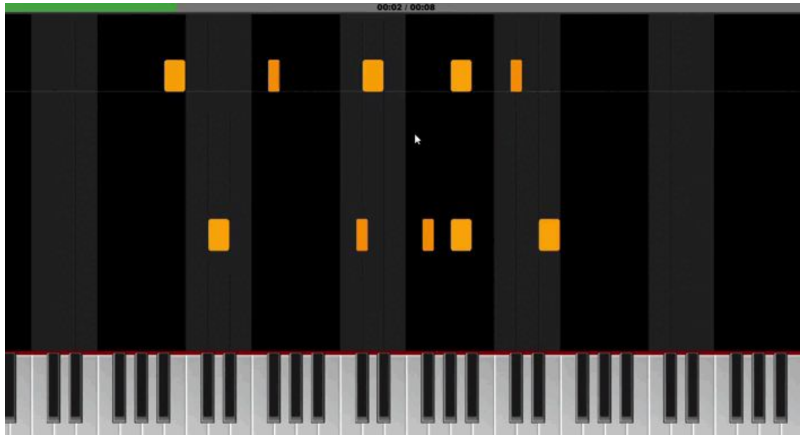
(Musical Instrument Digital Interface)

Introduction to MIDI files (Musical Instrument Digital Interface)

- What is a MIDI file ?



Introduction to MIDI files



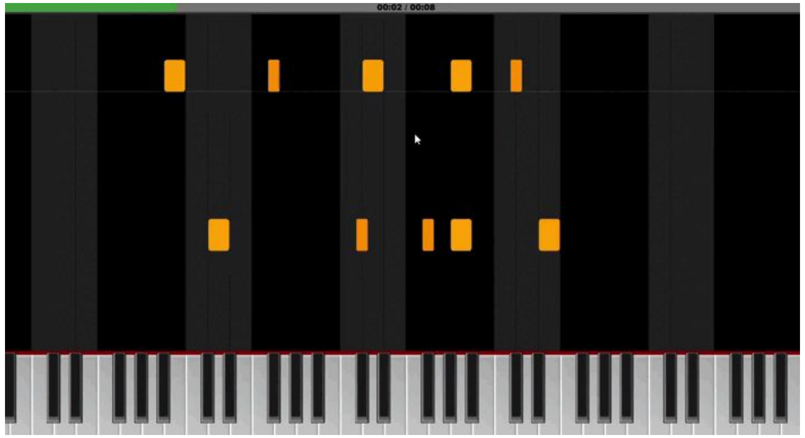
Digital instructions for music

(Condensed format => very small files)

Electronic Music

Introduction to MIDI files

Digital instructions for music

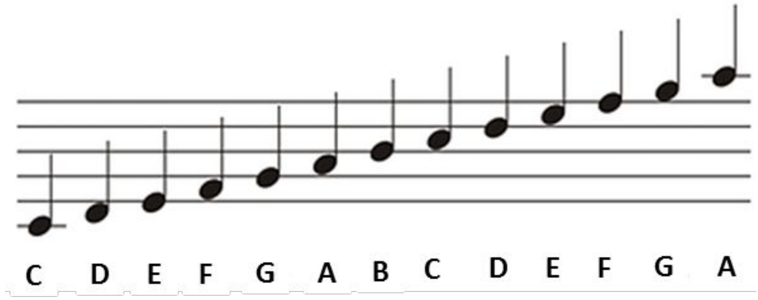


(Condensed format => very small files)

Electronic Music

Think of

Sheet Music



MIDI file structure

Header chunk

- Format
- Number of tracks
- Time Division

...

Track chunk #1

[event type, data1, data2, time]

[[on/off, velocity, note, time since last event]

Repeat for all events...

]

#2

Representing MIDI Data for Machine Learning

Representing MIDI Data for Machine Learning

Sequence of Events

[note, velocity, time]

X

```
[[47. 64.  0.]
 [54. 64.  0.]
 [62. 64.  0.]
 [69. 64.  0.]
 [73. 64.  0.]]
```

Predict

```
[47.  0. 37.]
```

y

Representing MIDI Data for Machine Learning

Sequence of Events

[note, velocity, time]

X

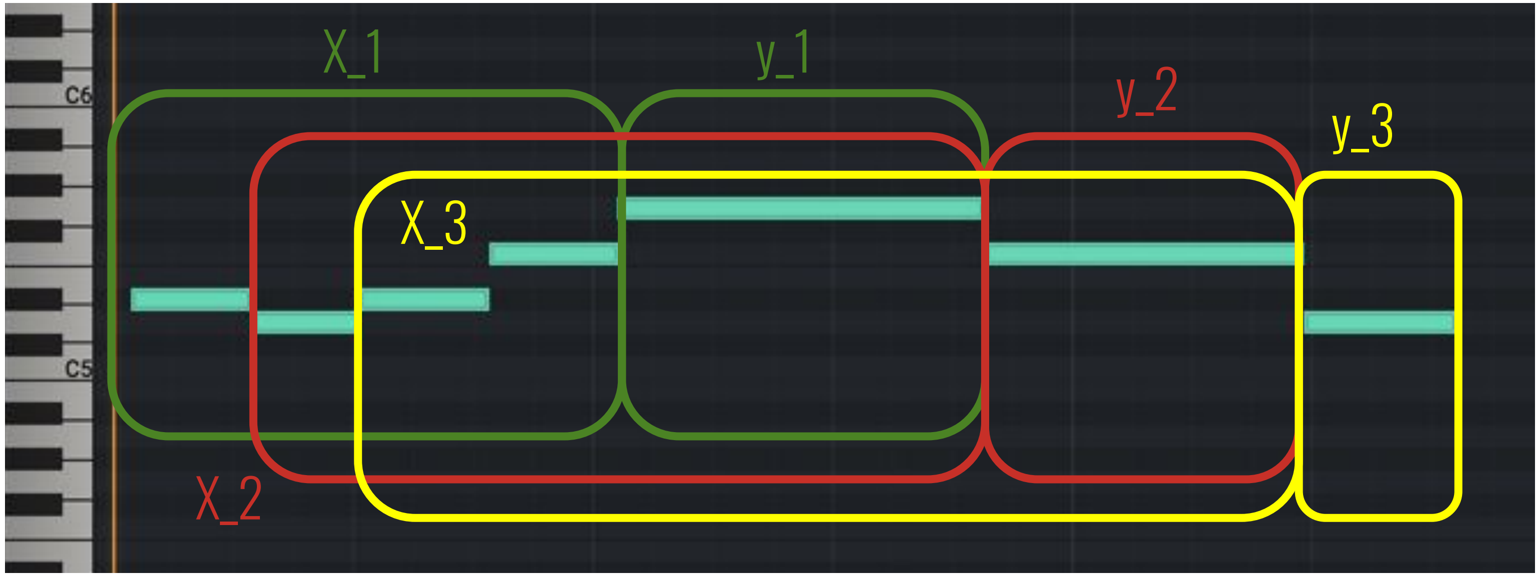
```
[[47. 64.  0.]
 [54. 64.  0.]
 [62. 64.  0.]
 [69. 64.  0.]
 [73. 64.  0.]]
```

Predict

```
[47.  0. 37.]
```

y

Choosing input and output size



Representing MIDI Data for Machine Learning

Sequence of Events

[note, velocity, time]

X

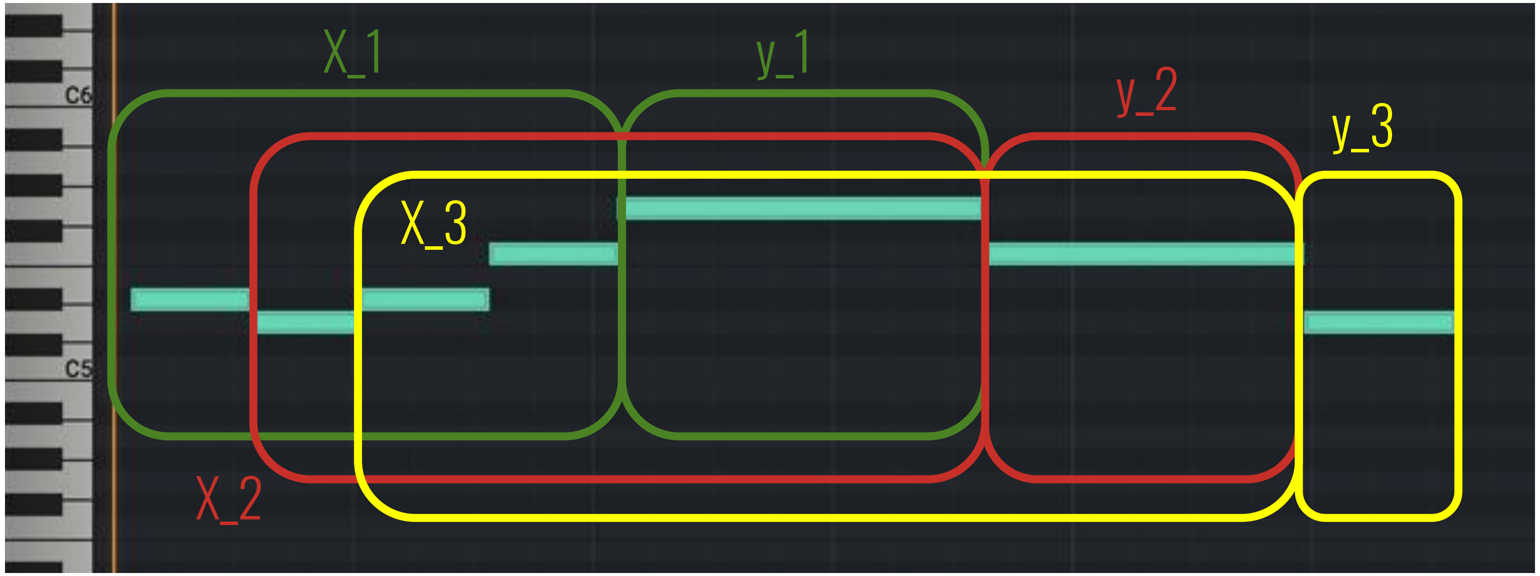
```
[[47. 64.  0.]
 [54. 64.  0.]
 [62. 64.  0.]
 [69. 64.  0.]
 [73. 64.  0.]]
```

Predict

```
[47.  0. 37.]
```

y

Choosing input and output size



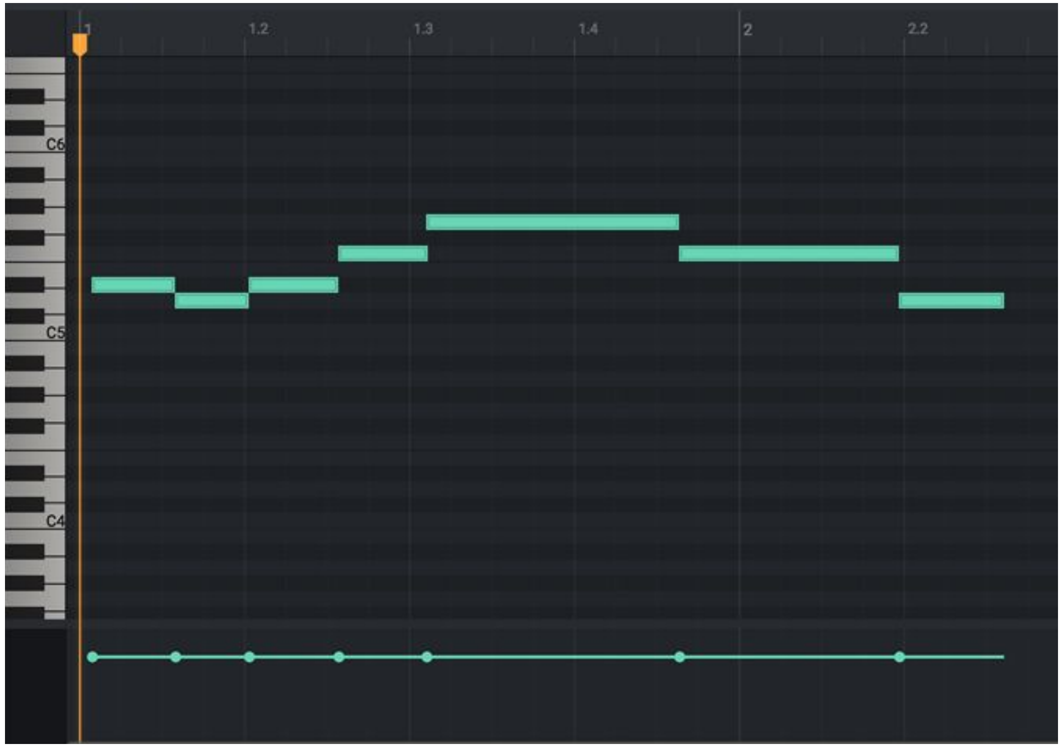
With N_notes in a song we get:

$$N_{datapoonts} = N_{notes} - N_{input} - N_{output}$$

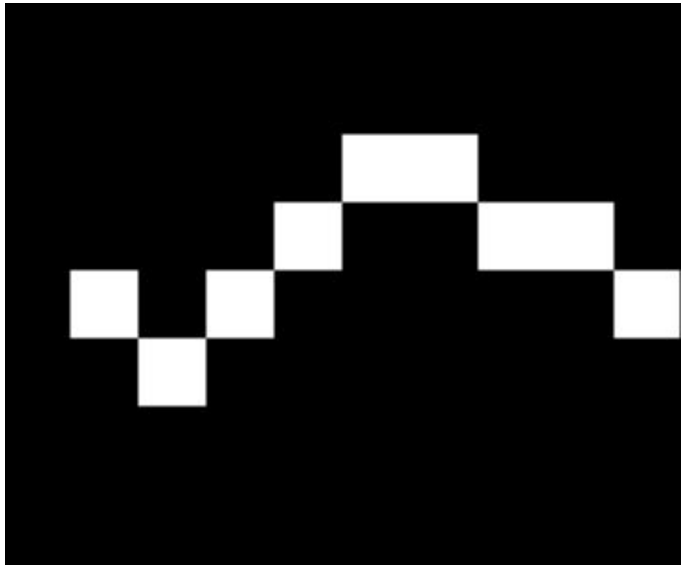
So a song with 100 notes, could give us 89 data points for 10 input and 1 output

Data - Representations

Sparse Matrix




```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 1. 1. 0.]
 [0. 1. 0. 1. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```



Representing MIDI Data for Machine Learning

- Extracting Musical Features with music21

Music21 is a Python toolkit for analyzing and manipulating symbolic music (e.g. MIDI).

Extract_features function:

1. Parsing the MIDI files as music21 objects
2. Extract features such as
   a. 🎼 Key, time signature, tempo
   b. 🎹 Pitch & chord histograms
   c. ⏱️ Duration stats
   d. 🔀 Interval distributions
3. Output = dictionary of features for each MIDI files

MIDI FILES

↓

MUSIC21

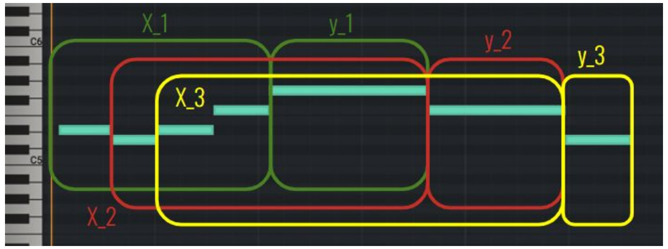
↓

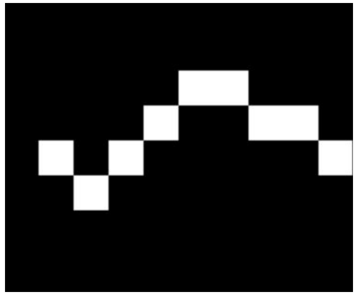
FEATURES

↓

ML

Let’s sum it all up





- Key, time signature, tempo
- Pitch & chord histograms
- Duration stats
- Interval distributions

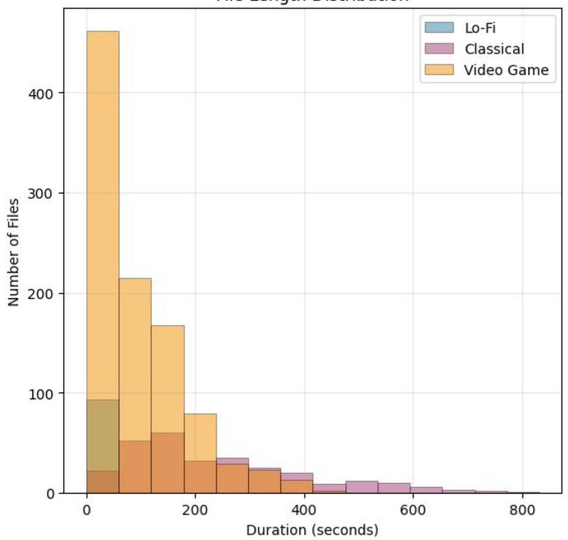
	Easy Data Handling	Fixed input size	Little musical Structure
Musical Structure	Variable input size	Sparse	Little data
Extracts Features	Suitable for Unsupervised Learning	Not suitable for music generation	

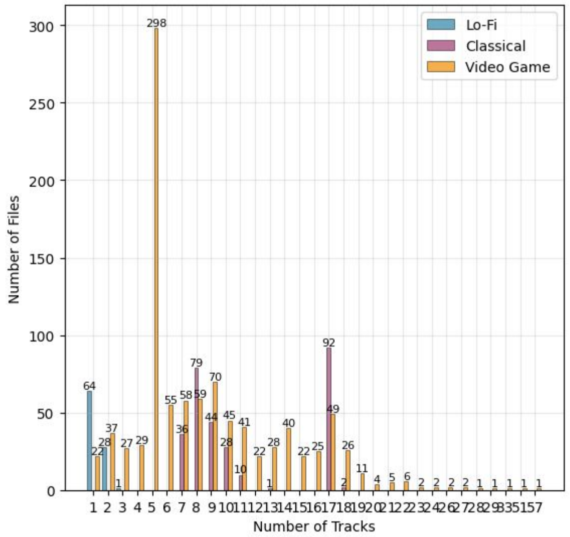
Datasets Used: Source, Structure and Statistics

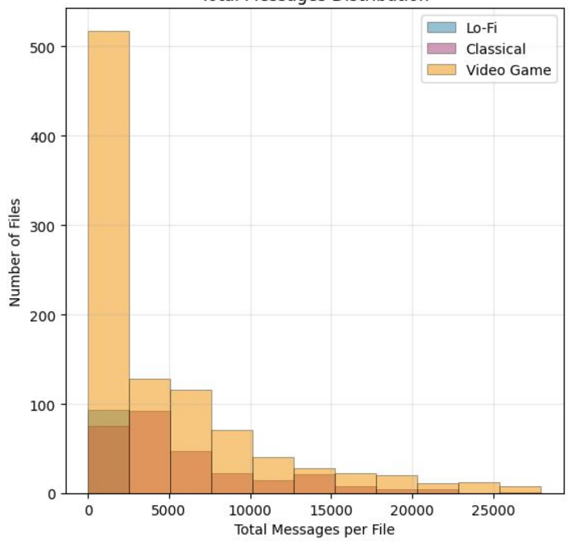
Datasets

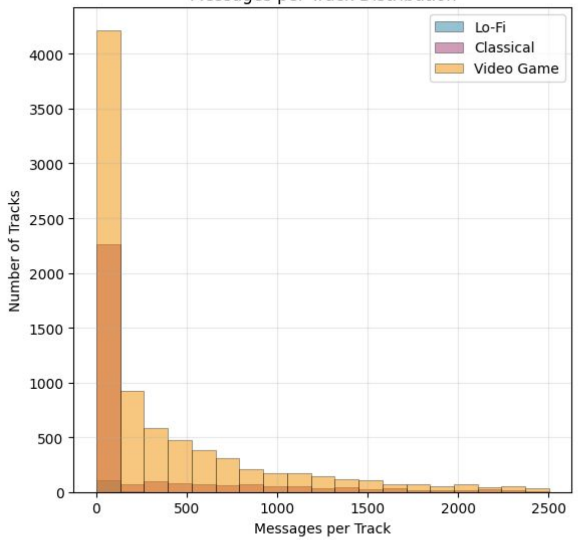
- **Lo-Fi**
  Short and simple
- **Video Game**
  Long and medium
- **Classical**
  Long and complex

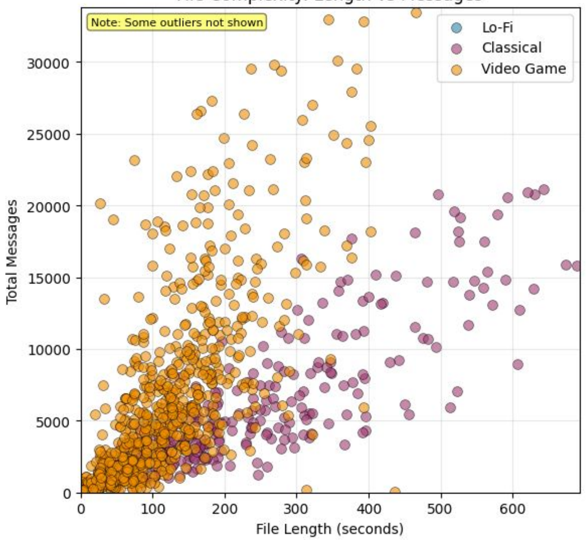
MIDI Datasets Comparative Analysis











Comparative Statistics

Metric	Lo-Fi	Classical	Video Game
Total Files	93	292	992
Avg Length (s)	9.3	254.8	10918.0
Max Length (s)	32.0	2011.1	10737617.4
Avg Tracks	1.3	11.2	8.7
Avg Msgs/File	96	6218	5342
Total Messages	8,910	1,815,652	5,299,385

Unsupervised Learning: Clustering

Clustering Musical Genres with Unsupervised Learning

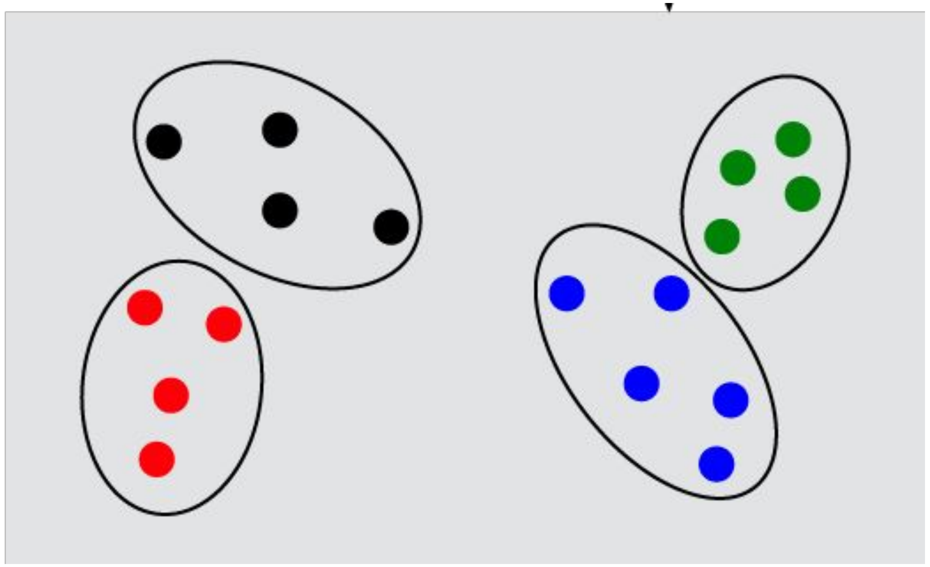
We clustered a selection of music files from three genres: Classical, Lofi, and Video Game music. (1385 MIDI files)

Methods Used:
1. KMeans
2. GMM
3. Spectral Clustering.

A grid search was used to optimize hyperparameters for each method, with the aim of identifying three clusters matching the genres.

The clusters were evaluated using a silhouette score.

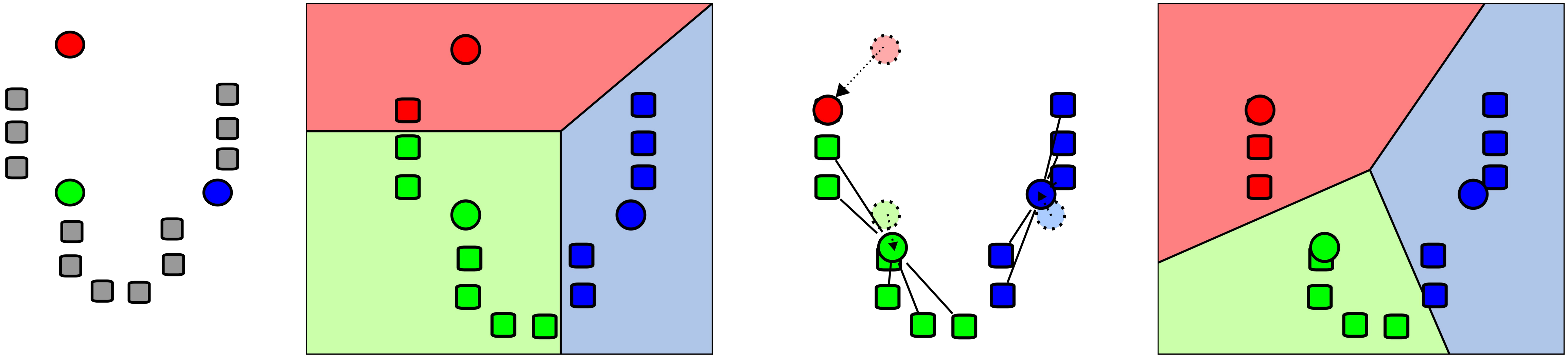
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \longrightarrow S = \frac{1}{n}\sum_{i=1}^{n} s(i)$$



KMeans – a Center-Based Partitioning Clustering Algorithm

Partitioning algorithms generate various clusterings and iteratively assign each observation to one of *k* mutually exclusive clusters

1. Randomly initialize *k* "means" (centroids) within the data domain
2. Assign each observation to the nearest mean
3. Compute the centroid of each cluster to update the means
4. Repeat steps 2 and 3 until convergence



Spectral Clustering – a Graph-Based Partitioning Clustering Algorithm

Instead of relying on distances alone, spectral clustering uses relationships between points to find clusters.

1. Construct a similarity graph from the data, that connects the data based on how close the data points are, the edges have weights according to how close it is.
2. Compute eigenvectors from the graph's Laplacian matrix (based on the graph's spectrum) to reduce the data to k-dimensional space.

$$L = D - W$$

Degree Matrix

Similarity Matrix

3. Apply a standard clustering algorithm (usually K-means) in this reduced space

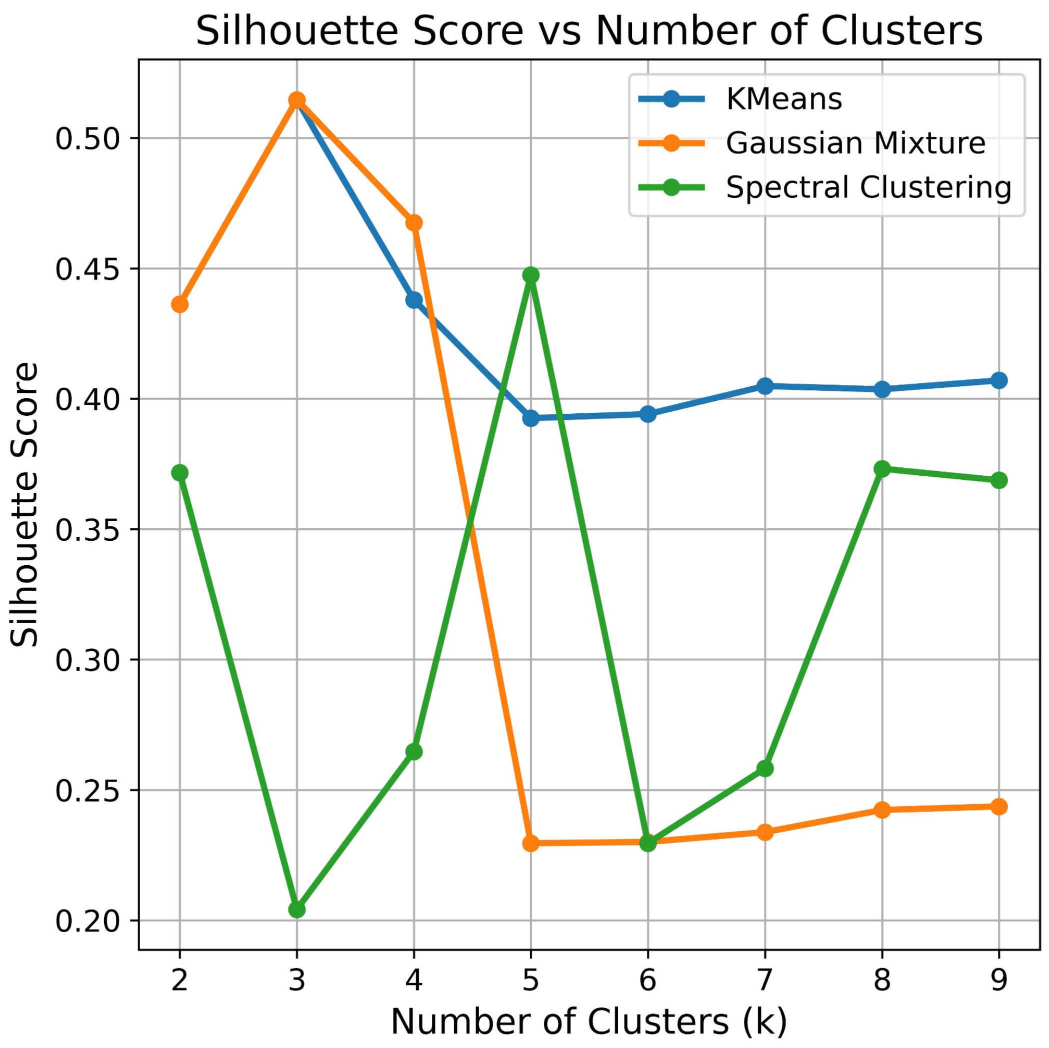
Gaussian Mixture Model

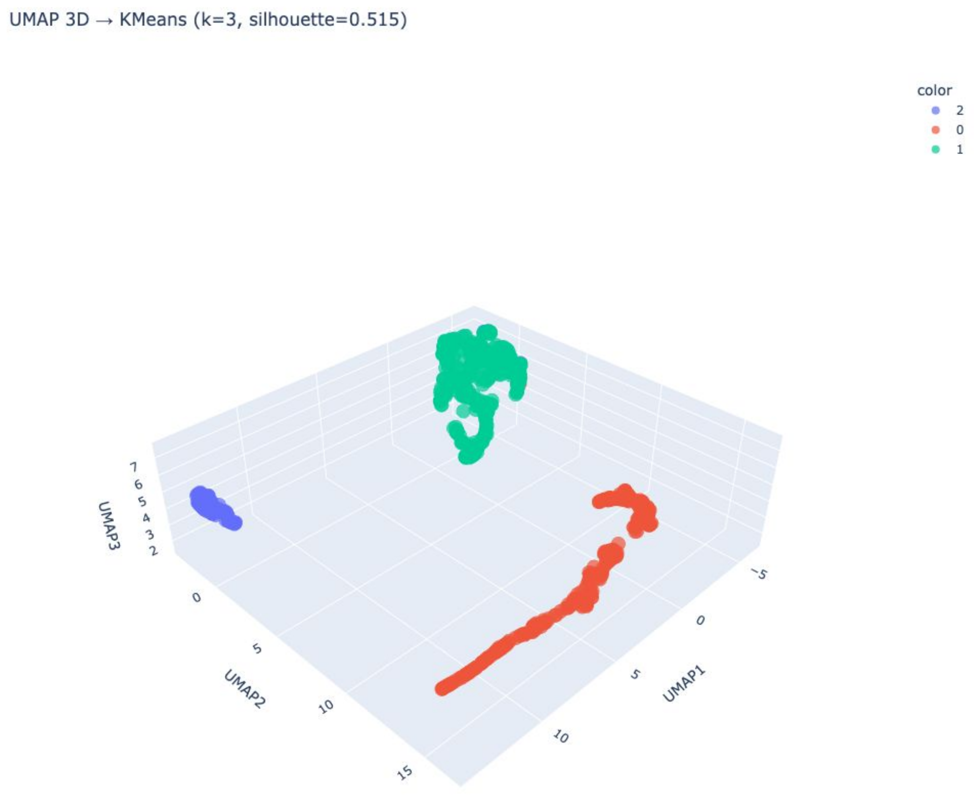
GMM assumes that the data is generated from a mix of several Gaussian distributions

1. The data is modeled as k overlapping clusters, where each cluster is a Gaussian with its own mean and spread (covariance)
2. The parameters of the Gaussians are optimized using the Expectation-Maximization (EM) algorithm
3. Each point is assigned a probability of belonging to each cluster
4. Final clusters are based on this probability

GMM is more flexible than K-means because it can model elliptical clusters.

Clustering Musical Genres with Unsupervised Learning





Clustering Classical Music: Identifying Composer Groups

We cluster a dataset of classical music pieces aiming to identify 19 distinct composers,

by applying the same three clustering methods and use as grid search to optimize for the number of clusters, k .

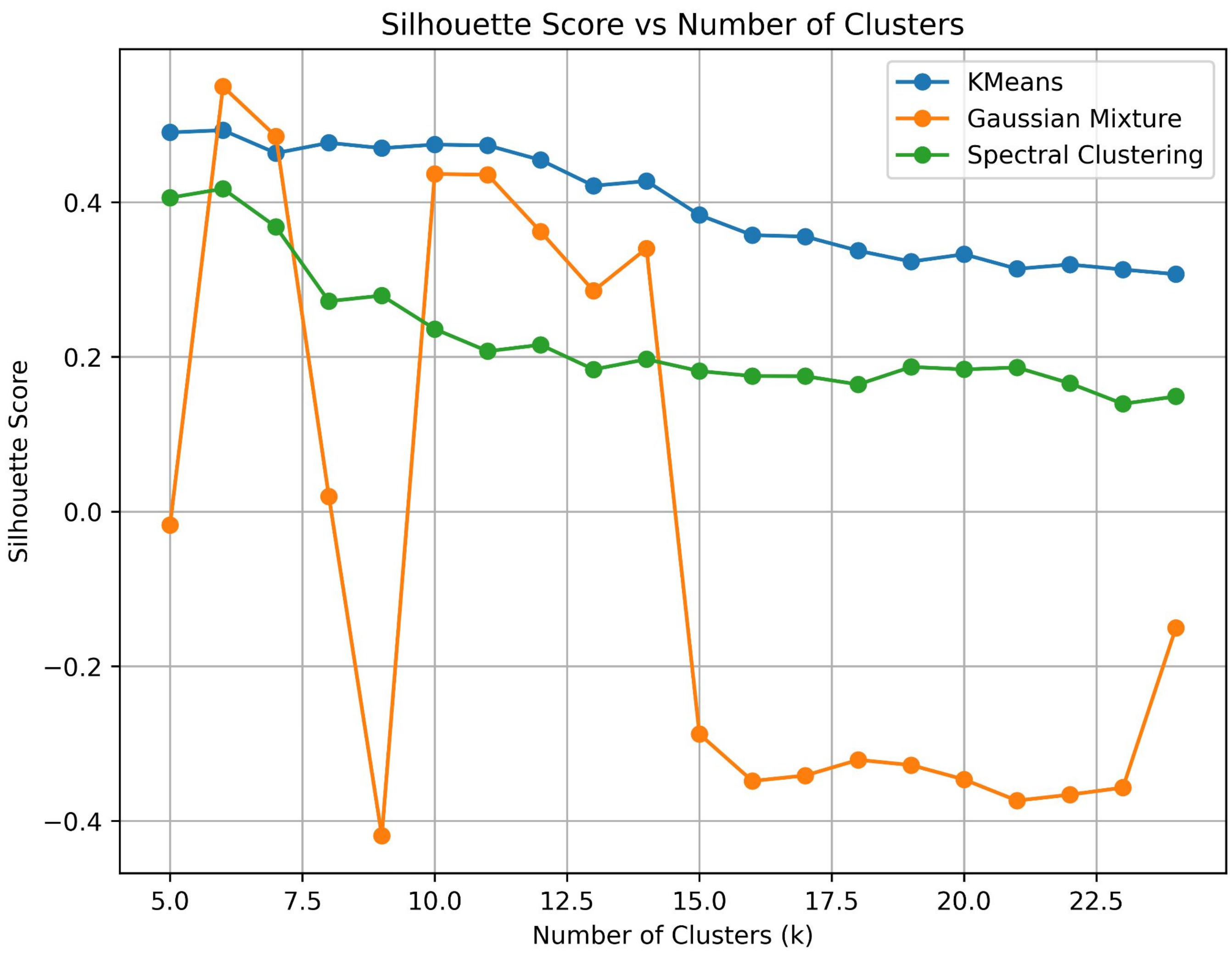
External validation metrics:

Normalized Mutual Information (NMI): Measures the amount of shared information between cluster assignments and true labels.

NMI = 1 $\longrightarrow$ Perfect match between clusters and true classes

NMI = 0 $\longrightarrow$ No mutual information

Clustering Classical Music: Identifying Composer Groups



Algorithm	NMI Score
KMeans (6 clusters)	0.073
KMeans (19 clusters)	0.178
GMM (6 clusters)	0.033
GMM (19 clusters)	0.193
Spectral (6 clusters)	0.105
Spectral (19 clusters)	0.194

Music Prediction

Music prediction with tree-based model

Treats each input independently

Music prediction with tree-based model

Treats each input independently

No sequential awareness or memory of past inputs

Music prediction with tree-based model

Treats each input independently

No sequential awareness or memory of past inputs

Limited context modeling

Music prediction with tree-based model

Treats each input independently

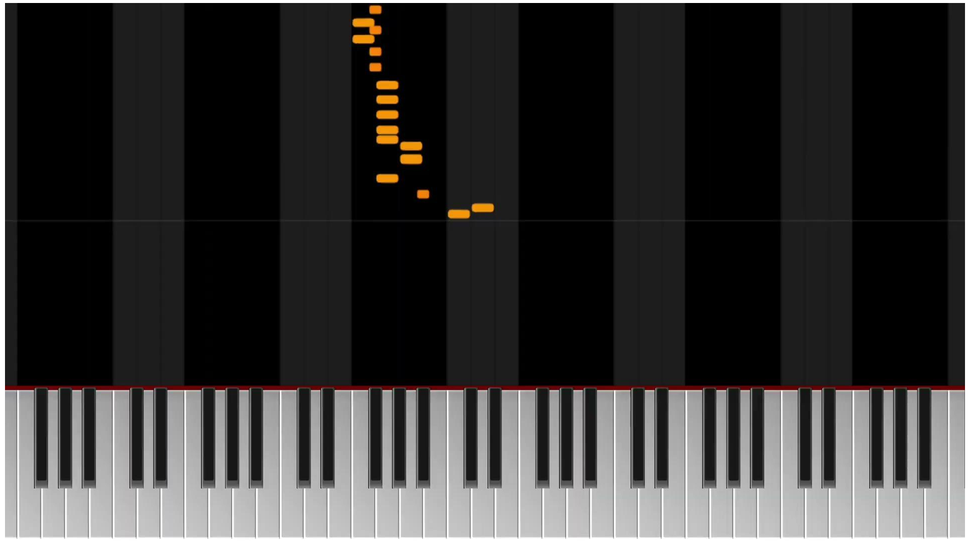
No sequential awareness or memory of past inputs

Limited context modeling

Not inherently generative

Generating music with tree based model....

Is this music?



(Music generated from random forest model trained on the lofi music data)

Could a Neural Network perform better?

Moving on

What are we optimizing ?

Loss functions

We usually use a loss on the form: $\mathcal{L} \sim y_{pred} - y_{target}$

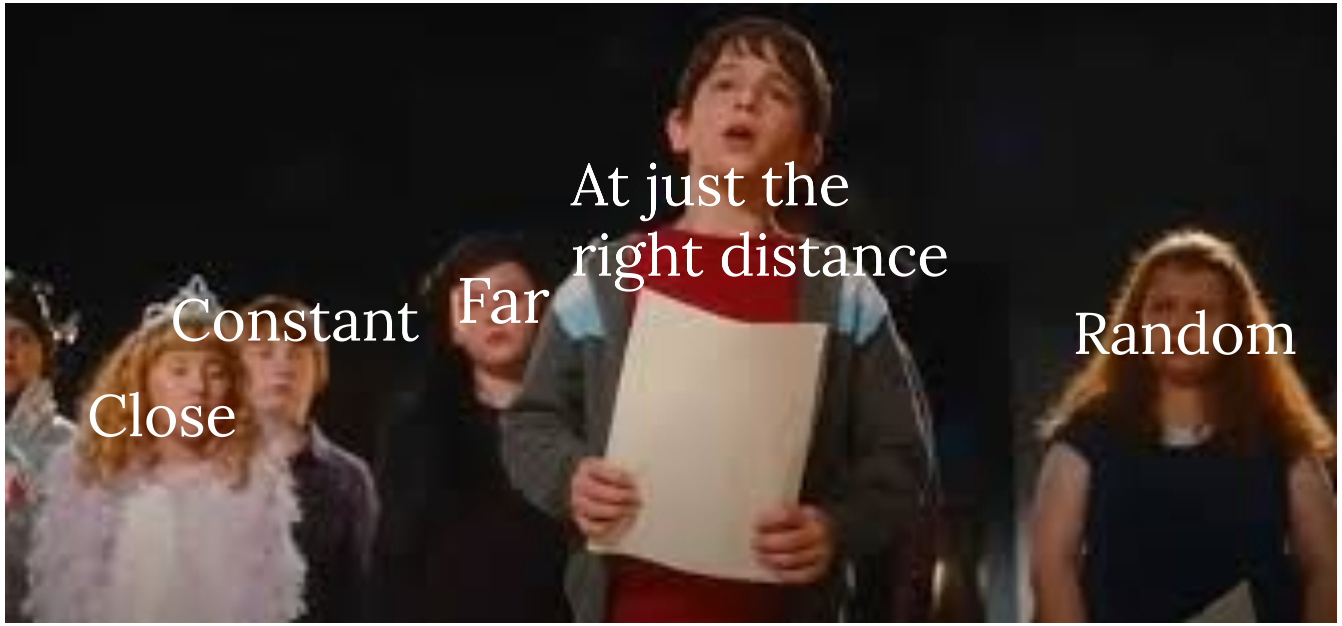
This is good for getting close to target values

Loss functions

We usually use a loss on the form: $\mathcal{L} \sim y_{pred} - y_{target}$

This is good for getting close to target values

But do we really want to "almost" hit the right note in music?

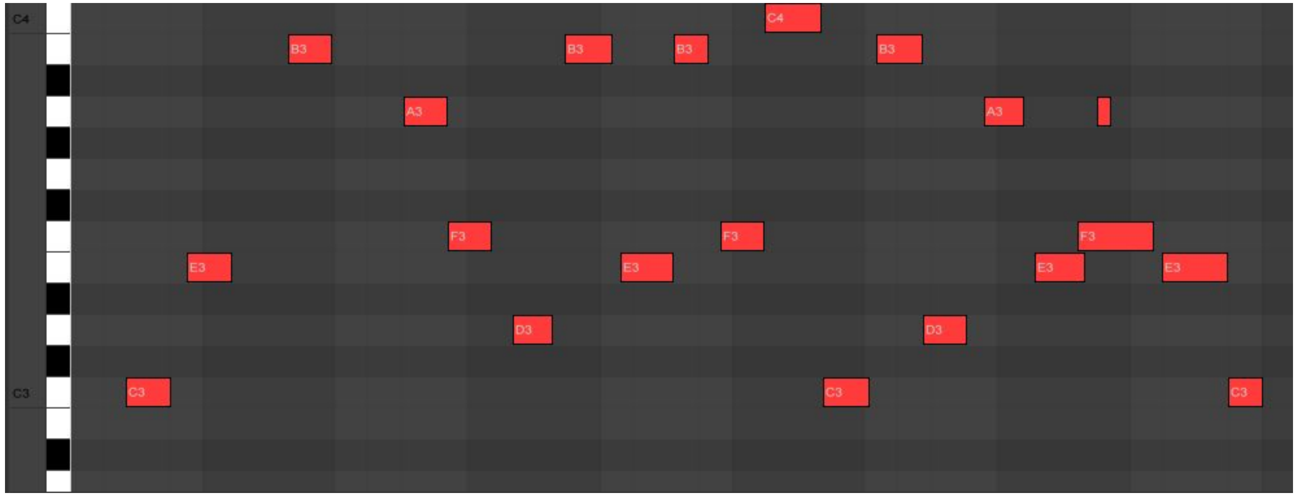


™

Music informed (MINN)

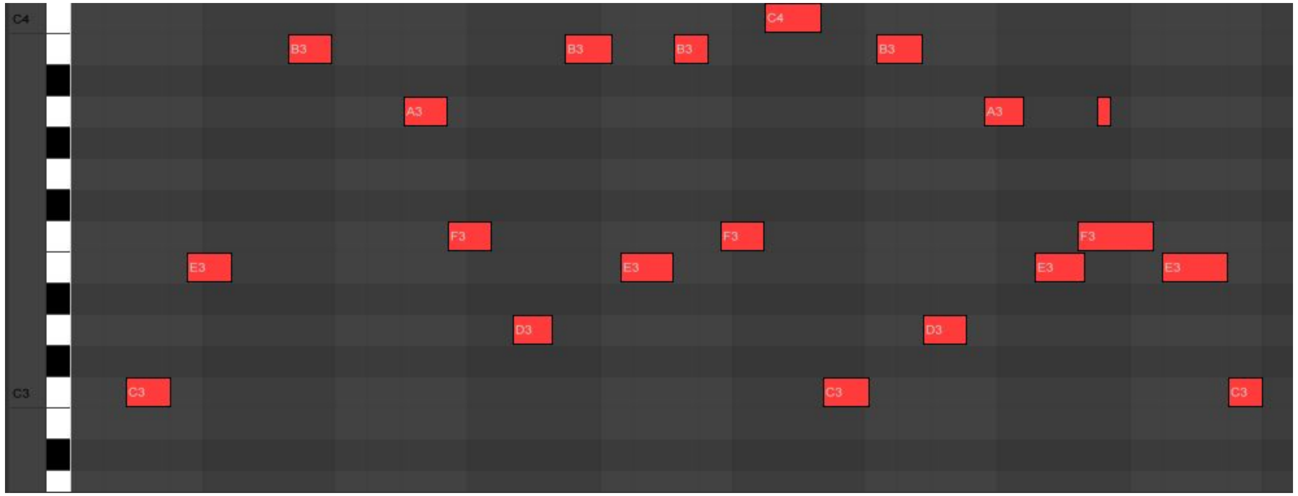
Loss Functions

Sometimes we have sequences like this:

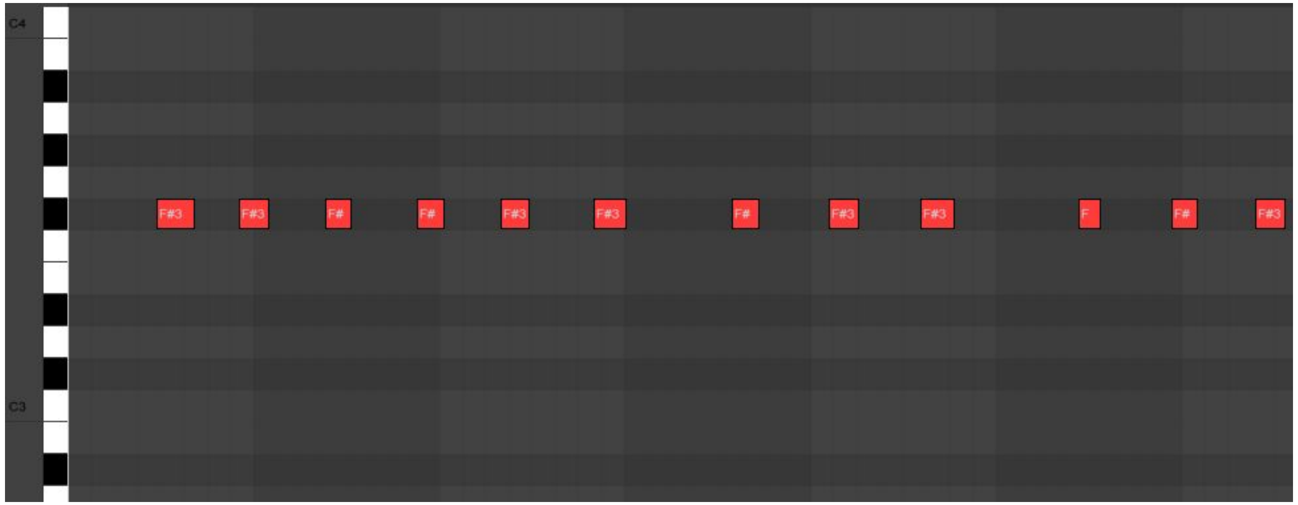


Loss Functions

Sometimes we have sequences like this:



And our models find a nice low loss by doing this:



Loss Functions

$$\alpha > 1$$

Why not punish it for making repeats ?

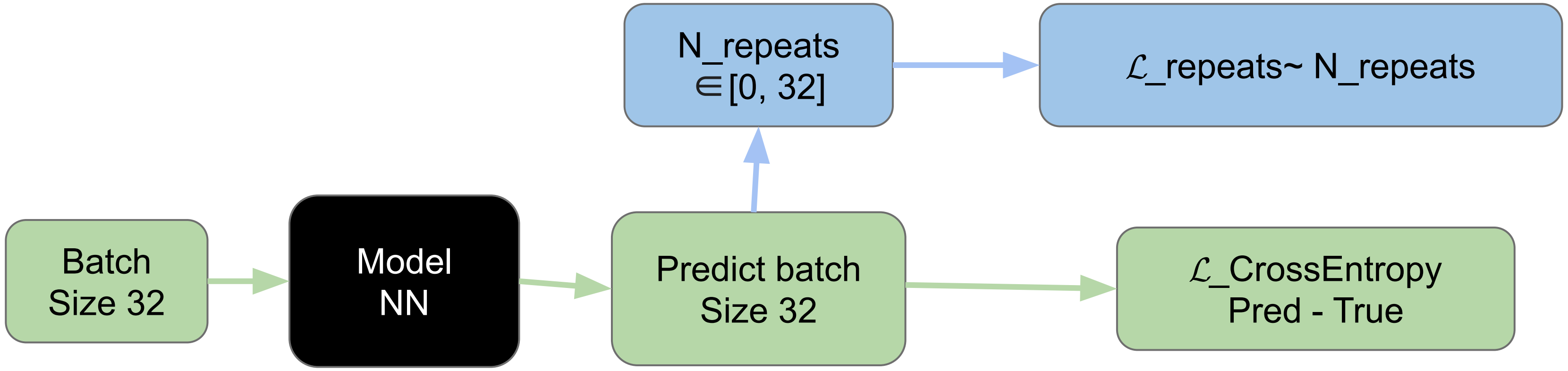
$$\mathcal{L}_{rep} = \lambda \cdot (N_{repeats})^{\alpha}$$

Loss Functions

$\alpha > 1$

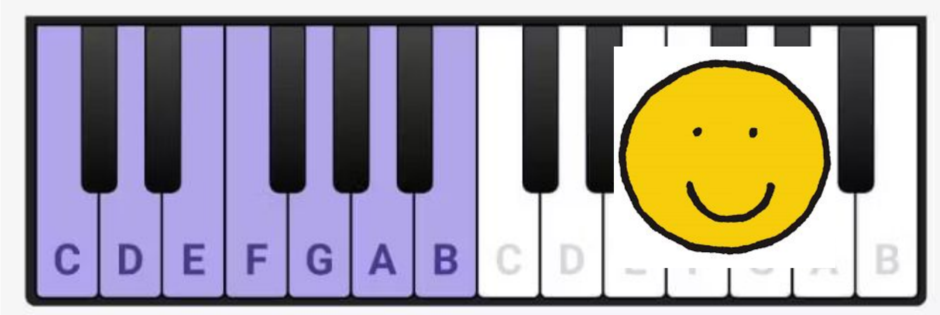
Why not punish it for making repeats ?

$$\mathcal{L}_{rep} = \lambda \cdot (N_{repeats})^{\alpha}$$



Loss functions

Music Theory 101

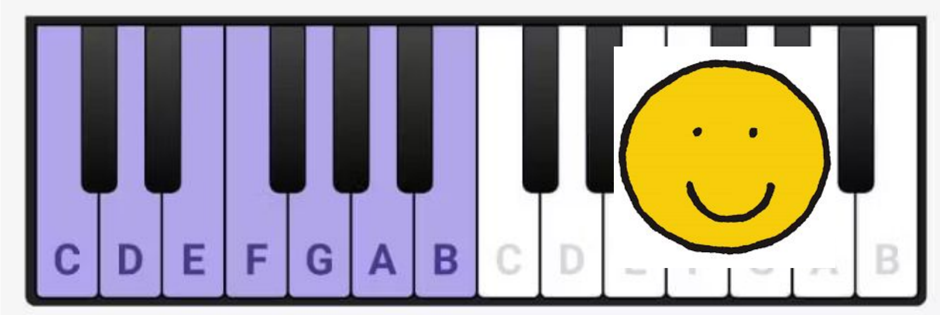


Some notes “like “each other.

These notes are not necessarily right next to each other

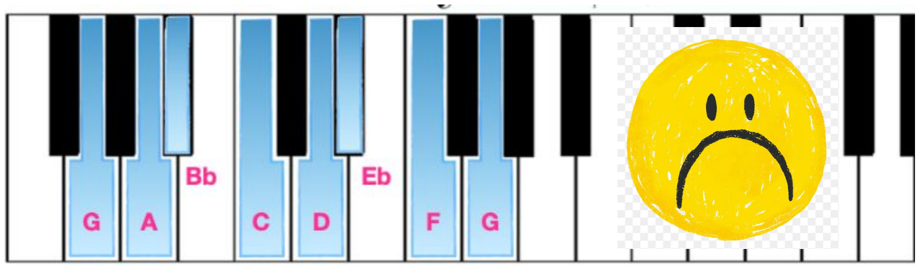
Loss functions

Music Theory 101



Some notes “like “each other.

These notes are not necessarily right next to each other



Notes close to each other, generally sounds worse

A NN does not know this !

It just tries to get as close as possible

Loss Functions

“ Have i seen my prediction in the input “ - loss
Try to produce something similar to the input

$$\alpha > 1$$

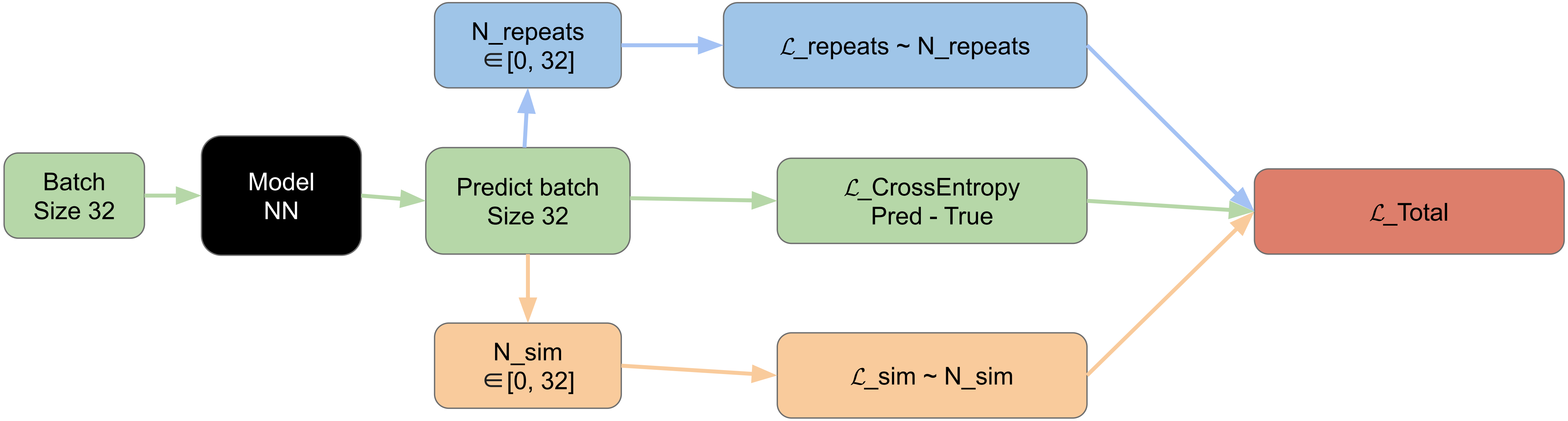
$$\mathcal{L}_{sim} = \lambda \cdot (N_{sim})^{\alpha}$$

Loss Functions

“ Have i seen my prediction in the input “ - loss
Try to produce something similar to the input

$$\alpha > 1$$

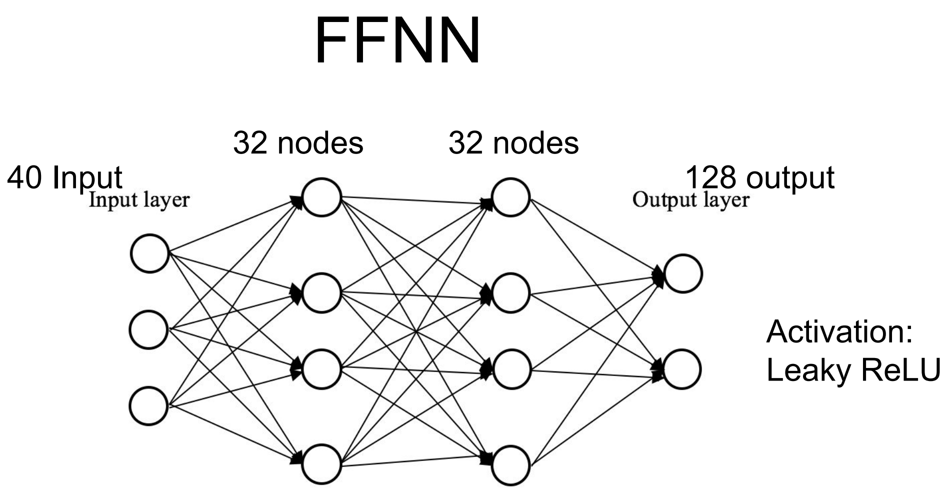
$$\mathcal{L}_{sim} = \lambda \cdot (N_{sim})^{\alpha}$$



FFNN

Let's start with the simplest approach:

We only look at pitch using a small network on a simple and small dataset (lofi → 91 songs → 4K data points)

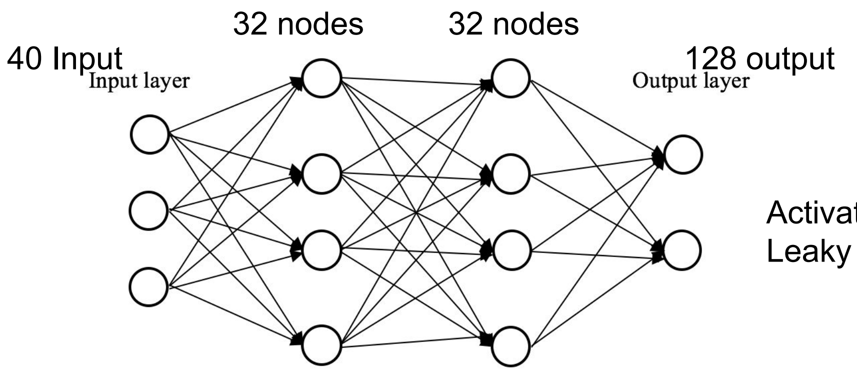


FFNN - Results

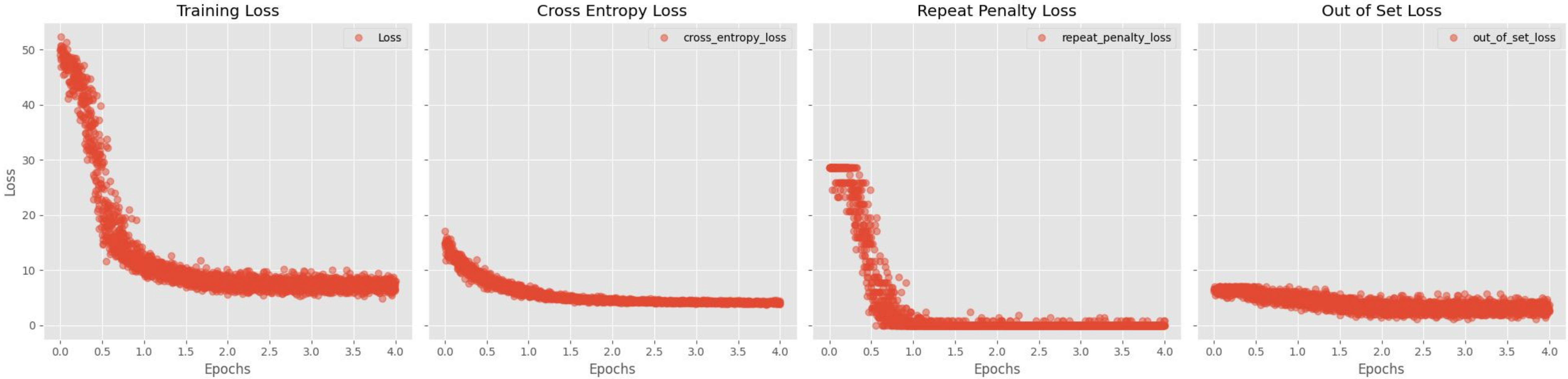
Let's start with the simplest approach:

We only look at pitch using a small network on a simple and small dataset (lofi → 91 songs → 4K data points)

FFNN



Activation:
Leaky ReLU

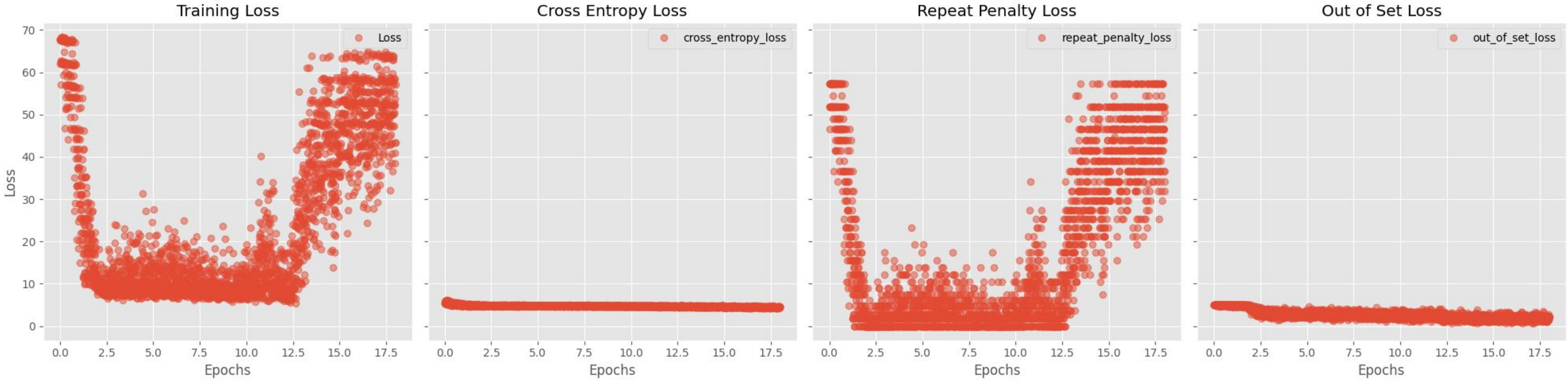


FFNN - Deep Model

For deep models we observe faster overtraining.

Converges to sequences of the same note

We don't get to do a lot of epochs for fine tuning, before it blows up



Other things we tried

- Long-Short Term Memory (LSTM)
- Variational Autoencoder (VAE)

Long-Short Term Memory (LSTM)

Captures patterns in long term memory

Long-Short Term Memory (LSTM)

Captures patterns in long term memory

Works on sequential data

Long-Short Term Memory (LSTM)

Captures patterns in long term memory

Works on sequential data

MID can be preprocessed to sequential data

Long-Short Term Memory (LSTM)

Captures patterns in long term memory

Works on sequential data

MID can be preprocessed to sequential data

Success?!

Long-Short Term Memory (LSTM)

Let's try something new!

Focus on the melody

Before

[velocity, pitch, time]

After

[interval, duration]

Long-Short Term Memory (LSTM)

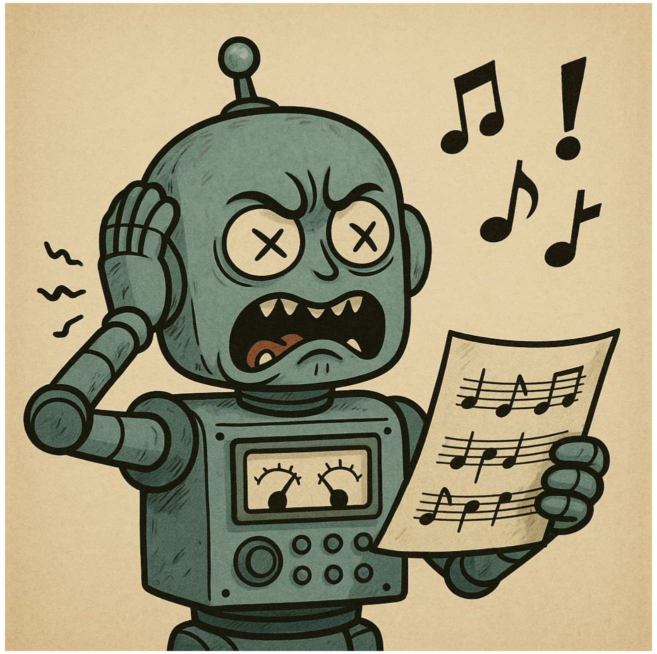
Let's try something new!

Focus on the melody

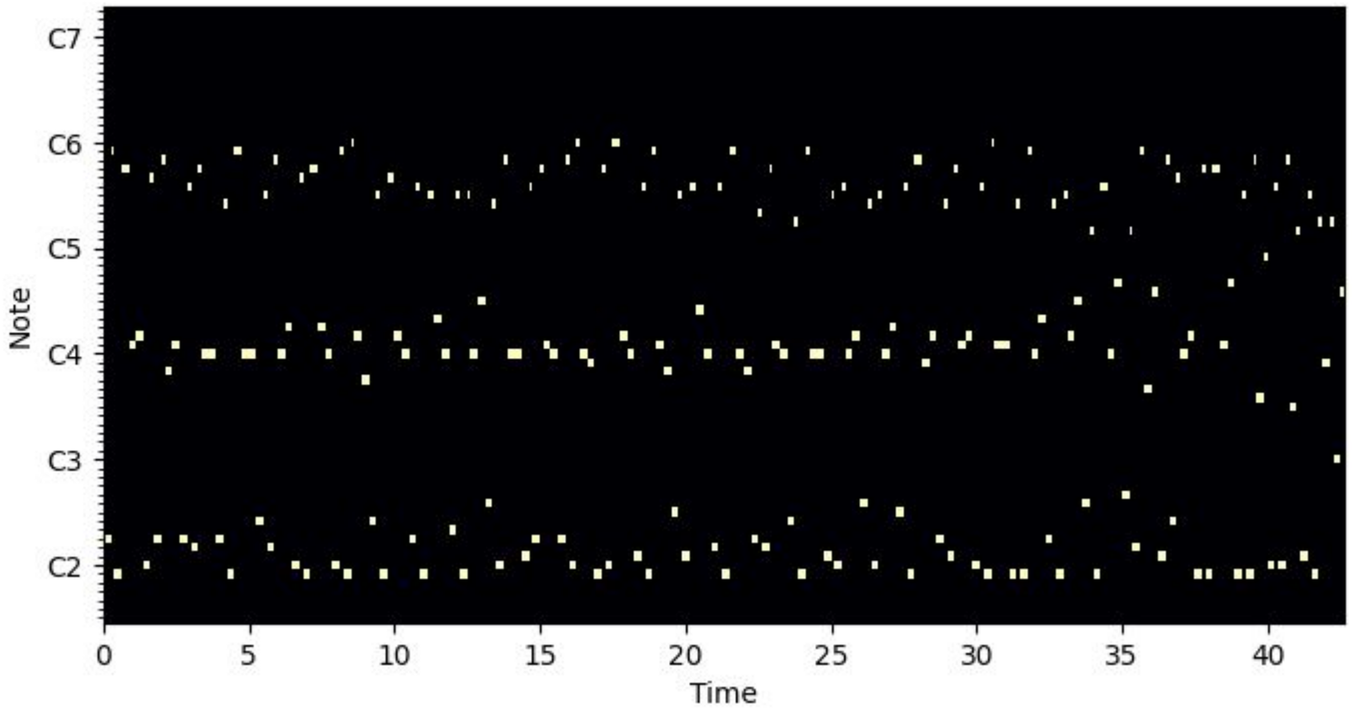
A tone-deaf LSTM

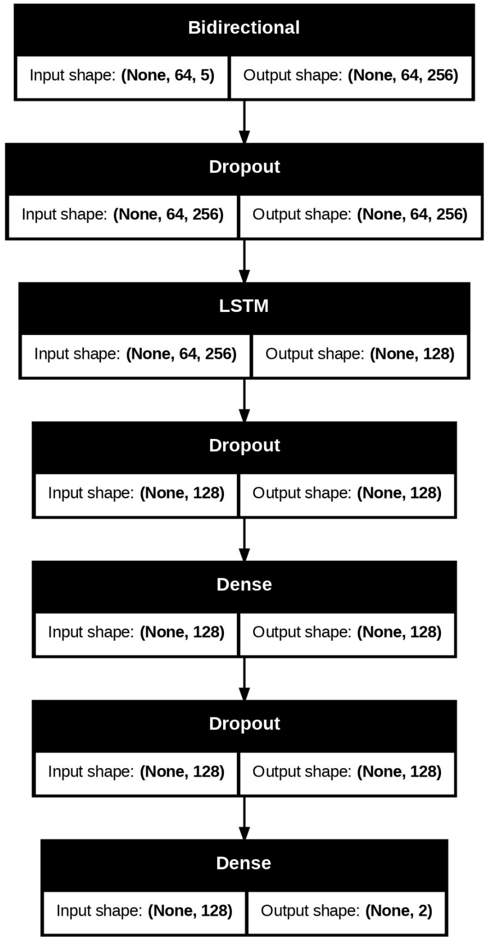
Before

[velocity, pitch, time]



Long-Short Term Memory (LSTM)

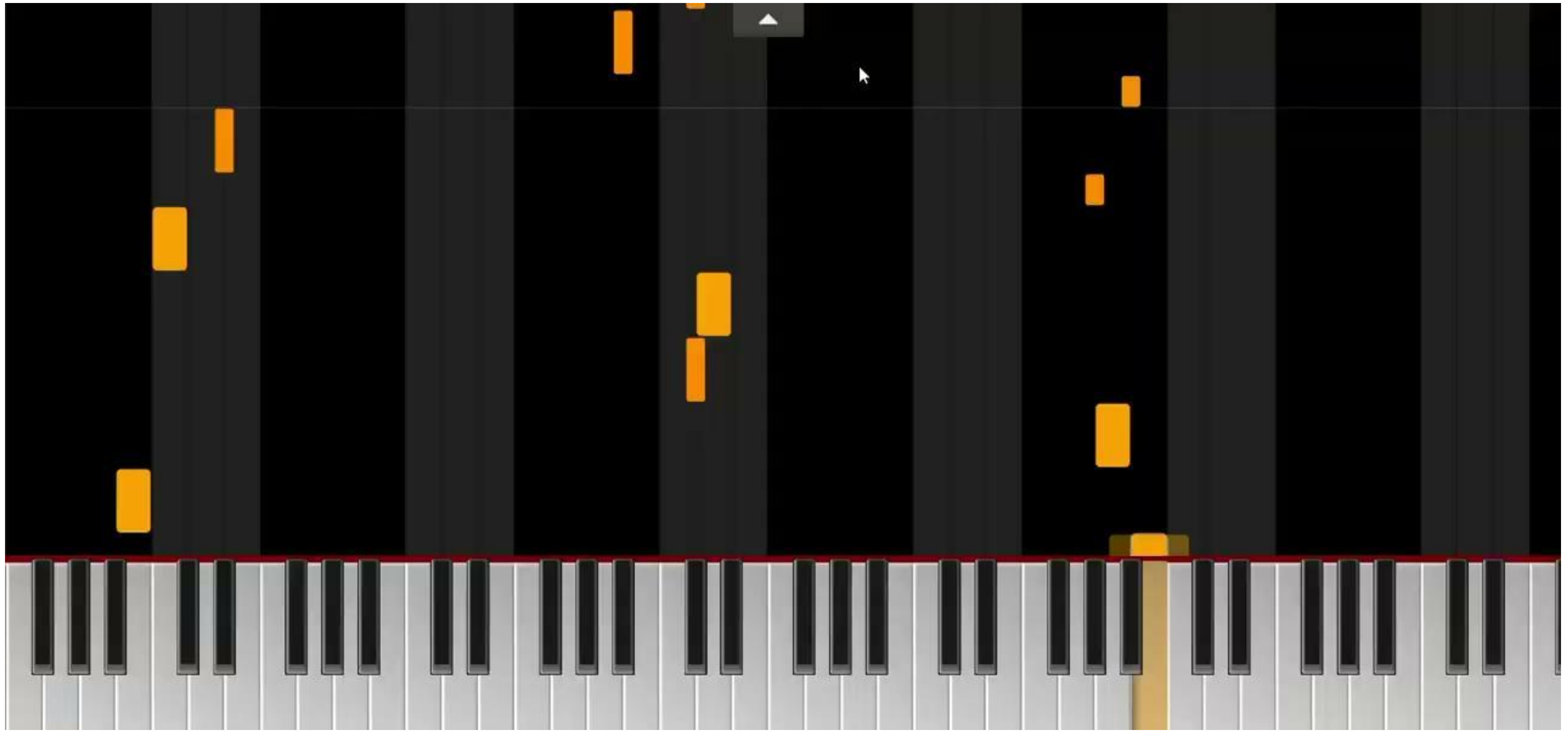




* Input data is seeded from training data

* Output is [interval, duration]

Long-Short Term Memory (LSTM)



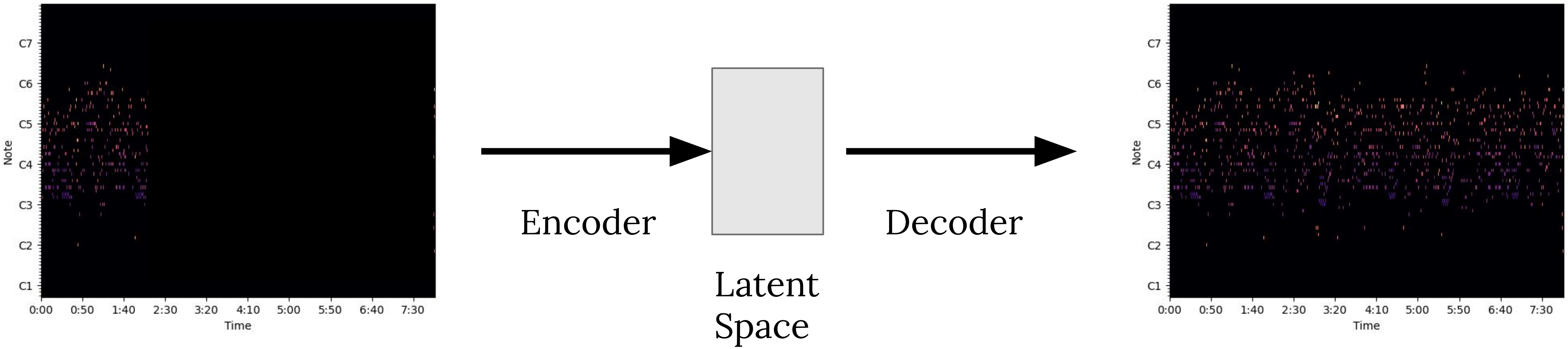
* Input data is seeded from training data

Variational Autoencoder (VAE)

Idea: Train a VAE by letting it guess the remaining part of a song

Variational Autoencoder (VAE)

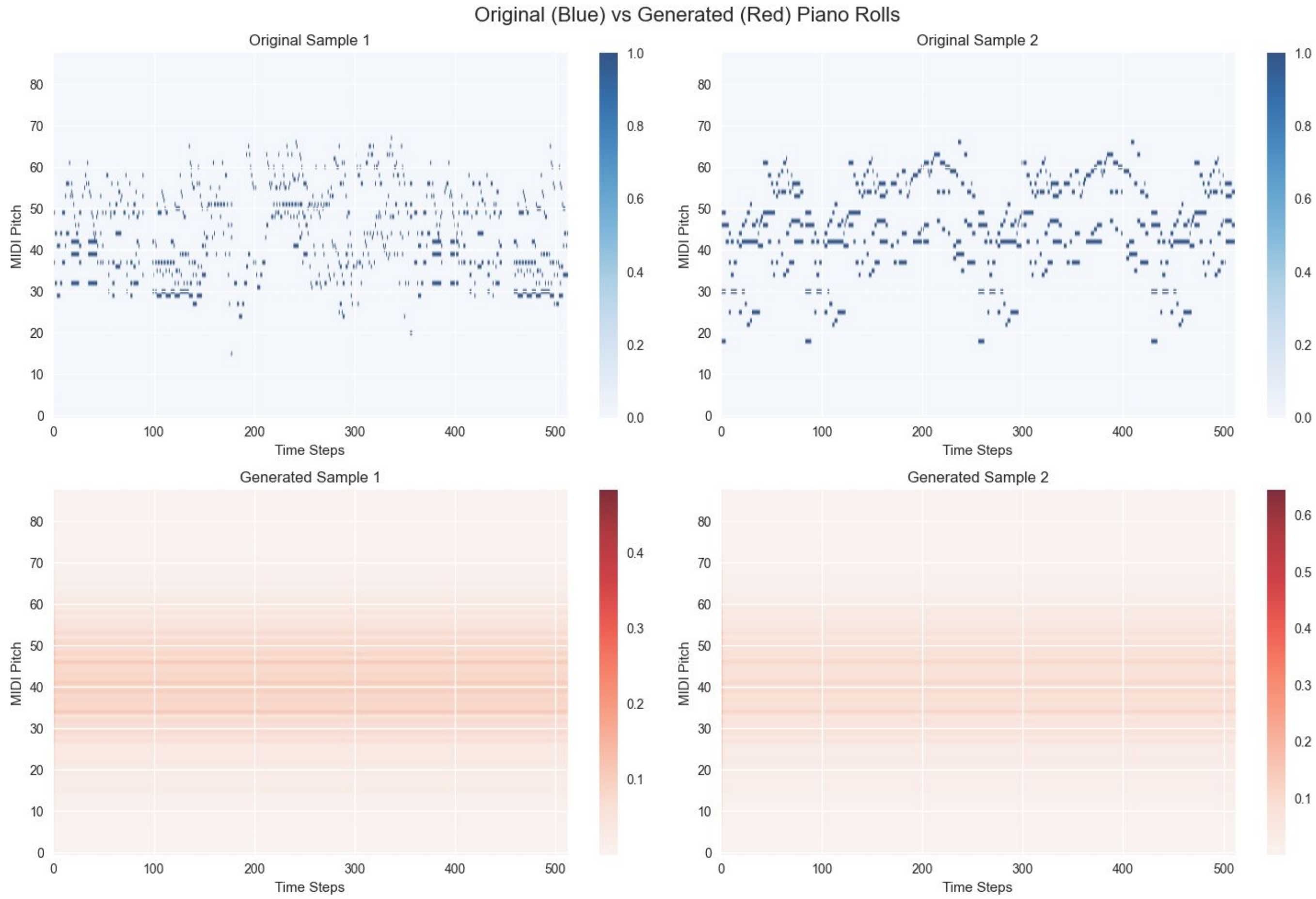
Idea: Train a VAE by letting it guess the remaining part of a song



Variational Autoencoder (VAE)

Idea: Train a VAE by letting it guess the remaining part of a song

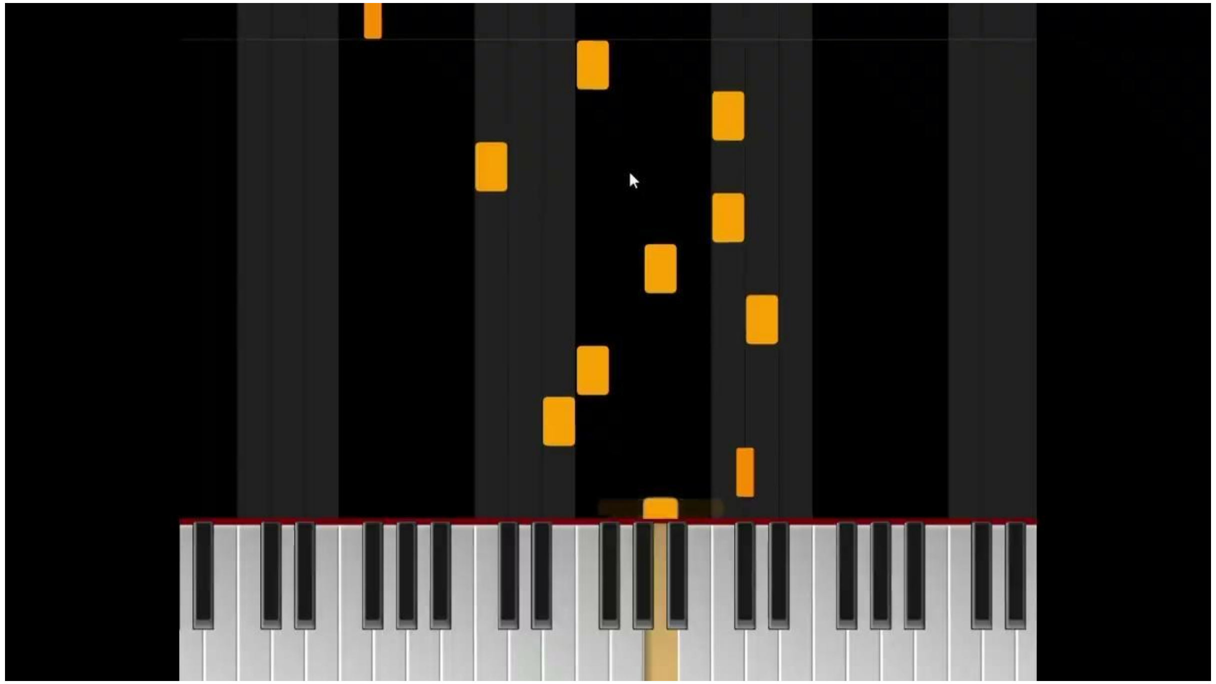
Reality: Sparse matrix representations don't work that well and continuous melodies generate low losses



Examples of Generated “Music”

FFNN with modified loss

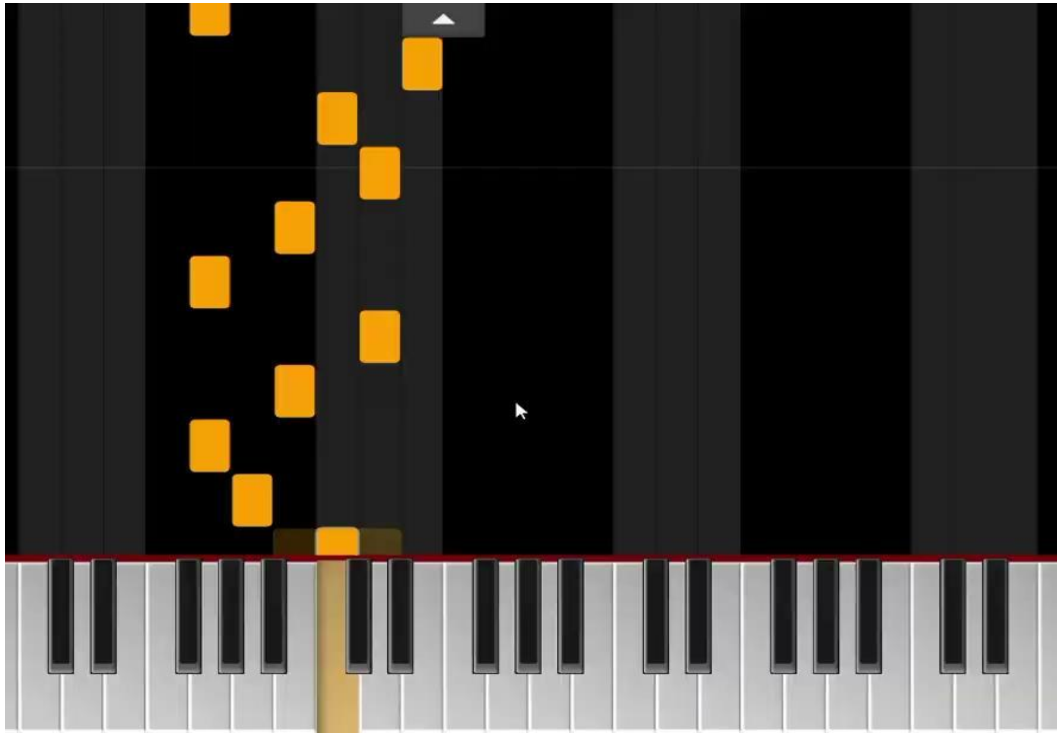
Input from training songs
80 note short term memory



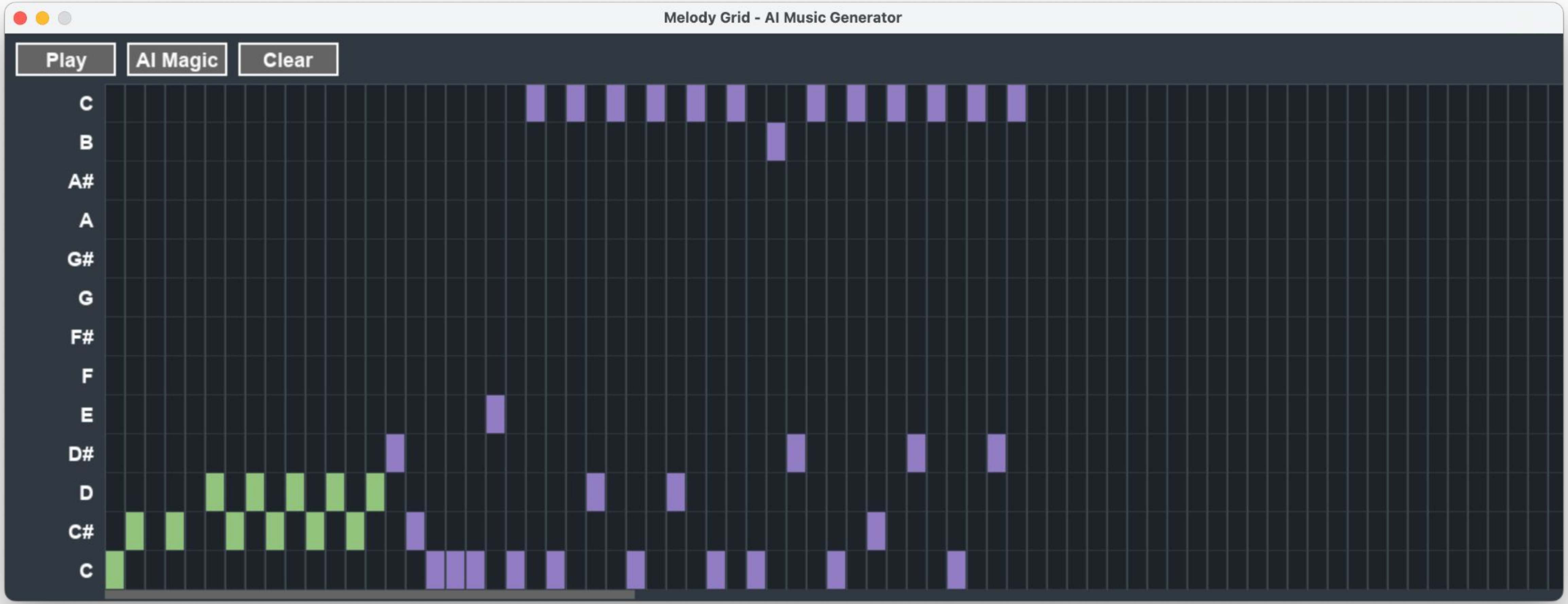
FFNN with modified loss

Input from training songs
80 note short term memory

Using a custom input
40 note short term memory



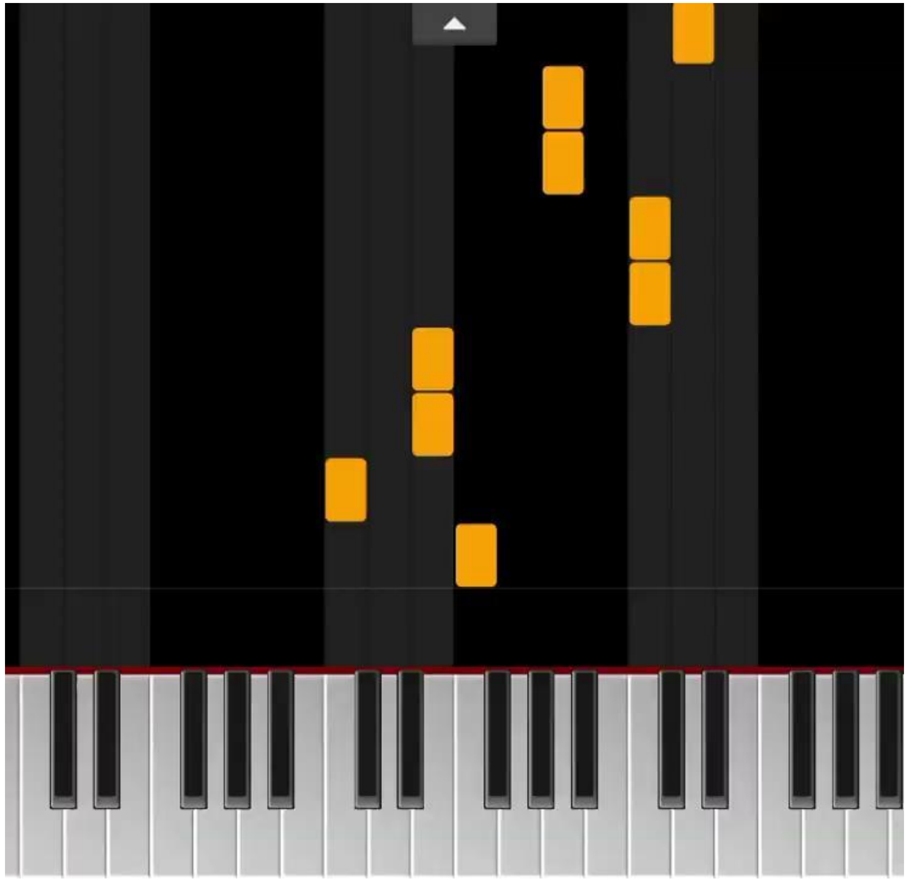
Live Demo


Melody Grid - AI Music Generator
Play
AI Magic
Clear
C
B
A#
A
G#
G
F#
F
E
D#
D
C#
C

Results

FFNN with modified loss

Convergence & overfitting

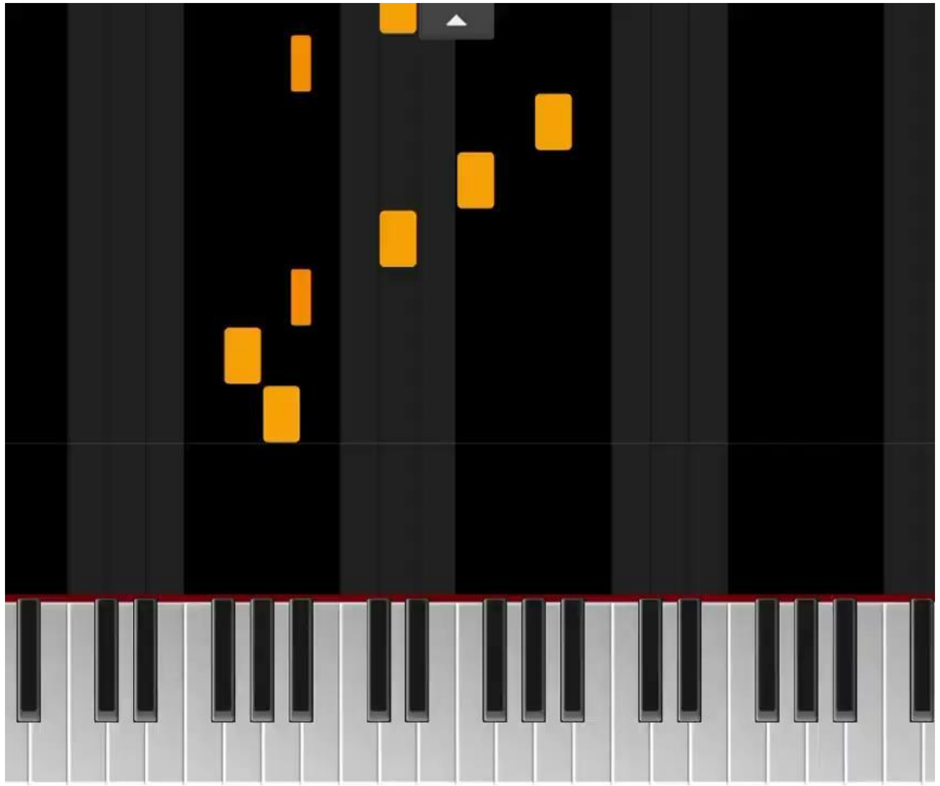


Results

FFNN with modified loss

Convergence & overfitting

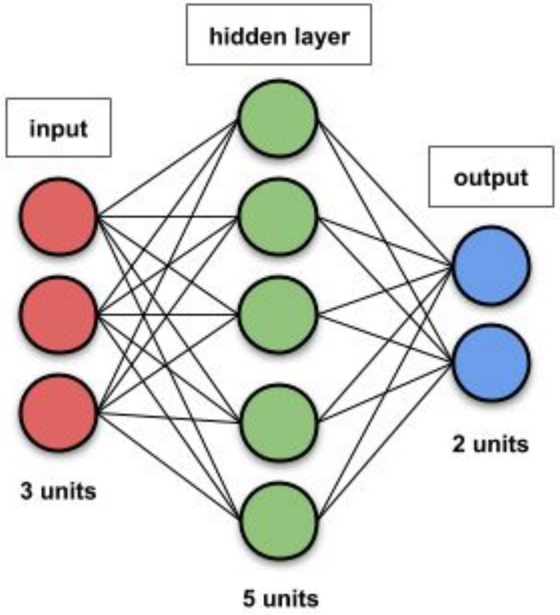
Underfitting



Takeaways

All models are equal but some are better

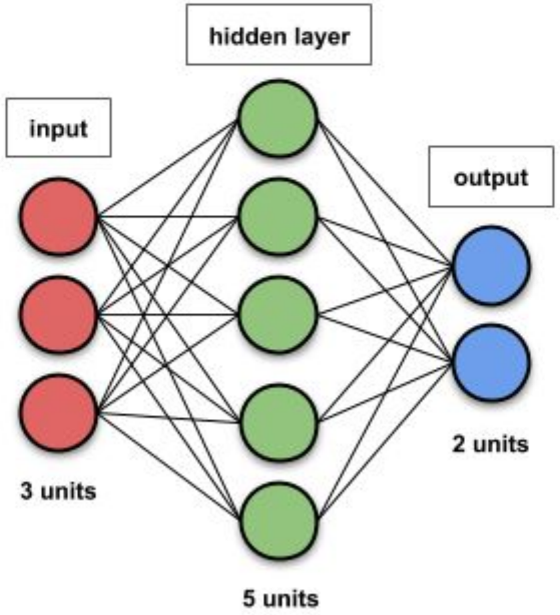
NN have potential



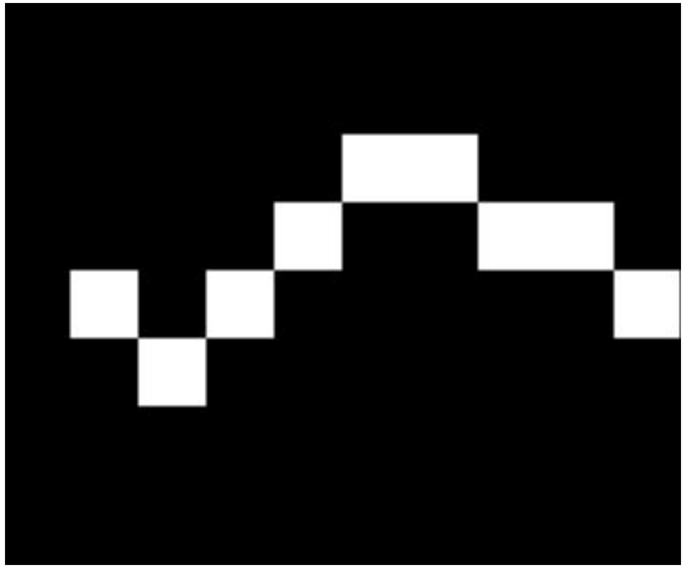
Takeaways

All models are equal but some are better

NN have potential



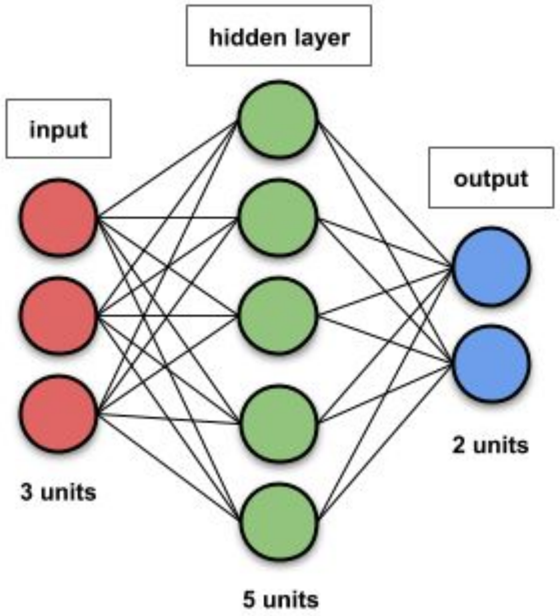
Data representation matters !



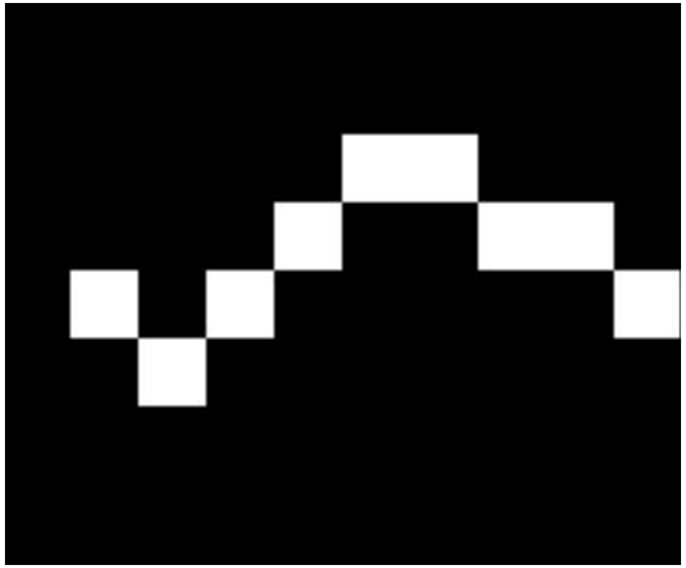
Takeaways

All models are equal but some are better

NN have potential

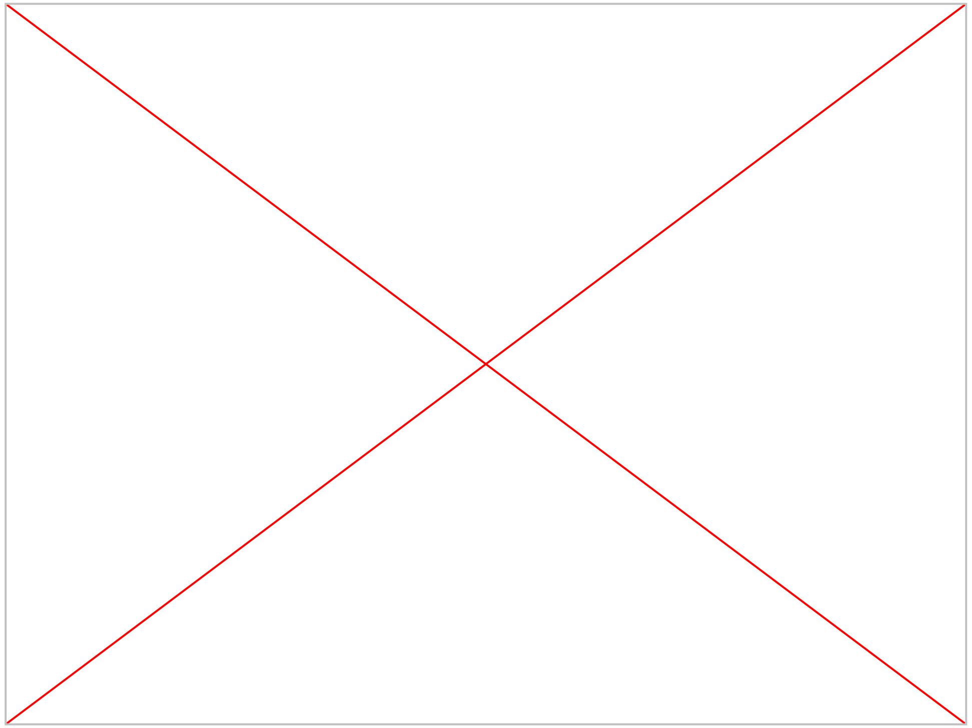


Data representation matters !



Don’t do tree based music


HONORABLE MENTION



Thank you

Appendix

Representing MIDI Data for Machine Learning

- Extracting Musical Features with music21

✅ Pros

- Reduces the dimensionality of each MIDI file – only 42–52 entries per dictionary, compared to thousands of raw events
- The compact and interpretable feature dictionaries make the data representation well-suited for unsupervised learning tasks like clustering

❌ Cons

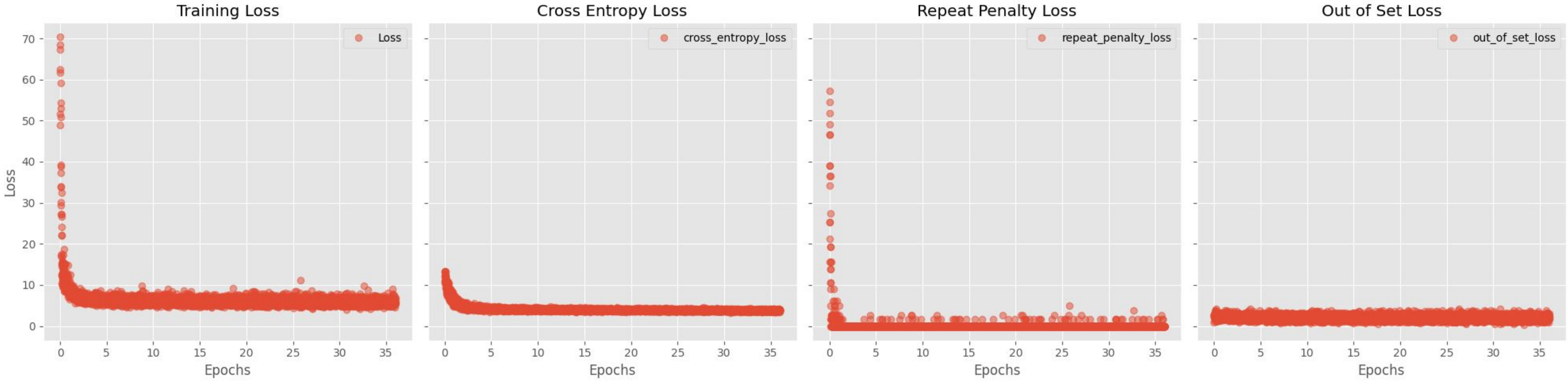
- Note order and rhythmic nuances are flattened — the music cannot be reconstructed, so this representation is unsuitable for music generation
- The built-in feature extraction in music21 is complex and relatively slow, especially for large datasets

FFNN - Big Layers

For too many nodes, we are not learning music features

Output sounds random and *out of tune*

We can keep training to reduce loss, but the output never “sounds like musik”



Teaser: There will be “good” songs in the end. Stay tuned

Representing MIDI Data for Machine Learning

Sequence of Events

Quick loading

No sparsity

Easy data handling

Fixed input size

No long term memory

Little musical structure (key, chords)

LSTM notes

- Trained differently than the FFNN.
- Focus on getting the interval right, not the pitch.
    - Therefore the LSTM is ‘tone deaf’
- A custom loss function is implemented to focus on the melody.

```python
def custom_loss(y_true, y_pred):
    interval_true = y_true[:, 0]
    interval_pred = y_pred[:, 0]
    duration_true = y_true[:, 1]
    duration_pred = y_pred[:, 1]

    # Base losses
    interval_loss = tf.keras.losses.mse(interval_true, interval_pred)
    duration_loss = tf.keras.losses.mse(duration_true, duration_pred)

    # Context penalty: large interval jumps are penalized
    if len(interval_pred.shape) > 1:
        # Assuming batch dimension exists
        interval_diffs = tf.abs(interval_pred[1:] - interval_pred[:-1])
        large_jump_penalty = tf.reduce_mean(
            tf.maximum(0.0, interval_diffs - 12.0)  # Penalize jumps > octave
        )
    else:
        large_jump_penalty = 0.0

    return 2.0 * interval_loss + duration_loss + 0.1 * large_jump_penalty
```

$$\mathcal{L}_{\text{musical}} = 2.0 \cdot \mathcal{L}_{\text{interval}} + \mathcal{L}_{\text{duration}} + 0.1 \cdot \mathcal{L}_{\text{jump}}$$

VAE notes

- Trained on sparse matrices.
- Songs are clipped to 512 ‘time steps’.
- As we realized no-one was doing this we focused on LSTM and FFNN instead.
- **Result**: The VAE generates continuous notes to reduce loss.