2023 October 26

Deep Learning for Music Analysis and Generation

MIDI Generation (I)

(x → MIDI)

Yi-Hsuan Yang Ph.D.
yhyangtw@ntu.edu.tw

Objectives

- **Language modeling** (LM) for symbolic-domain music generation
  - To account for ***long-range dependency***
  - Consider “musical events” (e.g., beginning and releasing of a note) as a **token**
  - Then build an LM
    - By **RNN**
    - By **Transformer**

```
bach piano_strings start tempo90 piano:v72:G1
piano:v72:G2 piano:v72:B4 piano:v72:D4 violin:v80:G4
piano:v72:G4 piano:v72:B5 piano:v72:D5 wait:12
piano:v0:B5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:4 piano:v0:G1 piano:v0:G2 piano:v0:B4 piano:v0:D4
violin:v0:G4 piano:v0:G4 wait:1 piano:v72:G5 wait:12
piano:v0:G5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:5 piano:v72:B5 wait:12
```

- **Token design**

- Symbolic-domain music generation with song-level structure: next lecture

Reference: Three ISMIR Tutorials

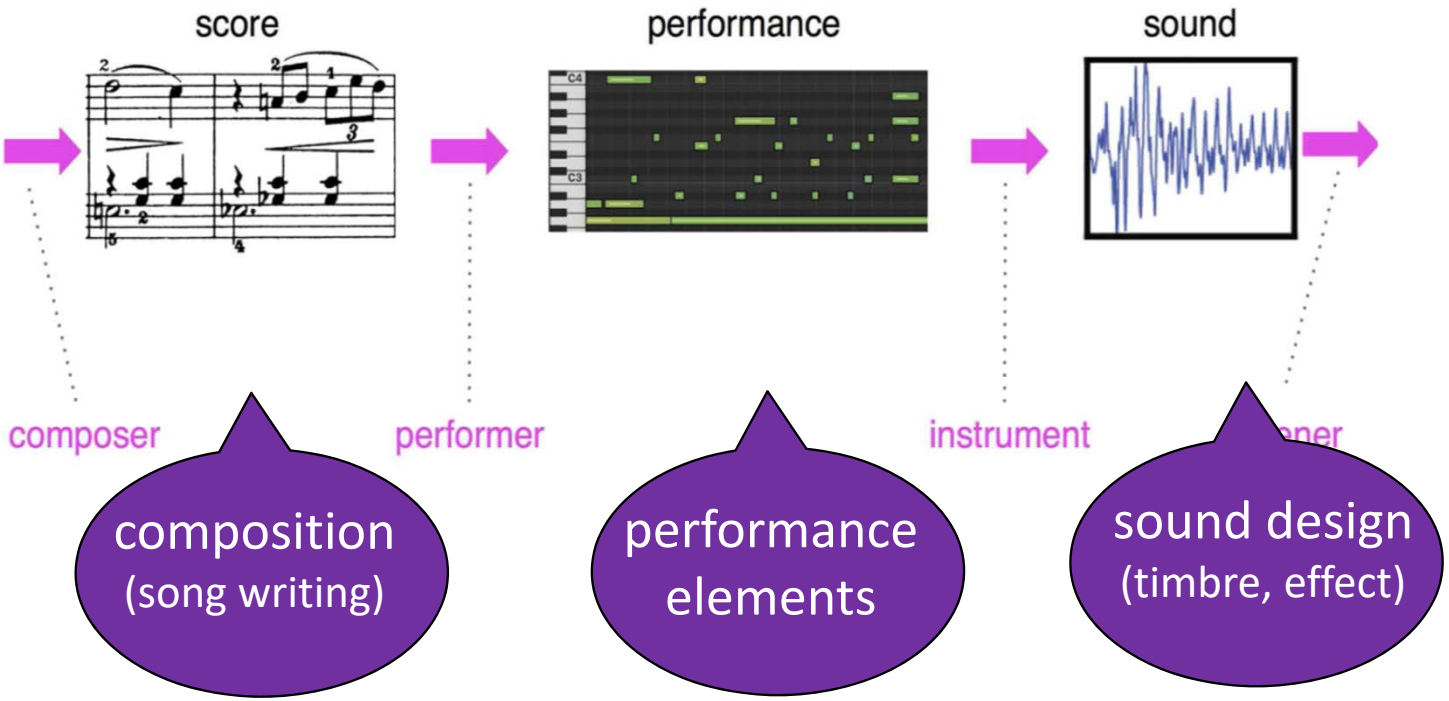
Tutorials at the International Society for Music Information Retrieval (ISMIR)

- "**Machine-Learning for Symbolic Music Generation"** by Pierre Roy (Spotify) and Jean-Pierre Briot (Sony CSL) at **ISMIR 2017**
  https://ismir2017.ismir.net/tutorials/index.html#T4
- "**Generating Music with GANs—An Overview and Case Studies**" by Hao-Wen Dong and Yi-Hsuan Yang (Academia Sinica) at **ISMIR 2019**
  https://salu133445.github.io/ismir2019tutorial/
- "**Designing Generative Models for Interactive Co-creation**" by Anna Huang (Google), Jon Gillick (UC Berkeley), Chris Donahue (Stanford) at **ISMIR 2021**
  https://github.com/chrisdonahue/music-cocreation-tutorial

Outline

- **General ideas**
- Image-based approach for symbolic-domain music generation
- Text-based approach for symbolic-domain music generation
  - RNN
  - Transformer
- Token design
- Building your first MIDI Transformer

Music Production Pipeline



Types of Music Generation Research

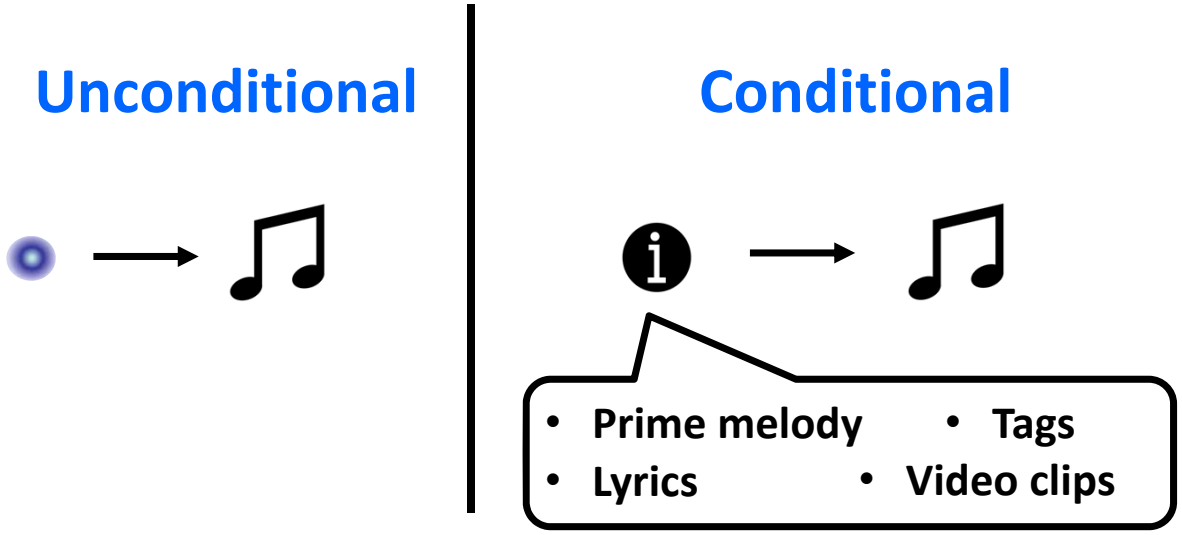
1. **Composition** (symbolic): MIDI scores (or other types of symbolic format)
2. **Performance** (symbolic): MIDI performances (scores + expression)
   a. Generate MIDI performances *from scratch* (i.e., generate scores and their expression at the same time)
   b. Given MIDI score, generate MIDI performance (e.g., VirtuosoNet [1])
3. **Audio**
   a. Generate audio *from scratch*
   b. Given MIDI, generate audio

Will talk about **1** and **2a** in this lecture

[1] Jeong et al, "VirtuosoNet: A hierarchical RNN-based system for modeling expressive piano performance," ISMIR 2019

Unconditional or Conditional Generation

- The generation of MIDI scores can also be conditional

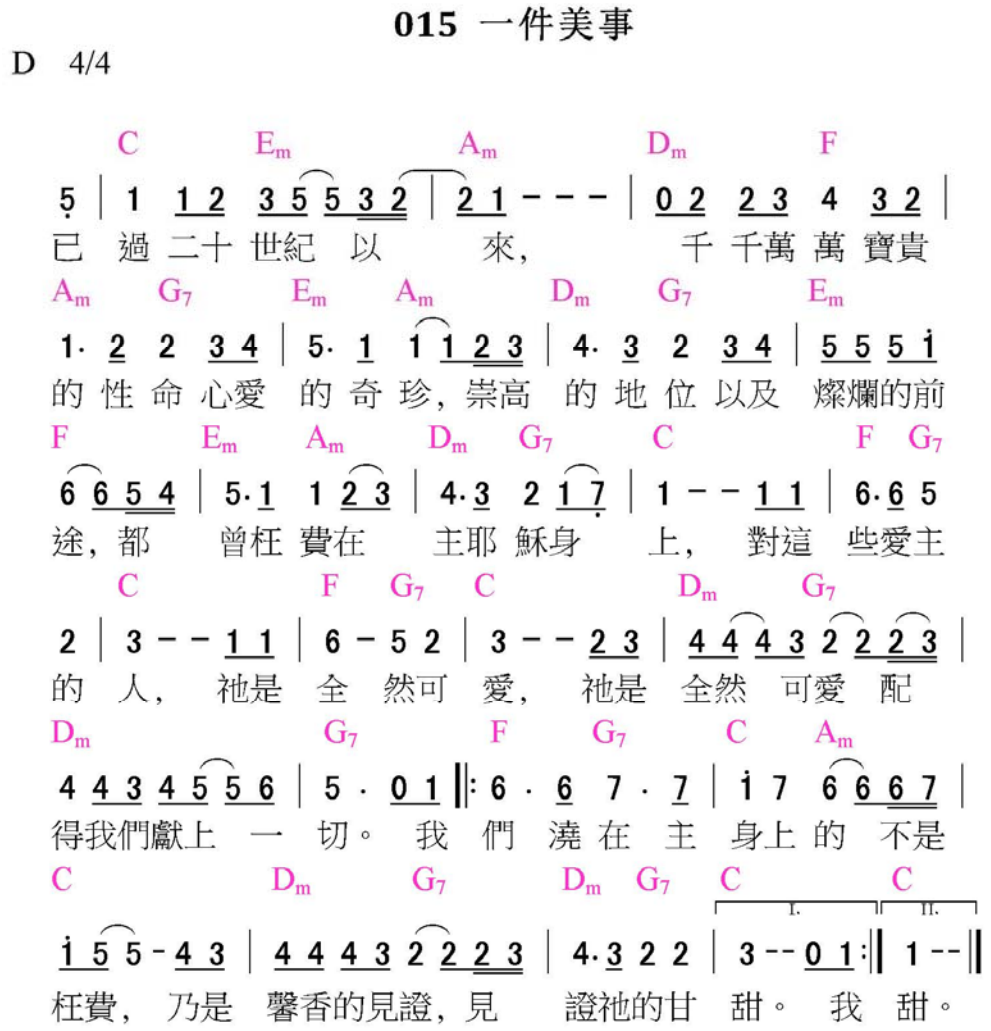


(Figure made by Hao-Min Liu)

Lead Sheet Generation

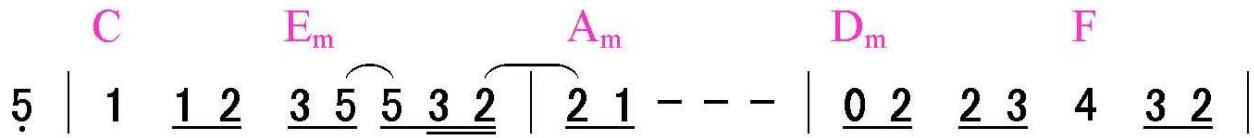
- Lead sheet
  - melody
  - chord
  - (lyrics)

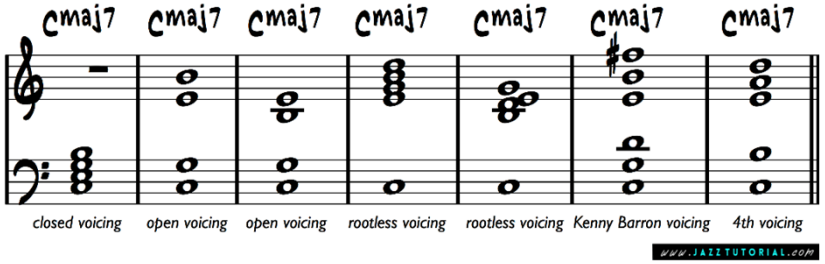
- Possible generation tasks
  - Given chord, generate melody
  - Given melody, generate chord (*melody harmonization*)
  - Or, generate both from scratch



Task Complexity: From Composition to Performance

- Lead sheet





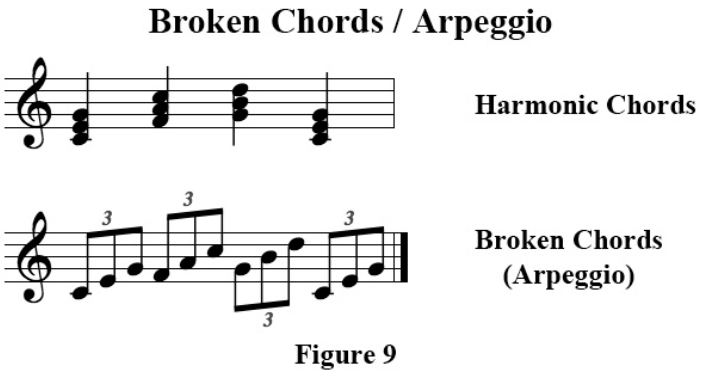
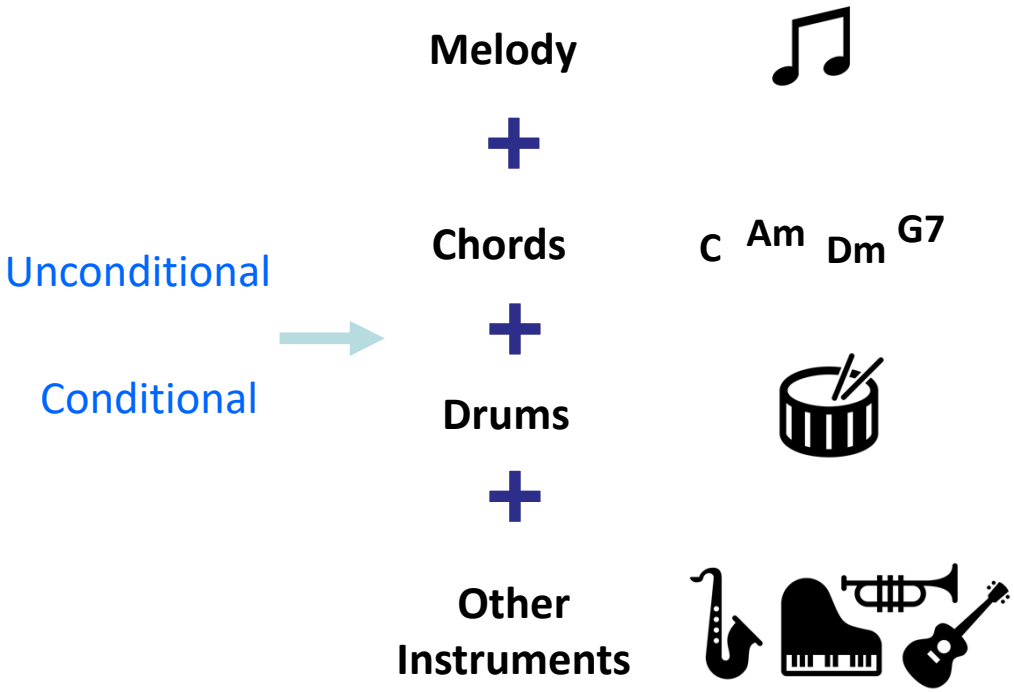


Figure 9

- Performance: expressive timing and dynamics

Task Complexity: From Single Track to Multiple Tracks

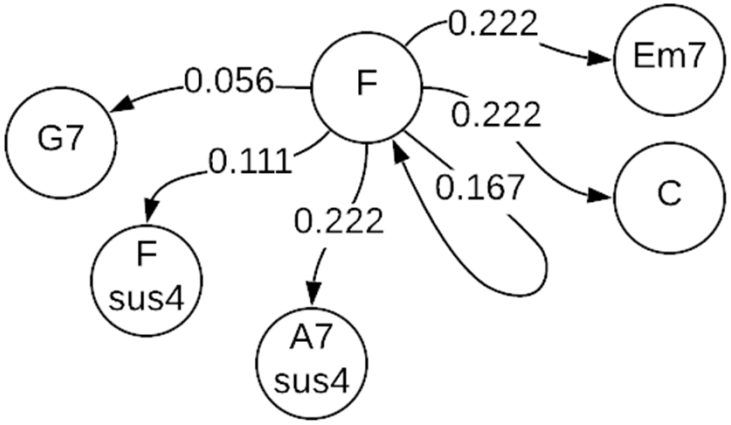
- Melody generation → piano generation → accompaniment generation



(Figure made by Hao-Min Liu)

Algorithmic Composition in the Old Days

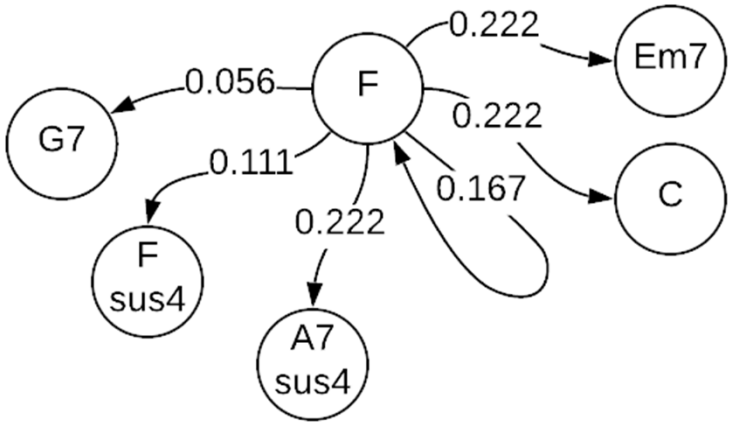
- Based on **probabilities**
- Given "x_{t-1}" (the last one), predict "x_t" (the current one)



https://towardsdatascience.com/markov-chain-for-music-generation-932ea8a88305

Algorithmic Composition in the Old Days

- **Rule-based**
  - safe but rigid
- **Statistical-based (e.g., Markov chain)**
  - limited flexibility



https://towardsdatascience.com/markov-chain-for-music-generation-932ea8a88305

Automatic Music Composition by Neural Networks

- The relationship between the input and output is modeled through a large number of layers, with “hidden states” that may not have a physical meaning

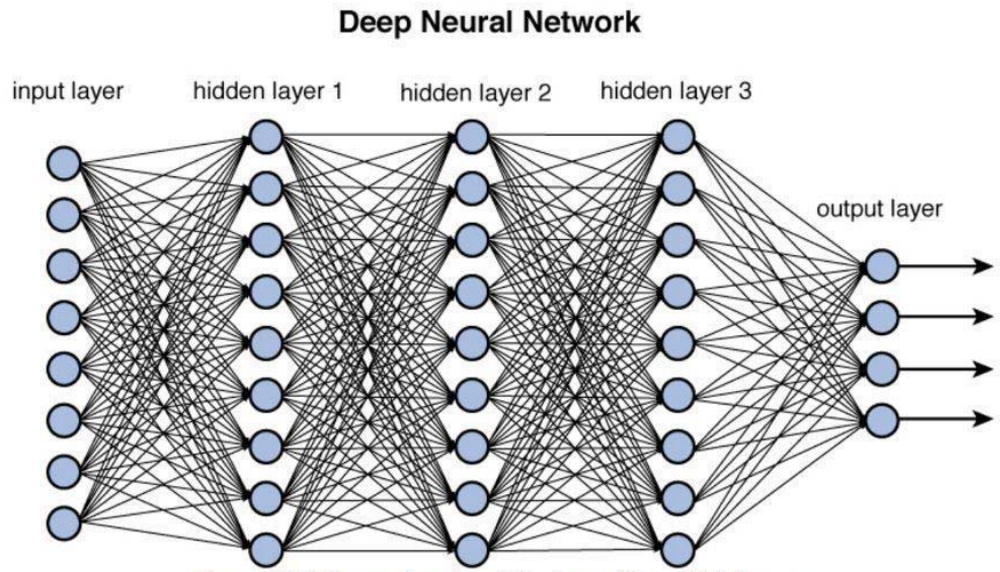
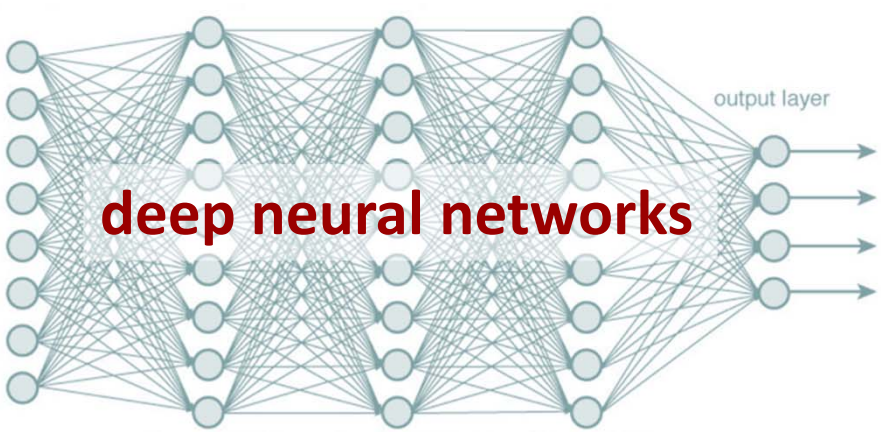


Figure 12.2 Deep network architecture with multiple layers.

https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Autoregressive Models for Music Generation

- Based on **probabilities**
- ~~Given "x_{t-1}" (the last one), predict "x_t" (the current one)~~



- Given "x_1, …, x_{t-2}, x_{t-1}" (all the past), predict "x_t" (the current one)

Autoregressive Models for Music Generation

- A piece of music is considered as a sequence of "**events**"
- A model is trained to predict "**what comes next**", given the history of previous events

$$P(x_t \mid x_{t-1}, x_{t-2}, x_{t-3}, \ldots), \; x_t \in S$$

a finite collection of possible events

"As we listen to melodies, our brain also guesses what's next"
https://bigthink.com/surprising-science/music-brain-predict

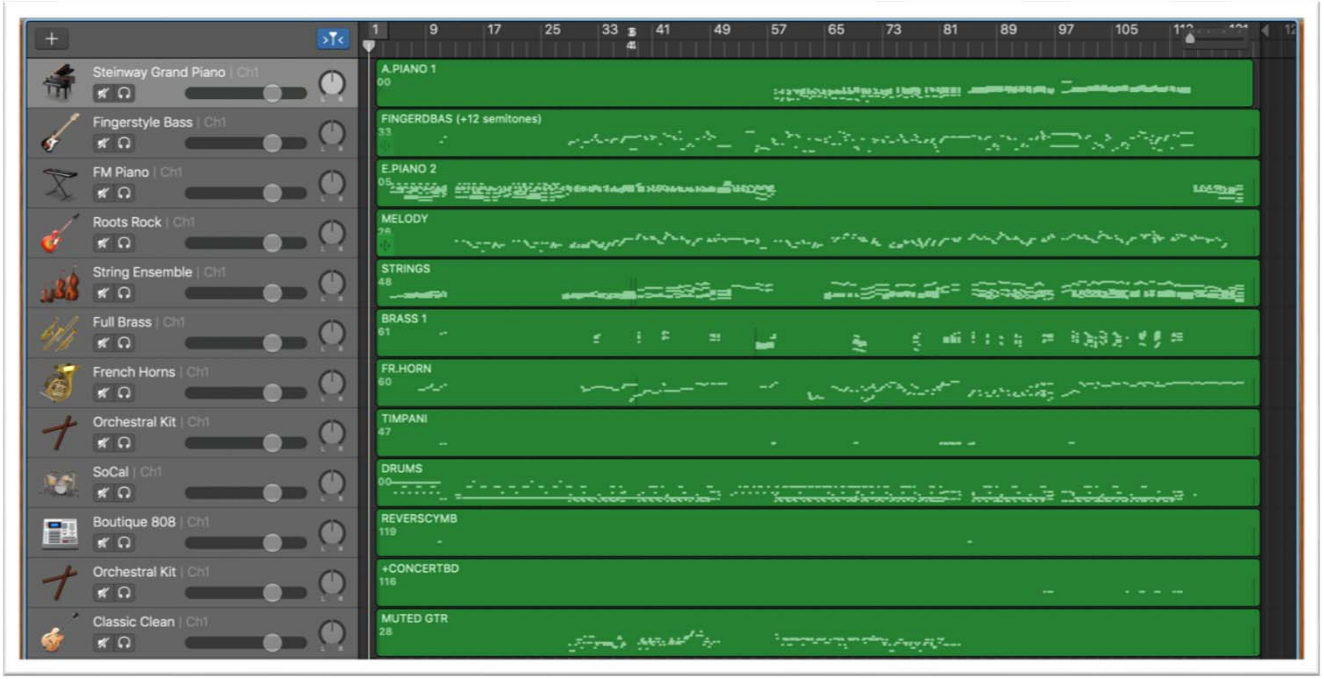
- Given a **prompt**, generate **continuation**
- But there are also *non-autoregressive* approaches

Outline

- General ideas
- **Image-based approach for symbolic-domain music generation**
- Text-based approach for symbolic-domain music generation
  - RNN
  - Transformer
- Token design
- Building your first MIDI Transformer

Two Main Approaches

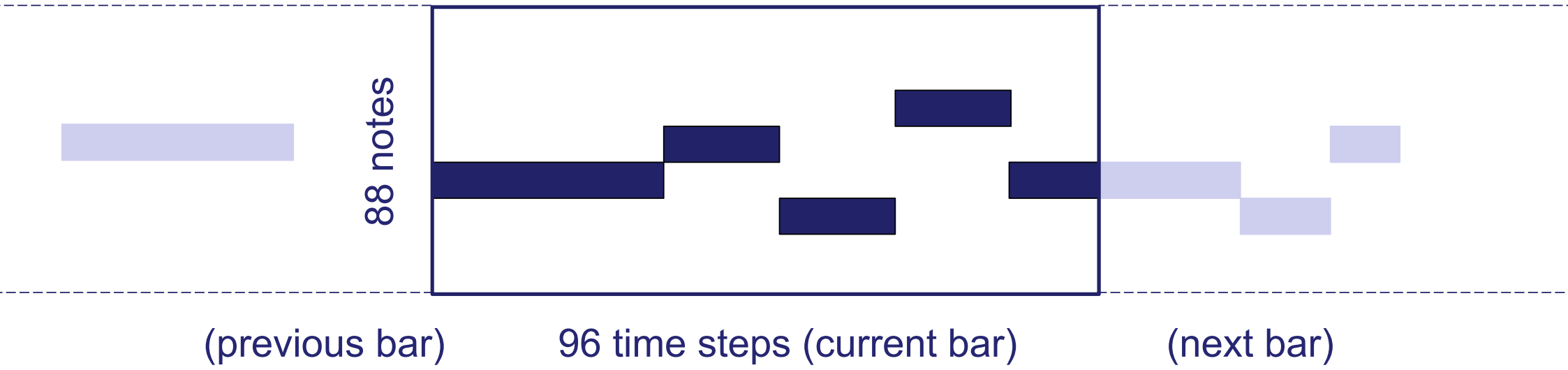
(1) Consider MIDI as **image** using the piano roll representation



(image from the internet)

Piano Roll

Represent each bar of a single-track MIDI as a fixed-size matrix (thus an image)

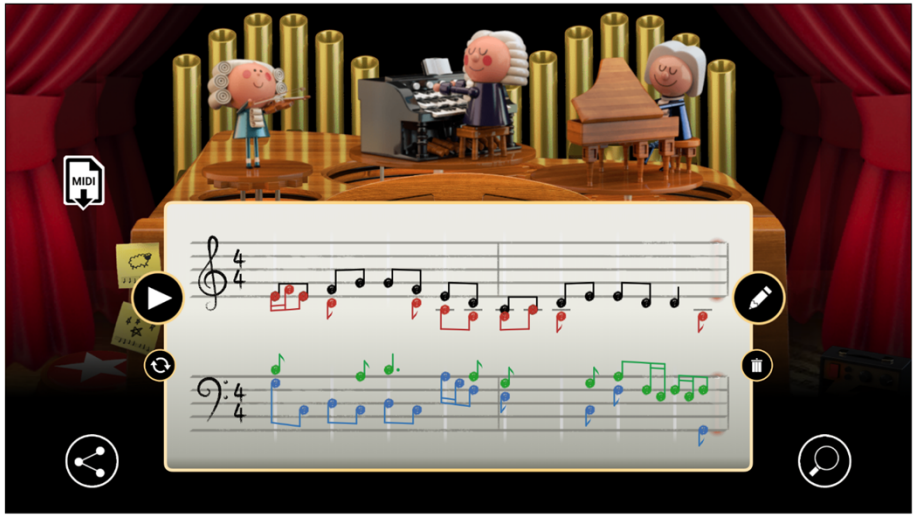


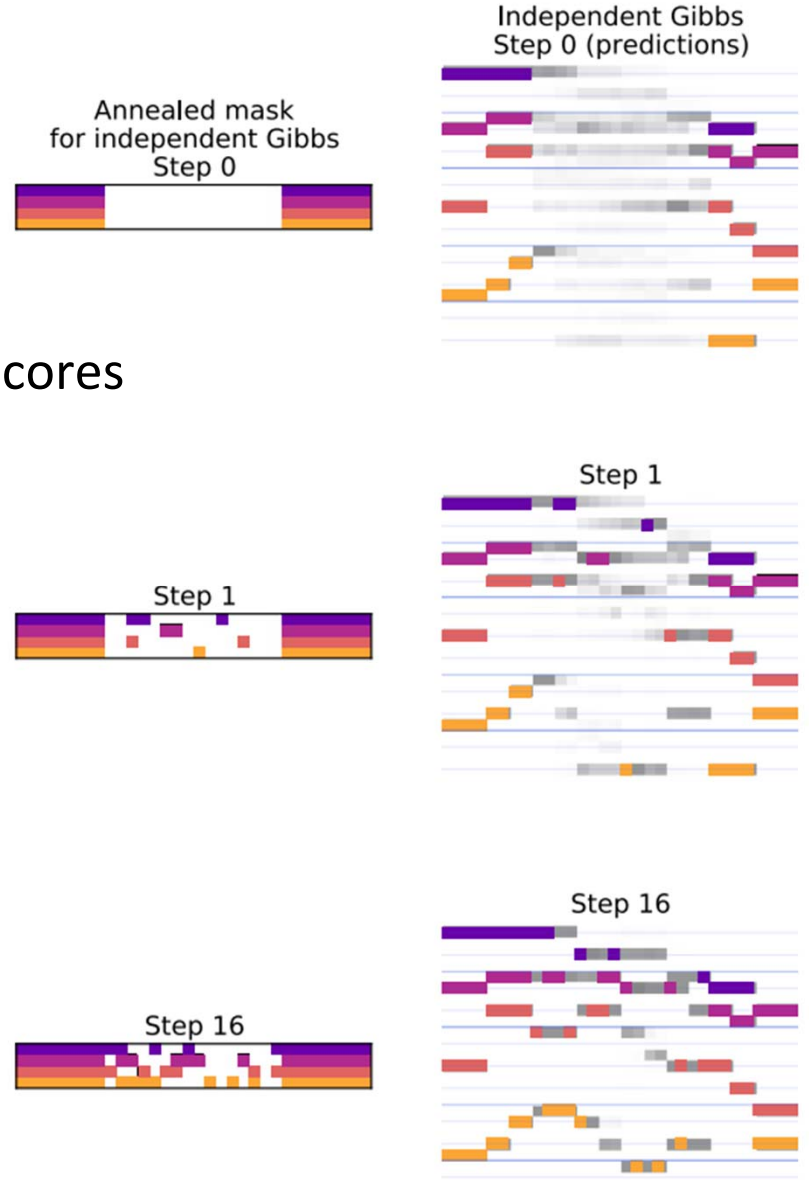
Ref: Dong et al, “Pypianoroll: Open source Python package for handling multitrack pianorolls”, ISMIR-LBD 2018

Coconet & Bach Doodle: Non-GAN CNN-based Approach

https://magenta.tensorflow.org/coconet

- For 2-bar piano roll "**infilling**": to complete *partial* scores
- Blocked-Gibbs sampling (non-autoregressive)



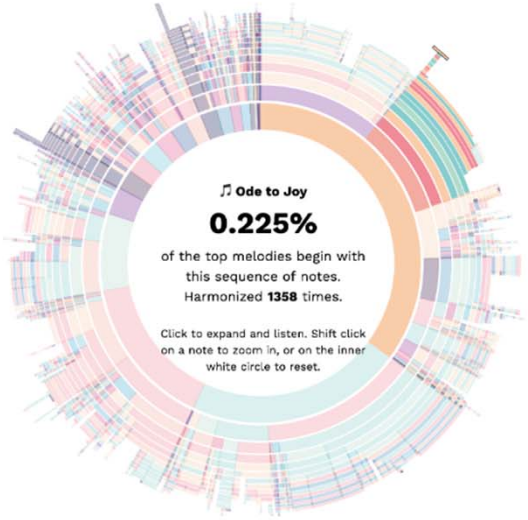


Ref: Huang et al, "Counterpoint by convolution", ISMIR 2017

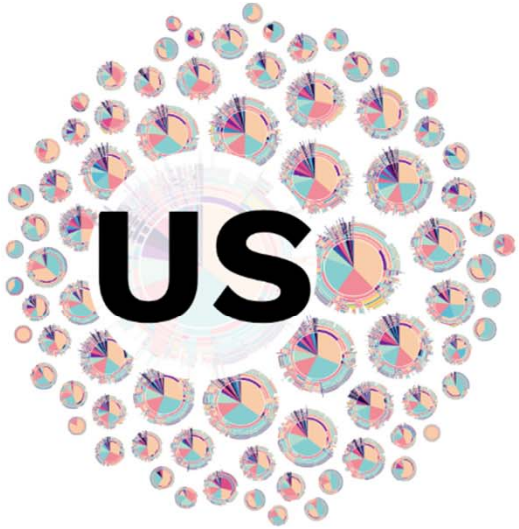
Coconet & Bach Doodle:
Non-GAN CNN-based Approach

https://magenta.tensorflow.org/datasets/bach-doodle

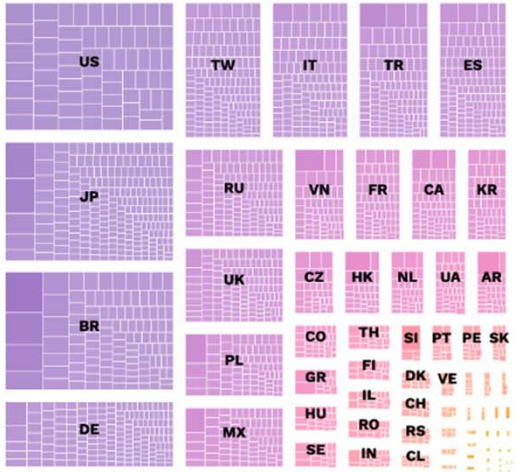
- "In three days, people spent 350 years worth of time playing with the Bach Doodle, and Coconet received more than 55 million queries"



Top overall repeated melodies



Top repeated melodies per country

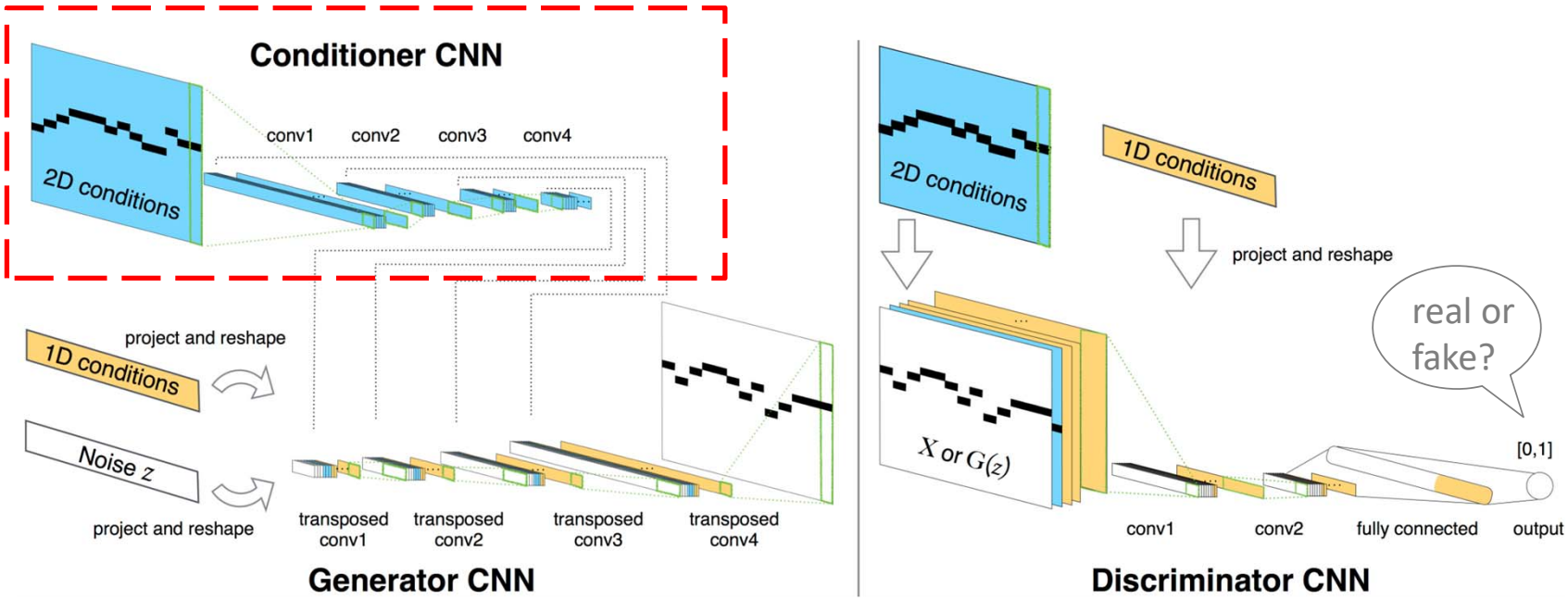


Unique regional hits

Ref: Huang et al, "The Bach Doodle: Approachable music composition with machine learning at scale", ISMIR 2019

MidiNet [yang17ismir]

- Convolutional GAN for **one-bar melody** generation
  - Turn a random vector into a piano-roll like matrix (non-autoregressive note-wise)
  - Conditioned on the previous bar (2D condition), or on the chord (1D condition)

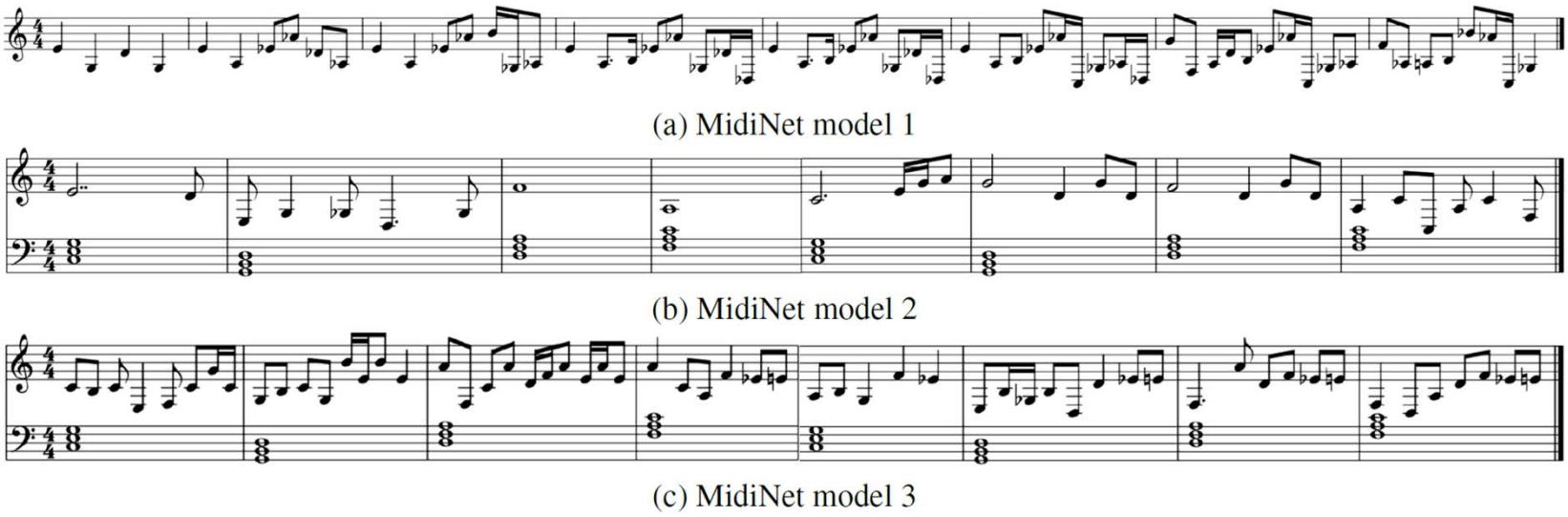


Ref: Yang et al, "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation", ISMIR 2017

MidiNet: Examples

https://richardyang40148.github.io/TheBlog/midinet_arxiv_demo.html

- Variants of MidiNet 1 2 3



(a) MidiNet model 1

(b) MidiNet model 2

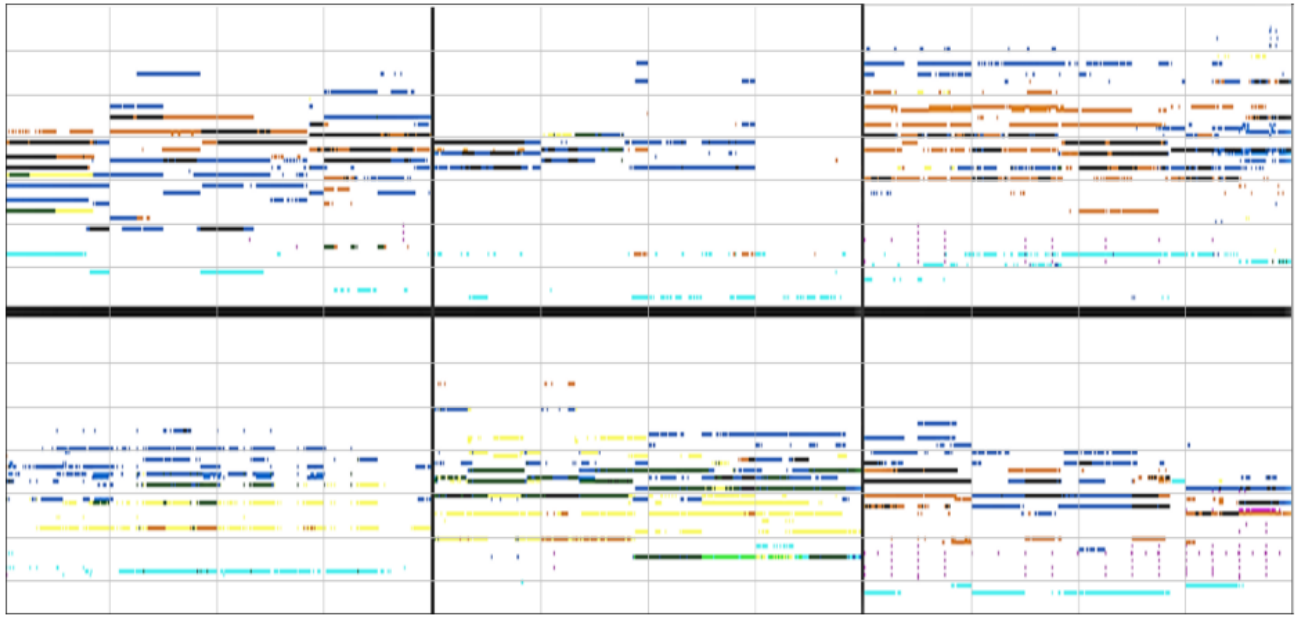
(c) MidiNet model 3

- With drums

Ref: Yang et al, "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation", ISMIR 2017

Multi-track Piano Roll

- Represent difference voices (tracks) in different channels of a fixed-size tensor

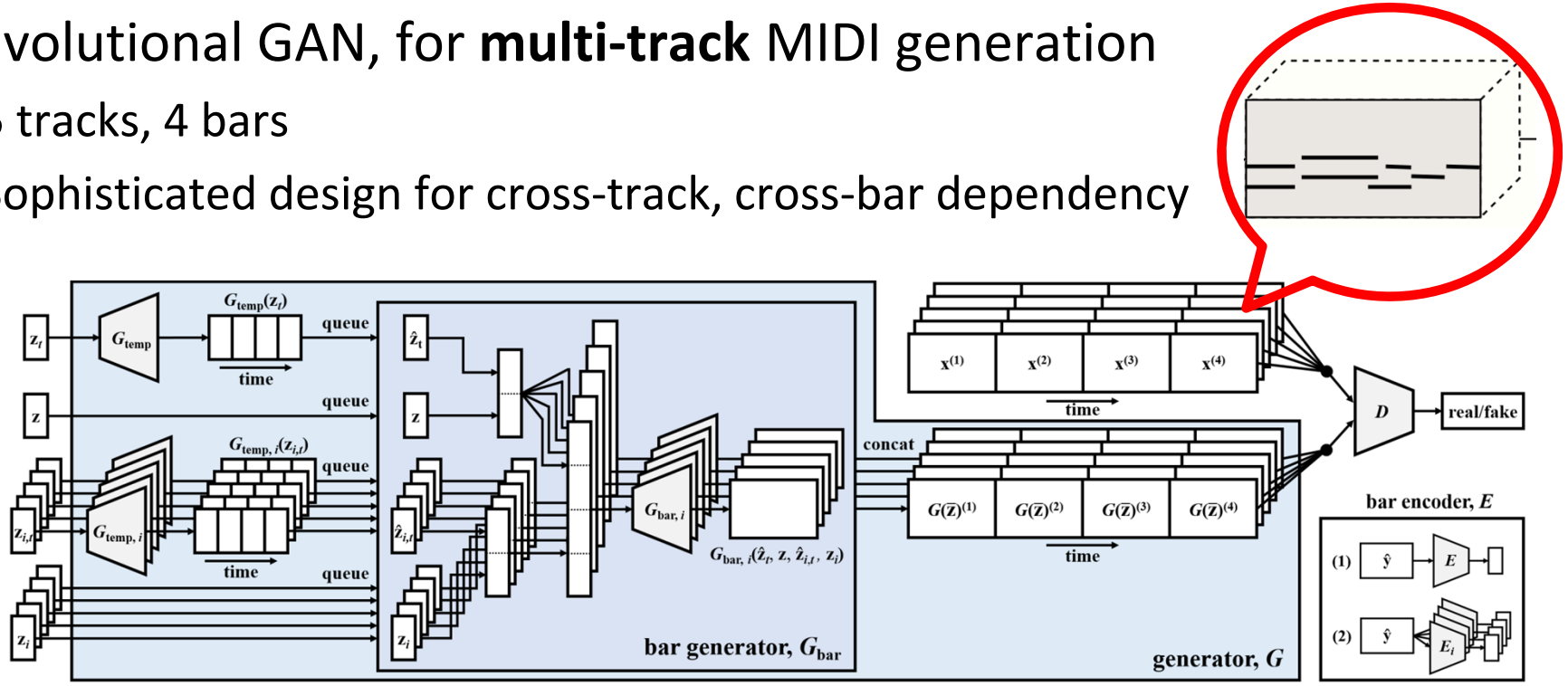


(different colors represent different tracks in this visualization)

MuseGAN [dong18aaai]

https://salu133445.github.io/musegan/

- Convolutional GAN, for **multi-track** MIDI generation
  - 5 tracks, 4 bars
  - Sophisticated design for cross-track, cross-bar dependency

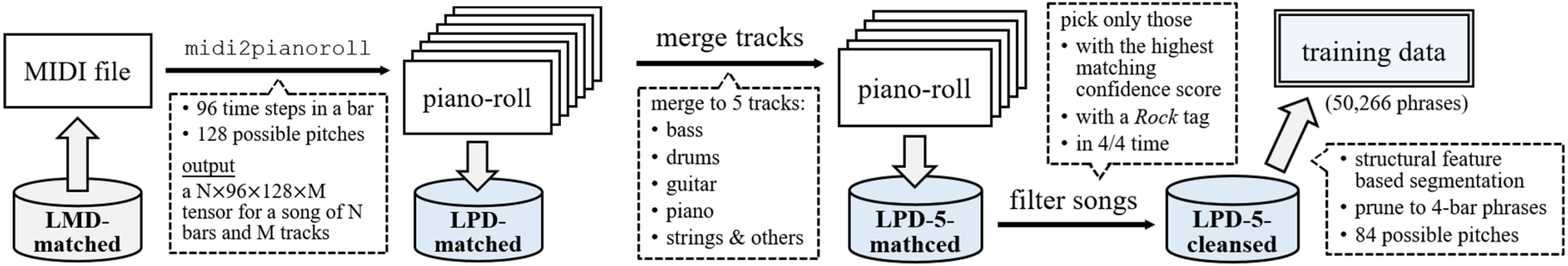


Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: Data

https://salu133445.github.io/musegan/dataset

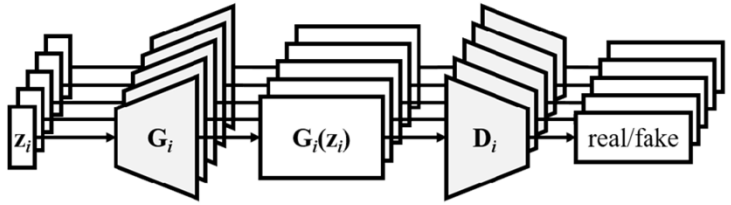
- LPD dataset: **128K MIDIs (piano-rolls)** from LMD (http://colinraffel.com/projects/lmd/)



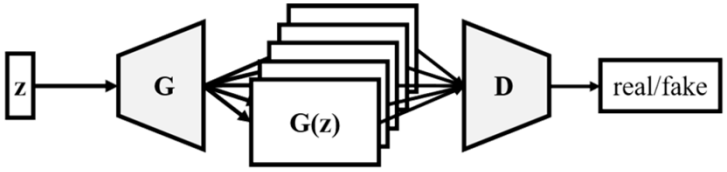
Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: Intra- & Inter-tracks

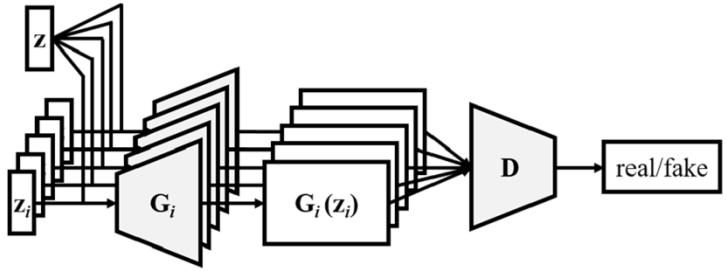
- Multi-track
  - Piano, guitar, bass, strings, drums
- **Hybrid model**
  - One "**shared**" (inter) z
  - Five "**private**" (intra) z_i
  - **Five** generators
  - **One** discriminator



(a) the jamming model



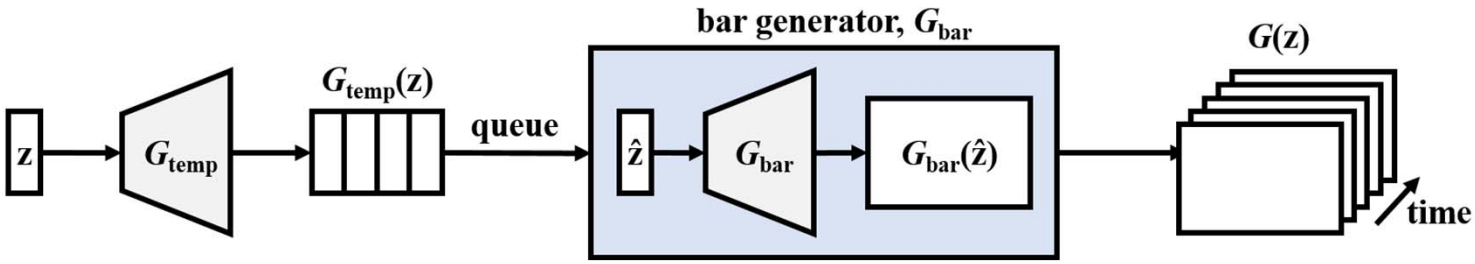
(b) the composer model



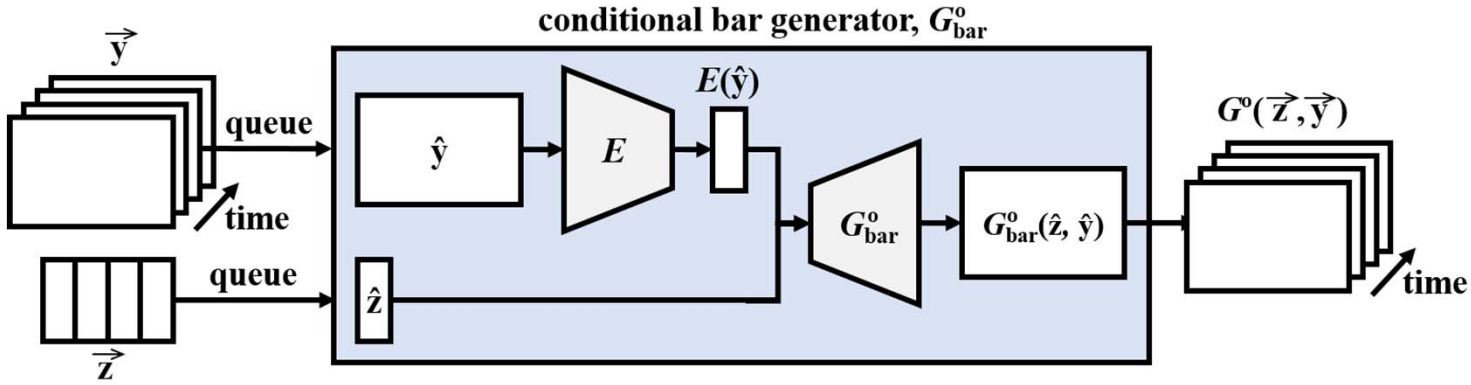
(c) the hybrid model

Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: Temporal Model



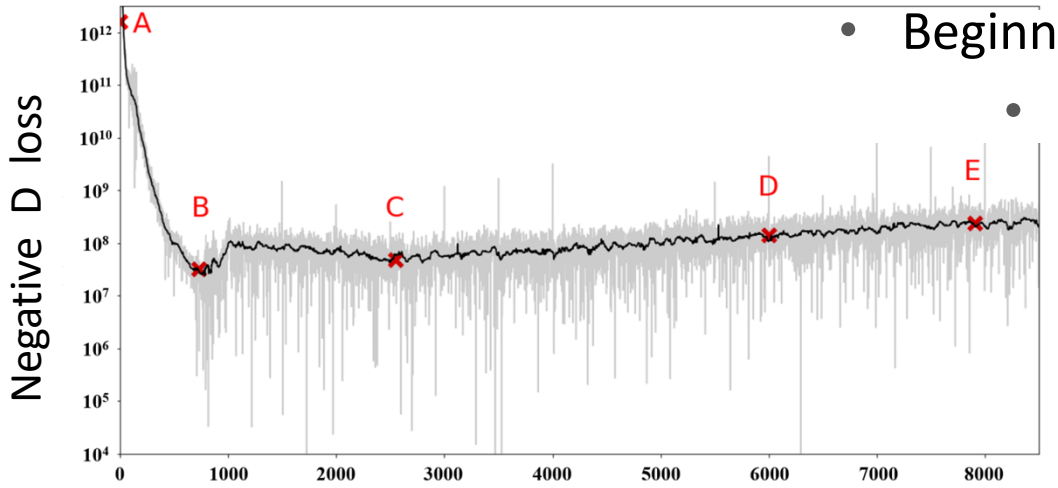
(a) generation from scratch



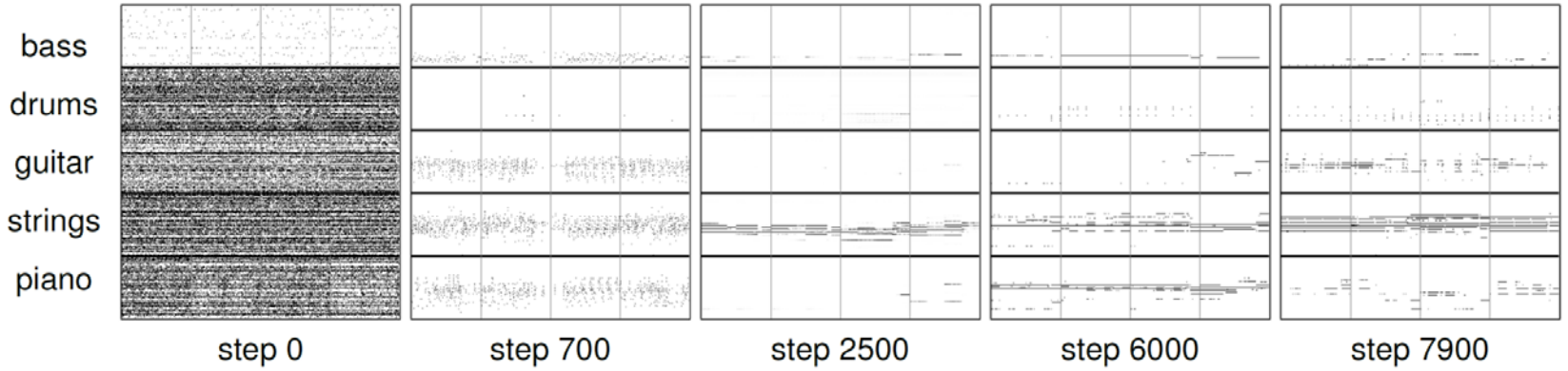
(b) track-conditional generation

Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: WGAN-gp



- Beginning – 500:1 D/G updates
- Later – 5:1 D/G updates
- Training time: <24 hours



Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: G & D

Input: $\mathbf{z} \in \Re^{128}$					
reshaped to $(1, 1) \times 128$ channels					
transconv	1024	2×1	$(2, 1)$	BN	ReLU
transconv	256	2×1	$(2, 1)$	BN	ReLU
transconv	256	2×1	$(2, 1)$	BN	ReLU
transconv	256	2×1	$(2, 1)$	BN	ReLU
transconv	128	3×1	$(3, 1)$	BN	ReLU
transconv	64	1×7	$(1, 7)$	BN	ReLU
transconv	K_{bar}	1×12	$(1, 12)$	BN	tanh
Output: $G_{\text{bar}}(\mathbf{z}) \in \Re^{96 \times 84 \times K_{\text{bar}}}$ (K_{bar}-track piano-roll)					

(b) the bar generator G_{bar}

- Grow **time steps** first
  - 1 → 2 → 4 → 8 → 16 → 32 → 96
- Then **notes** (freq)
  - octave (7)
  - then, pitch (84)

Input: $\tilde{\mathbf{x}} \in \Re^{4 \times 96 \times 84 \times 5}$ (real/fake piano-rolls of 5 tracks)				
reshaped to $(4, 96, 84) \times 5$ channels				
conv	128	$2 \times 1 \times 1$	$(1, 1, 1)$	LReLU
conv	128	$3 \times 1 \times 1$	$(1, 1, 1)$	LReLU
conv	128	$1 \times 1 \times 12$	$(1, 1, 12)$	LReLU
conv	128	$1 \times 1 \times 7$	$(1, 1, 7)$	LReLU
conv	128	$1 \times 2 \times 1$	$(1, 2, 1)$	LReLU
conv	128	$1 \times 2 \times 1$	$(1, 2, 1)$	LReLU
conv	256	$1 \times 4 \times 1$	$(1, 2, 1)$	LReLU
conv	512	$1 \times 3 \times 1$	$(1, 2, 1)$	LReLU
fully-connected	1024			LReLU
fully-connected	1			
Output: $D(\tilde{\mathbf{x}}) \in \Re$				

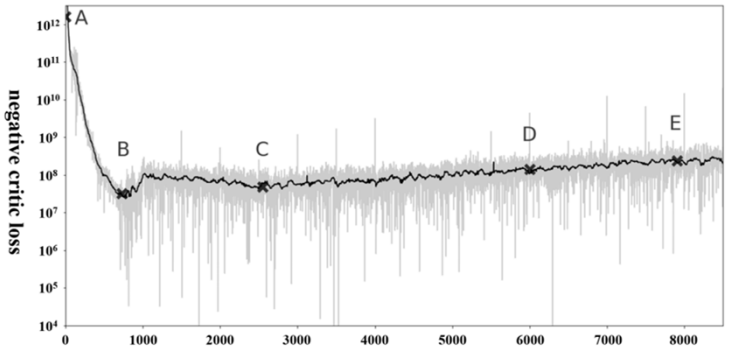
(c) the discriminator D

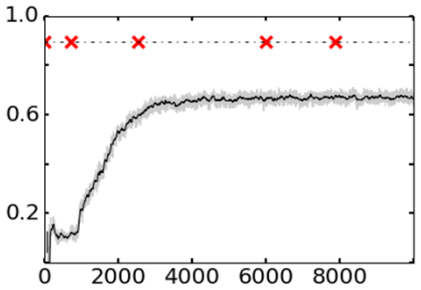
Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: Objective Metrics

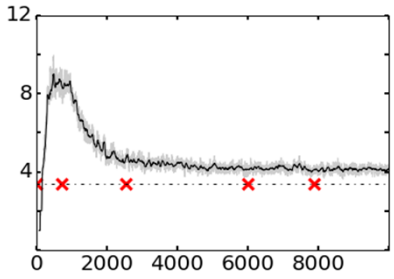
(implemented in https://github.com/salu133445/muspy)

- EB: **ratio of empty bars** (in %)
- UPC: **number of used pitch classes per bar** (from 0 to 12)
- QN: **ratio of "qualified" notes** (in %); we consider a note no shorter than three time steps (i.e. a 32th note) as a qualified note; QN shows if the music is overly fragmented
- DP, or **drum pattern**: ratio of notes in 8- or 16-beat patterns, common ones for Rock songs in 4/4 time
- TD: or **tonal distance**; the hamornicity between a pair of tracks; larger TD implies weaker inter-track harmonic relations



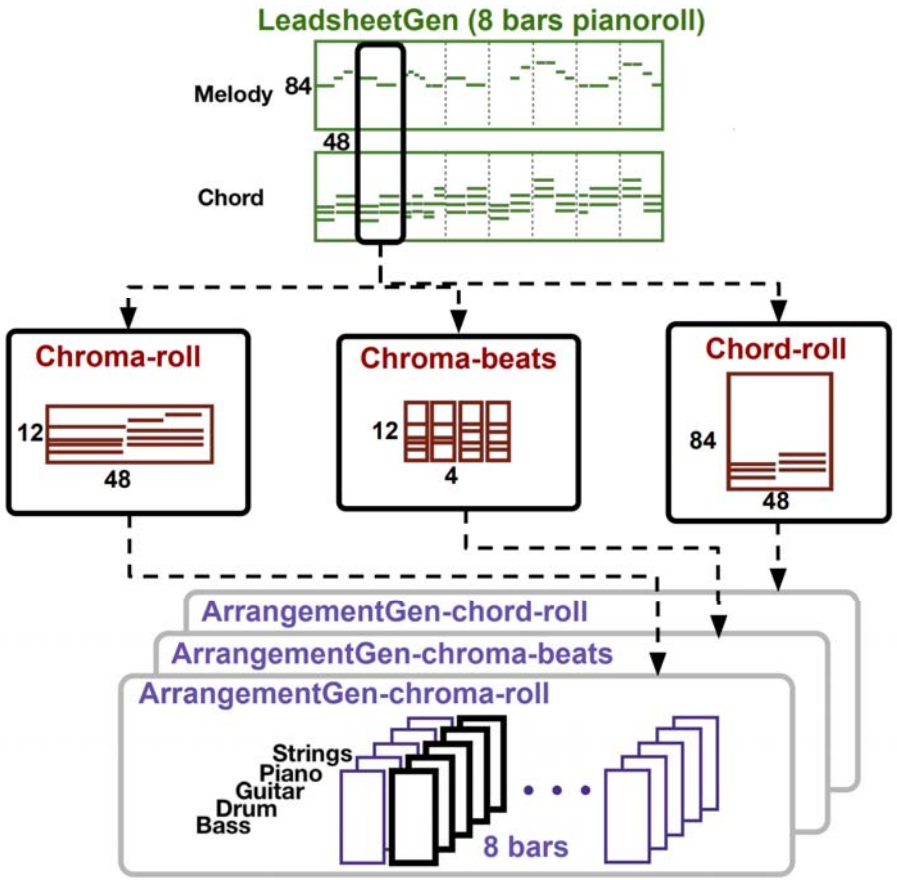


UPC

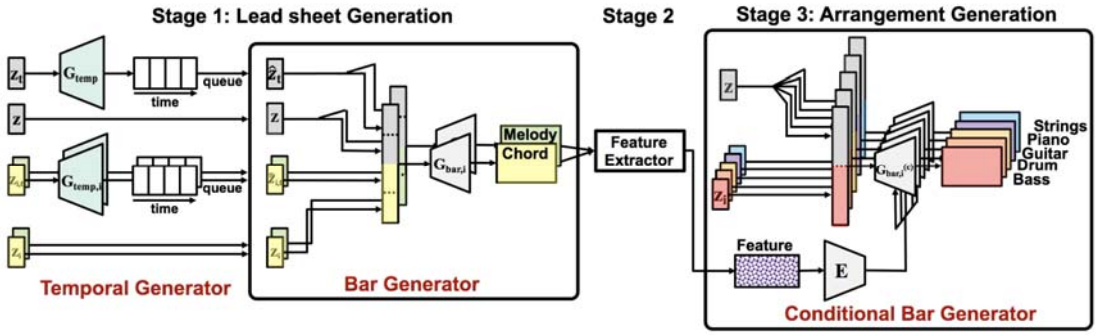


QN

LeadsheetGAN [liu18icmla]

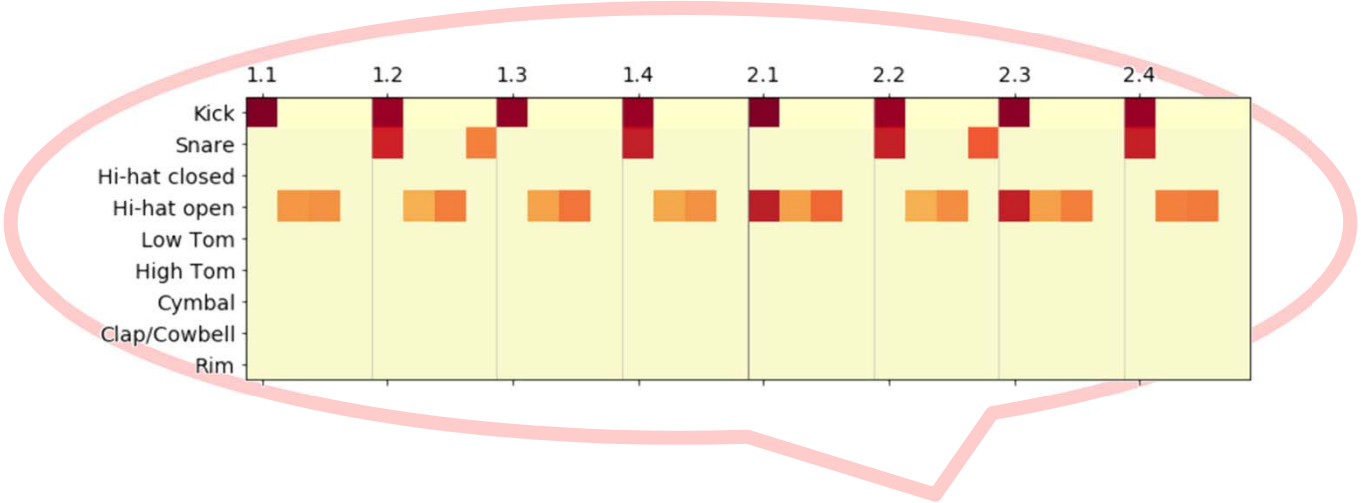


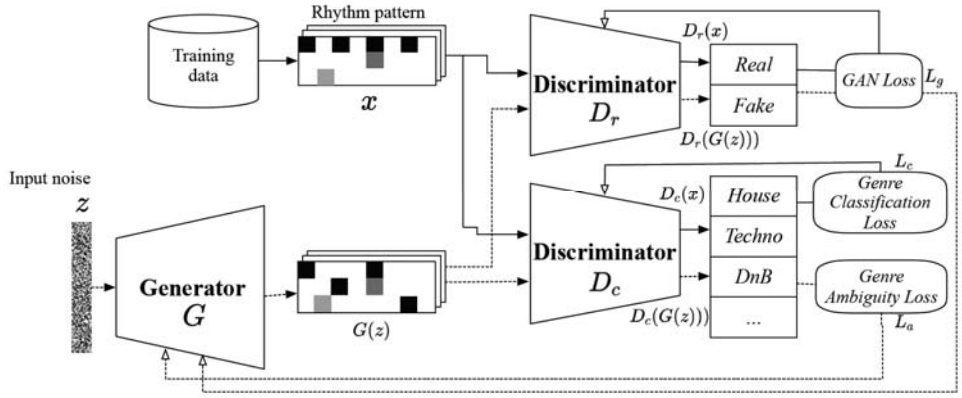
- Extension of MuseGAN
- Generate the *lead sheet* **first**
- Use features extracted from the lead sheet as condition to generate the multi-track arrangement



Ref: Liu & Yang, “Lead sheet generation and arrangement by conditional generative adversarial network”, ICMLA 2018

RhythmCAN [tokui20arxiv]

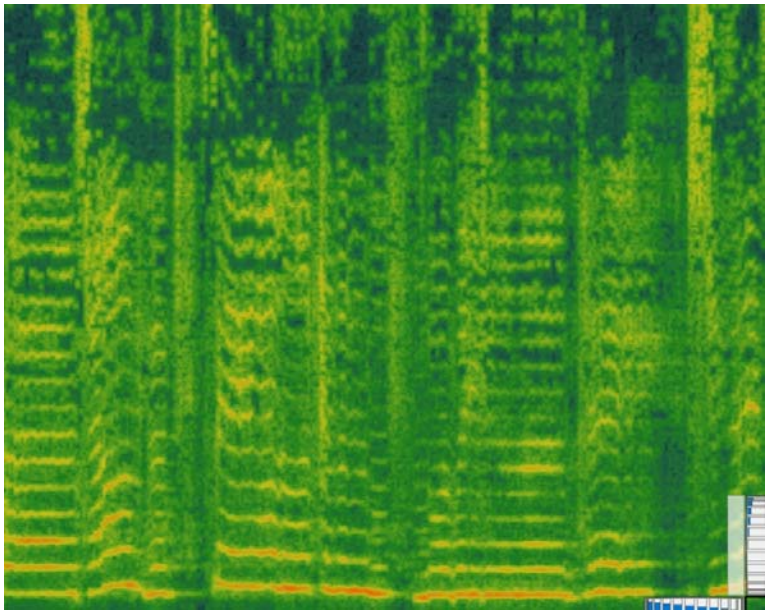




Tokui, “Can GAN originate new electronic dance music genres?—Generating novel rhythm patterns using GAN with genre ambiguity loss”, arXiv 2020

UNAGAN [liu20interspeech]

Unconditional generation of spectrograms
https://github.com/ciaua/unagan



singing

speech

violin

piano

Liu et al, “Unconditional audio generation with generative adversarial networks and cycle regularization”, INTERSPEECH 2020

Two Main Approaches

(1) Consider MIDI as **image** using the piano roll representation

- Works better for generating **patterns** (and spectrograms)
- **Less effective for modeling long sequences**

Outline

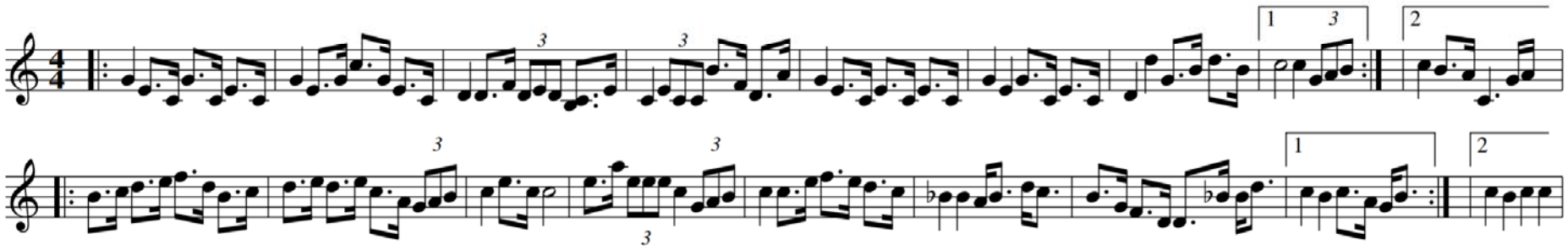
- General ideas
- Image-based approach for symbolic-domain music generation
- **Text-based approach for symbolic-domain music generation**
  - **RNN**
  - Transformer
- Token design
- Building your first MIDI Transformer

Two Main Approaches

(2) Consider music as **text**

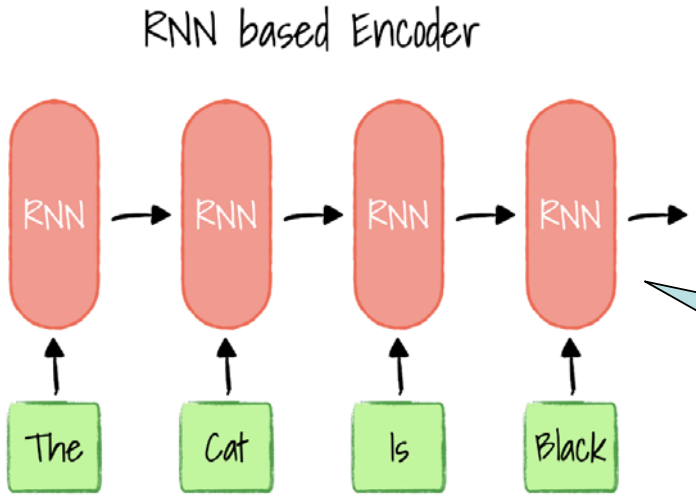
- Use “tokens” to represents “events” in music
- Becomes a “natural language generation” (NLG) problem
  - e.g., Folk RNN [sturm15ismir-lbd]

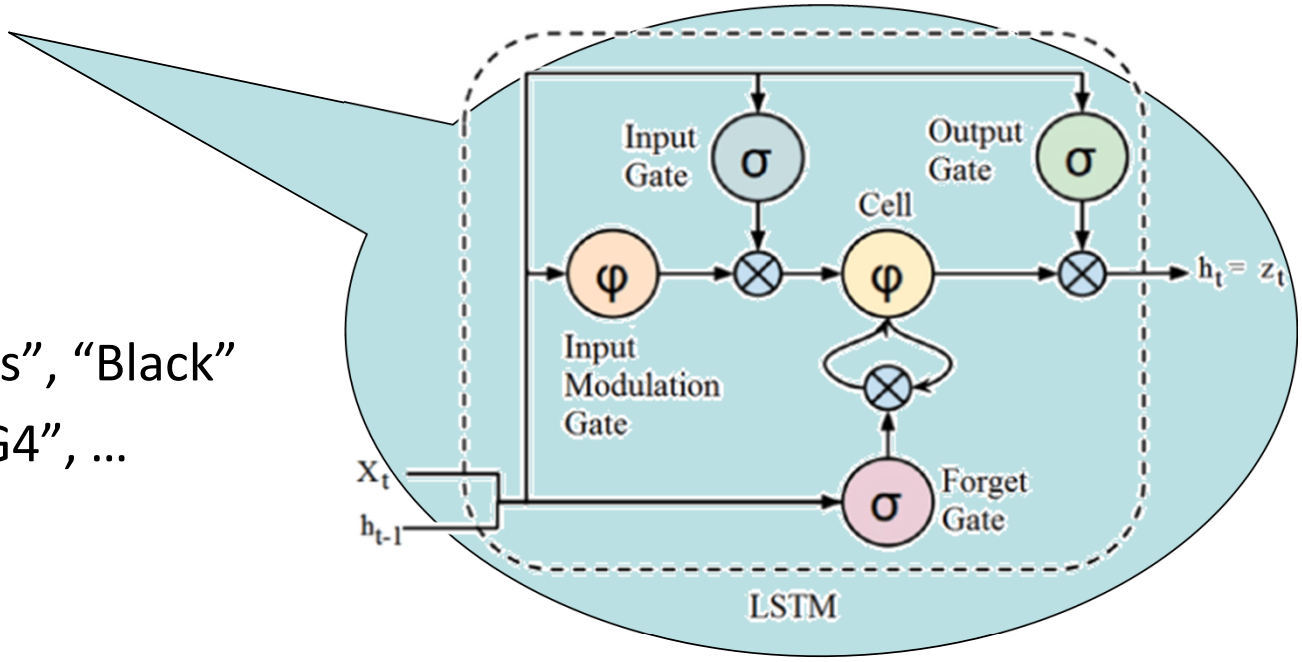
```
M:4/4
K:Cmaj
|: G 2 E > C G > C E > C | G 2 E > G c > G E > C | D 2 D > F (3 D E D [ B, C ] > E | C 2 (3 E C C B > F D > A |
G 2 E > C E > C E > C | G 2 E 2 G > C E > C | D 2 d 2 G > B d > B |1 c 4 c 2 (3 G A B :| |2 c 2 B > A C 3 G /2 A /2
|: B > c d > e f > d B > c | d > e d > e c > A (3 G A B | c 2 e > c c 4 | e > a (3 e e e c 2 (3 G A B |
c 2 c > e f > e d > c | _B 2 B 2 A < B d < c | B > G F > D D > _B B < d |1 c 2 B 2 c > A G < B :| |2 c 2 B 2 c 2 c 2 |
```



Ref: Sturm et al, “Folk music style modelling by recurrent neural networks with long short term memory units,” ISMIR-LBD 2015

RNNs





- Text tokens: “The”, “Cat”, “Is”, “Black”
- Music tokens: “C4”, “E4”, “G4”, …

RNN-based Symbolic Music Generation

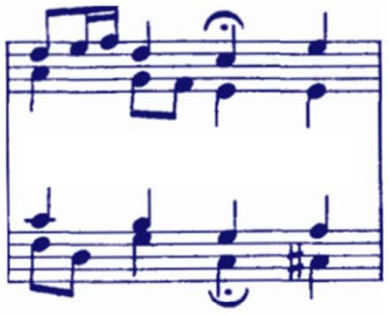
- Used to be the state-of-the-art
- Good for not-very-long sequences (e.g., 16 bars)

- Examples
  - **MelodyRNN** (2016): https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial
  - **C-RNN-GAN** (2016)
  - **Song From PI** (2017)
  - **DeepBach** (ICML 2017)

RNN Example 1: DeepBach

https://replicate.com/ghadjeres/deepbach

- Notes, voices, ***hold*** (`__`)
- **Fermeta** $\mathcal{F}$
  - End of a musical phrase
- **Subdivision** $\mathcal{S}$
  - Subdivision indexes of a beat



(a)

```
D5,__,E5,F5,D5,__,__,__,C5,__,__,__,E5
A4,__,__,__,G4,__,F4,__,E4,__,__,__,E4
C4,__,__,__,B3,__,__,__,G3,__,__,__,A3
F3,__,D3,__,G3,__,__,__,C2,__,__,__,C#2
 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1
 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0
```

(b)

Figure 3. Extract from a Bach chorale and its representation as four voice lists and two metadata lists ($\mathcal{S}$ and $\mathcal{F}$). The hold symbol is displayed as "__" and considered as a note.

- Dataset
  - 352 chorale harmonizations by J.S. Bach included in the music21 toolkit
  - Data augmentation by key transposition

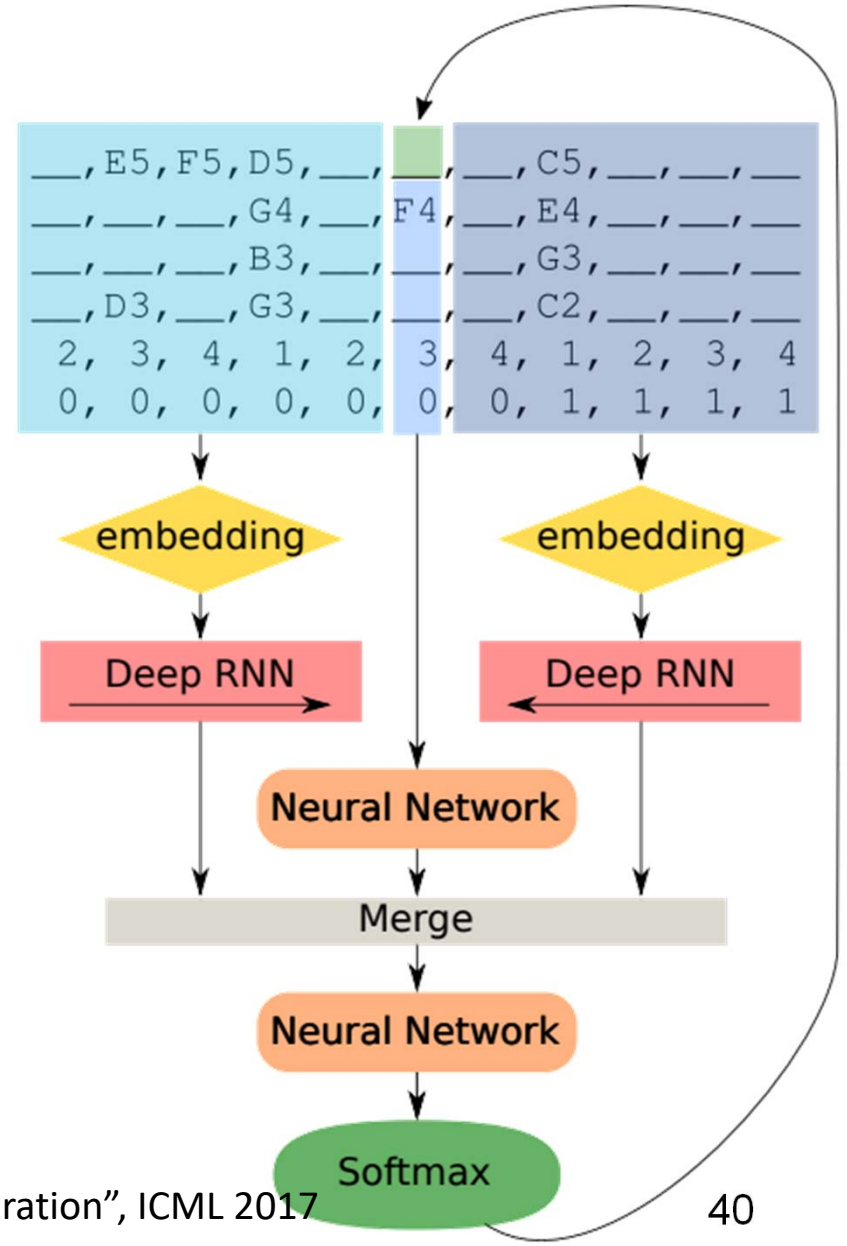
Ref: Hadjeres et al, "DeepBach: a steerable model for bach chorales generation", ICML 2017

DeepBach

- Four sub-networks
  - One RNN for the past
  - One RNN for the future (writing backward)
  - One for the concurrent notes
  - One that summarizes information from the above three

- Softmax
  - to turn the output into probabilities

$$\text{Softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j \in [K].$$



Ref: Hadjeres et al, "DeepBach: a steerable model for bach chorales generation", ICML 2017

DeepBach

- Pseudo-Gibbs sampling
  - Non-autoregressive
  - Can be used for infilling
  - But may be less effective for prompt continuation

Algorithm 1 Pseudo-Gibbs sampling

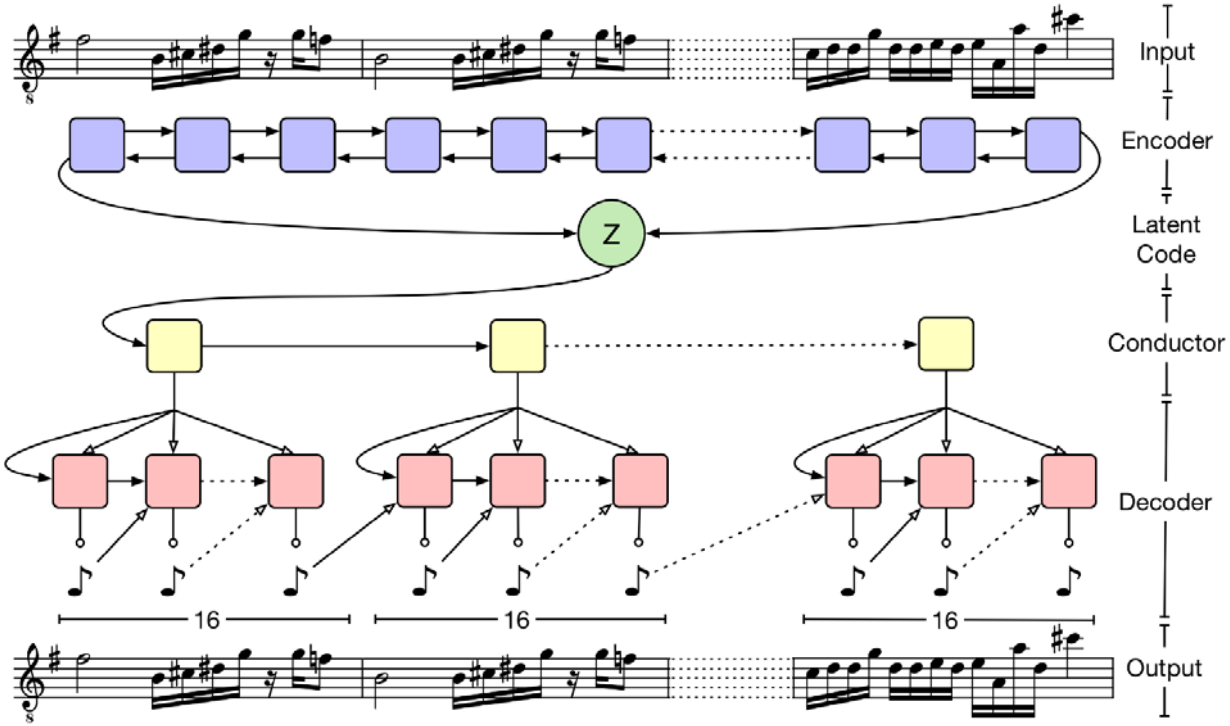
Input: Chorale length L, metadata $\mathcal{M}$ containing lists of length L, probability distributions (p_1, p_2, p_3, p_4), maximum number of iterations M
Create four lists $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3, \mathcal{V}_4)$ of length L
{The lists are initialized with random notes drawn from the ranges of the corresponding voices (sampled uniformly or from the marginal distributions of the notes)}
for m from 1 to M **do**
    Choose voice i uniformly between 1 and 4
    Choose time t uniformly between 1 and L
    Re-sample $\mathcal{V}_i^t$ from $p_i(\mathcal{V}_i^t | \mathcal{V}_{\setminus i,t}, \mathcal{M}, \theta_i)$
end for
Output: $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3, \mathcal{V}_4)$

Ref: Hadjeres et al, “DeepBach: a steerable model for bach chorales generation”, ICML 2017

RNN Example 2: Music VAE [roberts18icml]

https://magenta.tensorflow.org/music-vae

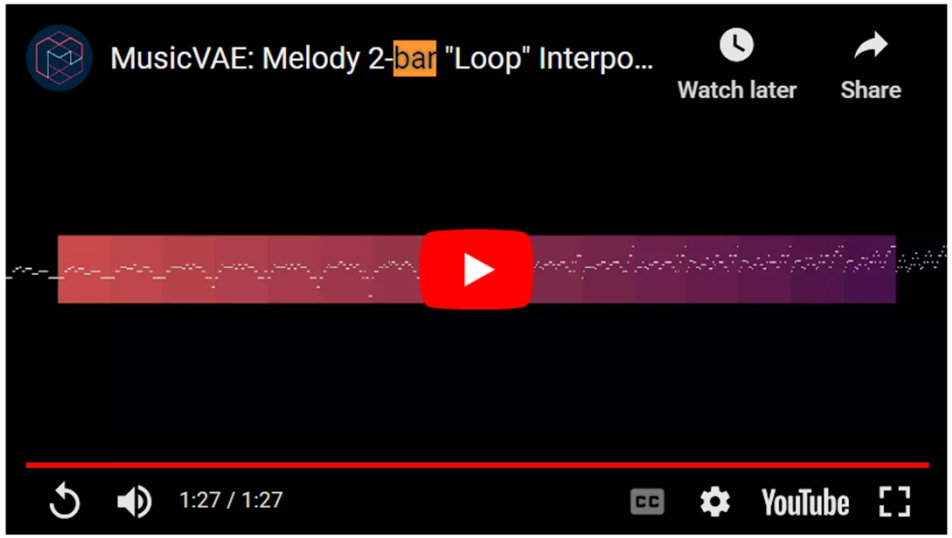
- For **melody** generation
- **Bidirectional encoder**
- **Hierarchical decoder**
  - **Bar-level** conductor
  - **Note-level** decoder
- **Recurrent VAE**
  - Two-layer LSTM for the encoder, conductor, decoder
  - Spherical Gaussian prior for the latent code *z*

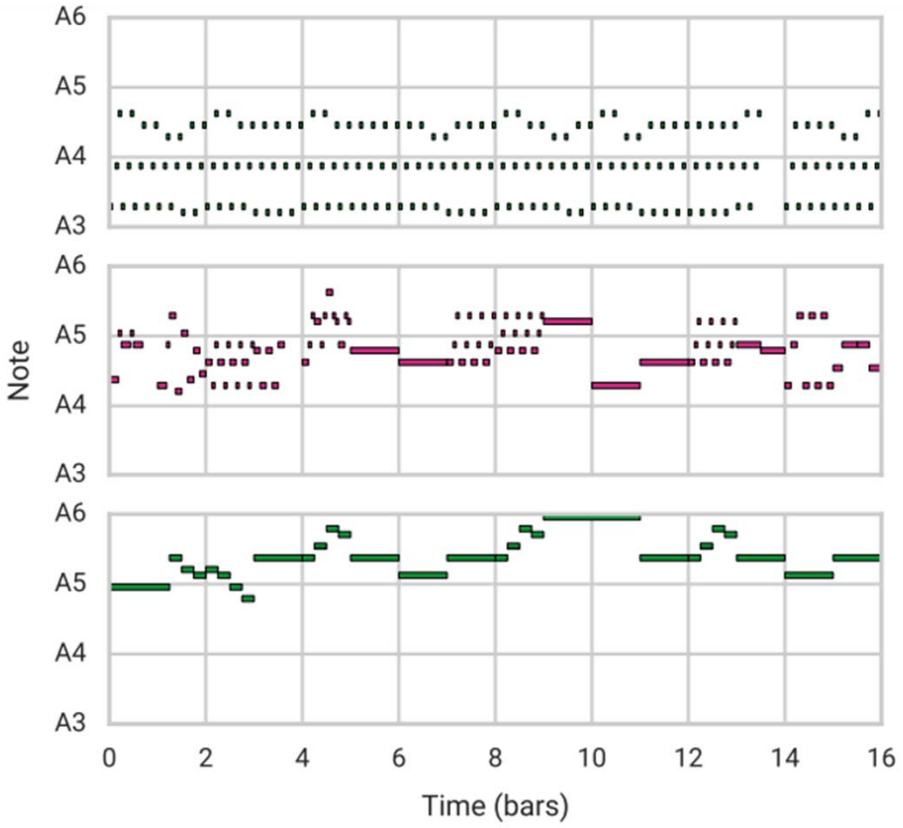


Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Music VAE [roberts18icml]

- Latent space manipulation
  - Attribute vectors
  - **Spherical interpolation**

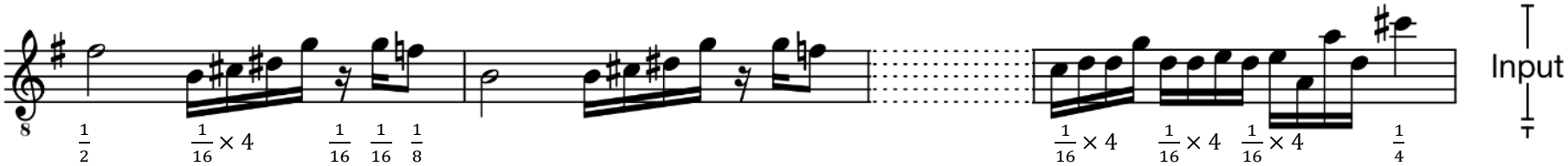




Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Music VAE [roberts18icml]

- Data representation
  - 2- and 16-bar melodies (monophonic note sequences)
  - **130- dimensional output space** (categorical distribution over tokens)
    - 128 "**note-on**" tokens for the 128 MIDI pitches
    - plus single tokens for "**note-off**" and "**rest**"

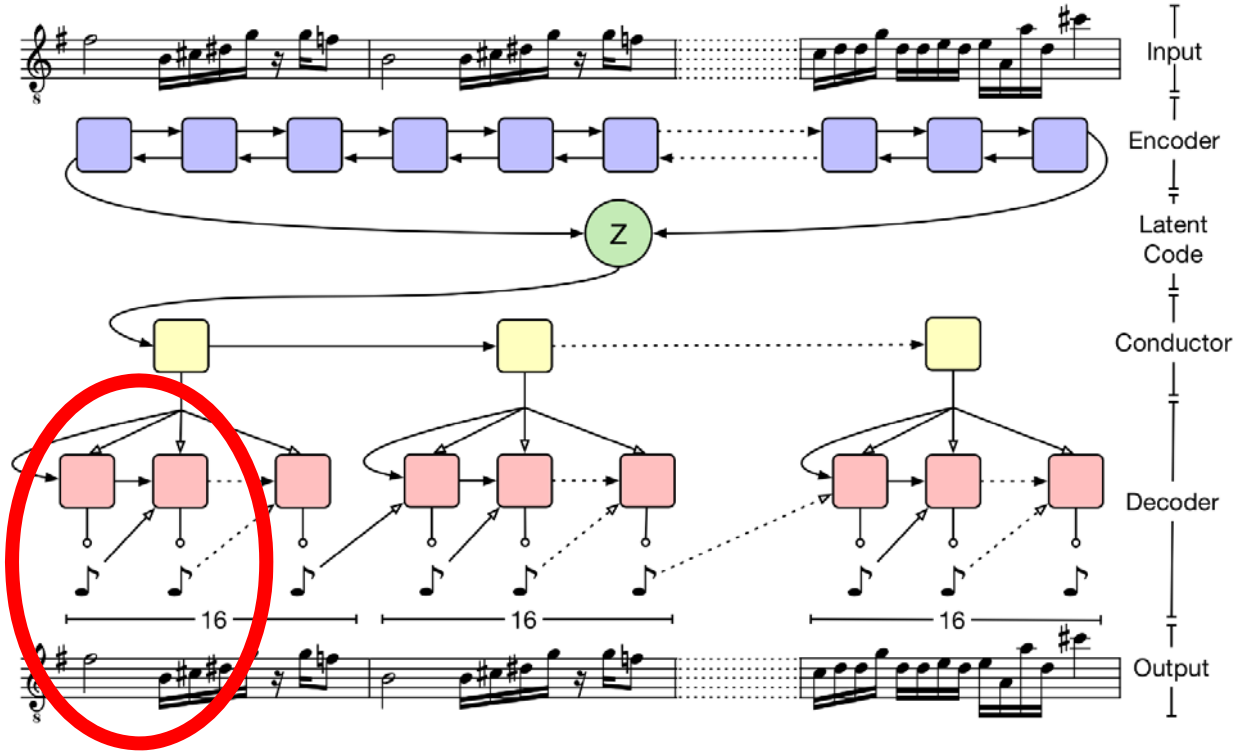


  - Model the melodies as sequences of 16th note events
    - Only consider 4/4 signature—each bar consists of 16 events
    - 2-bar data: sequence length 32
    - 16-bar data: sequence length 256

Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Music VAE [roberts18icml]

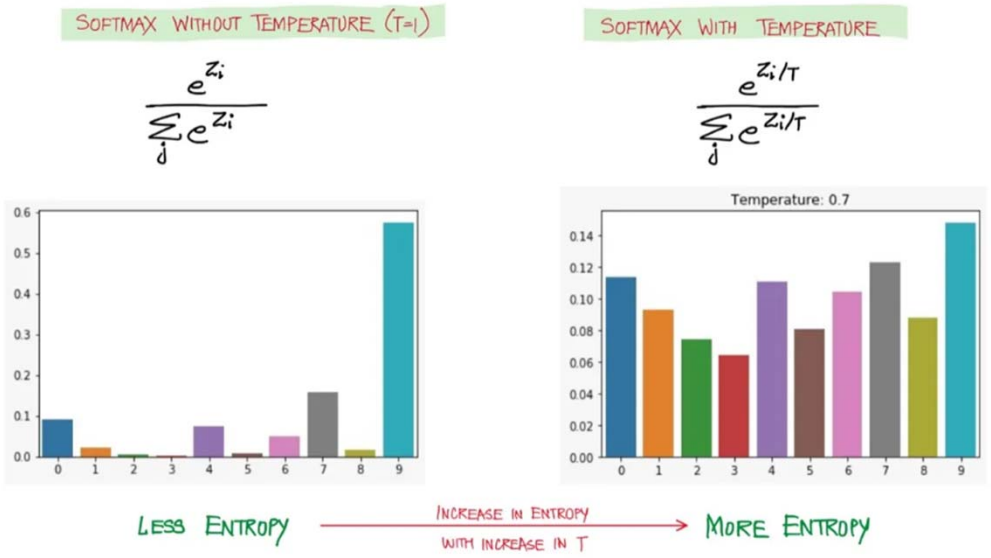
- Training
  - **Cross-entropy loss**
    - For each time step (token)
  - **Teacher forcing**
    - During *training* time, use observed sequence values (i.e. ground-truth samples) back into the RNN after each step
  - **Scheduled sampling**
    - To reduce "**exposure bias**," the discrepancy between training time and inference time sampling

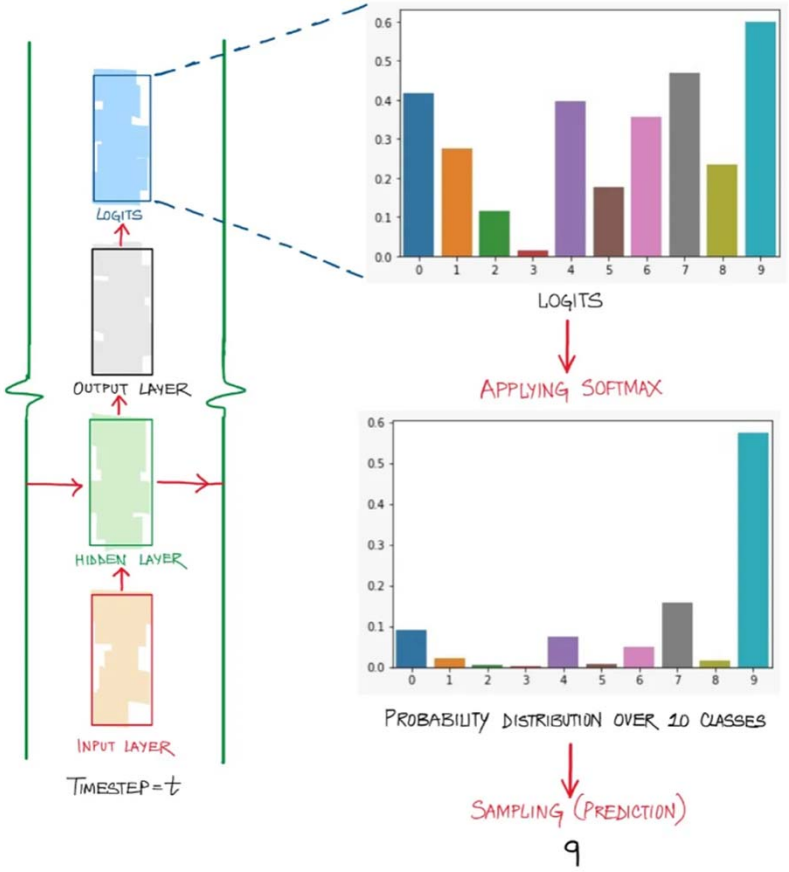


Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Music VAE [roberts18icml]

- Inference
  - **Softmax temperature**
  - https://medium.com/mlearning-ai/softmax-temperature-5492e4007f71



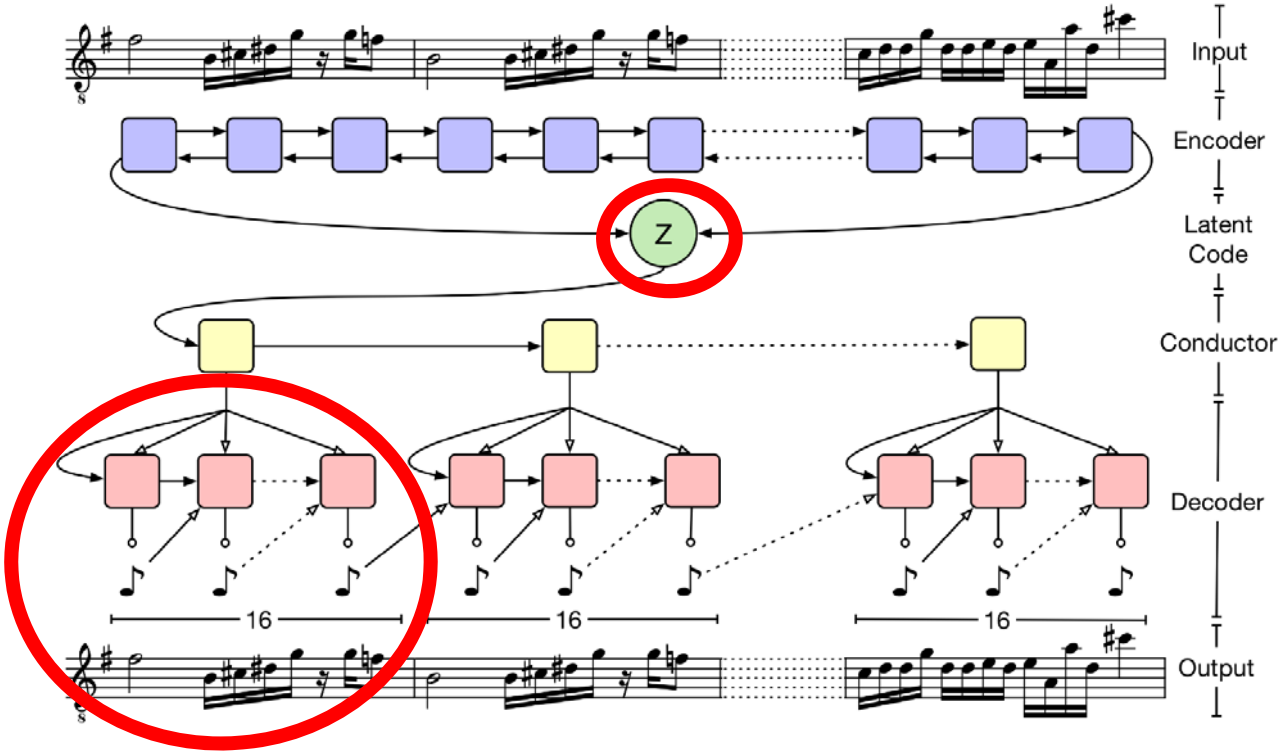


Training/Inference for Sequence Models (RNNs, Transformers)

- **Training**
  - Output a **softmax** distribution (sum-to-one)
  - Compare that softmax distribution (continuous-valued) with the groundtruth one-hot distribution (discrete) using **cross entropy**
  - *Teacher forcing*: **Do NOT really sample from that distribution**, but use the **groundtruth token** as the input for the next time step
- **Inference**
  - Output a **softmax** distribution (sum-to-one)
  - **Do sampling**
    - There are many sampling methods
    - May apply temperature
  - Use the **sampled token** as the input for the next time step
  - *Exposure bias:* The discrepancy between training time and inference time sampling

Music VAE [roberts18icml]

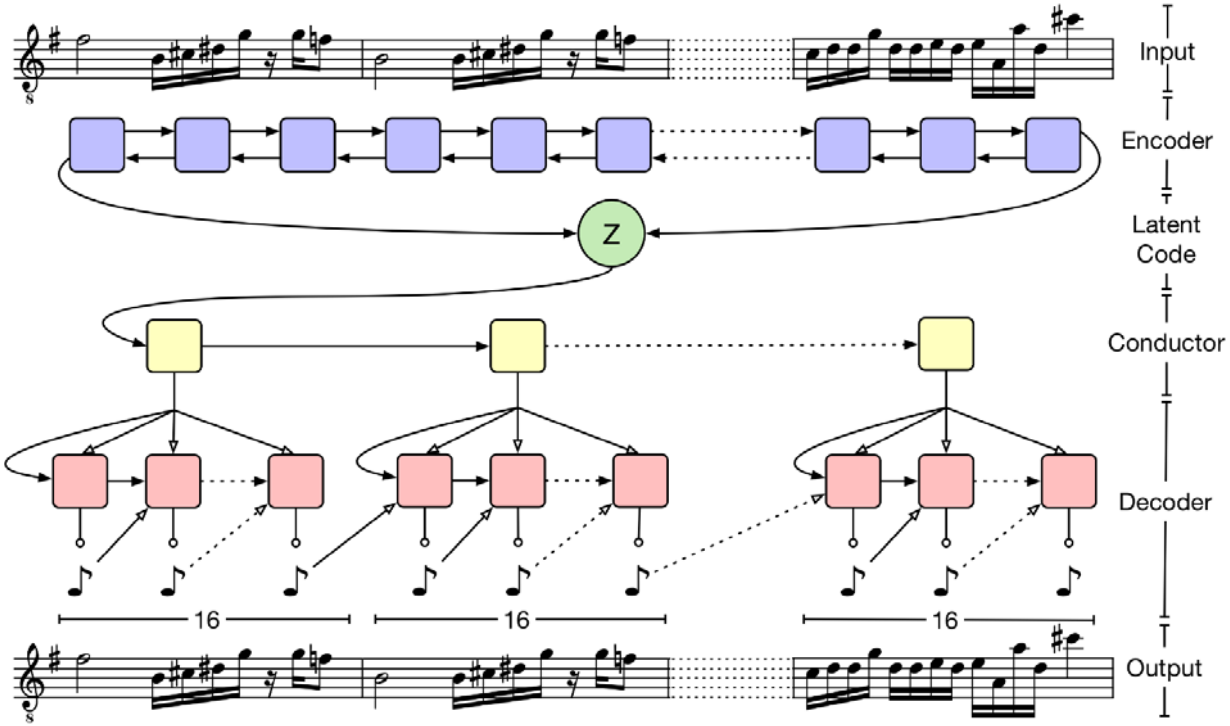
- Training
  - Cross-entropy loss
  - Teacher forcing
  - Avoid **“posterior collapse”**
    - Where the model learns to ignore the latent state
    - Reduce the effective scope of the decoder RNN by only allowing it to propagate state within an output subsequence
  - VAE training techniques: ***β*-VAE and *free bits***



Ref: Roberts et al, “A hierarchical latent vector model for learning long-term structure in music”, ICML 2018

Music VAE [roberts18icml]

- For **melody** generation
- For **bass** generation
- For **drum** generation
  - "For drum patterns, we mapped the 61 drum classes defined by the General MIDI standard to **9 canonical classes** and represented **all possible combinations of hits with 512 categorical tokens**"



Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Applications of MusicVAE

- Melody mixer
  https://experiments.withgoogle.com/ai/melody-mixer/view/

- Beat blender
  https://experiments.withgoogle.com/ai/beat-blender/view/

- Latent loops
  https://teampieshop.github.io/latent-loops/

RNN Example 3: Jazz RNN

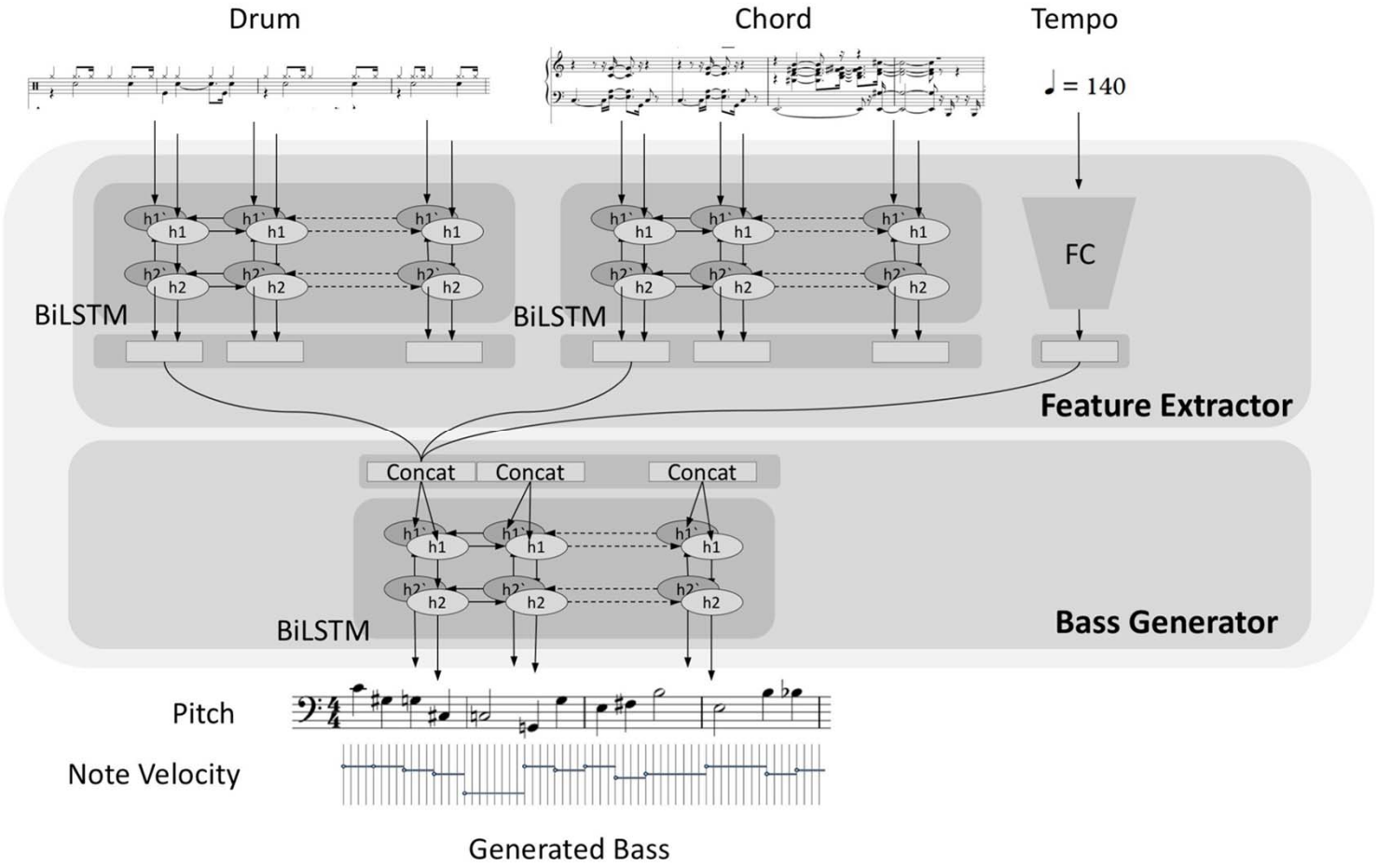
https://soundcloud.com/yating_ai/sets/ismir-2019-submission

- An RNN-based model that takes a random 8-bar chord sequence as condition to generate an **8-bar Jazz piano performances**
- Can also do **Pop-to-Jazz style transfer**
  - "This is done by using JazzRNN to pre-generate a large collection of note sequences and then finding the one (by machine) with the closest rhythmic pattern to that of the melody line extracted from the separated vocal part of a given Pop song."

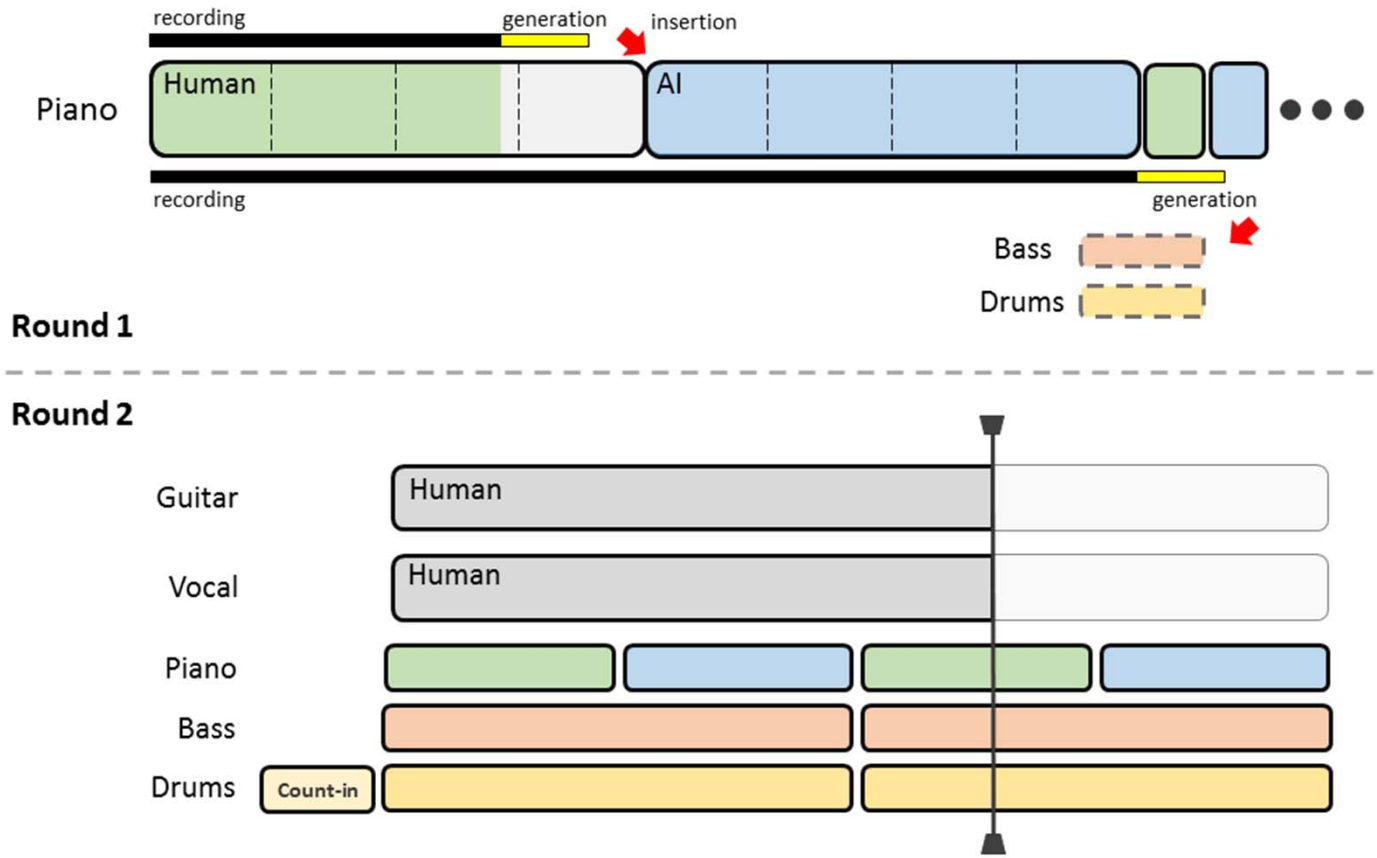
Ref: Yeh et al, "Learning to generate Jazz and Pop piano music from audio via MIR techniques", ISMIR-LBD 2019

Extension of Jazz RNN: Jazz Bass Player

https://ailabs.tw/human-interaction/ai-jazz-bass-player/

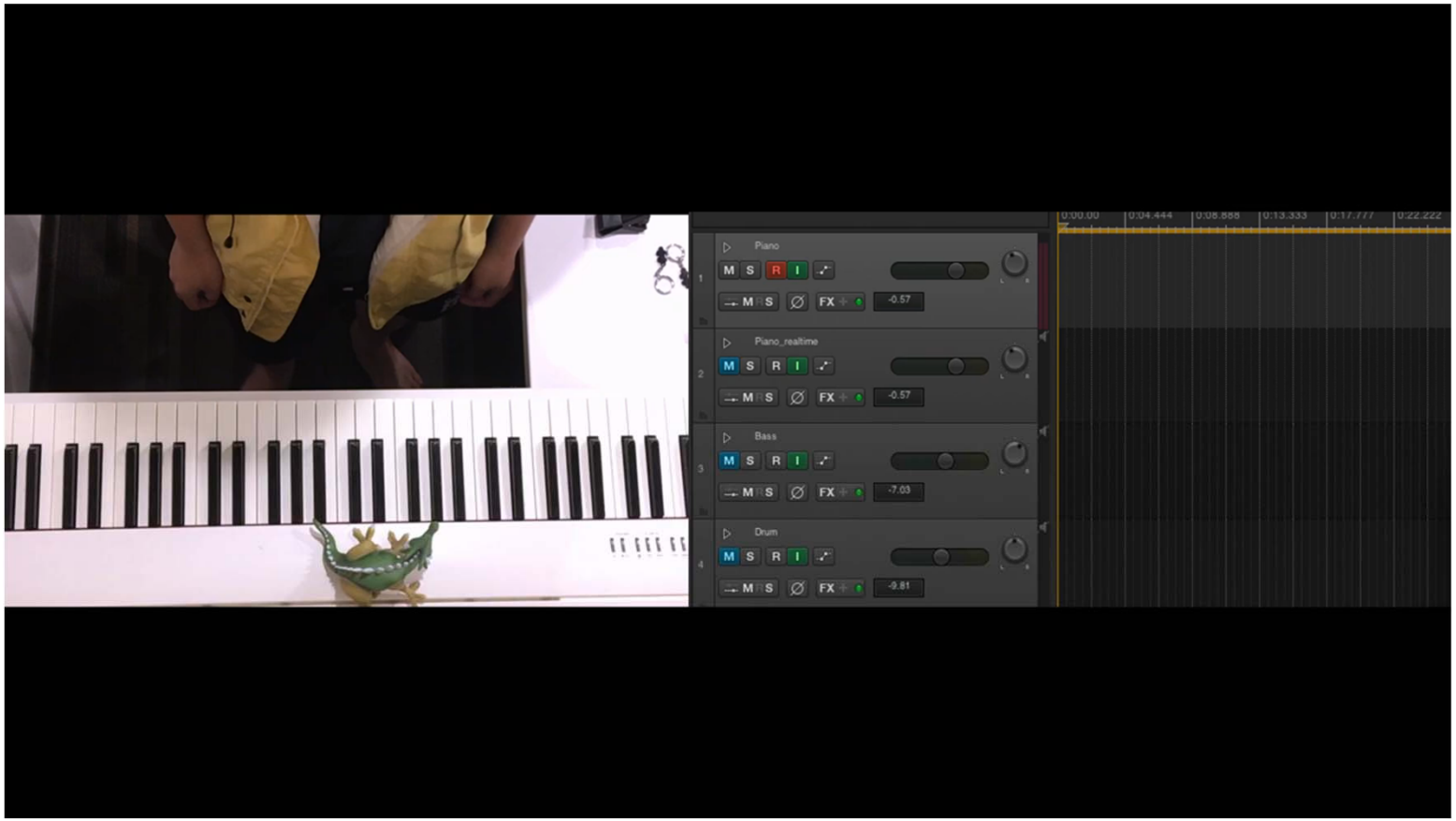


Extension of Jazz RNN: Jamming with Yating



Ref 1: Yeh et al., “Learning to generate Jazz and Pop piano music from audio via MIR techniques,” ISMIR-LBD 2019
Ref 2: Hsiao et al, “Jamming with Yating: Interactive demonstration of a music composition AI”, ISMIR-LBD 2019

Jamming with Yating Demo



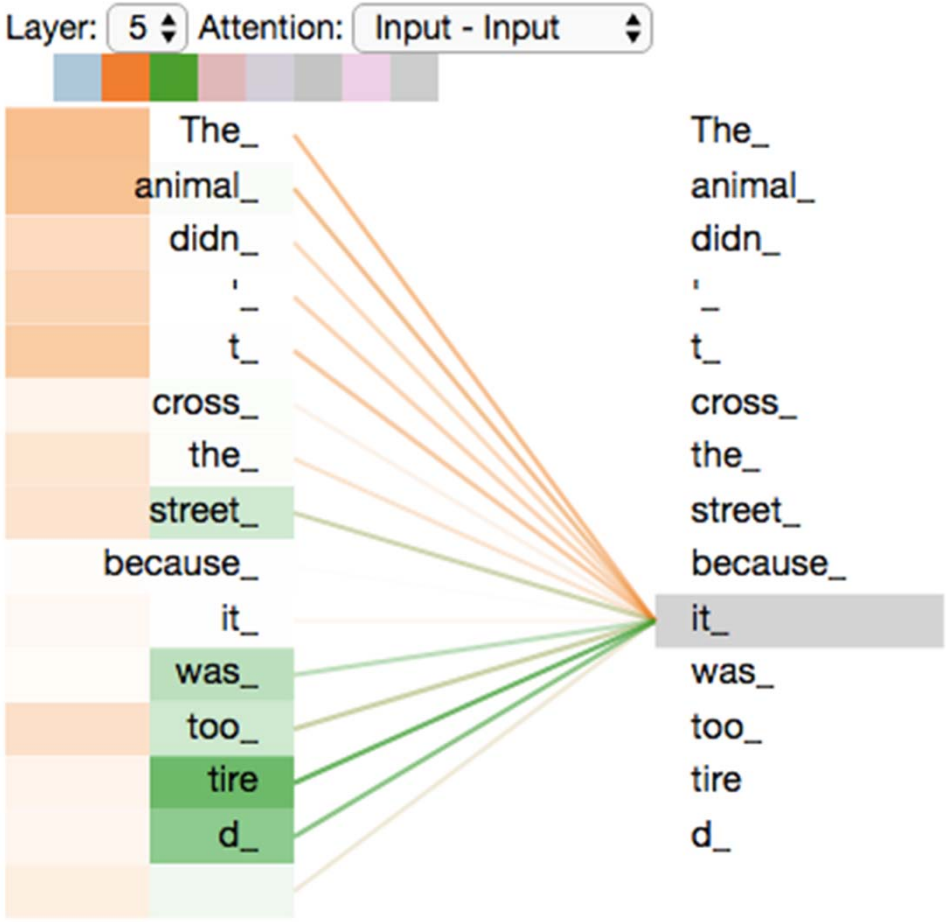
Ref: Hsiao et al, “Jamming with Yating: Interactive demonstration of a music composition AI”, ISMIR-LBD 2019

Outline

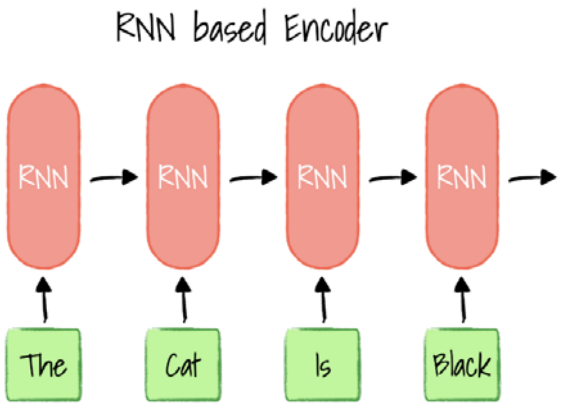
- General ideas
- Image-based approach for symbolic-domain music generation
- Text-based approach for symbolic-domain music generation
  - RNN
  - **Transformer**
- Token design
- Building your first MIDI Transformer

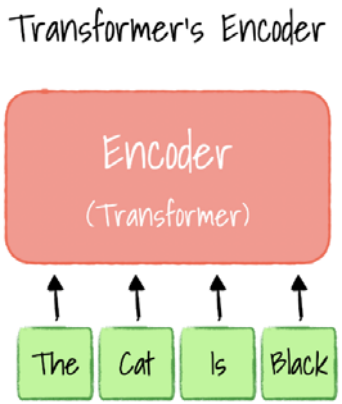
Transformers

- Attention mechanism
  - Maintain a latent vector for ***every*** token in the history
  - Compute the correlation between the input with all the latents in the history
  - Need "**positional encoding**"
- In contrast, there is only **one** latent vector for the entire sequence in RNNs



RNNs vs. Transformers





- **RNNs**
  - use only a latent vector
  - $y_t = f(y_{t-1}, h_{t-1})$

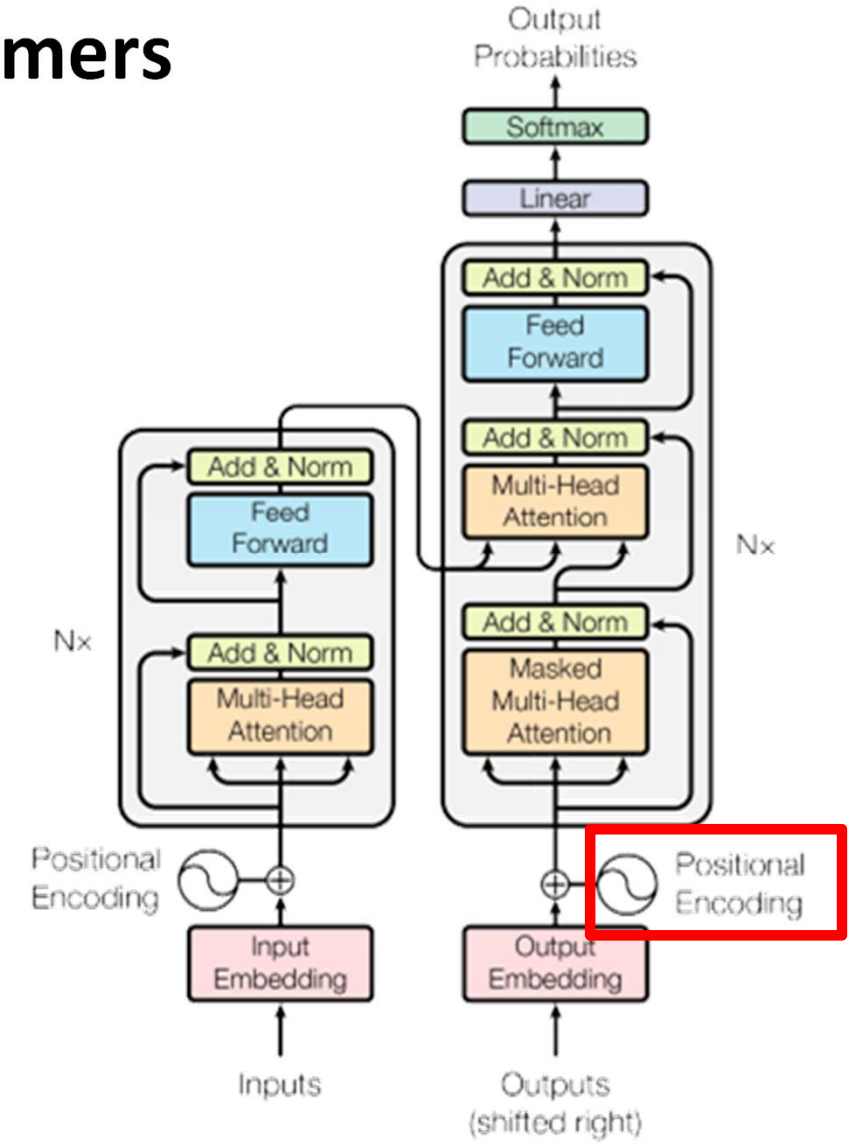
current output — last output — latent representation of the "history"

- **Transformers**
  - multiple latent vectors
  - $y_t = f(y_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L})$

latent representation of t-1 — latent representation of t-L

Transformers

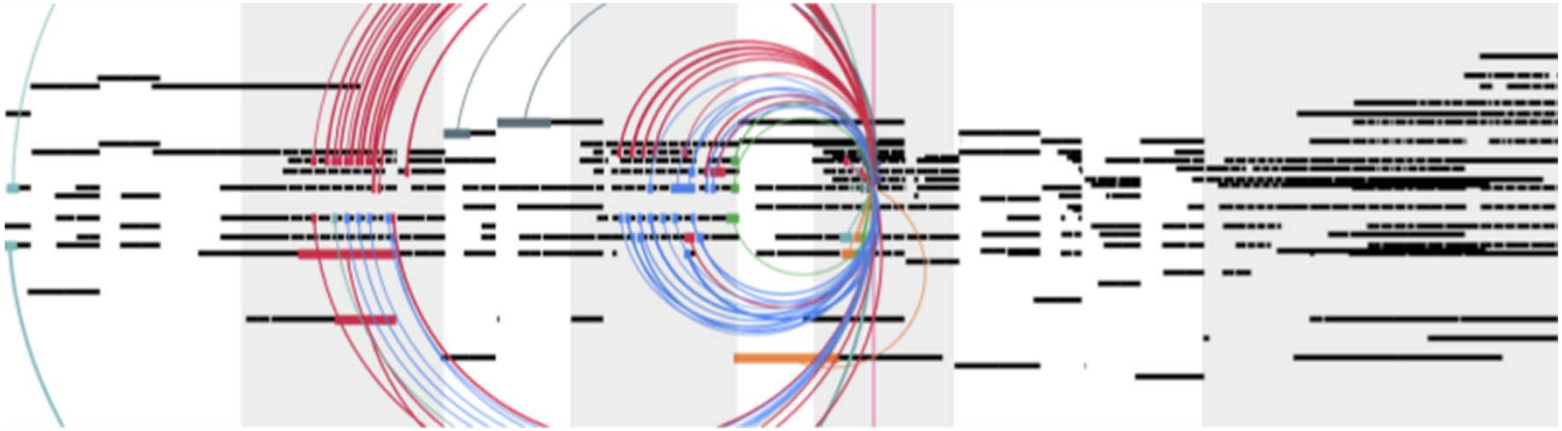
- There are encoders & decoders
  - Encoder: cross-attention
  - Decoder: self-attention
- **Decoder-only** architecture
  - Usually used for unconditional, or prompt-based generation
  - Mainly talk about this one today
- **Encoder/decoder** architecture
  - Usually used for sequence-to-sequence generation



Transformer Example 1: Music Transformer [huang19iclr]

https://magenta.github.io/listen-to-transformer/

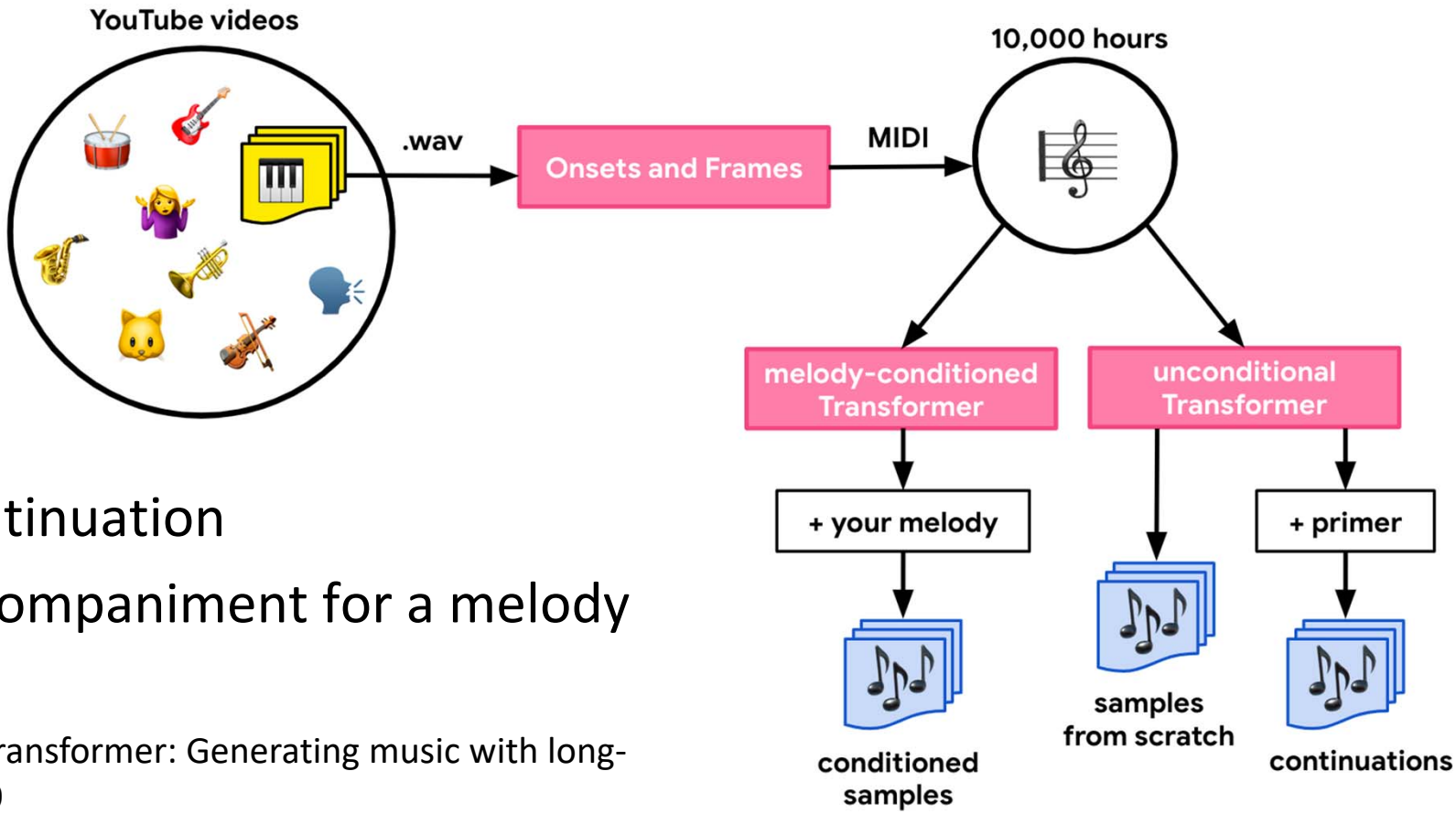
- Generate full piano performances (one note at a time)
  - "… can generate **minute-long** compositions (thousands of steps) with compelling structure, generate continuations that coherently elaborate on a given motif, and in a seq2seq setup generate accompaniments conditioned on melodies."



Ref: Huang et al, "Music Transformer: Generating music with long-term structure", ICLR 2019

Music Transformer [huang19iclr]

https://magenta.tensorflow.org/piano-transformer



- Generate continuation
- Generate accompaniment for a melody

Ref: Huang et al, "Music Transformer: Generating music with long-term structure", ICLR 2019

Transformer Example 2: MuseNet [payne19]

https://openai.com/blog/musenet/

- Generate multi-track MIDIs by a **72-layer** sparse Transformer decoder
  - *Instrument-related* tokens

```
bach piano_strings start tempo90 piano:v72:G1 piano:v72:G2
piano:v72:B4 piano:v72:D4 violin:v80:G4 piano:v72:G4
piano:v72:B5 piano:v72:D5 wait:12 piano:v0:B5 wait:5
piano:v72:D5 wait:12 piano:v0:D5 wait:4 piano:v0:G1 piano:v0:G2
piano:v0:B4 piano:v0:D4 violin:v0:G4 piano:v0:G4 wait:1
piano:v72:G5 wait:12 piano:v0:G5 wait:5 piano:v72:D5 wait:12
piano:v0:D5 wait:5 piano:v72:B5 wait:12
```

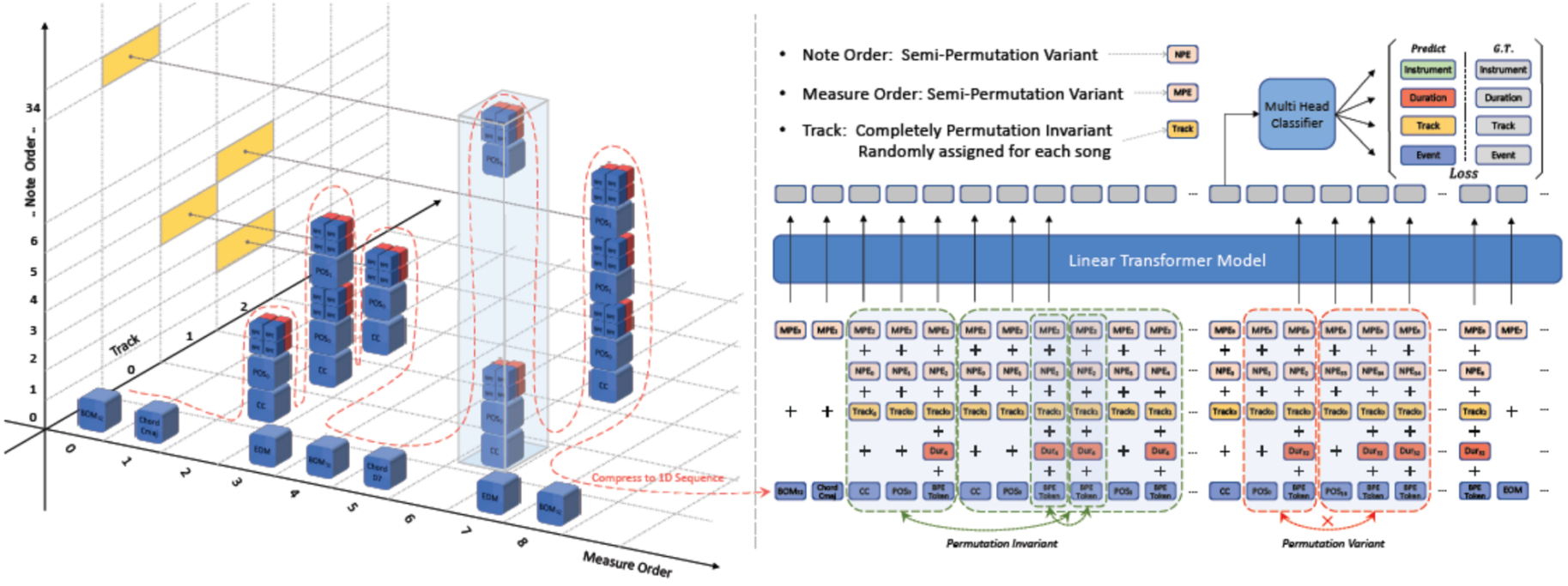
  - Another early multitrack Transformer: LakhNES [donahue19ismir]

Ref 1: Payne et al, “MuseNet,” OpenAI blog, 2019
Ref 2: Donahue et al, “LakhNES: Improving multi-instrumental music generation with cross-domain pre-training”, ISMIR 2019

Transformer Example 3: SymphonyNet [liu22ismir]

https://symphonynet.github.io/

- 12-layer Transformer decoder trained on 46,359 symphonic MIDI music



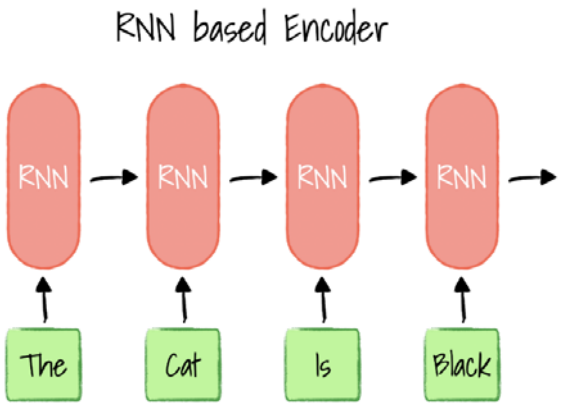
Ref: Liu et al, "Symphony generation with permutation invariant language model", ISMIR 2022

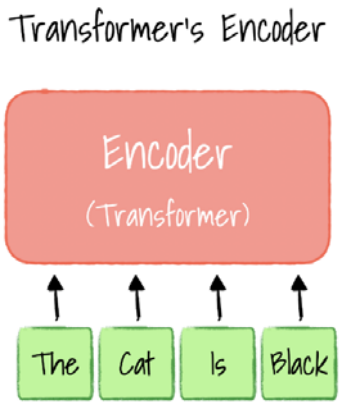
RNNs vs. Transformers

- **RNNs** work well when the sequence is ***not that long***
  - Melodies (e.g., Folk RNN, MusicVAE)
  - 8-bar piano performance (e.g., JazzRNN)

- **Transformers** work better for ***longer*** *sequences*
  - At the expense of computational power
    - An RNN-based model may contain **3** layers or so
    - A Transformer-based model may use up to >**72** layers … (e.g., MuseNet)
    - Usually, a **12**-layer Transformer decoder (with about 40M learnable parameters) works okay
  - Represent **the SOTA** approach for sequence modeling, and for building **LLMs**

RNNs vs. Transformers





- **RNNs**
  - use only a latent vector
  - $y_t = f(y_{t-1}, h_{t-1})$
  - **training: slower**
  - **inference: faster**

- **Transformers**
  - multiple latent vectors
  - $y_t = f(y_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L})$
  - **training: faster** (teacher forcing → parallel training)
  - **inference: slower**

Key to the Text-based Approach

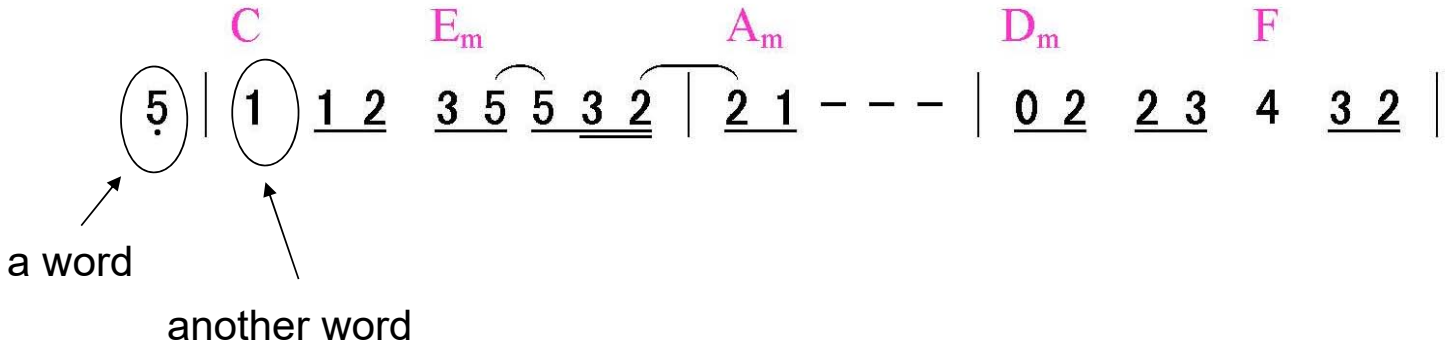
- **Neural sequence model**
  - RNNs, hierarchical RNN, Transformers
  - *Attention mechanism design*
- **Token representation of music**
  - MIDI-like, …
  - *Token design*

Outline

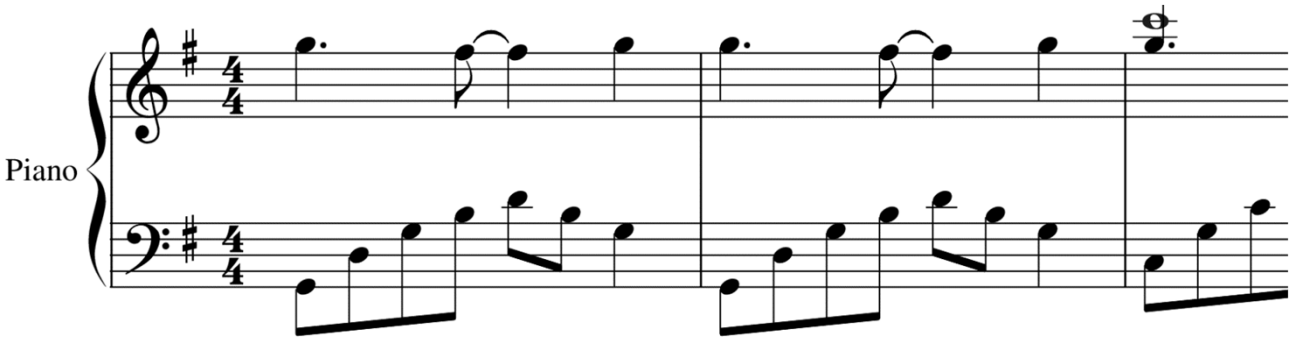
- General ideas
- Image-based approach for symbolic-domain music generation
- Text-based approach for symbolic-domain music generation
- **Token design**
- Building your first MIDI Transformer

Representing Music as Text

- Representing melody is easy

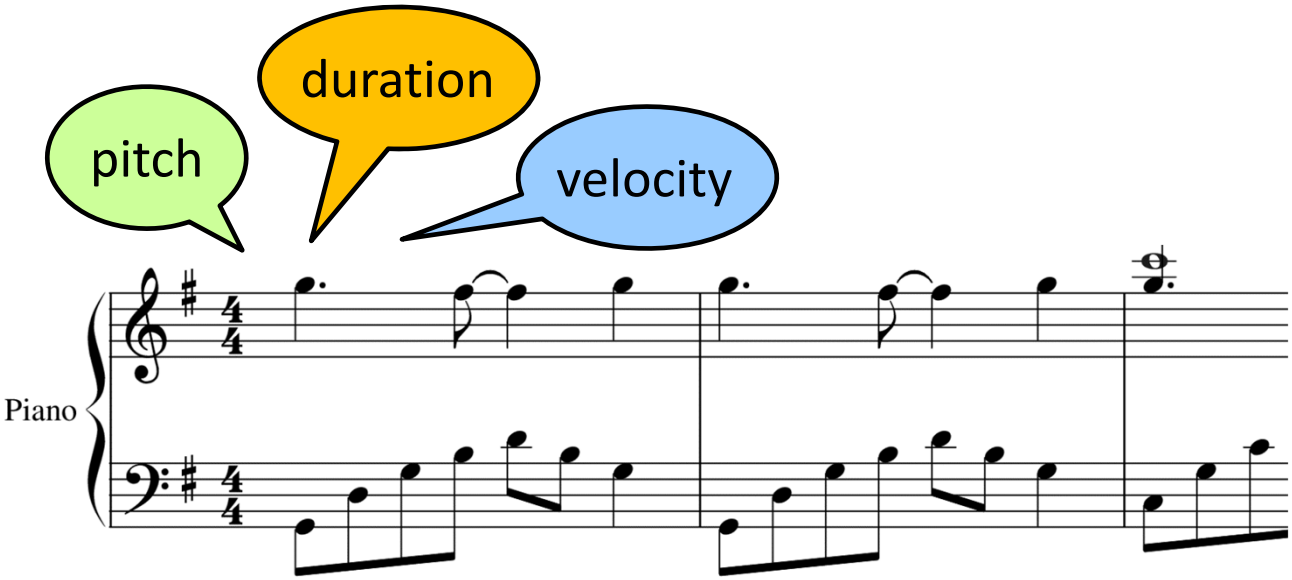


- How about??



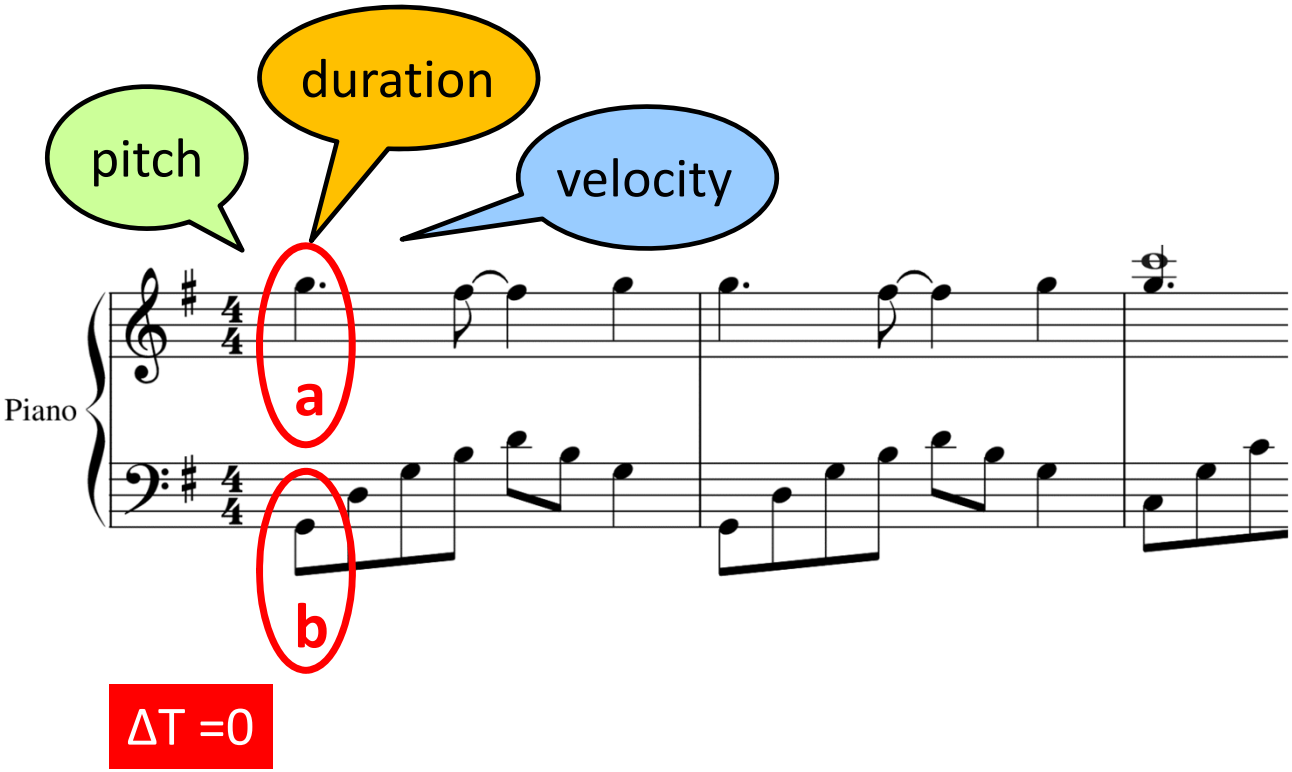
MIDI-like Representation

- Represent a music piece as a *token sequence*
- Describe a musical note of a MIDI performance by **three** tokens



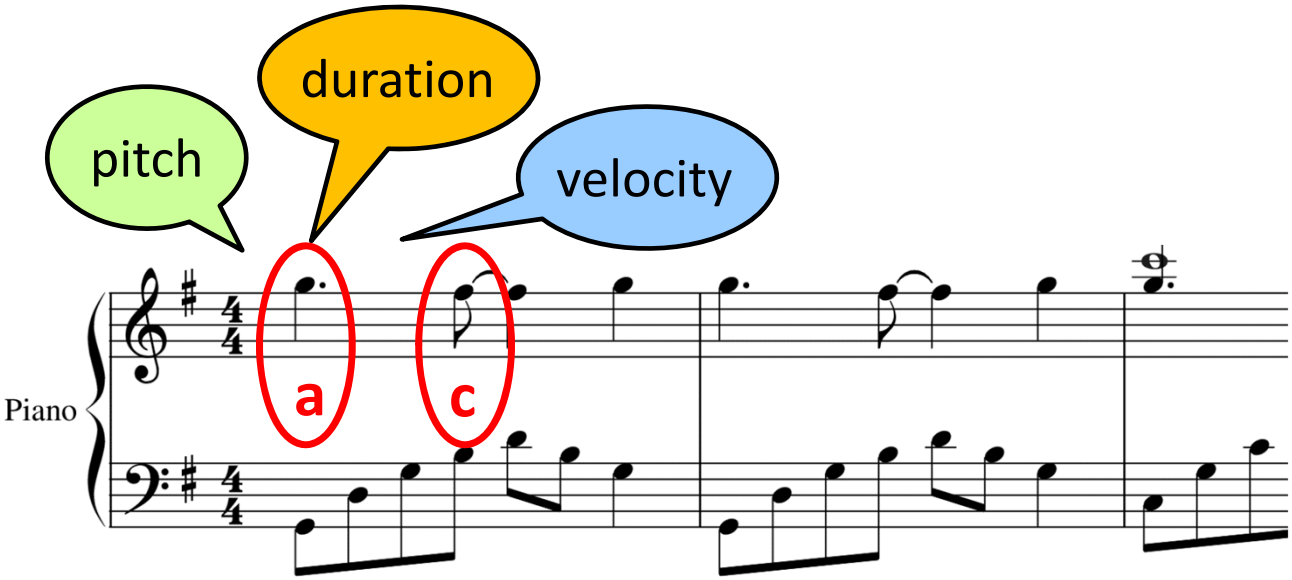
MIDI-like Representation

- And, one additional token to mark "**time-shift**" (i.e., ΔT; interval time)
  - To deal with concurrent notes



MIDI-like Representation

- And, one additional token to mark "**time-shift**" (i.e., ΔT; interval time)
  - To deal with concurrent notes

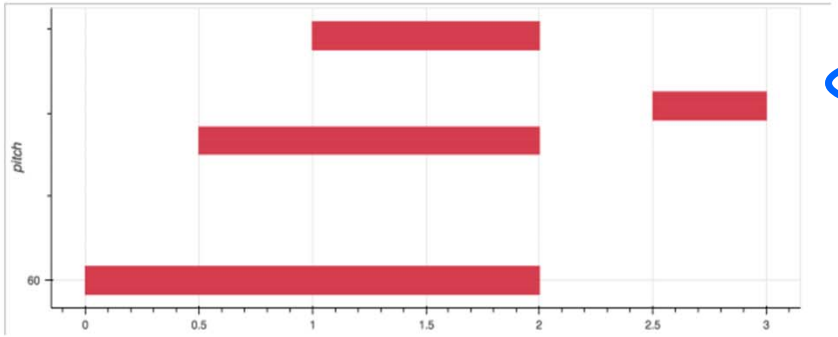


ΔT =50ms

An Example: PerformanceRNN [simon17]

https://magenta.tensorflow.org/performance-rnn

- Generate piano performances with **expressive timing** + **dynamics**
  - 128 **note-on** events + 128 **note-off** events: start or release a MIDI note
  - 100 **time-shift** events in increments of 10 ms up to 1 second; move forward in time to the next note event
  - 32 **velocity** events, corresponding to MIDI velocities quantized into 32 bins

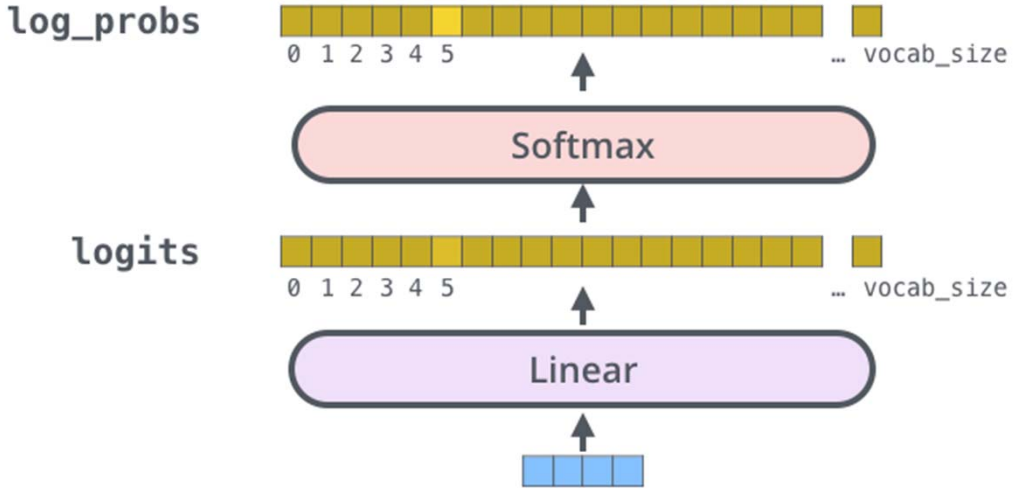


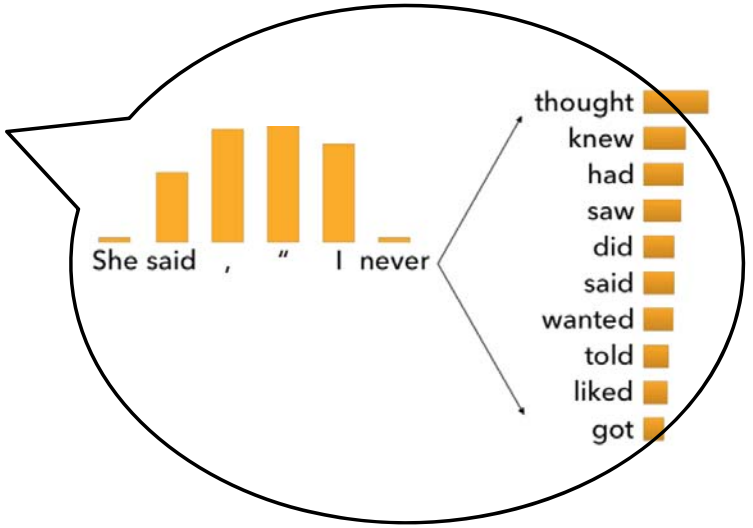

```
SET_VELOCITY<80>, NOTE_ON<60>
TIME_SHIFT<500>, NOTE_ON<64>
TIME_SHIFT<500>, NOTE_ON<67>
TIME_SHIFT<1000>, NOTE_OFF<60>, NOTE_OFF<64>,
NOTE_OFF<67>
TIME_SHIFT<500>, SET_VELOCITY<100>, NOTE_ON<65>
TIME_SHIFT<500>, NOTE_OFF<65>
```

Ref: Simon & Oore, "Performance RNN: Generating music with expressive timing and dynamics," 2017

Issues of the “Time-Shift” Token

- There is a “**sampling**” mechanism at Transformers’ output
  - **Errors in Time-Shift accumulate** → the generated music lacks stable rhythmic structure
  - The model does not have built-in notion of beats/downbeats





(images from the internet)

Another Example: OpenAI's MuseNet

v — **velocity** (volume), in [0, 127]
v0 — **offset** of a note
v72 — **onset** of a note
G1 — **pitch**
piano — **instrument**
wait — move the **time** marker

```
bach piano_strings start tempo90 piano:v72:G1
piano:v72:G2 piano:v72:B4 piano:v72:D4 violin:v80:G4
piano:v72:G4 piano:v72:B5 piano:v72:D5 wait:12
piano:v0:B5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:4 piano:v0:G1 piano:v0:G2 piano:v0:B4 piano:v0:D4
violin:v0:G4 piano:v0:G4 wait:1 piano:v72:G5 wait:12
piano:v0:G5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:5 piano:v72:B5 wait:12
```

(v72: onset of the piano note G1)

Another Example: OpenAI's MuseNet

v — **velocity** (volume), in [0, 127]
v0 — **offset** of a note
v72 — **onset** of a note
G1 — **pitch**
piano — **instrument**
wait — move the **time** marker

```
bach piano_strings start tempo90 piano:v72:G1
piano:v72:G2 piano:v72:B4 piano:v72:D4 violin:v80:G4
piano:v72:G4 piano:v72:B5 piano:v72:D5 wait:12
piano:v0:B5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:4 piano:v0:G1 piano:v0:G2 piano:v0:B4 piano:v0:D4
violin:v0:G4 piano:v0:G4 wait:1 piano:v72:G5 wait:12
piano:v0:G5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:5 piano:v72:B5 wait:12
```

(special "wait" token to indicates the delta time between events)

Another Example: OpenAI's MuseNet

v — **velocity** (volume), in [0, 127]
v0 — **offset** of a note
v72 — **onset** of a note
G1 — **pitch**
piano — **instrument**
wait — move the **time** marker

```
bach piano_strings start tempo90 (1)piano:v72:G1
(2)piano:v72:G2 (3)piano:v72:B4 (4)piano:v72:D4 (5)violin:v80:G4
(6)piano:v72:G4 (7)piano:v72:B5 (8)piano:v72:D5 wait:12
piano:v0:B5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:4 piano:v0:G1 piano:v0:G2 piano:v0:B4 piano:v0:D4
violin:v0:G4 piano:v0:G4 wait:1 piano:v72:G5 wait:12
piano:v0:G5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:5 piano:v72:B5 wait:12
```

(all these eight notes are played at the same time)

Another Example: OpenAI's MuseNet

v — **velocity** (volume), in [0, 127]
v0 — **offset** of a note
v72 — **onset** of a note
G1 — **pitch**
piano — **instrument**
wait — move the **time** marker

```
bach piano_strings start tempo90 piano:v72:G1
piano:v72:G2 piano:v72:B4 piano:v72:D4 violin:v80:G4
piano:v72:G4 piano:v72:B5 piano:v72:D5 wait:12
piano:v0:B5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:4 piano:v0:G1 piano:v0:G2 piano:v0:B4 piano:v0:D4
violin:v0:G4 piano:v0:G4 wait:1 piano:v72:G5 wait:12
piano:v0:G5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:5 piano:v72:B5 wait:12
```

(duration of the note G1: 12+5+12+4=33 ms)

Another Example: OpenAI's MuseNet

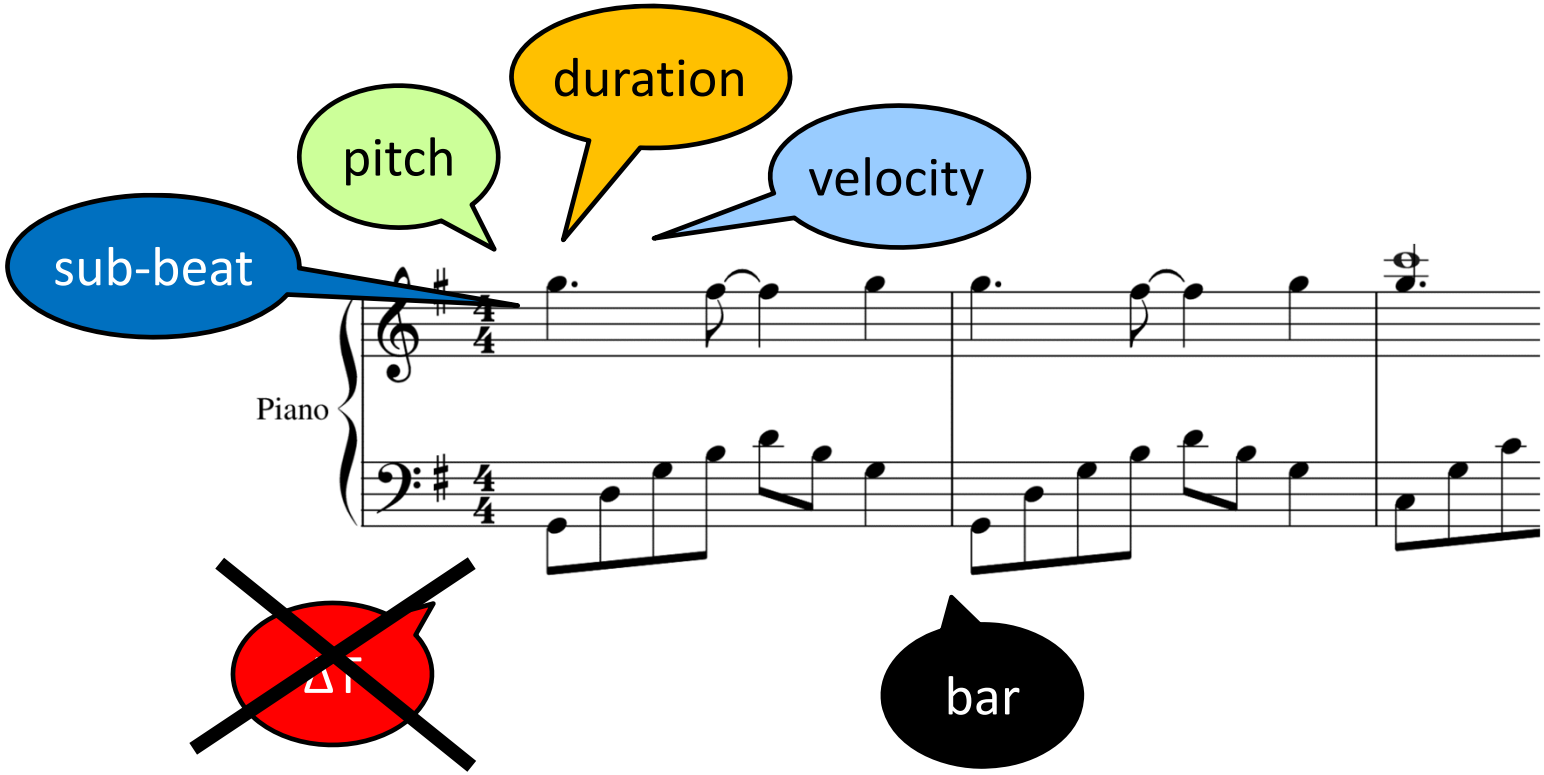
v — **velocity** (volume), in [0, 127]
v0 — **offset** of a note
v72 — **onset** of a note
G1 — **pitch**
piano — **instrument**
wait — move the **time** marker

```
bach piano_strings start tempo90 piano:v72:G1
piano:v72:G2 piano:v72:B4 piano:v72:D4 violin:v80:G4
piano:v72:G4 piano:v72:B5 piano:v72:D5 wait:12
piano:v0:B5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:4 piano:v0:G1 piano:v0:G2 piano:v0:B4 piano:v0:D4
violin:v0:G4 piano:v0:G4 wait:1 piano:v72:G5 wait:12
piano:v0:G5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:5 piano:v72:B5 wait:12
```

No bar lines!

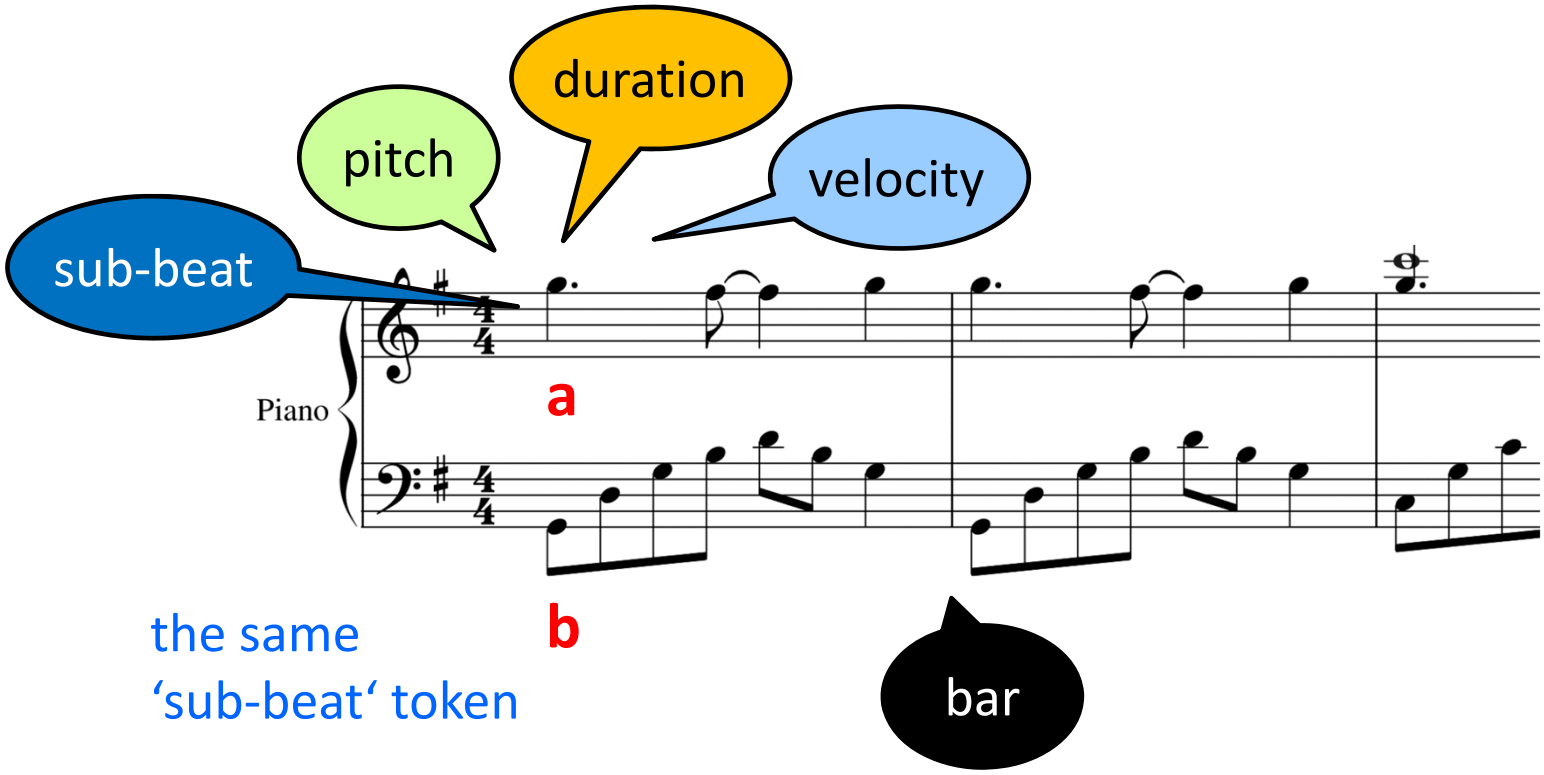
REMI: Revamped MIDI Representation

- REMI: An alternative token representation of music
  - **Mark the bar lines** by "**bar**" tokens
  - **Indicate the position of a note within a bar** by "**sub-beat**" tokens



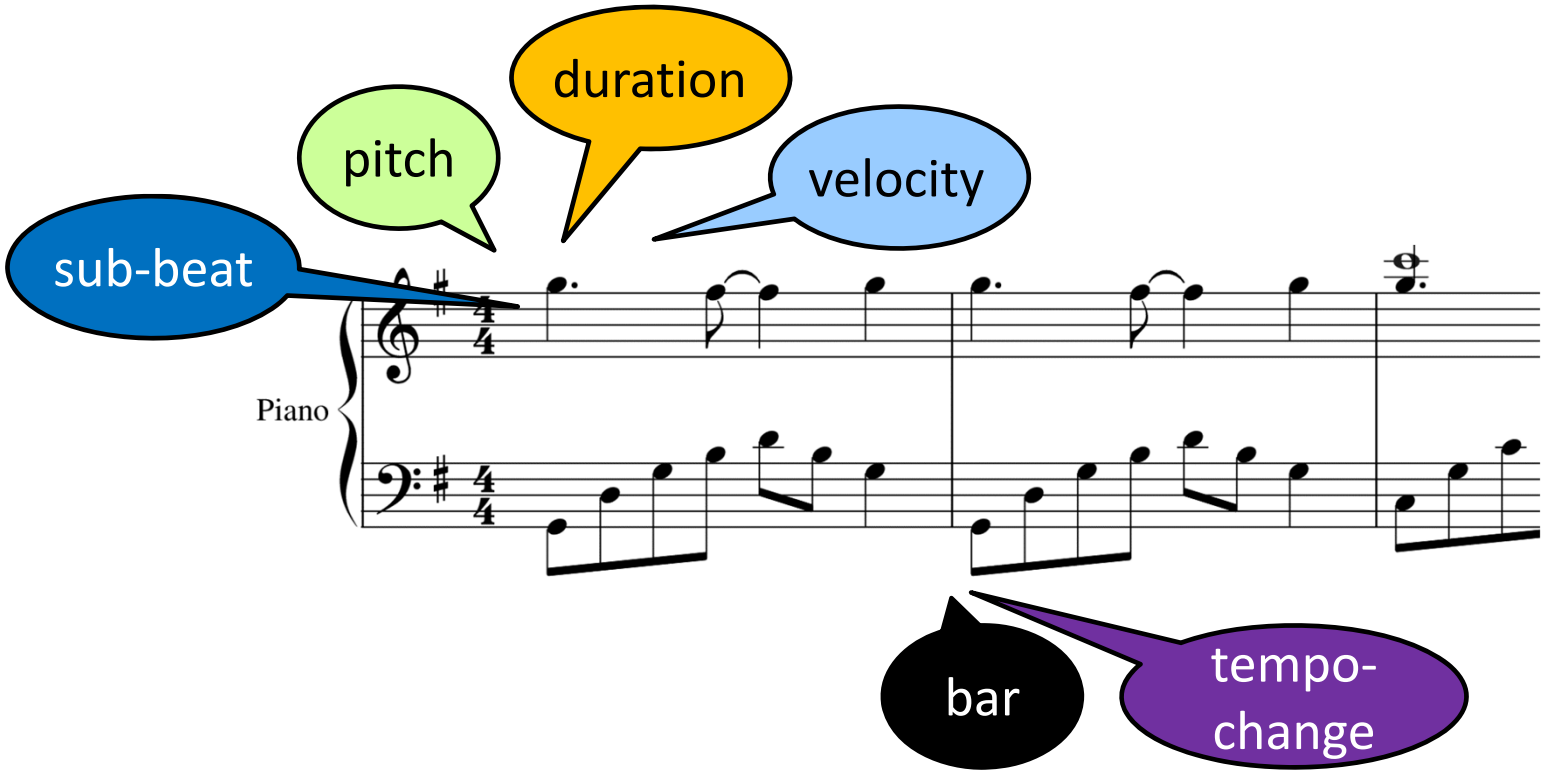
REMI: Revamped MIDI Representation

- REMI: An alternative token representation of music
  - **Mark the bar lines** by "**bar**" tokens
  - **Indicate the position of a note within a bar** by "**sub-beat**" tokens



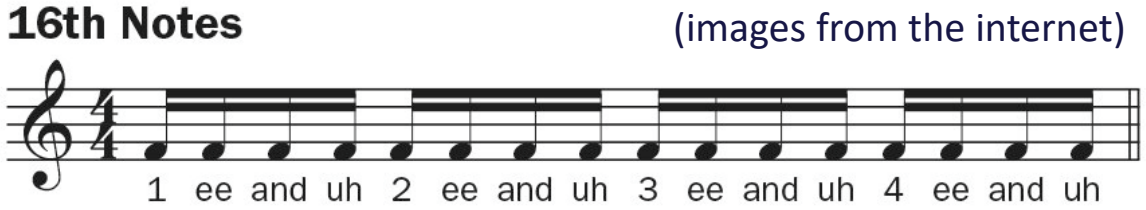
REMI: Revamped MIDI Representation

- REMI: An alternative token representation of music
  - Add **bar-wise tempo-change** tokens when needed to account for **tempo variations within a song** (the "tempo-change" tokens are after "bar" tokens)



REMI: Revamped MIDI Representation

- REMI: An alternative token representation of music
  - **Mark the bar lines** by "**bar**" tokens and **divide a bar into 16 discrete sub-beats**
  - **Symbolic timing + explicit time grid** (instead of using relative absolute time intervals)



  - **"Tempo-change" tokens** to facilitate **symbolic-to-absolute timing conversion**
  - How to get bar/sub-beat info?
    - MIDI files: already there
    - Transcribed performances: use beat/downbeat tracking

Ref: Huang et al, "Pop Music Transformer: Beat-based modeling and generation of expressive Pop piano compositions", ACM Multimedia 2020

REMI: Revamped MIDI Representation

- REMI: An alternative token representation of music
  - Use `**Sub-beat, Bar, and Tempo**` instead of **Time-Shift**
  - Use `**Note-Duration**` instead of **Note-Off**
    - note-on and note-off are native MIDI events
    - but using note-duration seems to have better musical meaning
  - Add `**Chord**` tokens

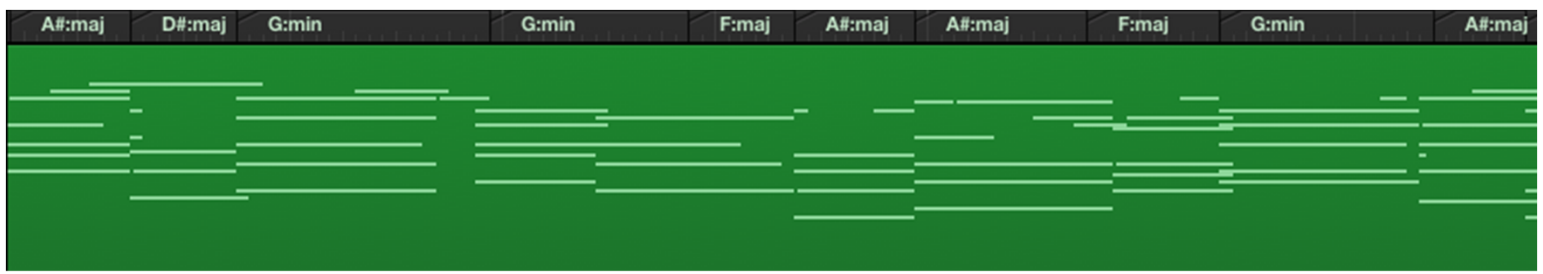
	MIDI-like	REMI
Note onset	NOTE-ON (0-127)	NOTE-ON (0-127)
Note offset	NOTE-OFF (0-127)	NOTE DURATION (32th note multiples)
Note velocity	NOTE VELOCITY (32 bins)	NOTE VELOCITY (32 bins)
Time grid	TIME-SHIFT (10-1000ms)	POSITION (16 bins) & BAR (1)
Tempo changes	✗	TEMPO (30-209 BPM)
Chord	✗	CHORD (60 types)

Table 1: A comparison of the commonly-used MIDI-like event representation with the proposed one, REMI. In the brackets, we show the corresponding ranges.

REMI: Modeling the Chord

Before (no chord information)

After (with chord information)



* Both <u>encode</u> and <u>generate</u> the chord information

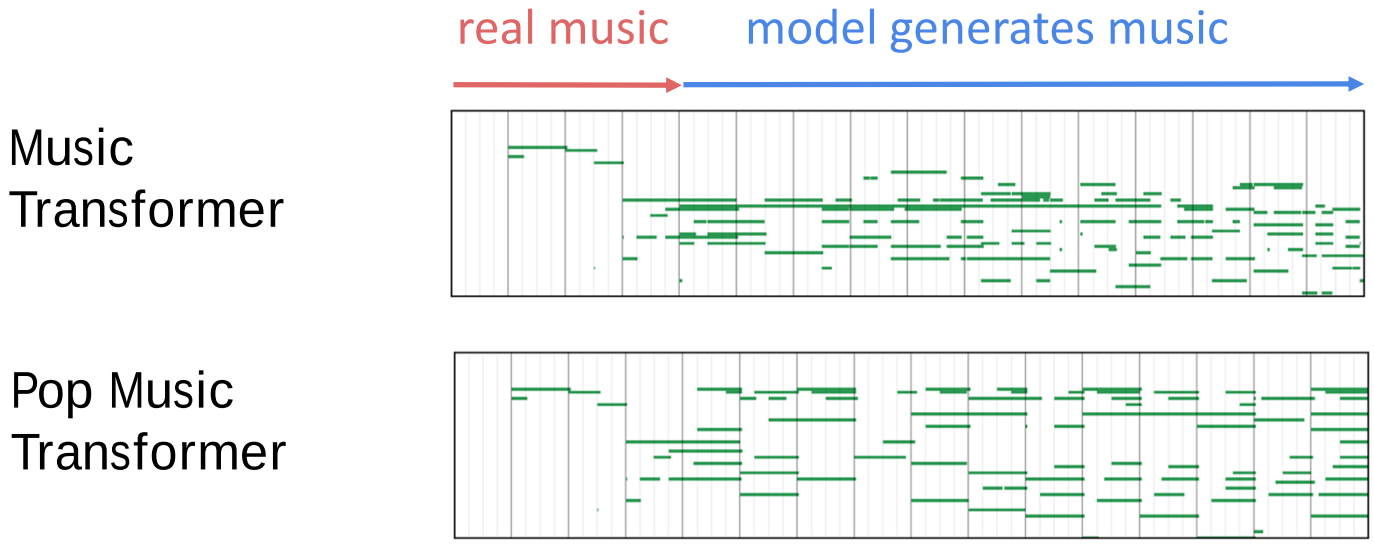
Ref: Huang et al, “Pop Music Transformer: Beat-based modeling and generation of expressive Pop piano compositions”, ACM Multimedia 2020

piano

Pop Music Transformer [huang20mm]

https://github.com/YatingMusic/remi

- Adopt the **REMI** representation
- Outperforms the Music Transformer ("MIDI-like") for generating pop piano performances



Ref: Huang et al, "Pop Music Transformer: Beat-based modeling and generation of expressive Pop piano compositions", ACM Multimedia 2020

Pop Music Transformer [huang20mm]

Pop Music Transformer ("REMI") outscores Music Transformer ("Baselines 1 & 3") in a user study that involves 76 raters

Pairs		Wins	Losses	p-value
REMI	Baseline 1	103	49	5.623e-5
REMI	Baseline 3	92	60	0.0187
Baseline 3	Baseline 1	90	62	0.0440

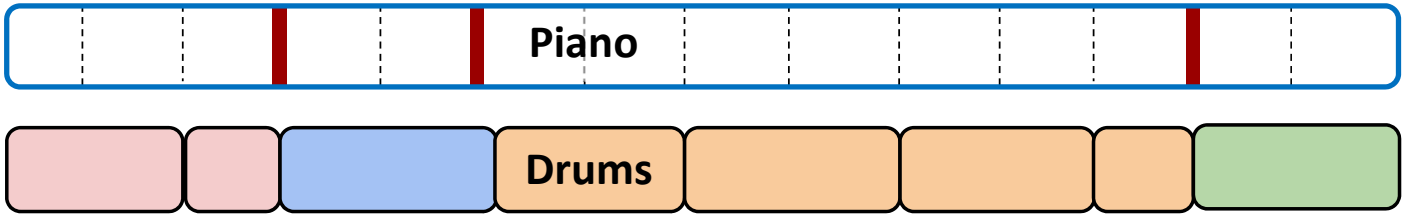
Table 5: The result of pairwise comparison from the user study, with the p-value of the Wilcoxon signed-rank test.

Method	Note offset	Time grid	TEMPO	CHORD	Beat STD	Downbeat STD	Downbeat salience
Baseline 1	NOTE-OFF	TIME-SHIFT (10-1000ms)			0.0968	0.3561	0.1033
Baseline 2	DURATION	TIME-SHIFT (10-1000ms)			0.0394	0.1372	0.1651
Baseline 3	DURATION	TIME-SHIFT (16th-note multiples)			0.0396	0.1383	0.1702
REMI	DURATION	POSITION & BAR	✓	✓	0.0386	0.1376	0.2279
	DURATION	POSITION & BAR	✓		0.0363	0.1265	0.1936
	DURATION	POSITION & BAR		✓	0.0292	0.0932	0.1742
	DURATION	POSITION & BAR			0.0199	0.0595	0.1880
Real data					0.0607	0.2163	0.2055

Pop Music Transformer

piano + drum guitar

- **Easier to add drums** (via structure analysis and grooving analysis)
  - https://soundcloud.com/yating_ai/sets/ai-piano-generation-demo-202004
  - https://soundcloud.com/yating_ai/sets/ai-pianodrum-generation-demo-202004



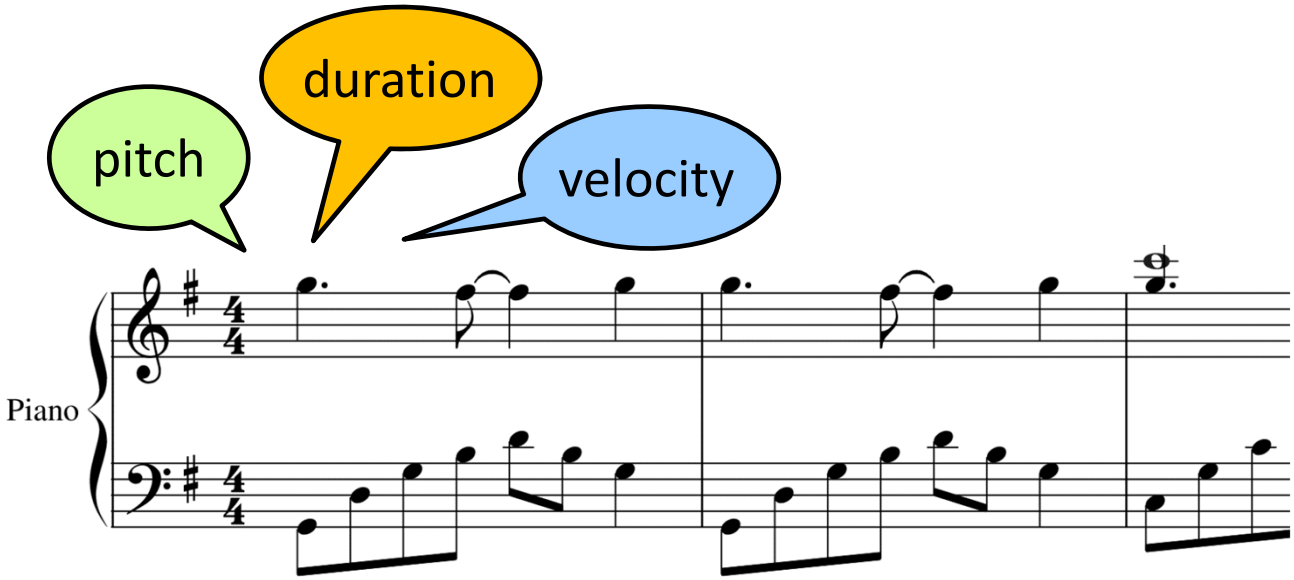
(Figure made by Wen-Yi Hsiao)

- Extensions can also generate **guitar tabs** [chen20ismir]
  - https://soundcloud.com/yating_ai/ai-guitar-tab-generation-202003/s-KHozfW0PTv5

Ref 1: Huang et al, "Pop Music Transformer: Beat-based modeling and generation of expressive Pop piano compositions", ACM Multimedia 2020
Ref 2: Chen et al, "Automatic composition of guitar tabs by Transformers and groove modeling", ISMIR 2020

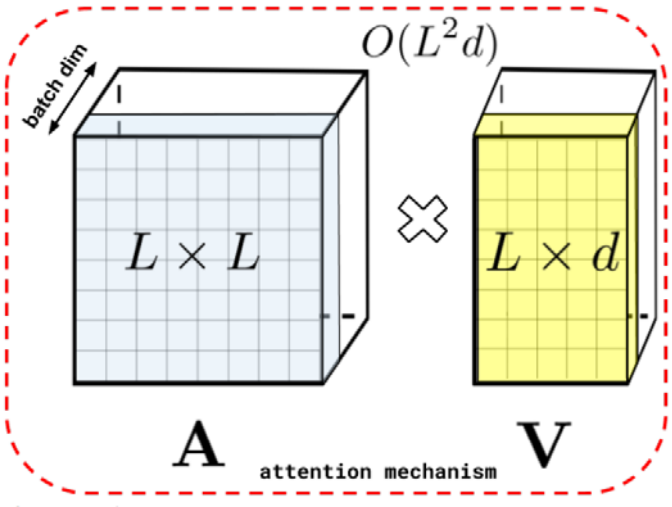
But…

- Both MIDI-like and REMI use multiple tokens to represent a musical note; isn't that inefficient?



The Problem: Transformers are Expensive

- **Transformers**
  - multiple latent vectors
  - $y_t = f(y_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L})$
  - Memory complexity: **O(L^2)**, where *L* denotes the length of the attention window (i.e., the history) because of the *L* x *L* similarity matrix
  - Need to cut a music pieces into **segments** to fit the VRAM



(Figure from the Performer paper)

The Problem: Transformers are Expensive

	Representation	Model	Attn. window
Music Transformer (Huang et al. 2019)	MIDI-like	Transformer	2,048
MuseNet (Payne 2019)	MIDI-like*	Transformer	4,096
LakhNES (Donahue et al. 2019)	MIDI-like*	Transformer-XL	512
(Choi et al. 2020)	MIDI-like	Transformer	2,048
Pop Music T. (Huang and Yang 2020)	REMI	Transformer-XL	512
Transformer VAE (Jiang et al. 2020)	MIDI-like	Transformer	128
Guitar Transformer (Chen et al. 2020)	REMI*	Transformer-XL	512
Jazz Transformer (Wu and Yang 2020)	REMI*	Transformer-XL	512
MMM (Ens and Pasquier 2020)	MIDI-like*	Transformer	2,048

- Most people use **512-token** attention window
  - But, a piano cover of a pop song can contain up to **5k** tokens
  - **Segmentation** of music pieces makes it hard to learn **long-term patterns**

Compound Word Transformer [hsiao21aaai]

https://github.com/YatingMusic/compound-word-transformer

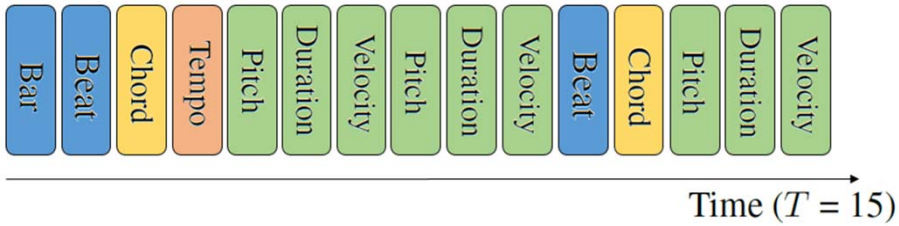
	GPU memory	Training time (single GPU)	Inference time (on GPU)	Quality MOS
Transformer-XL	4-17 GB	3-7 days	2x real time	3.0-**3.3**
CP Transformer (ours)	**4 GB**	**1 day**	**8x real time**	**3.3**

- **Compound Word**: grouping of tokens
- Shorter training time → easier to try different ideas
- Shorter inference time → good for real-time applications

Ref: Hsiao et al, "Compound Word Transformer: Learning to compose full-song music over dynamic directed hypergraphs," AAAI 2021

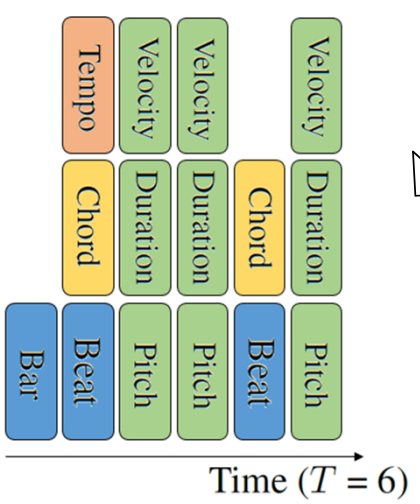
Compound Word Transformer [hsiao21aaai]

- Unlike the case in text, the vocabulary of music involves tokens of various token types (e.g., note-related, metric-related)



- Therefore, we can

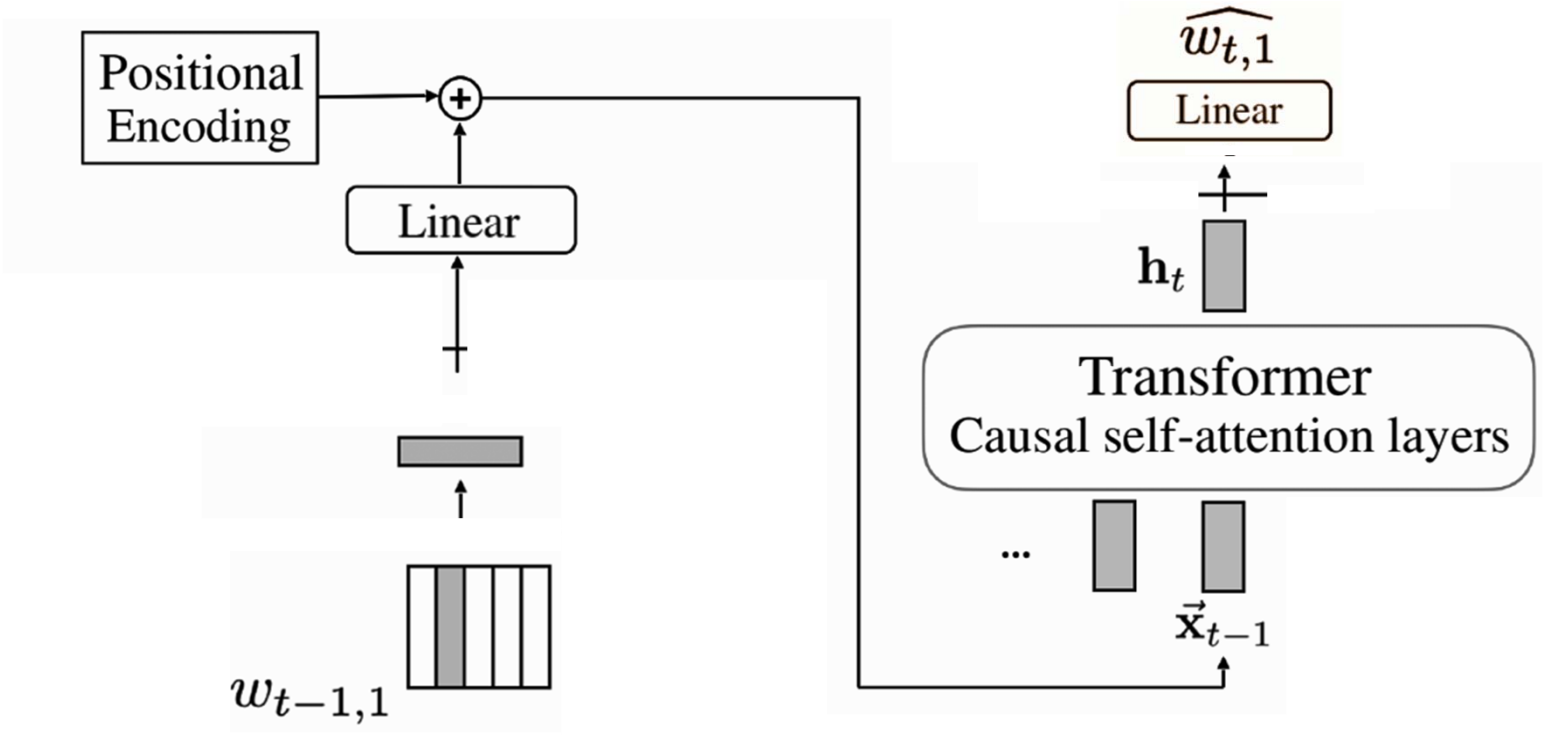
1. **group tokens** into "compound words" (e.g., pitch + duration + velocity)
2. **predict multiple tokens** of various types **at once in a single time step**
3. the embeddings of the predicted tokens are combined to be used as input to the next time step



Ref: Hsiao et al, "Compound Word Transformer: Learning to compose full-song music over dynamic directed hypergraphs," AAAI 2021

Original Transformer

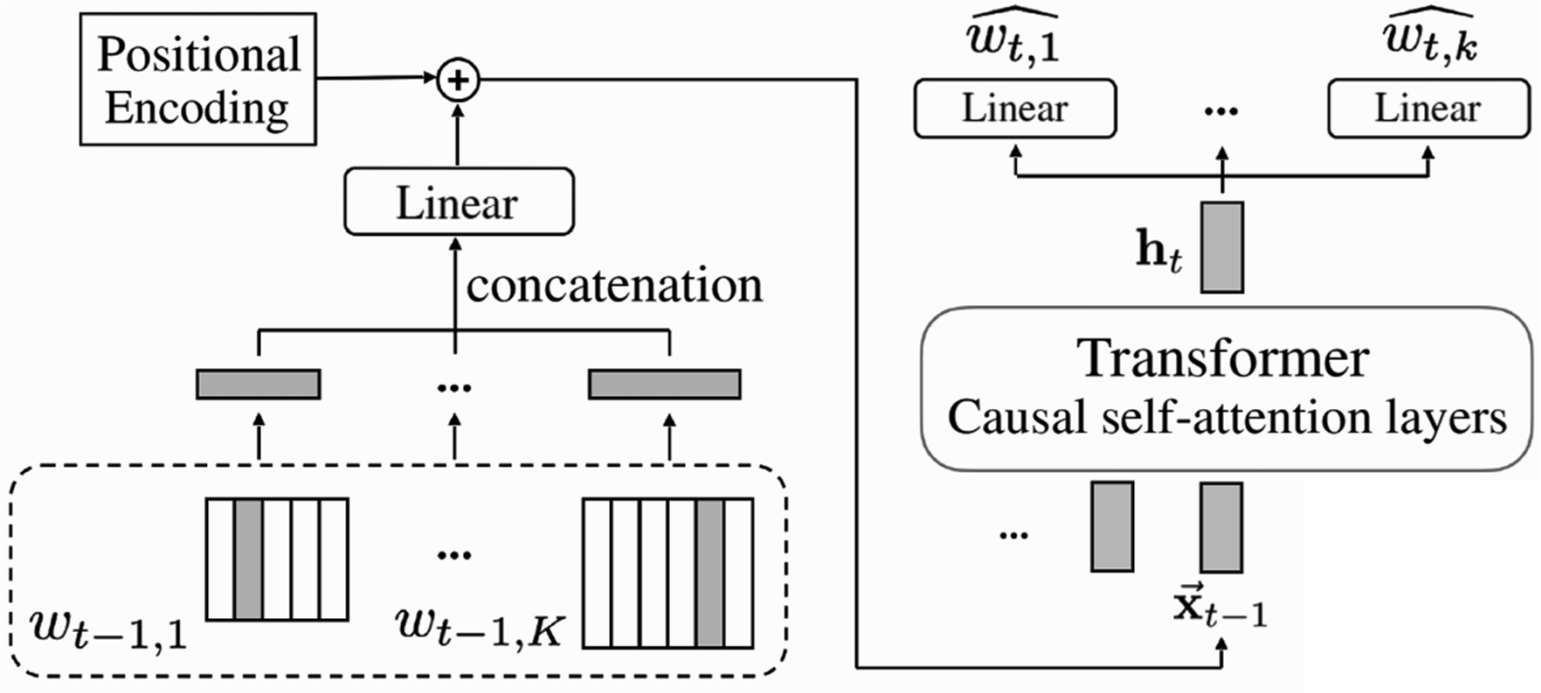
single output at each time step



single input at each time step

Compound Word Transformer

multiple output at each time step



multiple input at each time step

Transformer vs. CP Transformer

- **Transformer**
  - $y_t = f(y_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L})$
  - attention window: ***L* tokens**

- **CP Transformer**
  - multiple output: $\text{cp}_t = [y_t^{(1)}, y_t^{(2)}, \ldots, y_t^{(K)}]$
  - multiple input: $\text{cp}_{t-1} = [y_{t-1}^{(1)}, y_{t-1}^{(2)}, \ldots, y_{t-1}^{(K)}]$
  - $\text{cp}_t = f(\text{cp}_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L})$
  - `cp` is a super token
  - The latent vectors `h` are associated with each `cp`
  - Attention window: ***L* super-tokens**, or *LK* tokens

Compound Word Transformer [hsiao21aaai]

- We can now feed a **whole song** (up to 10,240 tokens) to our Transformer decoder for training on a single GPU with **11GB** VRAM
- The model converges within 1.5 days using an NVIDIA RTX 2080 Ti

Task	Representation + model@loss	Training time	GPU memory	Inference (/song) time (sec)	Inference (/song) tokens (#)
Conditional	Training data	—	—	—	—
	Training data (randomized)	—	—	—	—
	REMI + XL@0.44	3 days	4 GB	88.4	4,782
	REMI + XL@0.27	7 days	4 GB	91.5	4,890
	REMI + linear@0.50	3 days	17 GB	48.9	4,327
	CP + linear@0.27	0.6 days	10 GB	29.2	18,200
Unconditional	REMI + XL@0.50	3 days	4 GB	139.9	7,680
	CP + linear@0.25	1.3 days	4 GB	19.8	9,546

Ref: Hsiao et al, "Compound Word Transformer: Learning to compose full-song music over dynamic directed hypergraphs," AAAI 2021

Compound Word Transformer [hsiao21aaai]

- **Linear Transformer** [katharopoulos20icml]: **O(*L*)**
  → save GPU space
- **CP**: shorter sequence length
  → save GPU space further, and converge much faster

Representation + model@loss	Training time	GPU memory
Training data	—	—
Training data (randomized)	—	—
REMI + XL@0.44	3 days	4 GB
REMI + XL@0.27	7 days	4 GB
REMI + linear@0.50	3 days	17 GB
CP + linear@0.27	0.6 days	10 GB

Ref: Katharopoulos et al, "Transformers are RNNs: Fast autoregressive Transformers with linear attention", ICML 2020

Customed Design for Different Token Types

Different embedding size & sampling policy

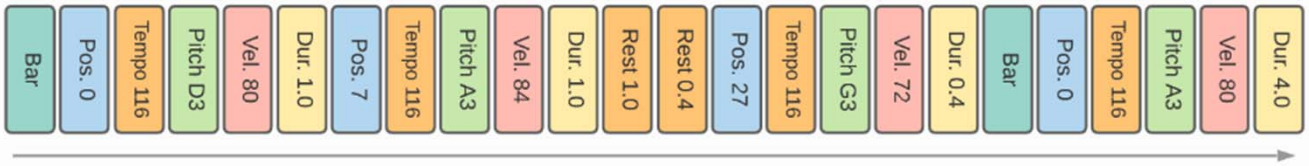
Repre.	Token type	Voc. size $\lvert\mathcal{V}_k\rvert$	Embed. size (d_k)	Sample$_k(\cdot)$ τ	ρ
CP	[track]	2 (+1)	3	1.0	0.90
	[tempo]	58 (+2)	128	1.2	0.90
	[position/bar]	17 (+1)	32	1.2	0.90
	[chord]	133 (+2)	256	1.0	0.90
	[pitch]	86 (+1)	512	1.0	0.90
	[duration]	17 (+1)	128	2.0	0.90
	[velocity]	24 (+1)	128	5.0	1.00
	[family]	4	32	1.0	0.99
	total	341 (+9)	—	—	—
REMI	total	338	512	1.2	0.90

Table 3: Details of the CP representation in our implementation, including that of the sampling policy (τ-tempered top-ρ sampling). For the vocabulary size, the values in the parentheses denote the number of special tokens such as [ignore].

Library: MidiTok

https://github.com/Natooz/MidiTok

REMI



class **miditok.REMI**(*tokenizer_config: TokenizerConfig = None, max_bar_embedding: int | None = None, params: str | Path = None*)

REMIPlus (multi-track extension of REMI)

class **miditok.REMIPlus**(*tokenizer_config: TokenizerConfig = None, max_bar_embedding: int | None = None, params: str | Path = None*)

MIDI-Like

class **miditok.MIDILike**

CPWord

class **miditok.CPWord**

Octuple

class **miditok.Octuple**

Outline

- General ideas
- Image-based approach for symbolic-domain music generation
- Text-based approach for symbolic-domain music generation
- Token design
- **Building your first MIDI Transformer**

Building Your First MIDI Transformer Decoder

- Step (1): Task & Data
- Step (2): Token representation
- Step (3): Choose a model
- Step (4): Train the model till the loss is sufficiently low
- Step (5): Do inference
- Step (6): Listen to the generated music!
- Step (7): Do evaluation

DIY Step (1): Task & Data

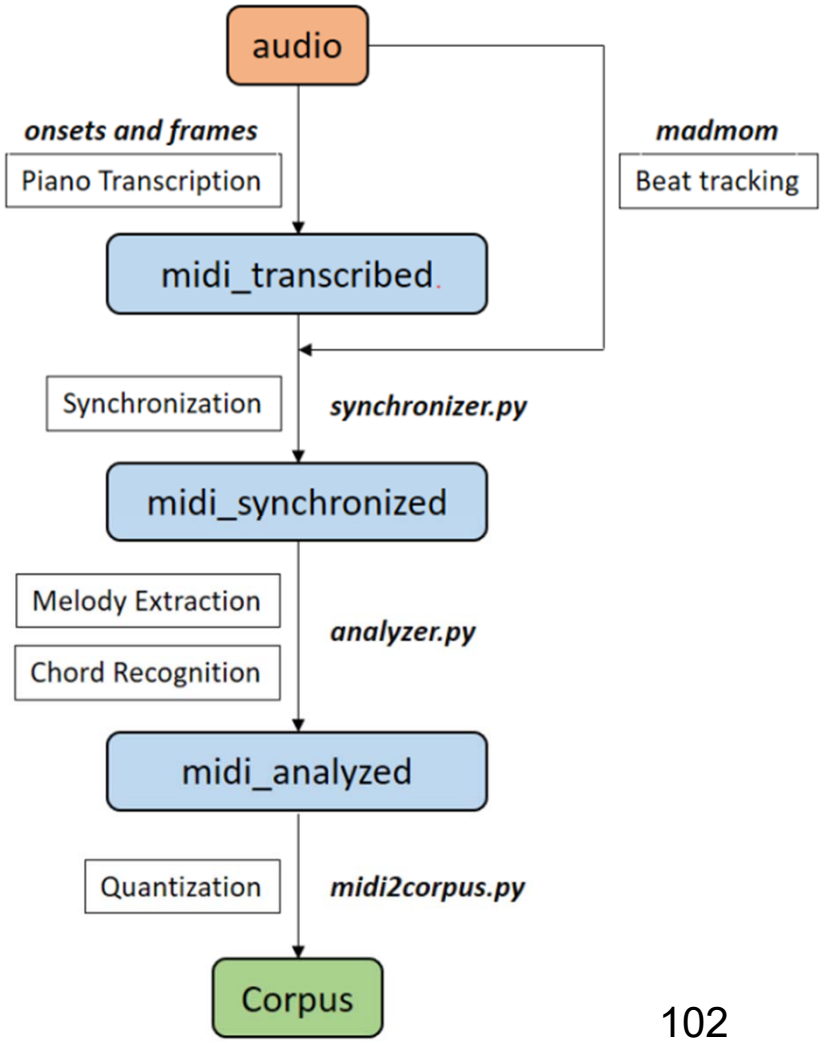
- Choose a task, and a dataset
  - https://github.com/affige/genmusic_demo_list
- List of symbolic musical datasets
  - https://github.com/wayne391/symbolic-music-datasets
- Three relatively more frequently studied cases
  - **Lead sheets** (MIDI score) — Jazz Transformer (ISMIR'20), Compose & Embellish (ICASSP'23)
  - **Multi-track MIDI** (MIDI score) — Multitrack Music Transformer (ICASSP'23)
  - **Piano performance** — Pop Music Transformer (MM'20), Compose & Embellish (ICASSP'23)

DIY: To Prepare a Dataset of Piano Performances

https://github.com/YatingMusic/compound-word-transformer/blob/main/dataset/Dataset.md

- Piano transcription
  - ("Onsets and frames" is no longer the SOTA → lecture 7)
- Beat and downbeat tracking
- Melody extraction
- Chord recognition
- Quantization

- Additional steps
  - Auto-tagging
  - Source separation



DIY Step (2): Choose A Token Representation

- Single-track
  - MIDI-like, REMI, CP, etc
- Multi-track
  - REMI+, Octuple, MMM, MuMIDI, etc
- Depend on your VRAM, you may need to decide a sequence length and cut your musical pieces into segments

DIY Step (3): Choose A Model

- PyTorch 2 Transformers
  https://pytorch.org/blog/accelerated-pytorch-2/

- Huggingface Transformer
  https://huggingface.co/docs/transformers/index

- x-Transformers
  https://github.com/lucidrains/x-transformers

DIY Step (4): Train the Model

- Train the model till the loss is ***sufficiently*** low
- Our experience on pop piano
  - Training loss around 0.30 gives better result
  - The model overfits and generates overly repeating content when the loss is too low
  - Keep several checkpoints and listen to the result to choose one

- No need to consider validation data
  - Unless you are building a conditional generation model with, for example, an encoder/decoder architecture

DIY Step (5): Do Inference

https://towardsdatascience.com/decoding-strategies-that-you-need-to-know-for-response-generation-ba95ee0faadc

- Greedy
- Beam search
- Random sampling
- Temperature
- **Top-K Sampling**
  - Only top K probable tokens should be considered for a generation
- **Nucleus Sampling**
  - Focuses on the smallest possible sets of Top-V words such that the sum of their probability is ≥ p

DIY Step (5): Do Inference

- **Tweak the sampling parameters** to strike a better balance between **diversity** and **quality**
  - The sampling parameters can influence the generation result **A LOT**
    - But, oftentimes people did not describe how they choose the sampling parameters for their proposed models and the evaluated baseline models, leading to unfair performance comparison
  - **A good practice in our Compose & Embellish (ICASSP'23) paper**

> Nucleus sampling [21] with temperatured softmax is employed during inference. We discover that the temperature τ and probability mass truncation point p greatly affect the intra-sequence repetitiveness and diversity of generated lead sheets. Thus, we follow [22] and search within $\tau = \{1.2, 1.3, 1.4\}$ and $p = \{.95, .97, .98, .99\}$ to find a combination with which our lead sheet model generates outputs with the closest mean perplexity (measured by the model itself) to that of validation real data. Finally, $\tau = 1.2$ and $p = .97$ are cho-

Ref: Wu et al, "Compose & Embellish: Well-structured piano performance generation via a two-stage approach", ICASSP 2023

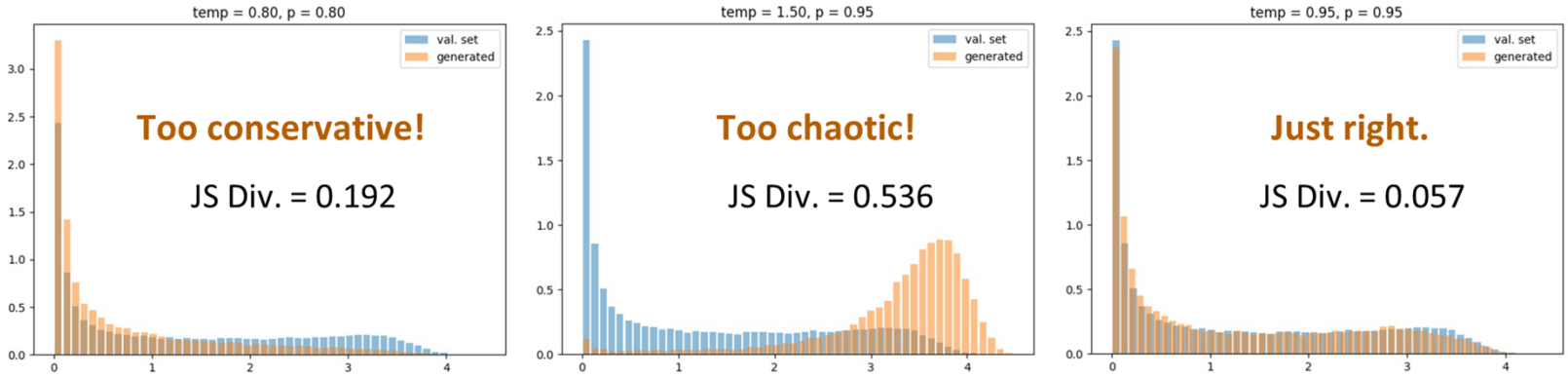
DIY Step (5): Do Inference

- **Nucleus Sampling w/ Temperature** [Holtzman et al., 2019]
  - Temperature -- **Reshape distribution**
  - Top-*p* -- **Truncate long tail**

→ collectively control how **aggressive** the sampling is

(Slide made by Shih-Lun Wu)

- Goal: match the **entropy histogram** during **sampling** to that computed on **val. set** by tuning temp. and top-*p*



Ref: Wu et al, “Compose & Embellish: Well-structured piano performance generation via a two-stage approach”, ICASSP 2023

DIY Step (6): Listen to the Generated Music!

https://www.fluidsynth.org/

https://github.com/bzamecnik/midi2audio

- Use a synthesizer if you prefer to listen to audio

FluidSynth

A SoundFont Synthesizer

FluidSynth is a real-time software synthesizer based on the SoundFont 2 specifications and has reached widespread distribution. FluidSynth itself does not have a graphical user interface, but due to its powerful API several applications utilize it and it has even found its way onto embedded systems and is used in some mobile apps.

```
from midi2audio import FluidSynth
```

Play MIDI:

```
FluidSynth().play_midi('input.mid')
```

Synthesize MIDI to audio:

```
# using the default sound font in 44100 Hz sample rate
fs = FluidSynth()
fs.midi_to_audio('input.mid', 'output.wav')
```

DIY Step (7): Do Evaluation

- More on this in the next lecture