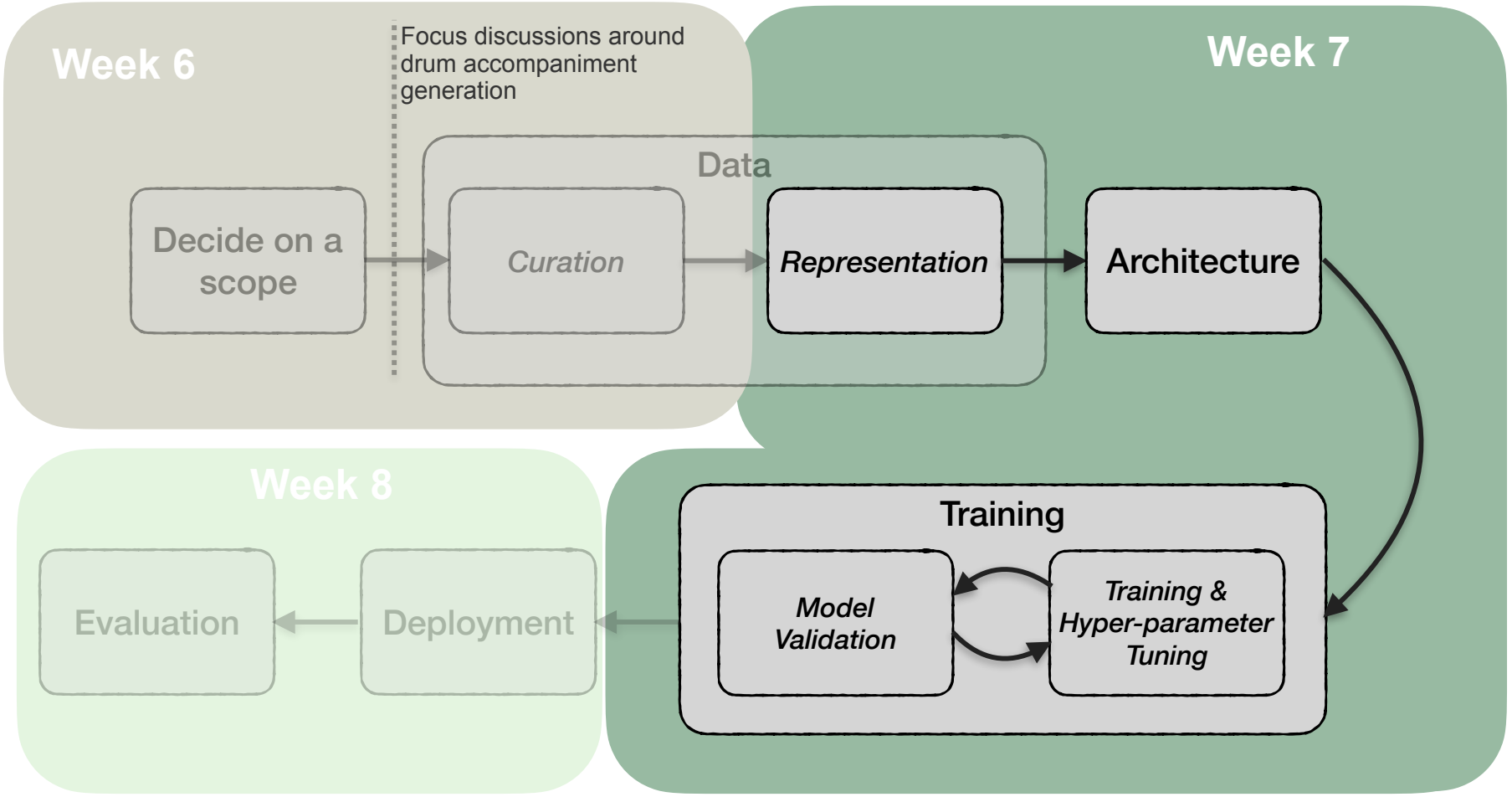
Designing NN-Based Generative Models of Music (Part 2)

Computational Music Creativity, SMC (2022/23)

Prepared by Behzad Haki, February 2023

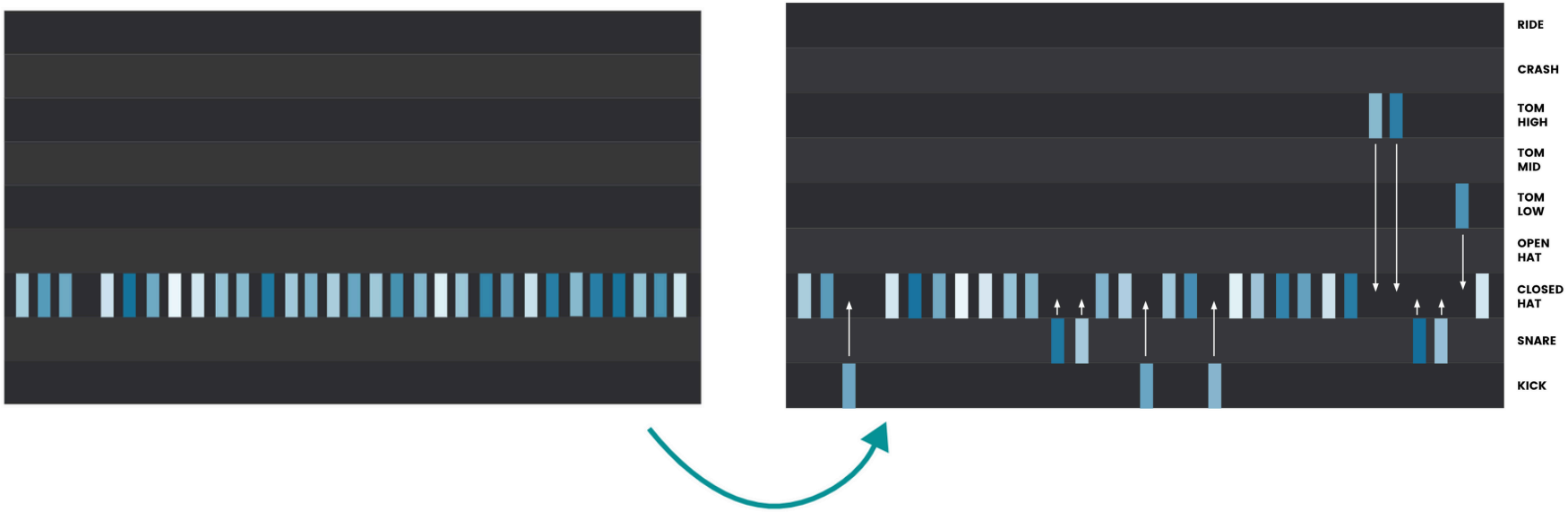
Focus of this week!



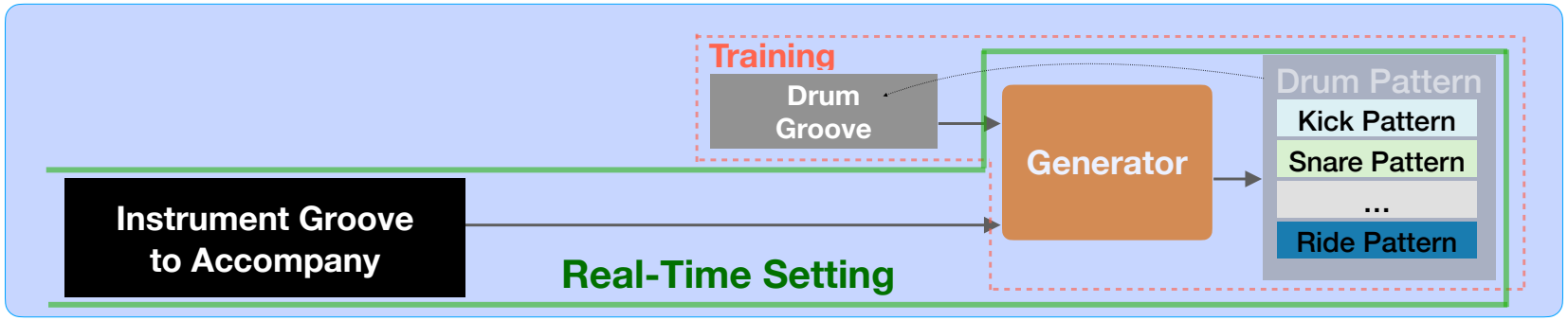
RECAP:

Create a real-time drum generation system that accompanies a given instrumental performance

Groove To Drum Generator



Unaware of the Type of Instrument to Accompany



Data: Representation

Most generative models of symbolic music are adapted from either NLP or Image domains

As such, musical information are generally represented as

(1) **Sentences**

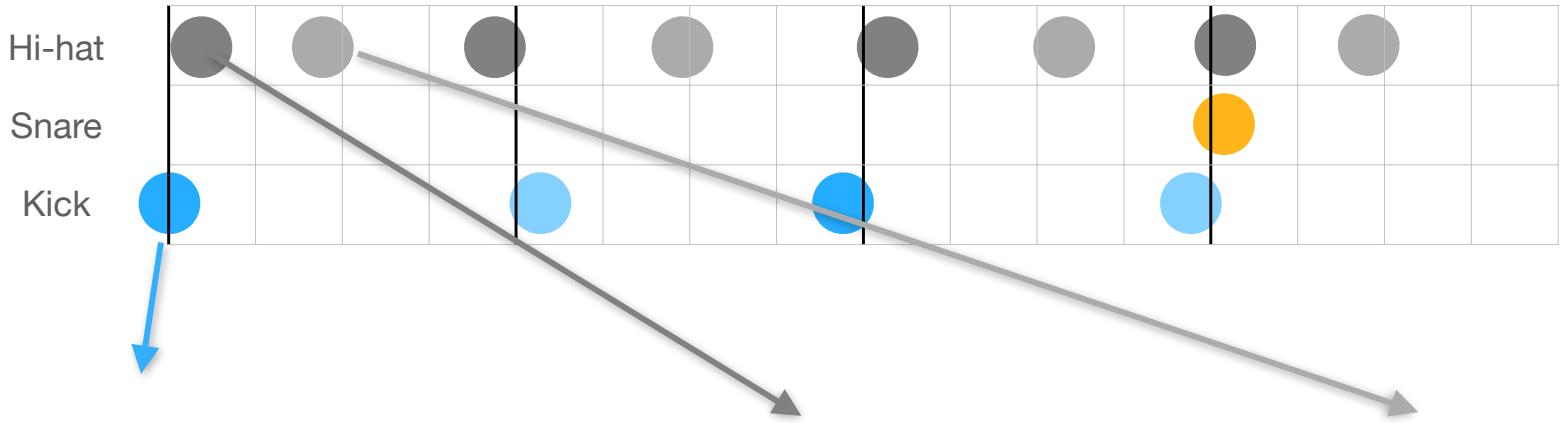
A sequence of words (tokens) describing the events occurring in a score in an ordered manner

(2) **Images**

An image of a piano roll, or sheet music of a score

Data: Representation As Words

Step 1: Describe the events as a sequence of words

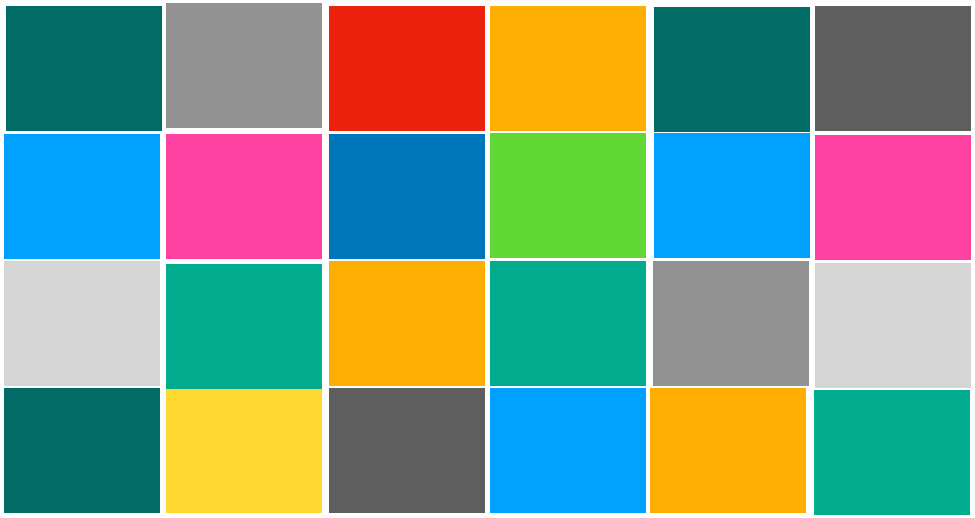


KickOnset, Velocity_100, *Silence_0.3*, HiHatOnset, Velocity_80, *Silence_2*, HiHatOnset, Velocity_0.4, …

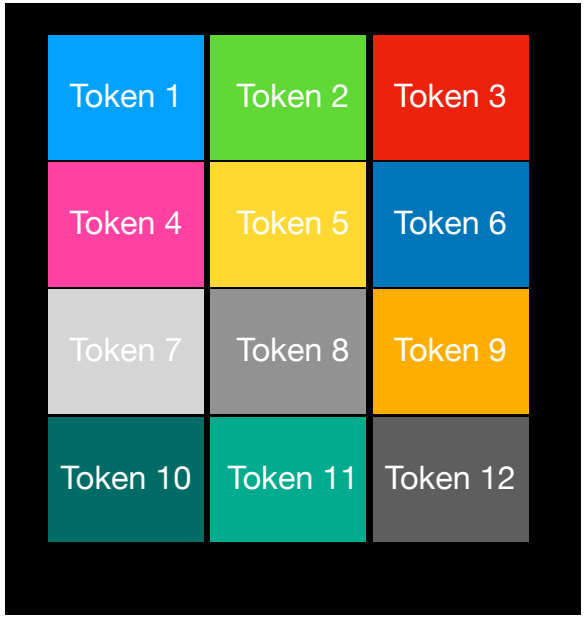
Data: Representation As Words

Step 2: Figure out how many unique words exist!

We call these unique words “Tokens”!



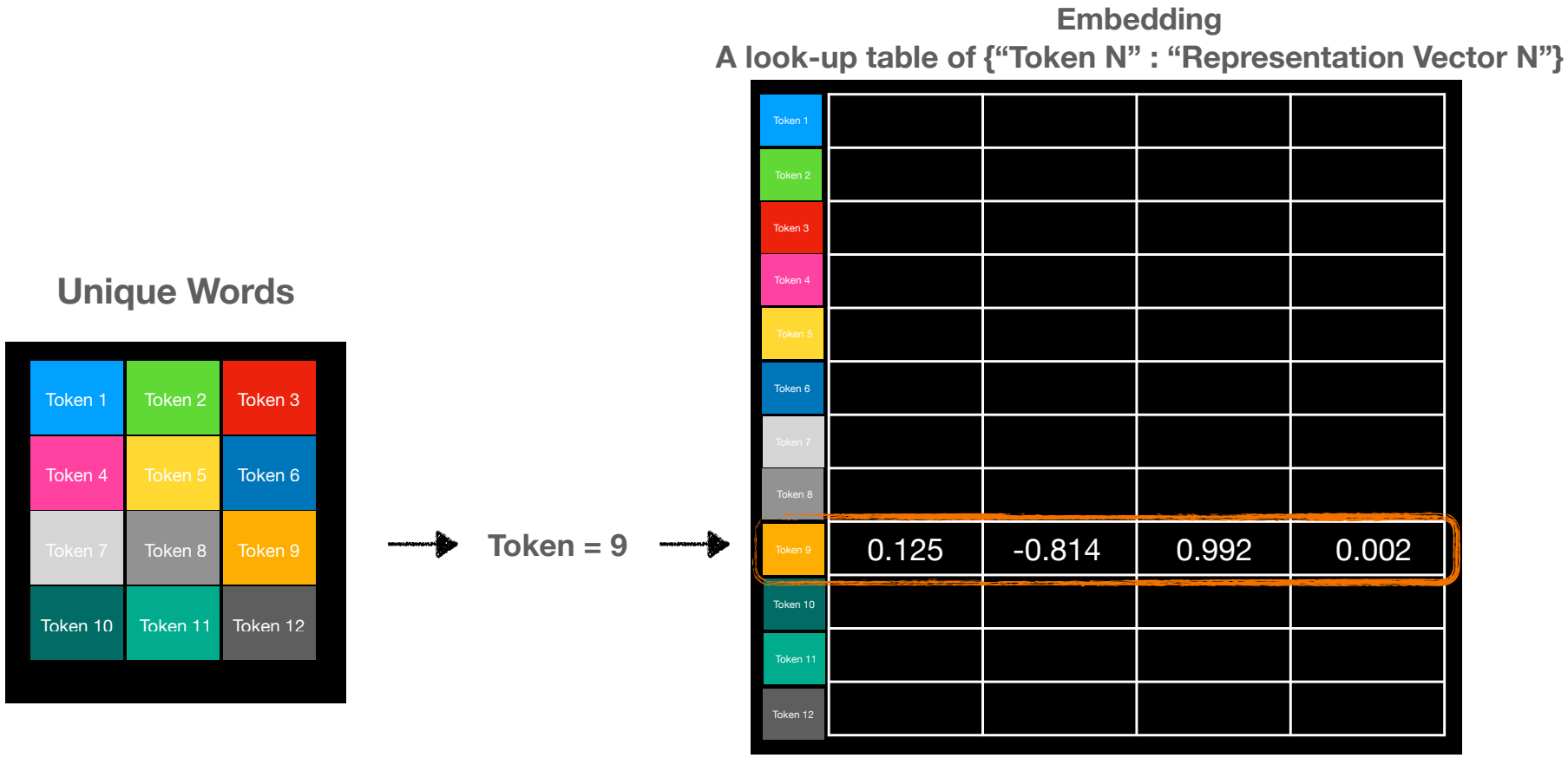
Unique Words



Data: Representation As Words

Step 3: Embed the tokens into a vector

In other words, represent each token using a unique vector



Data: Representation As Words

Why do we do this? Why don't we just assign equally distanced vectors to the tokens?

1. Similar words will end up with closer embeddings!

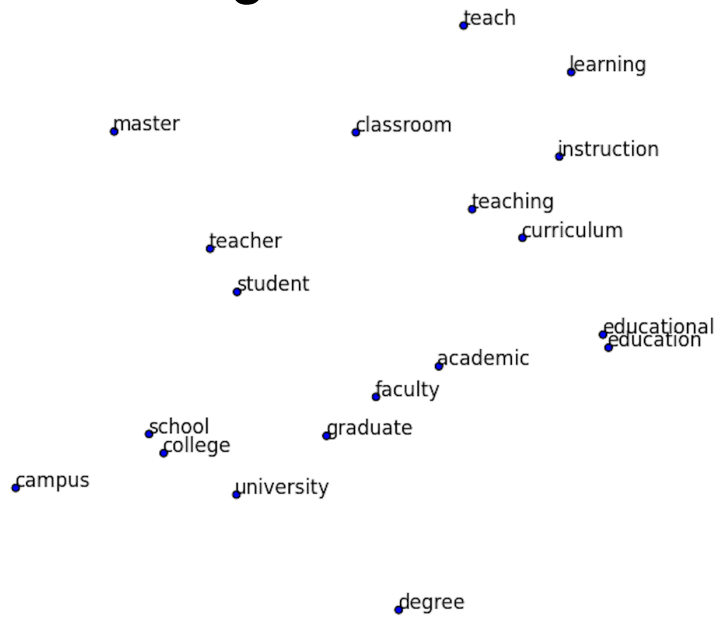
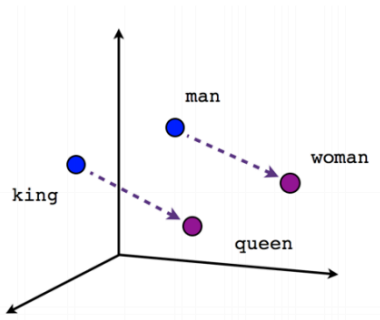


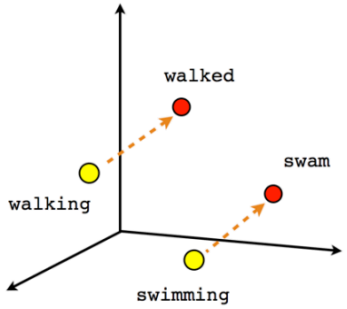
Image from https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12

Data: Representation As Words

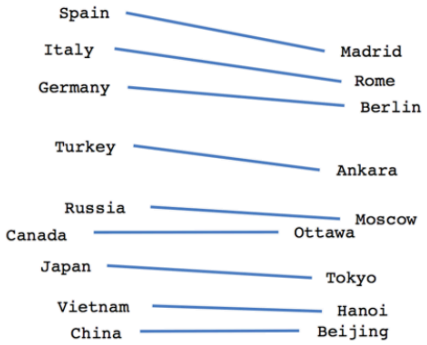
2. The distances between the embeddings are semantically meaningful!



Male-Female



Verb tense



Country-Capital

Plots from https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12

3. How do we come up with these vectors?

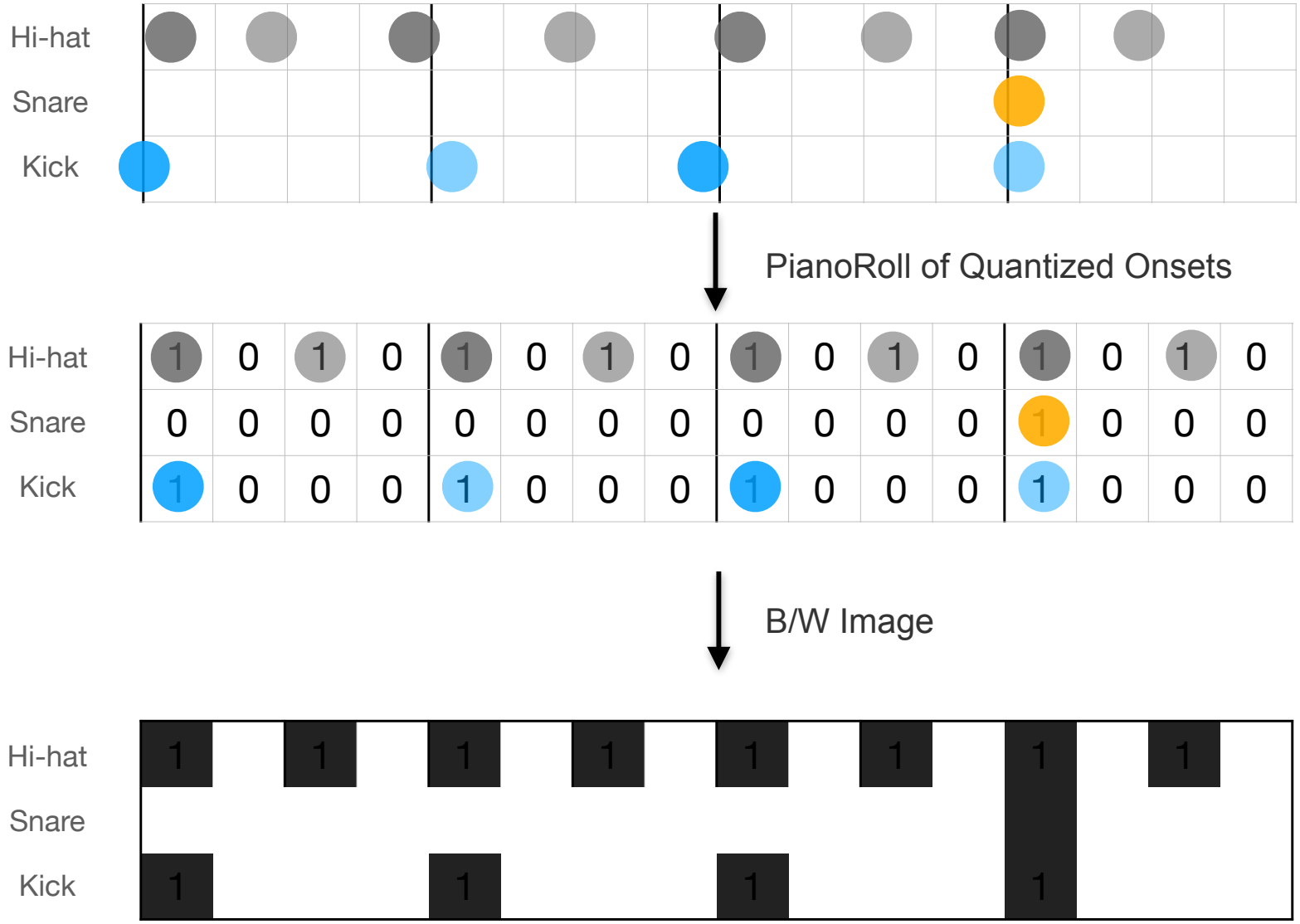
The model keeps updating the embedding vectors during the training process so as to come up with optimal representations

Data: Representation As Words

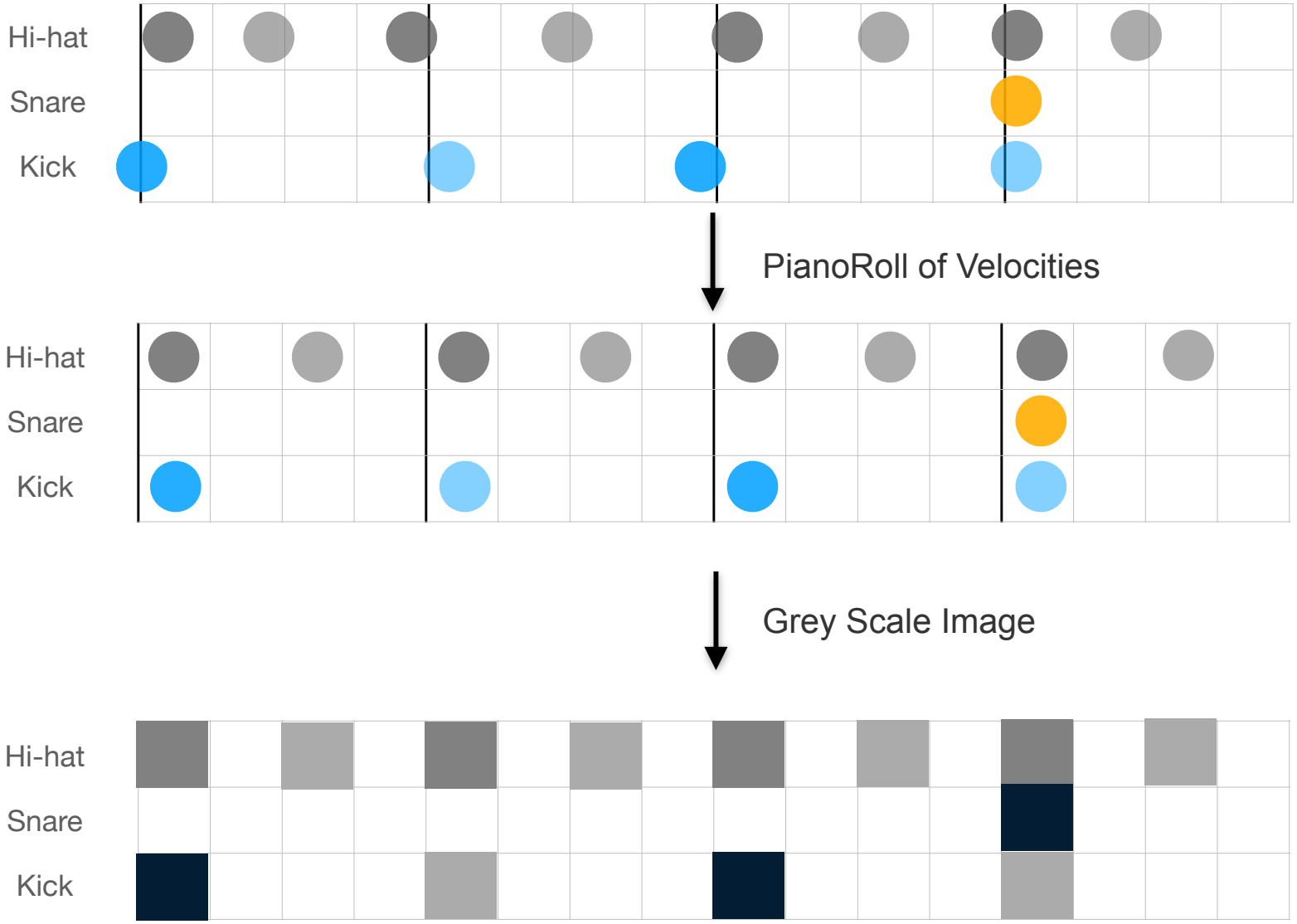
Many different tokenization techniques have been experimented with.

Check out https://github.com/Natooz/MidiTok/ for a review

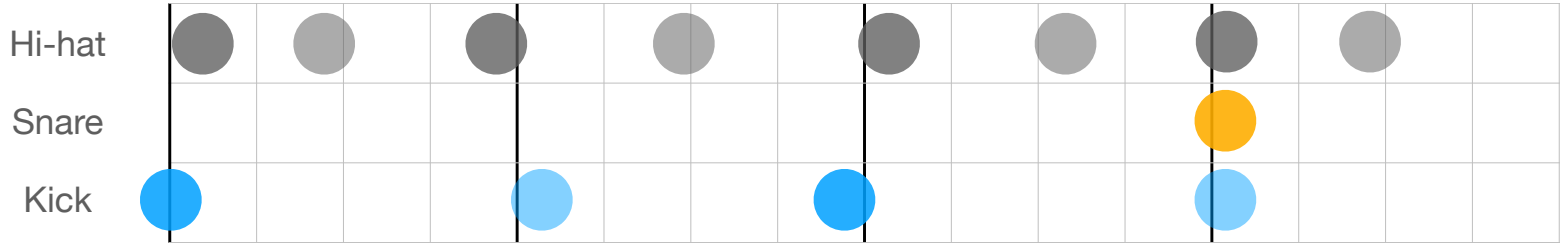
Data: Image-Like Representation



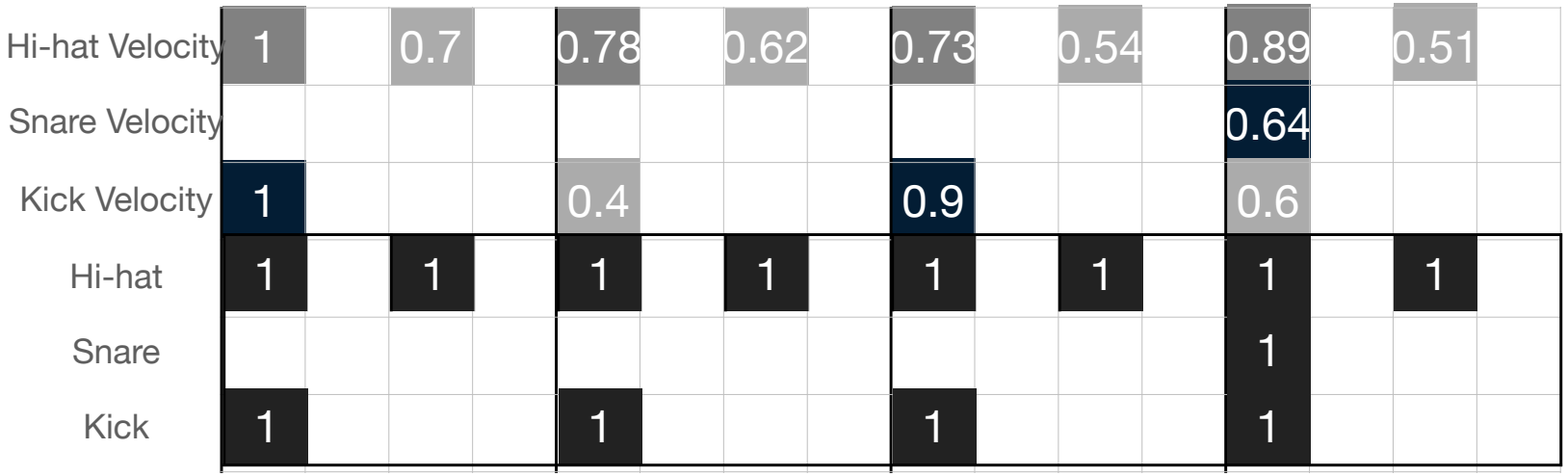
Data: Image-Like Representation



Data: Image-Like Representation



Combine Velocity and Onsets (Hits)

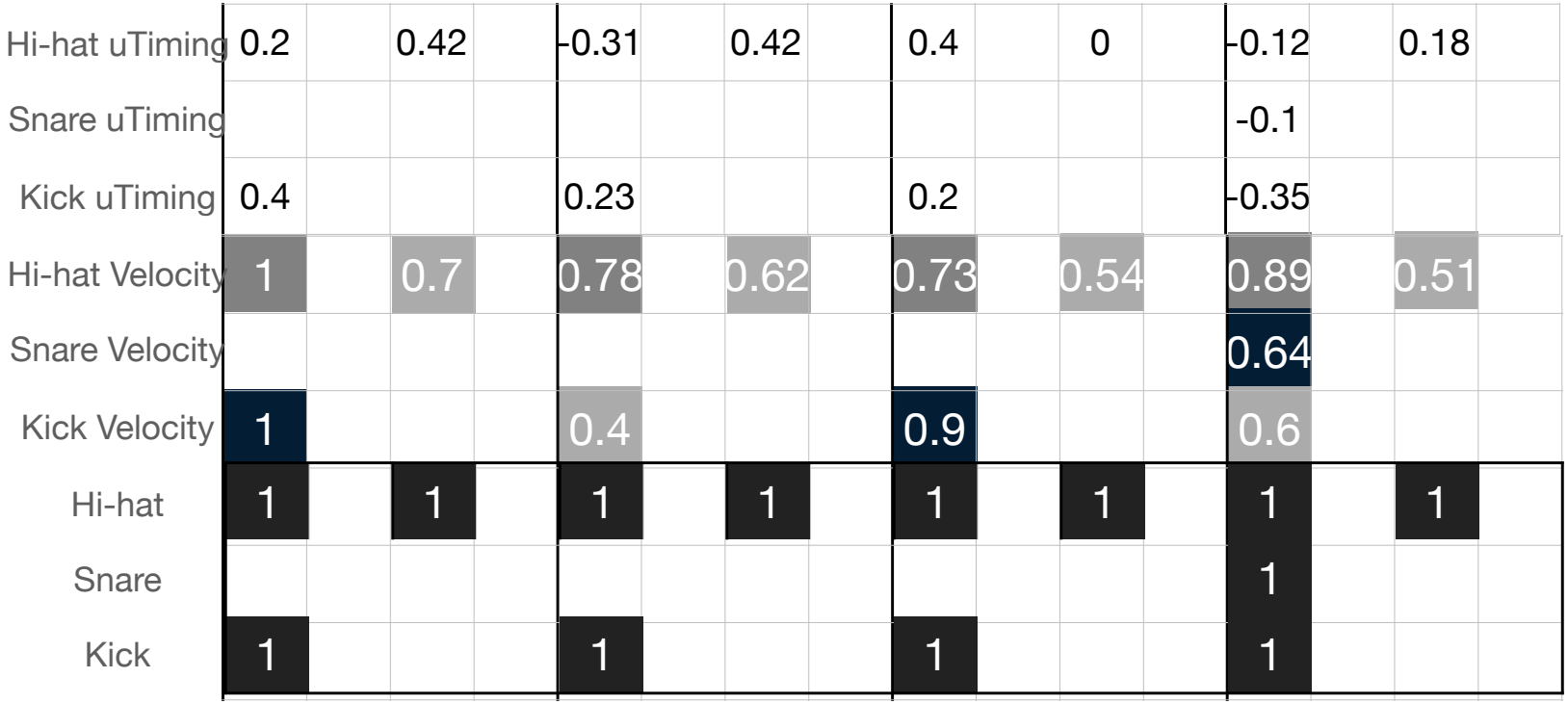


What about timing?

Data: Image-Like Representation

What about timing?

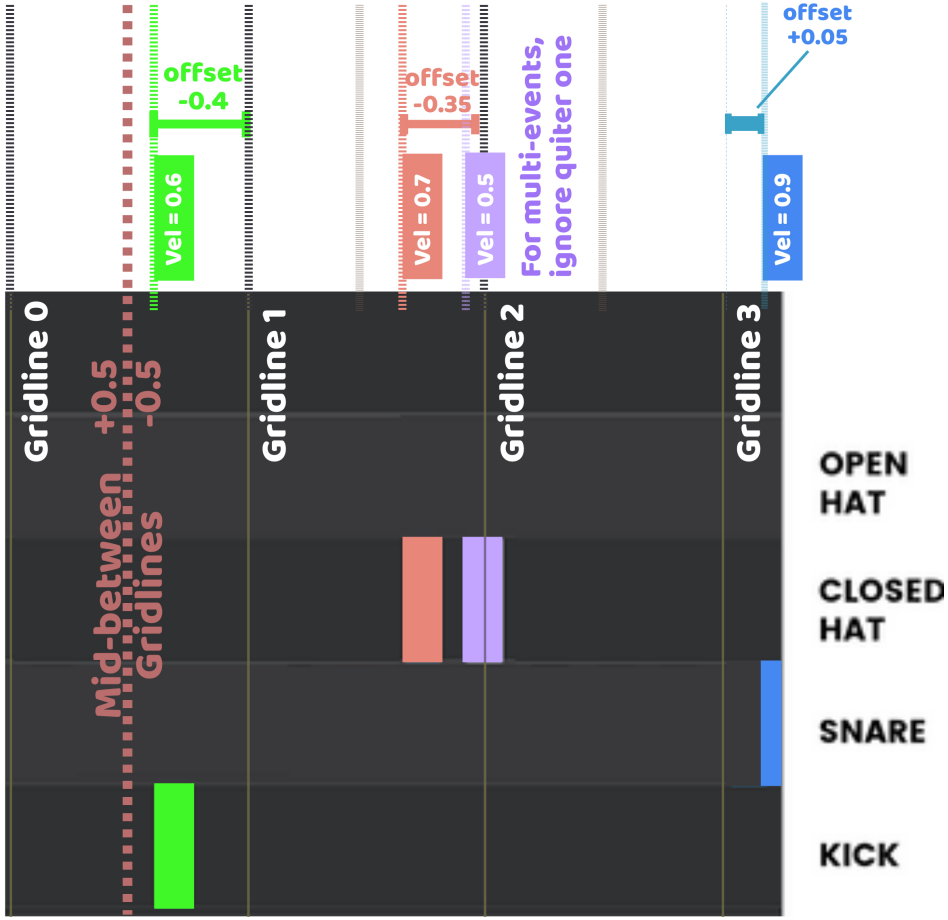
Add timing info, as a relative offset from the closest gridline



Data: We'll Represent Our Drum Patterns Using the Second Approach

	Gridline 0	Gridline 1	Gridline 2	Gridline 3
OH Offset	0		0	0
CH Offset	0	0	-0.35	0
Snare Offset	0	0	0	0.05
Kick Offset	0	-0.4	0	0
OH Velocity	0	0	0	0
CH Velocity	0	0	0.7	0
Snare Velocity	0	0	0	0.9
Kick Velocity	0	0.6	0	0
OH Hit	0	0	0	0
CH Hit	0	0	1	0
Snare Hit	0	0	0	1
Kick Hit	0	1	0	0

4 x 12



Architecture: Models for Sequences (RNN)

Overview of Discussions:

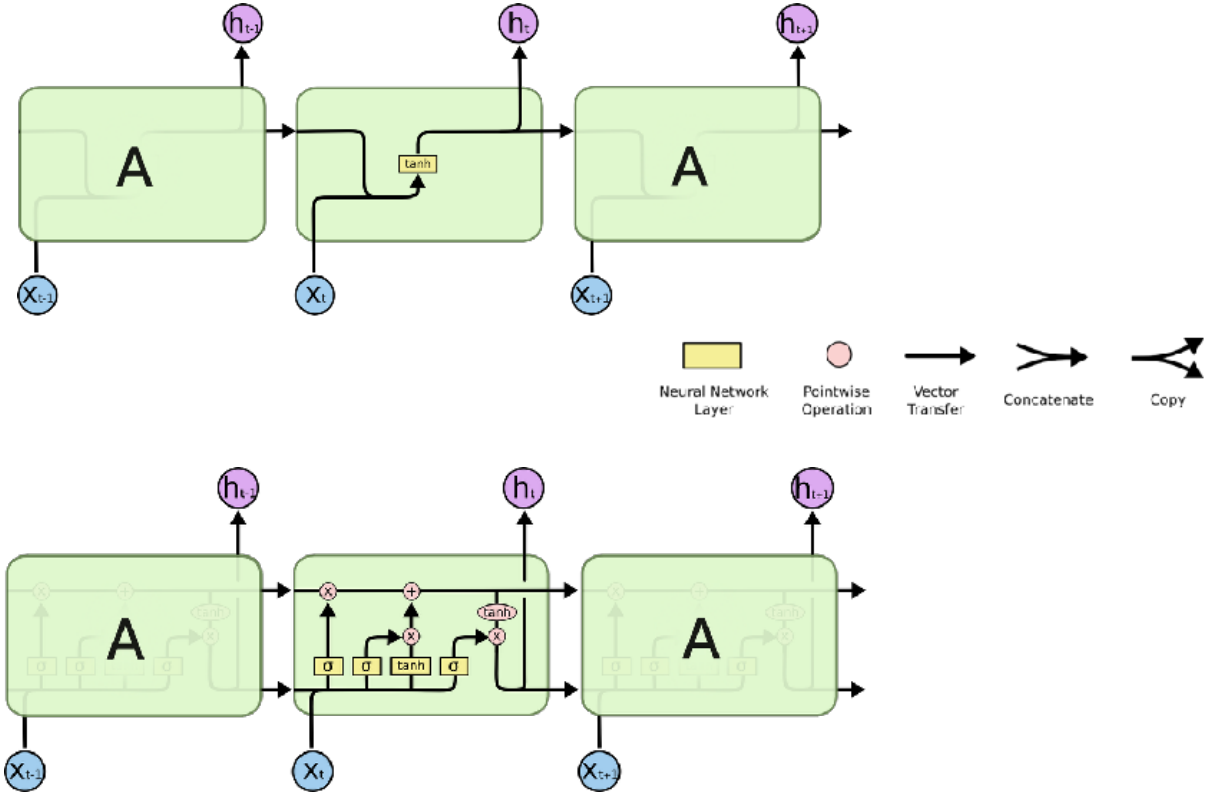
- **Models for sequential symbolic data**

- **Variational vs. Non-variational models**

Architecture: Models for Sequences (RNN)

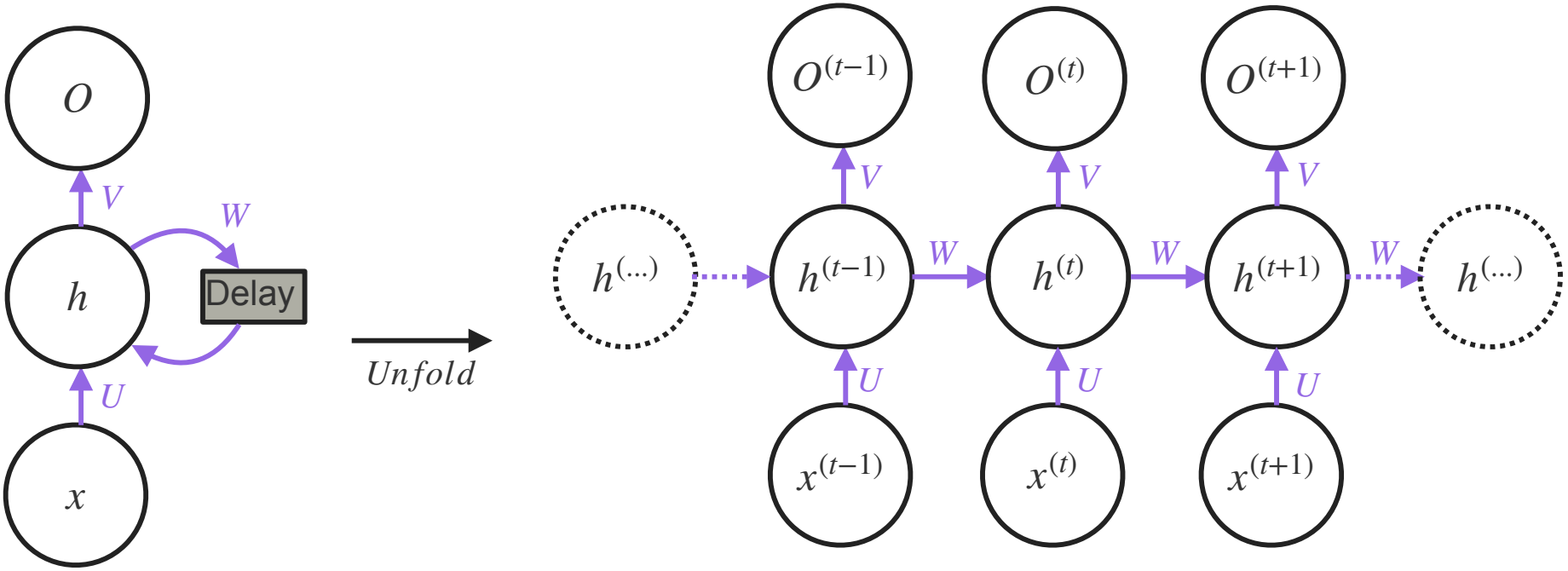
Two possible candidates:

1. RNNs

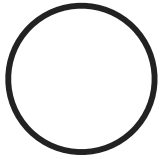


Figures from https://colah.github.io/posts/2015-08-Understanding-LSTMs/, for a detailed explanation refer to this source

Architecture: Models for Sequences (RNN)



$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}x^{(t)}$$
$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{a}^{(t)})$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$



Change Over Time

Constant Over Time

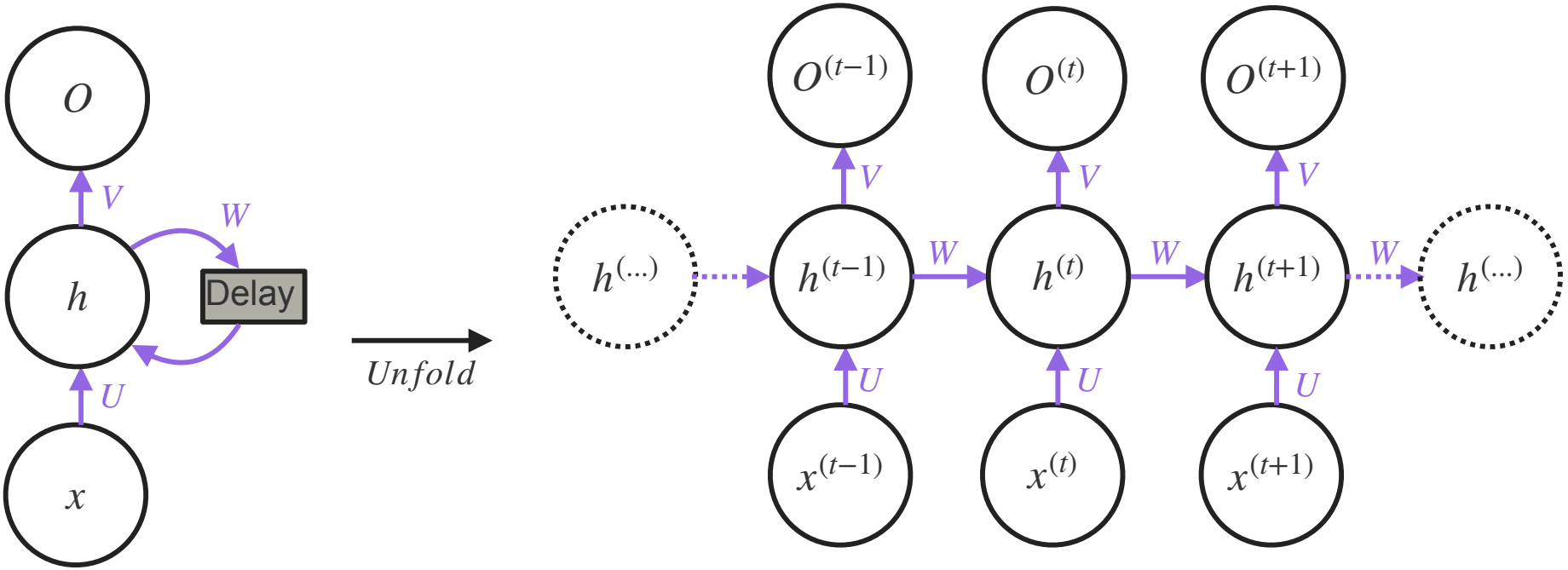
Architecture: Models for Sequences (RNN)

Pros:

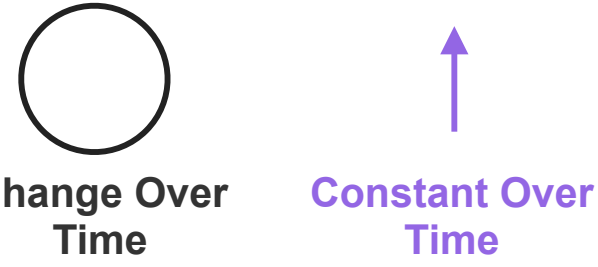
1. Can model inputs of arbitrary length
2. Regardless of length, model size is constant

Cons:

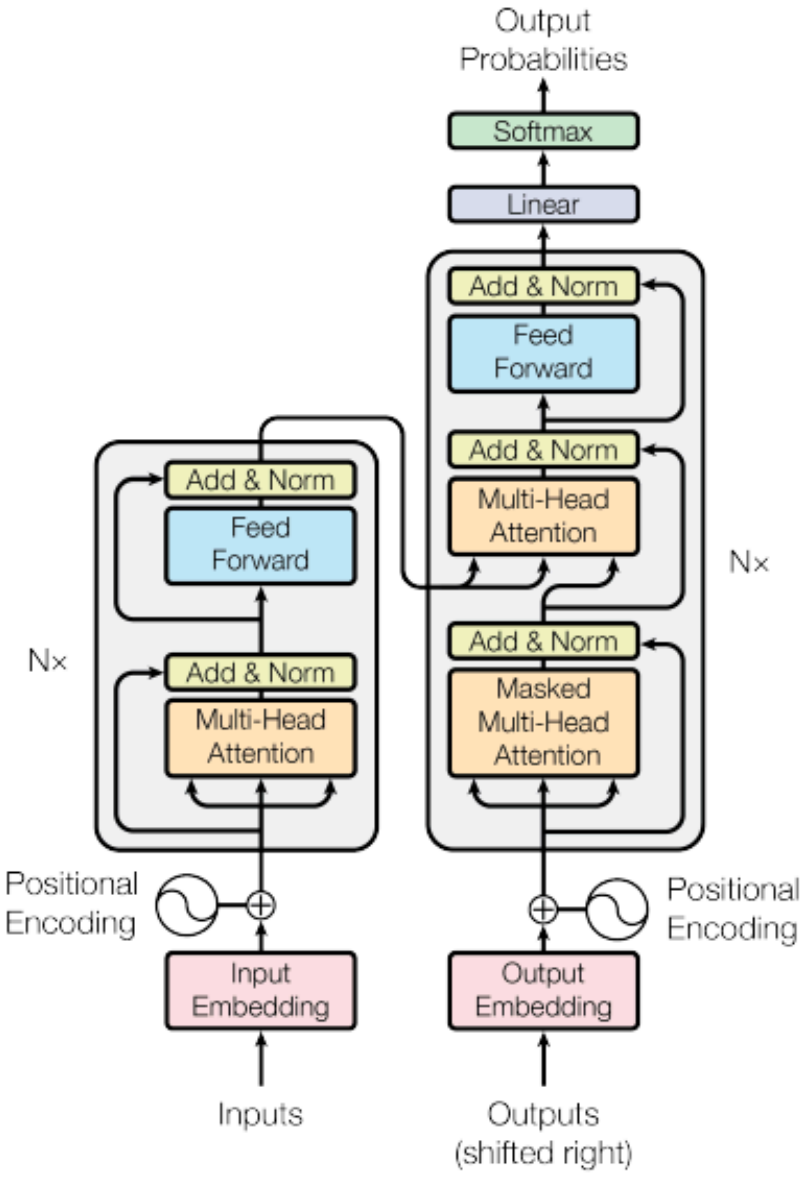
1. Recursive computation —> slow!
2. Exploding or vanishing gradient



$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = tanh(a^{(t)})$$
$$o^{(t)} = c + Vh^{(t)}$$



Architecture: Models for Sequences (Transformers)



Architecture: Models for Sequences (Transformers)

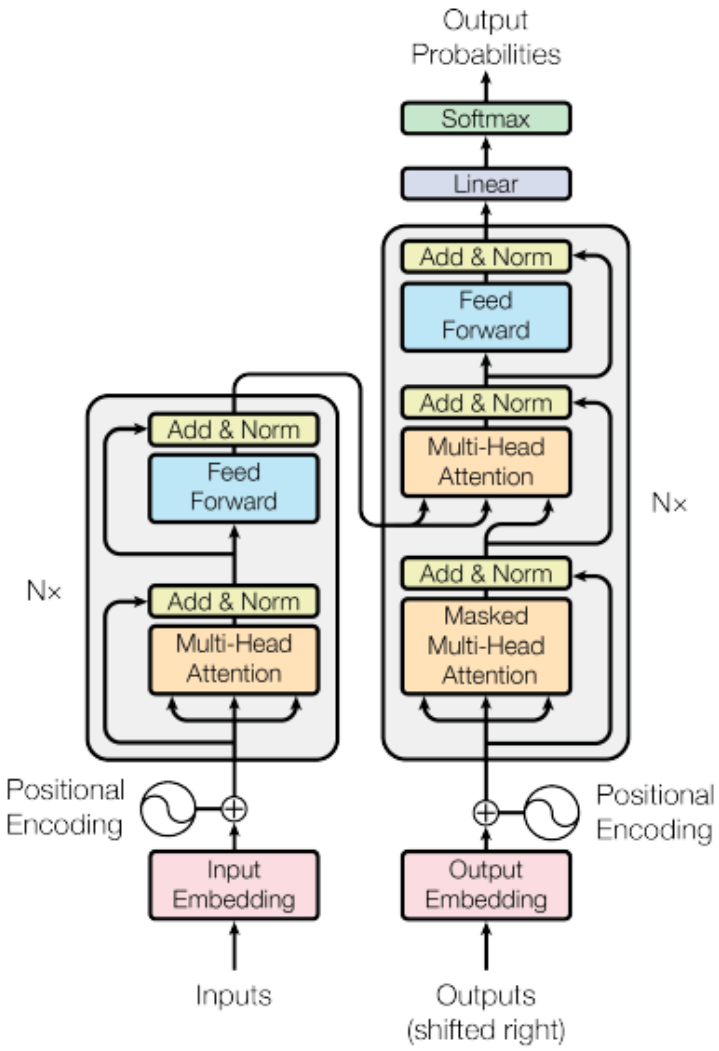
Two possible candidates:
1. Transformers

Pros:

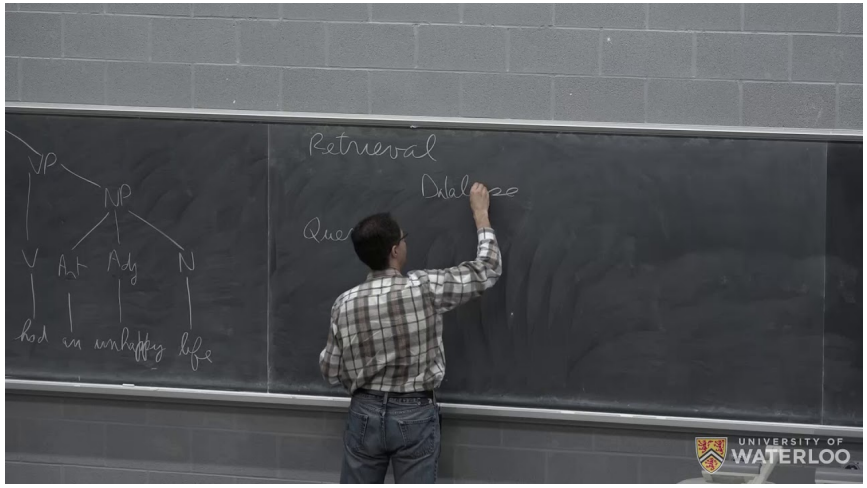
1. **Can use inputs of arbitrary length**
2. **They don't memorize the past events into a single vector! —> can deal with much longer sequences**
3. **Single forward path during training (and inference in case of variations such as BERT)**

Cons:

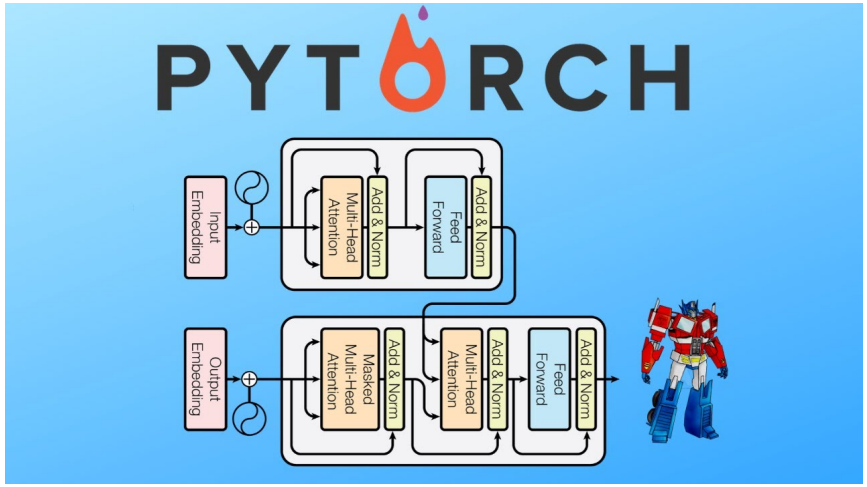
1. **Costly to train**
2. **Longer sequences need larger models**
3. **Require a lot of memory**



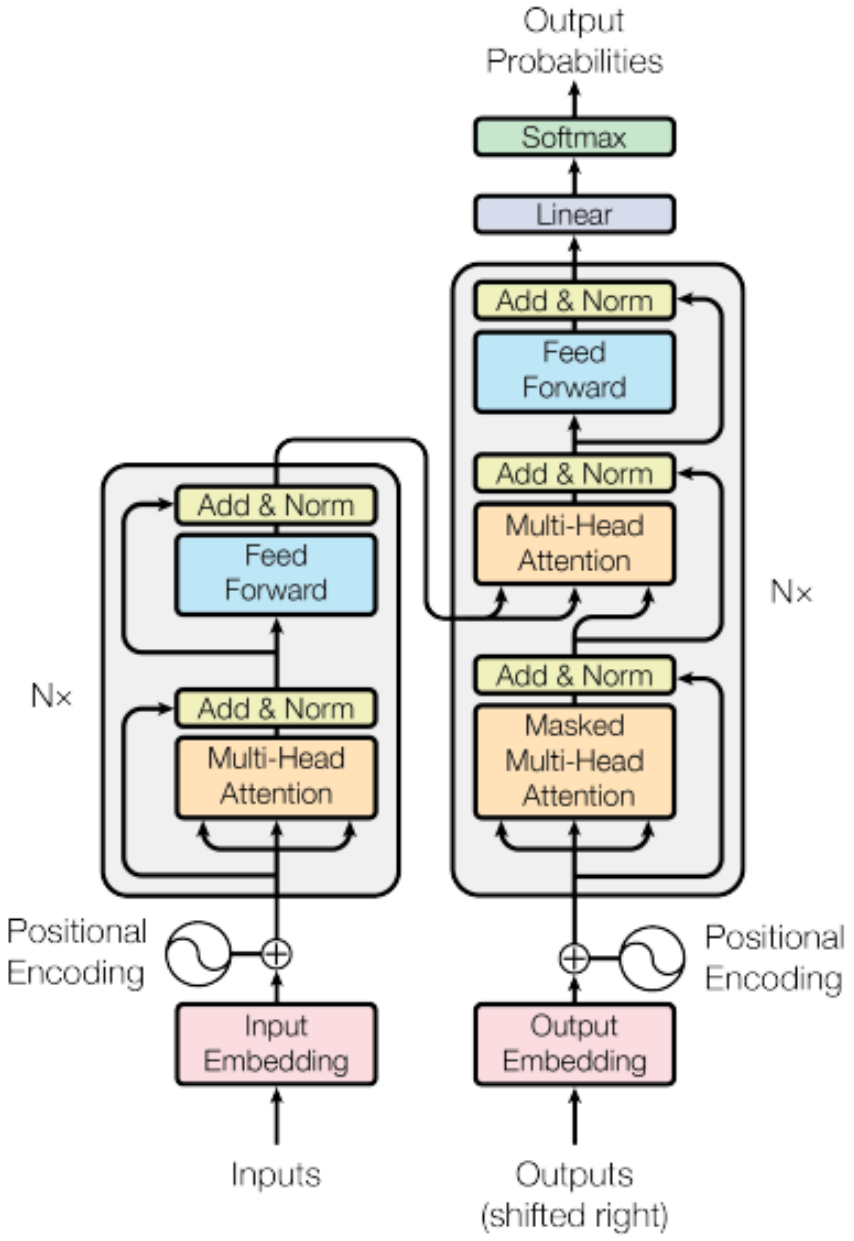
Architecture: Transformers



https://youtu.be/OyFJWRnt_AY

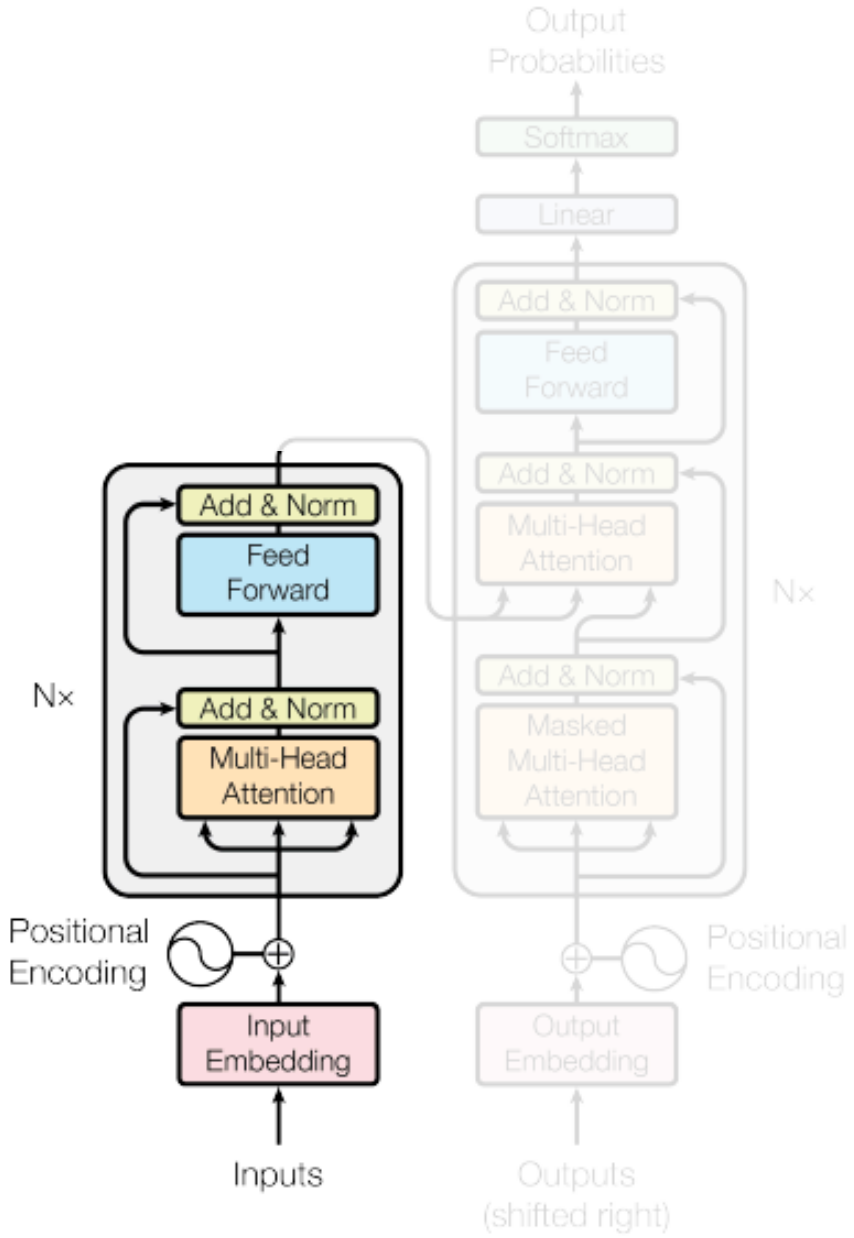


https://youtu.be/U0s0f995w14



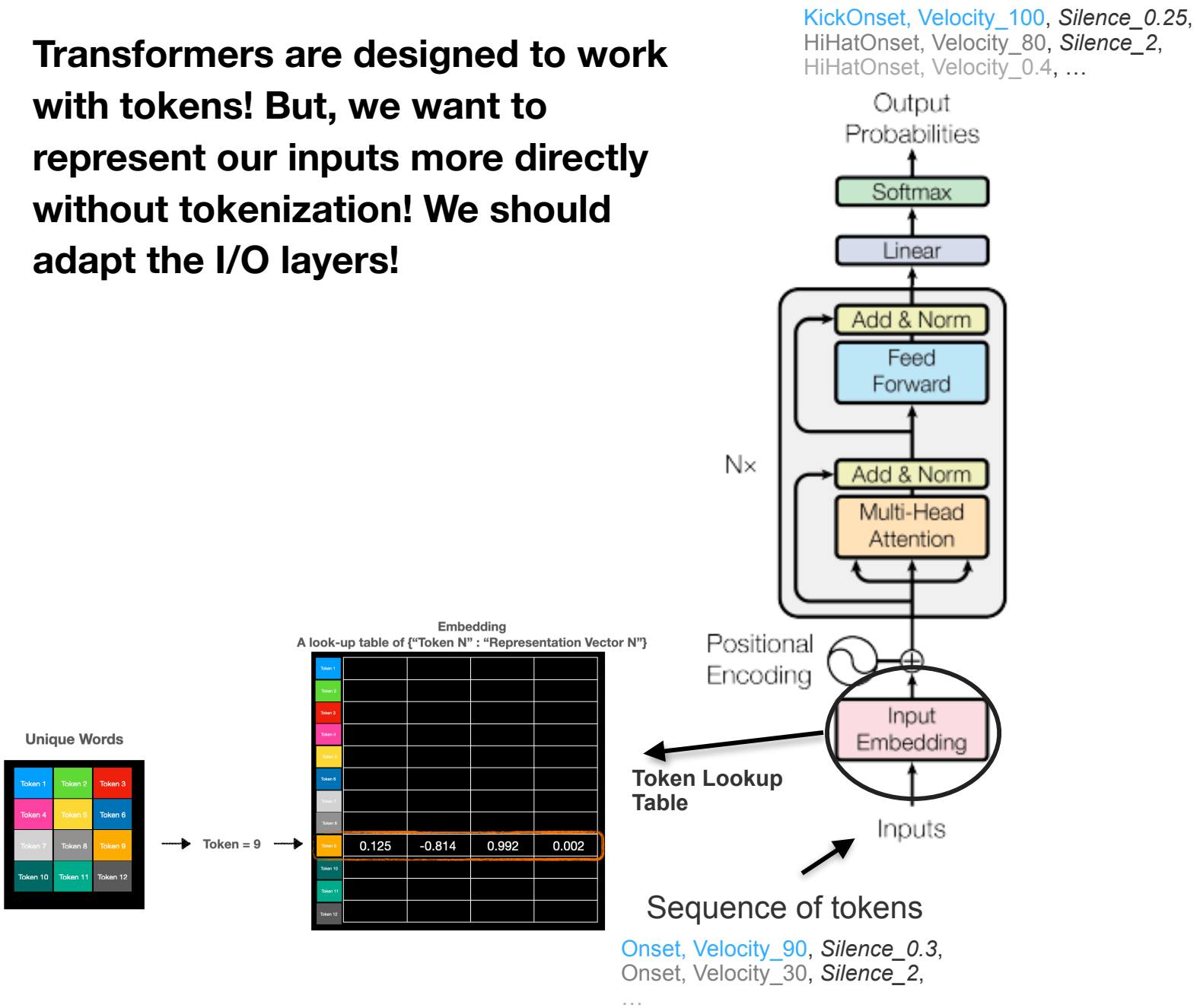
Architecture

We'll use the encoder section of transformer for our task



Architecture

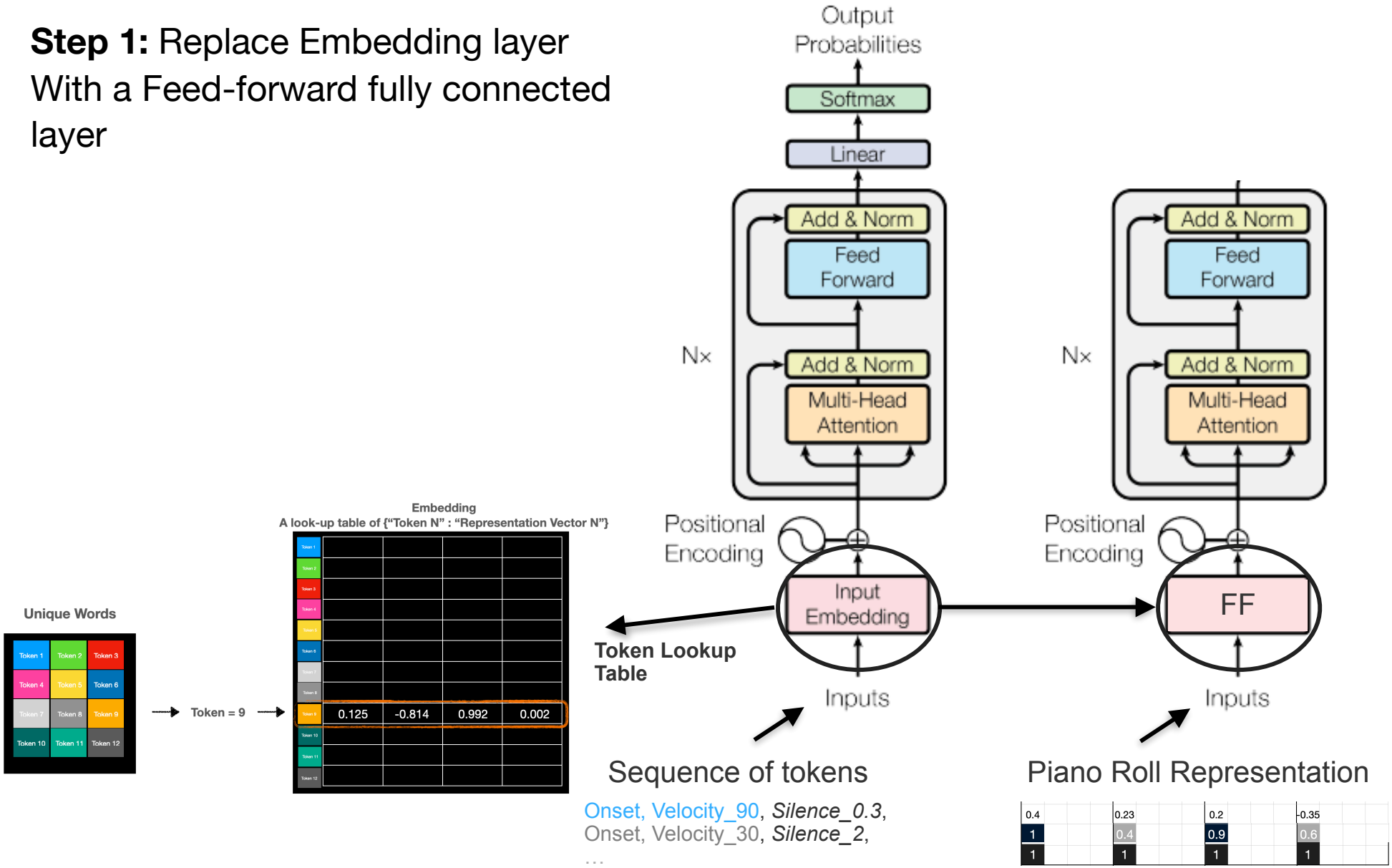
Transformers are designed to work with tokens! But, we want to represent our inputs more directly without tokenization! We should adapt the I/O layers!



Architecture:
Transformer for Piano Rolls Input/Outputs

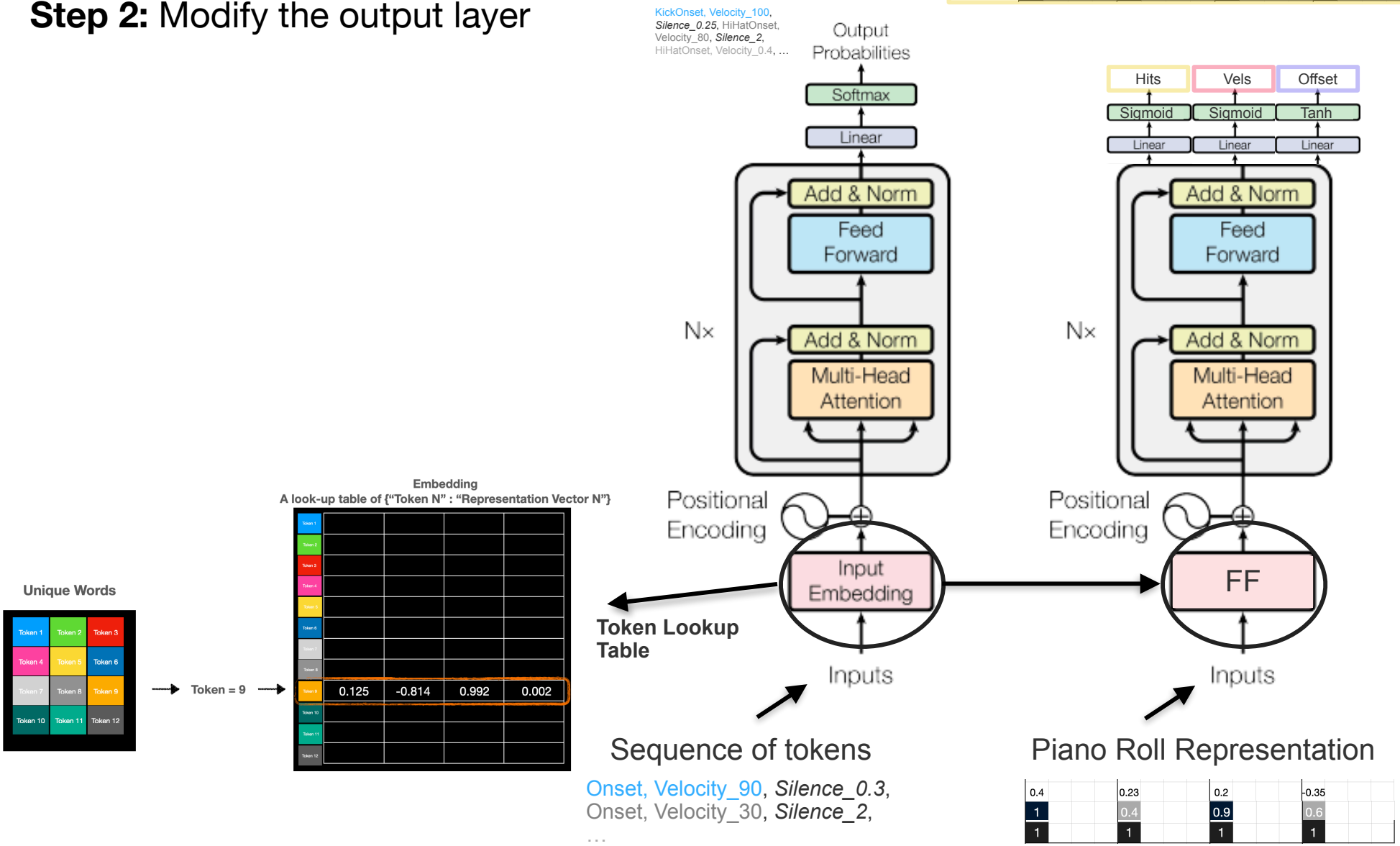
KickOnset, Velocity_100, *Silence_0.25*,
HiHatOnset, Velocity_80, *Silence_2*,
HiHatOnset, Velocity_0.4, …

Step 1: Replace Embedding layer
With a Feed-forward fully connected layer



Architecture:
Transformer for Piano Rolls Input/Outputs

Step 2: Modify the output layer



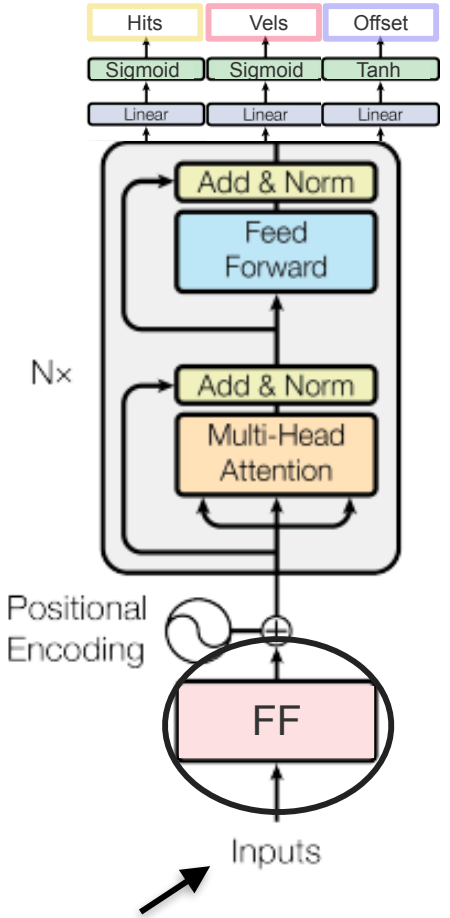
Architecture

A few limitations:

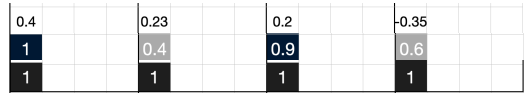
1. We haven't implemented any control parameters for the generations
2. We can't generate any variations for a given input! (For one input groove, we get a single drum pattern)
3. We can't generate from scratch using this model!

We'll try to make improvements to the architecture by modifying it into a Variational Auto-encoder (VAE)

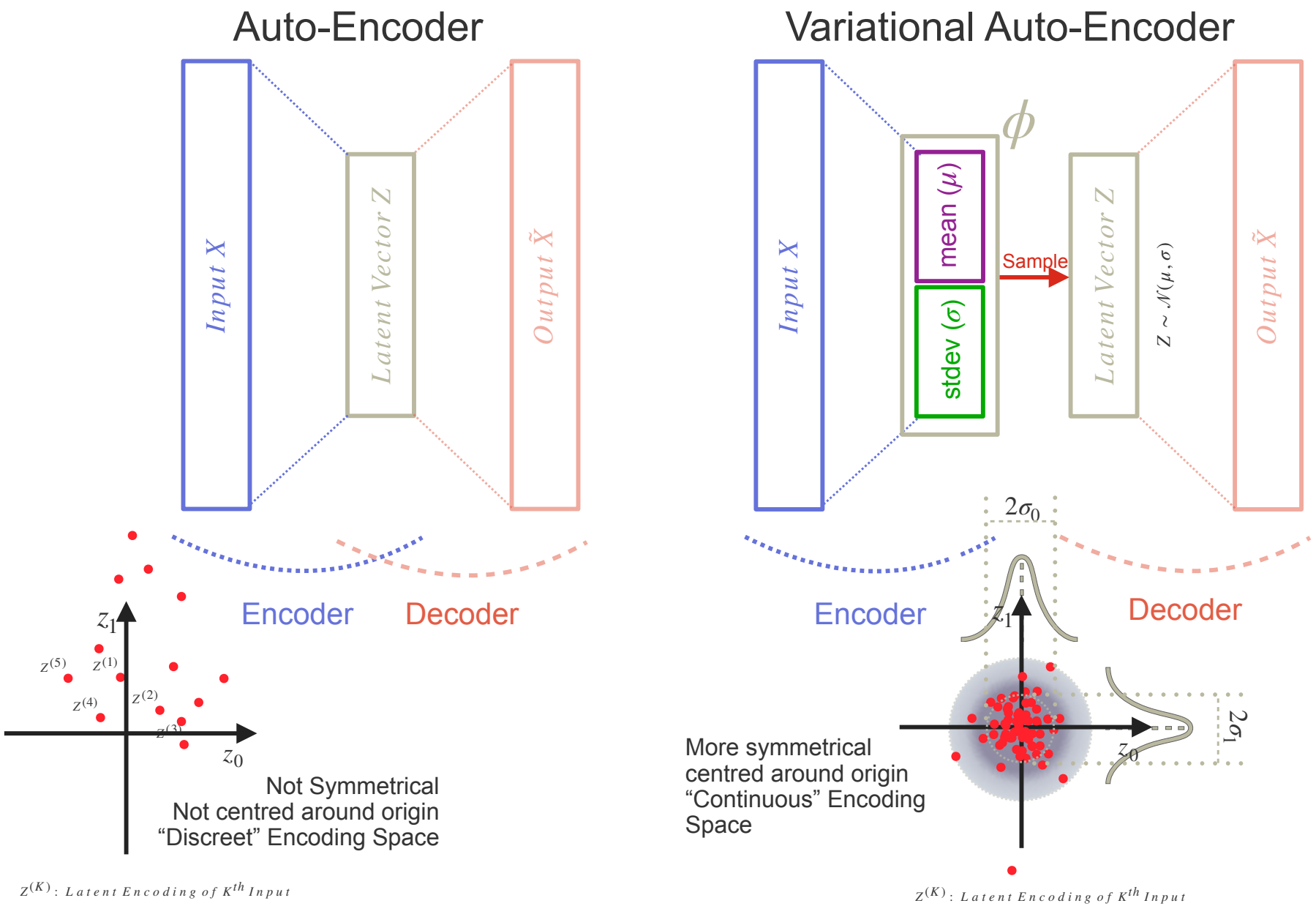
Drum Pattern



Input Monotonic Groove



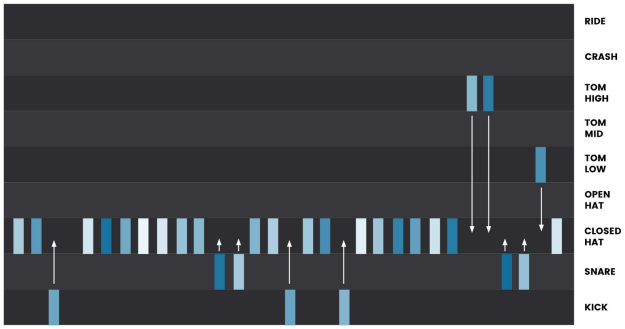
Architecture

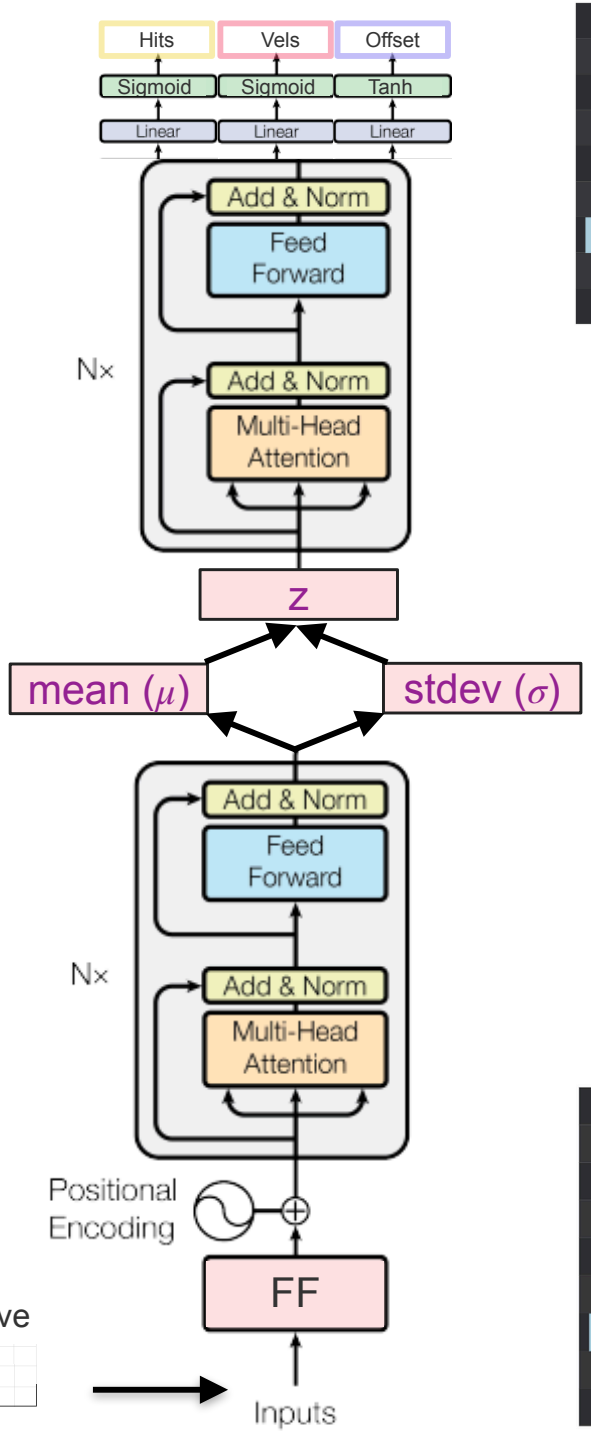


Read this if you're interested: https://towardsdatascience.com/generating-new-faces-with-variational-autoencoders-d13cfcb5f0a8

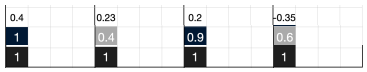
Architecture

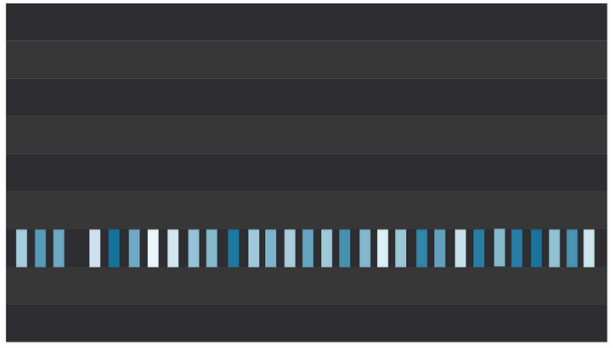
Drum Pattern





Input Monotonic Groove





Training

Two sets of Parameters:

1. Trainable:

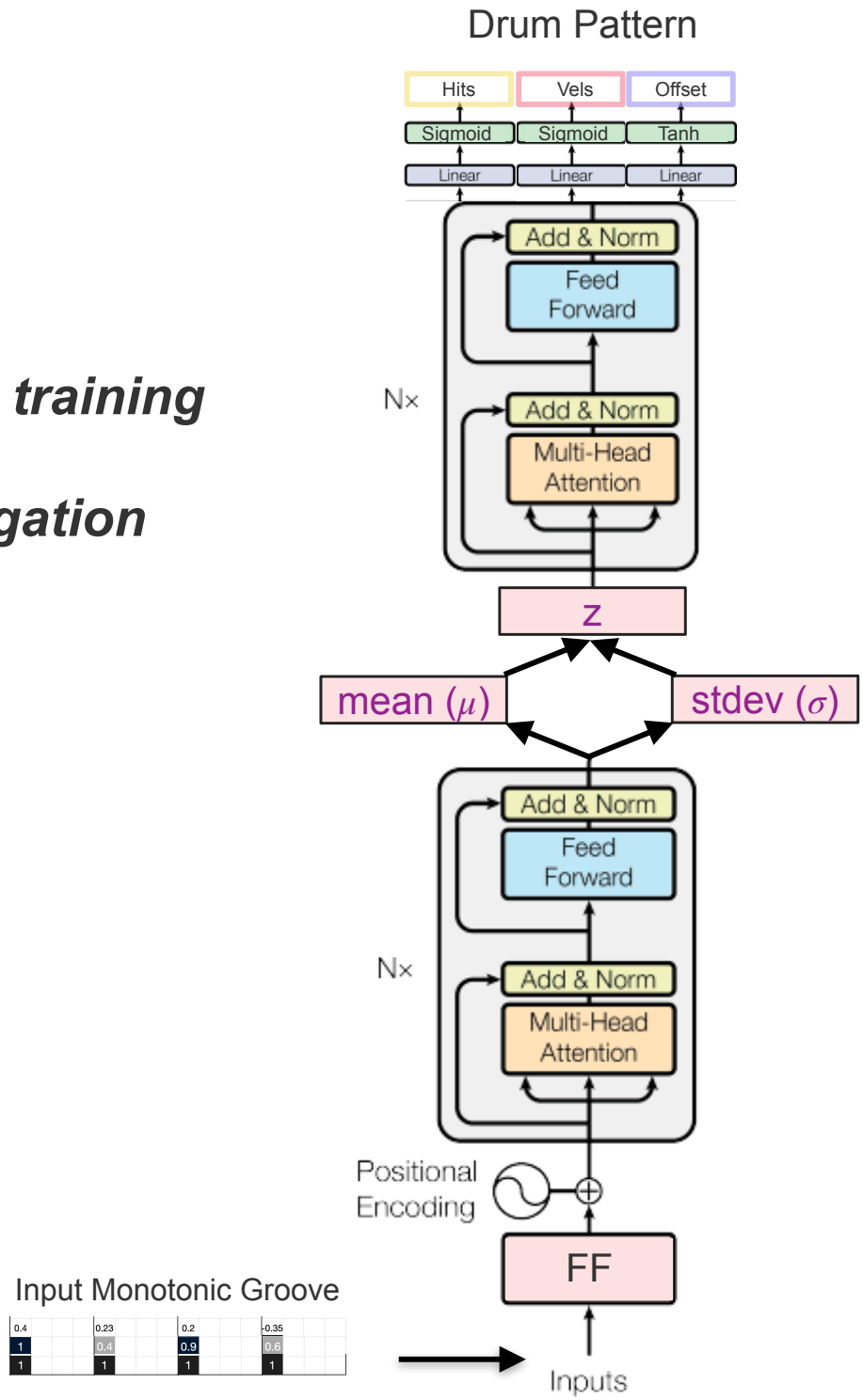
1. Weights of Network
2. Biases of Network

Learnt from training data using back-propagation

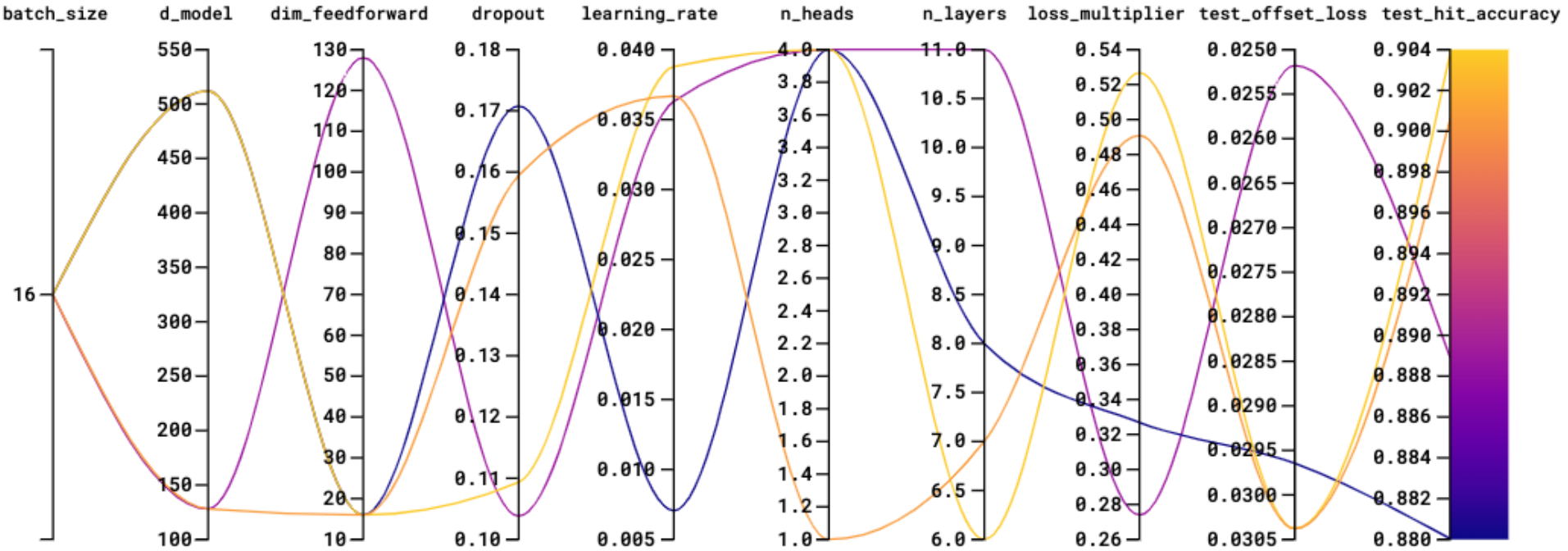
2. Hyper-Parameters (Non-trainable):

1. Number of Enc/Dec layers
2. FF dimensionality
3. Z dimensionality

…



Training



2. Hyper-Parameters (Non-trainable):

1. Number of Enc/Dec layers
2. FF dimensionality
3. Z dimensionality

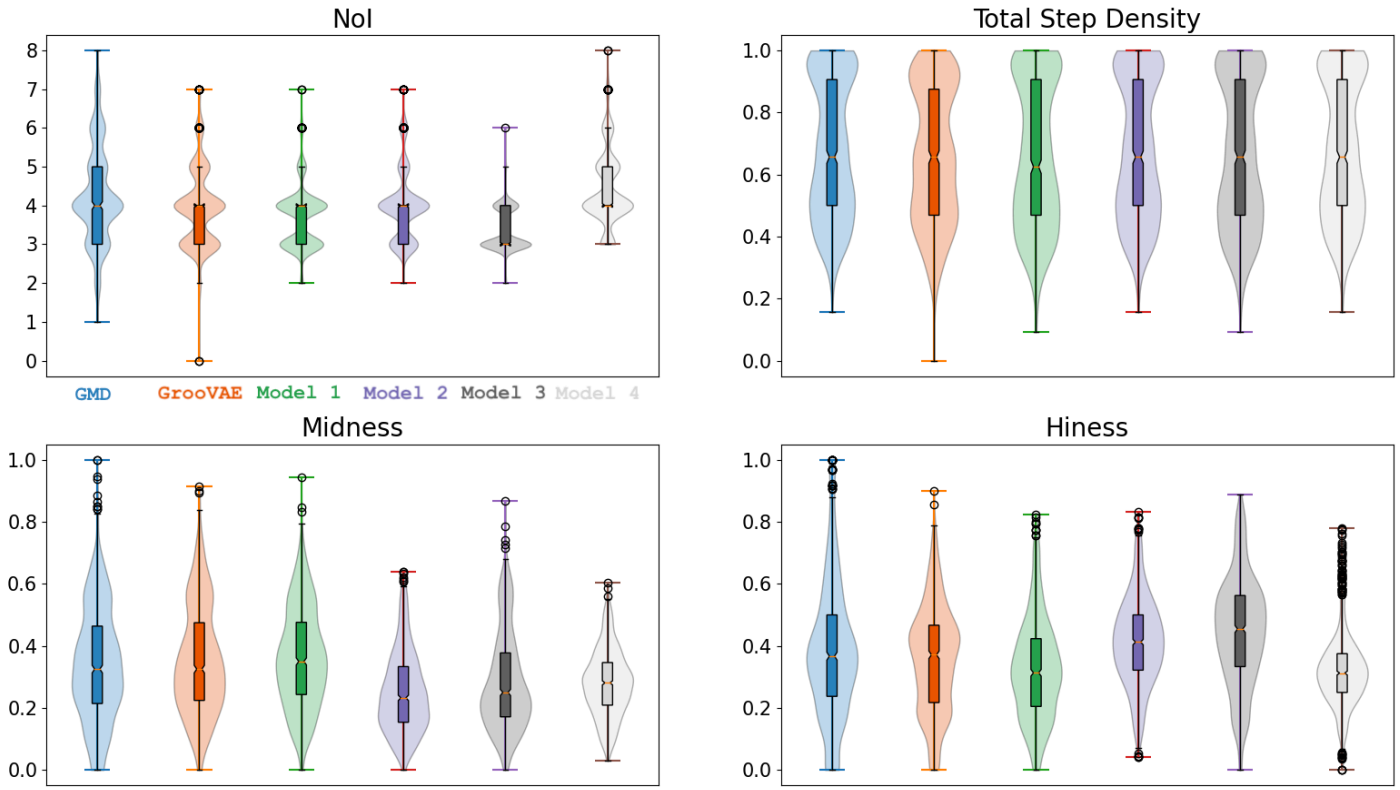
…

Tuned by Trial and Error
(AKA Hyper-parameter Tuning)

Read more:
https://docs.wandb.ai/guides/sweeps

Validation by Global Comparison (Feature-based)

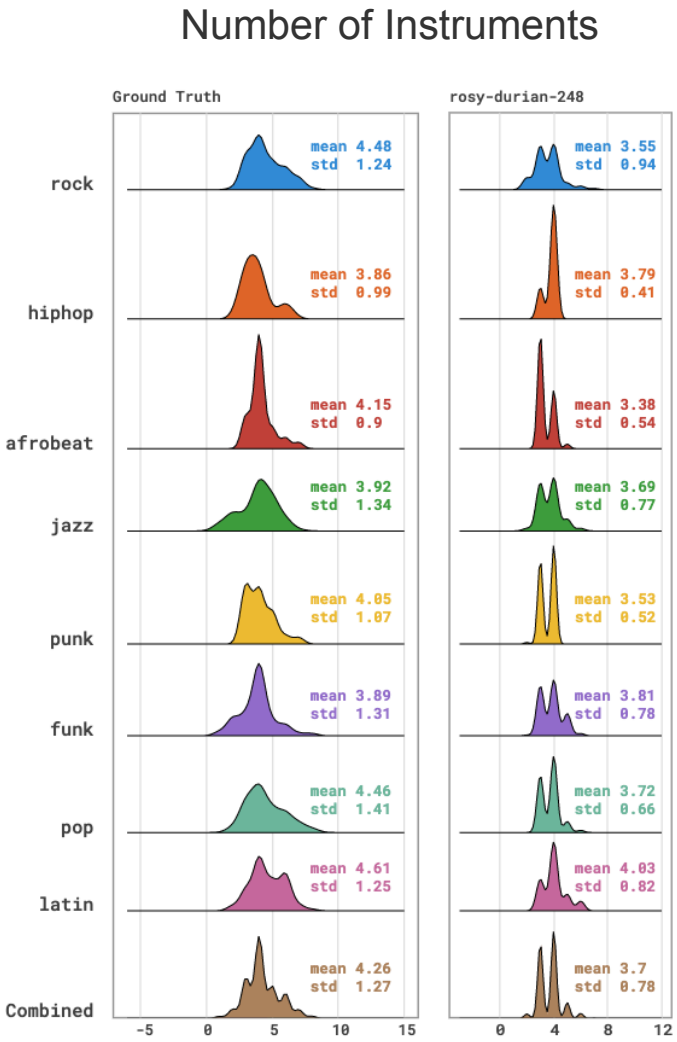
Extract musical features and compare models against one another by comparing the global distribution of features



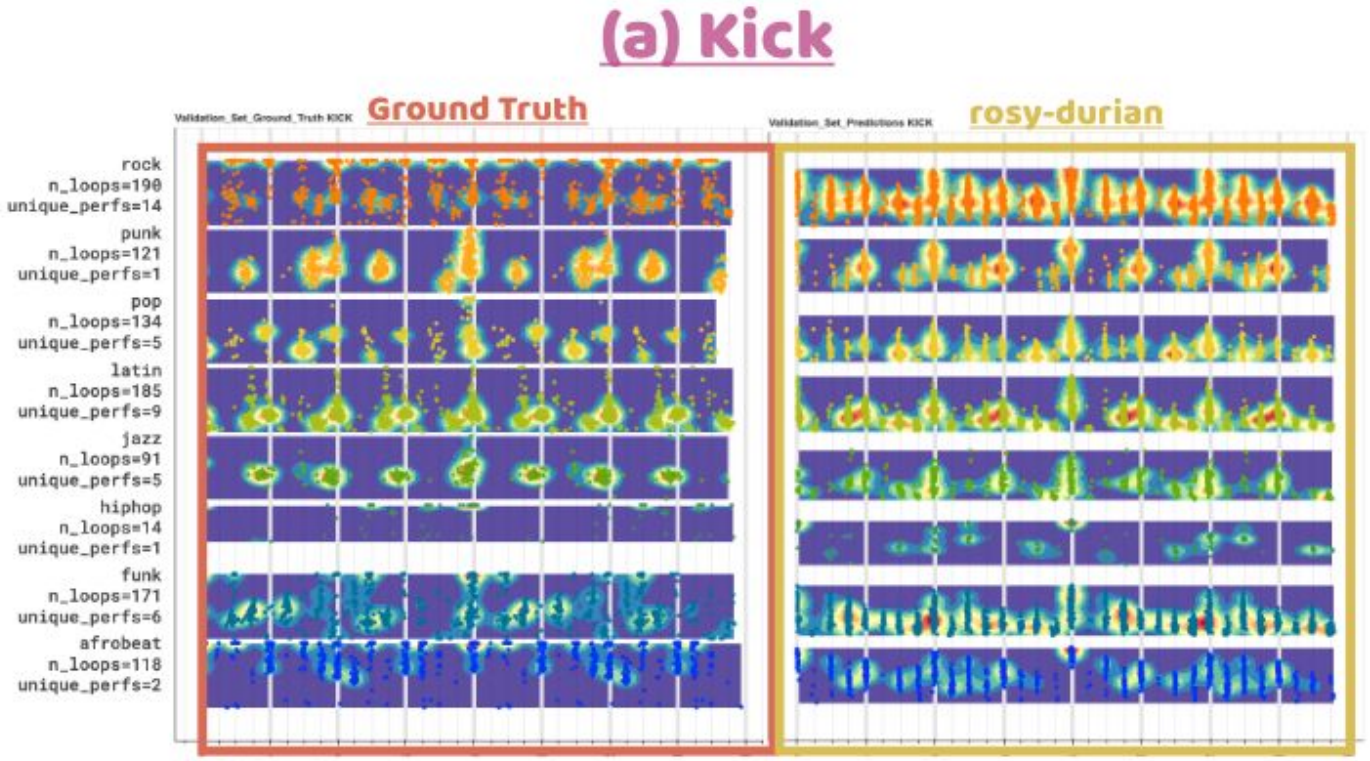
See more here:
https://wandb.ai/anonmmi/AIMC2022/reports/Absolute-Analysis--VmlldzoxOTU2OTc1

Validation by Global Comparison (Feature-based)

Now do the comparison per genre

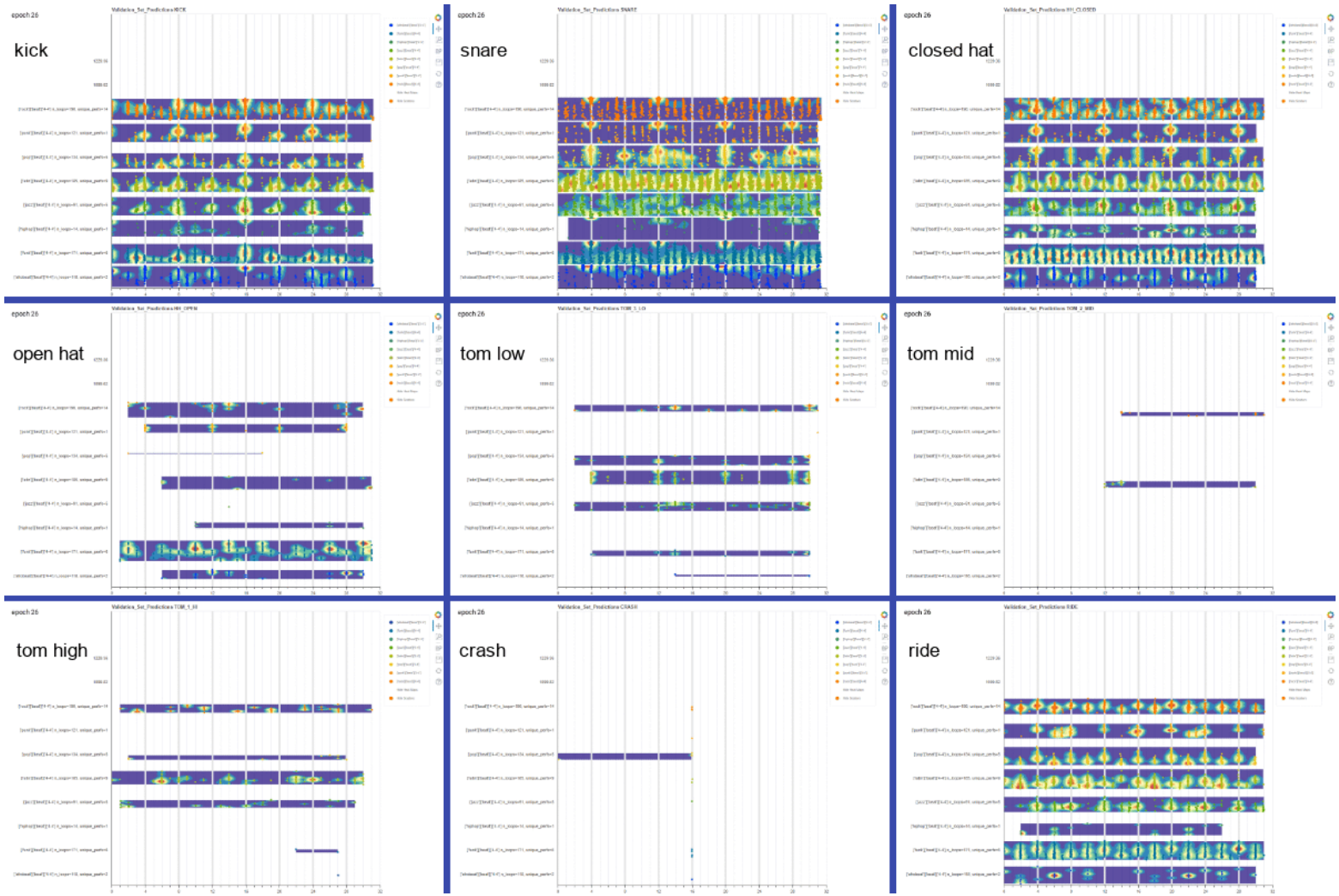


Validation by Global Comparison (Velocity Heat-maps)



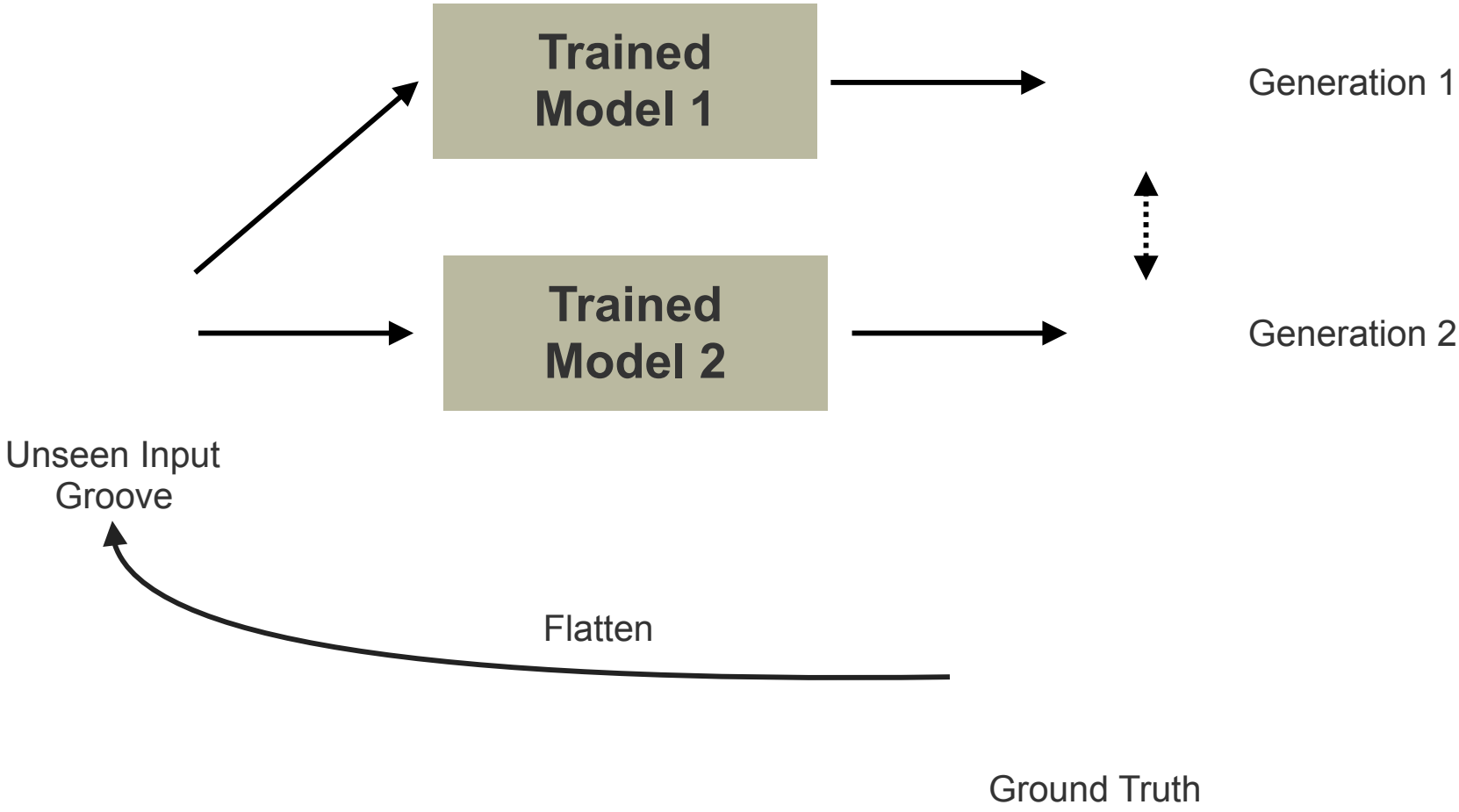
See more here:
https://wandb.ai/anonmmi/AIMC2022/reports/Velocity-heatmaps-for-Run-northern-sweep-26---VmlldzoxNTAxNjMy

Validation by Global Comparison (Velocity Heat-maps)



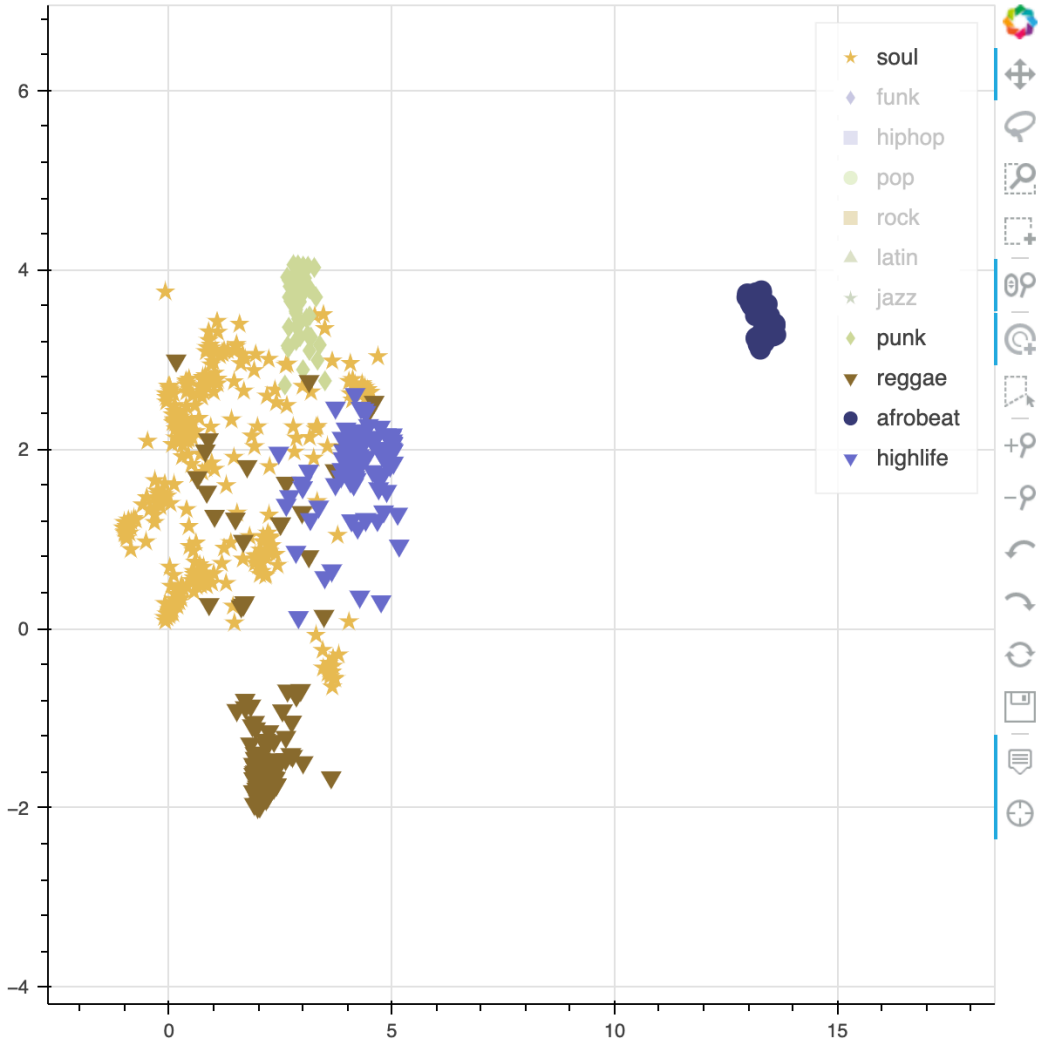
Validation by Listening (Subjective)

Use a held-out portion of data to evaluate the performance



Validation of Embeddings

Use dimension reduction techniques (such as UMAP, T-SNE) to visualize the clustering of embeddings



Check out in supplementary material:

UMAP_drawn_river_6.html
UMAP_noble-field-7.html

Additional Reading

For detailed discussions on topics discussed today:

https://behzadhaki.com/blog/2022/trainingGrooveTransformer/