Machine Learning Applications for Live Computer Music Performance

Ted Moore, Doctoral Fellow Music Composition
University of Chicago
Digital Media Workshop, Fall 2019

My Practice

- Composer (electronics + acoustic instruments)
- Improviser (electronics w/ acoustic collaborators)
- Coder (SuperCollider, Processing, openFrameworks, Python)
- Theatrical Sound Designer
- Aesthetics: noise, improvisation, glitch

Interest in Machine Learning

- Music Information Retrieval workshop at CCRMA summer 2018
- In what new ways can I approach sound?
- What can an algorithm do for (with) me? What can it tell me?
- Computational thinking
- What other routes are there to the same goal?

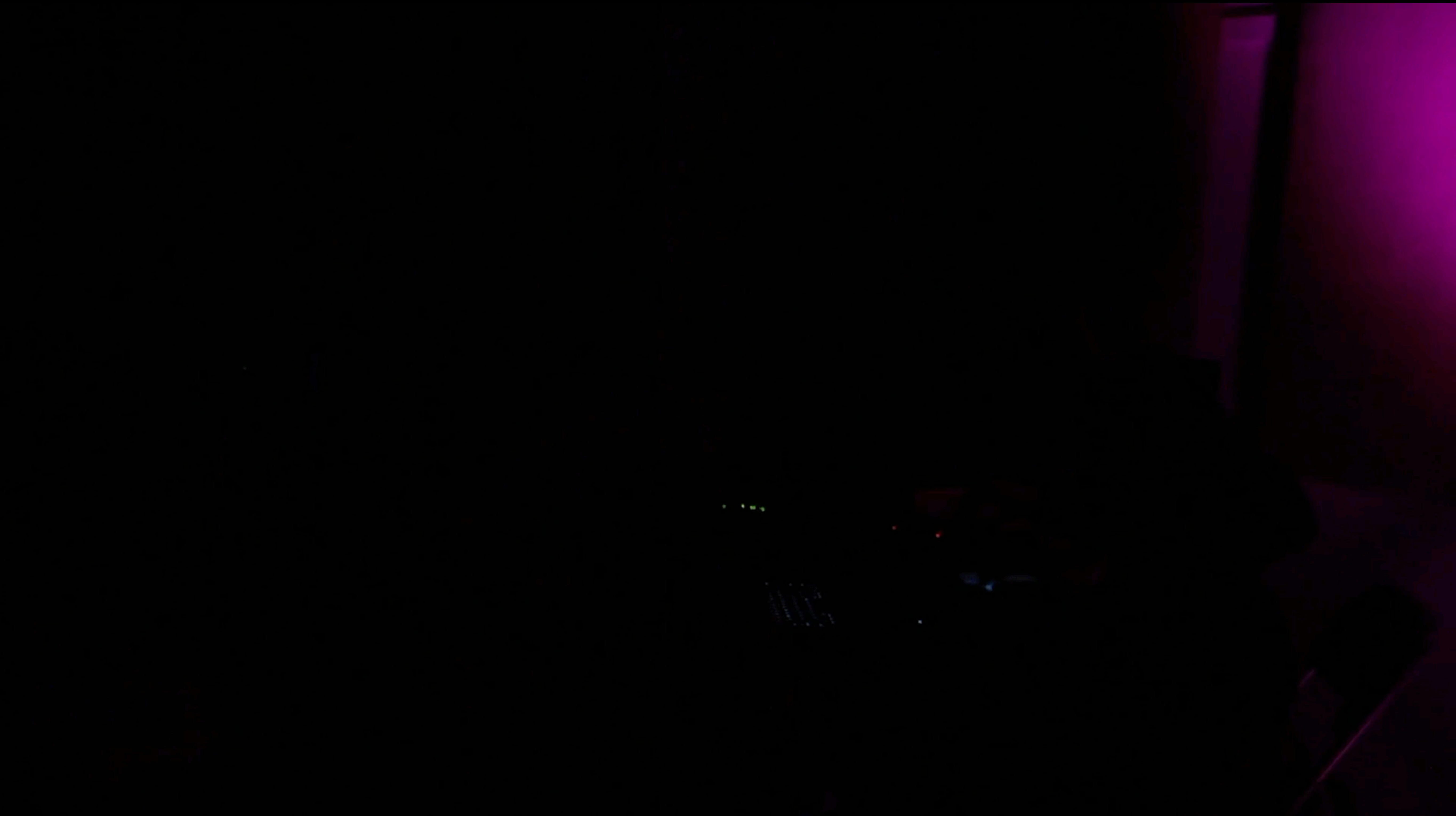
What is the goal?

Make sounds and art forms that I find artistically compelling.

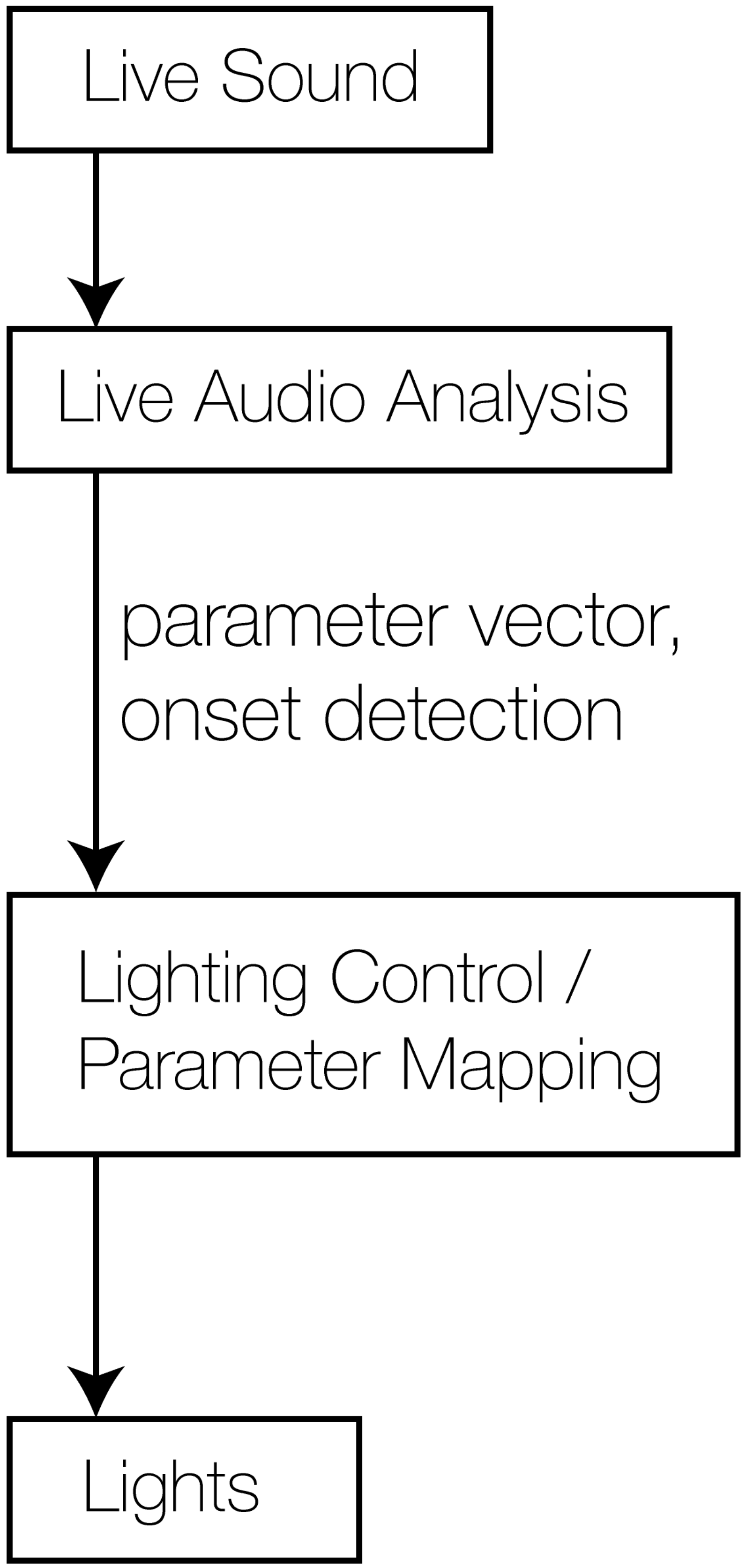
Today I share 3 examples of using these tools in that pursuit.

live computer music performance

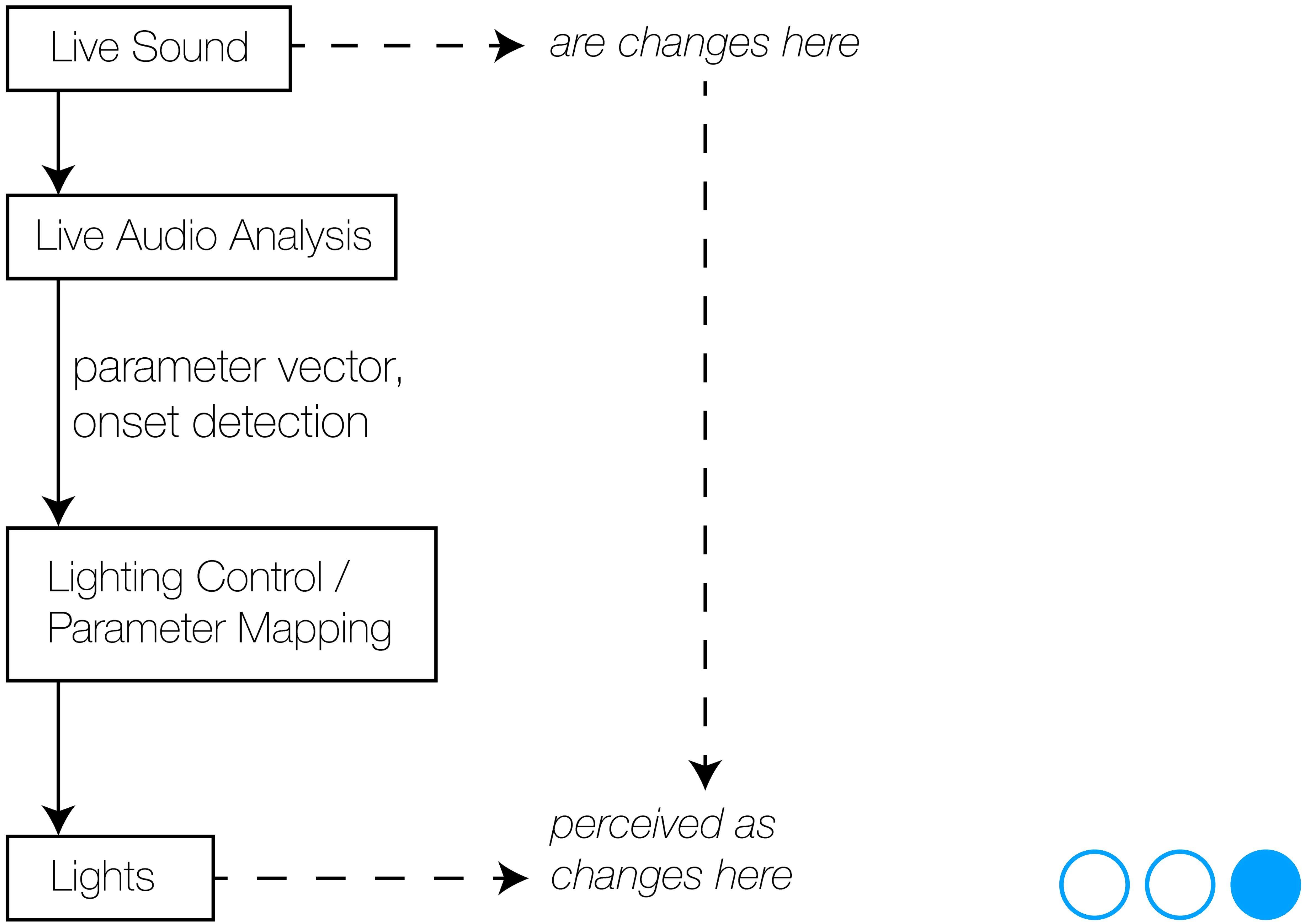
1. Live Sound Classification



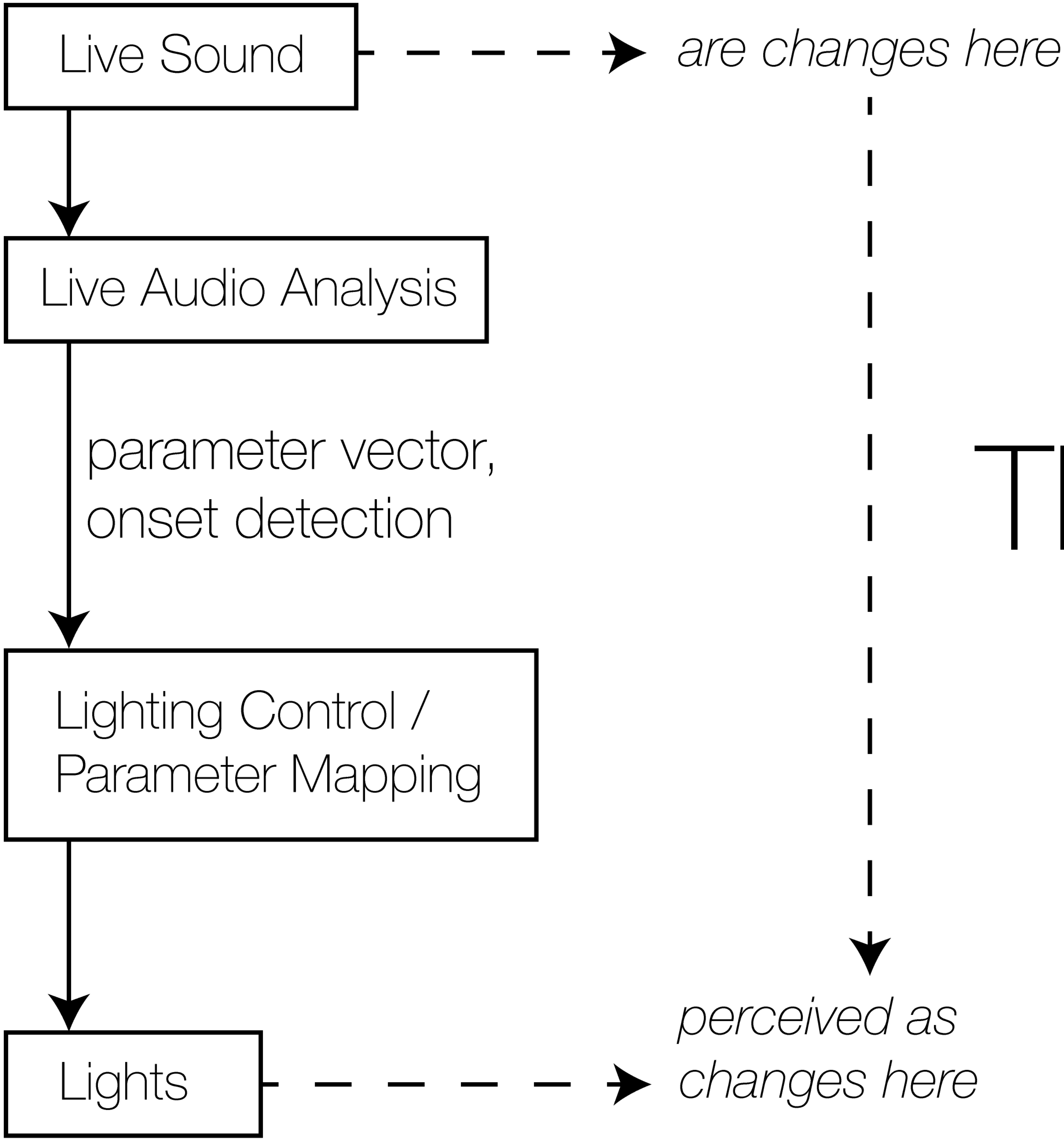
Machine ListeningSystem



Machine ListeningSystem

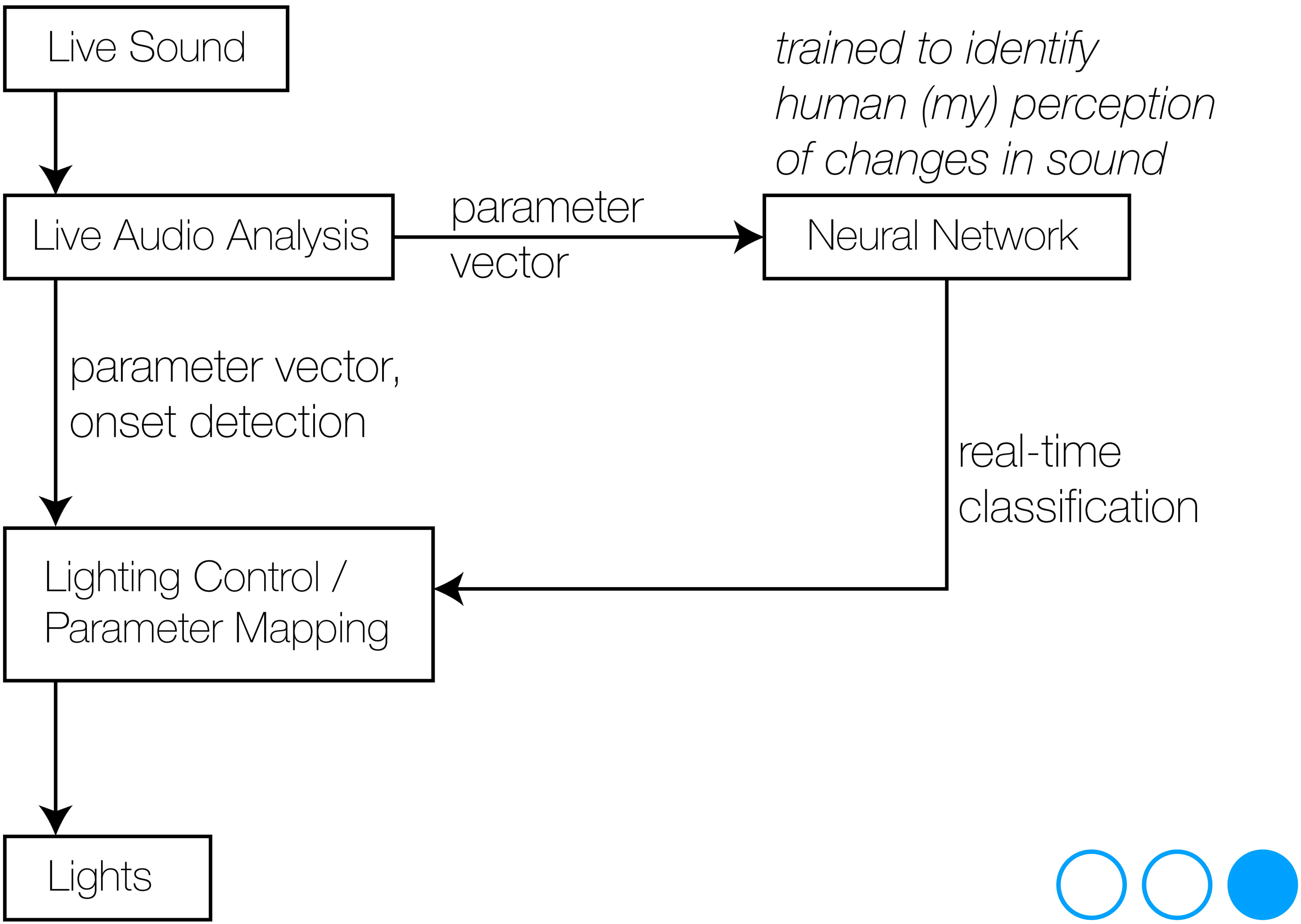


Machine ListeningSystem



The Goal

Machine Learning System



distorted_noise
high_squeal
low_impulses
sus_noise_quiet

```
NeuralNetwork {
    var <>net, <>learningRate, e = 2.71828, <shape, <>activation, <>normalizedRanges;

    *new {
        arg shape,learningRate = 0.05,activation = "relu",normalizedRanges;
        ^super.new.init(shape,learningRate,activation,normalizedRanges);
    }

    init {
        arg shape_,learningRate_ = 0.05,activation_ = "relu",normalizedRanges_;
        shape = shape_;
        activation = activation_;
        learningRate = learningRate_;
        normalizedRanges = normalizedRanges_;

        net = shape.collect({
            arg nNeurons, i;
            var data = (
                vals:Array.fill(nNeurons,{0}),
            );
            if(i > 0,{
                // not input layer;
                data.biases = Array.fill(nNeurons,{rrand(-1.0,1.0)});
                data.weights = Array.fill(shape[i],{
                    Array.fill(shape[i-1],{rrand(-1.0,1.0)});
                });
            });
            data;
        });
```

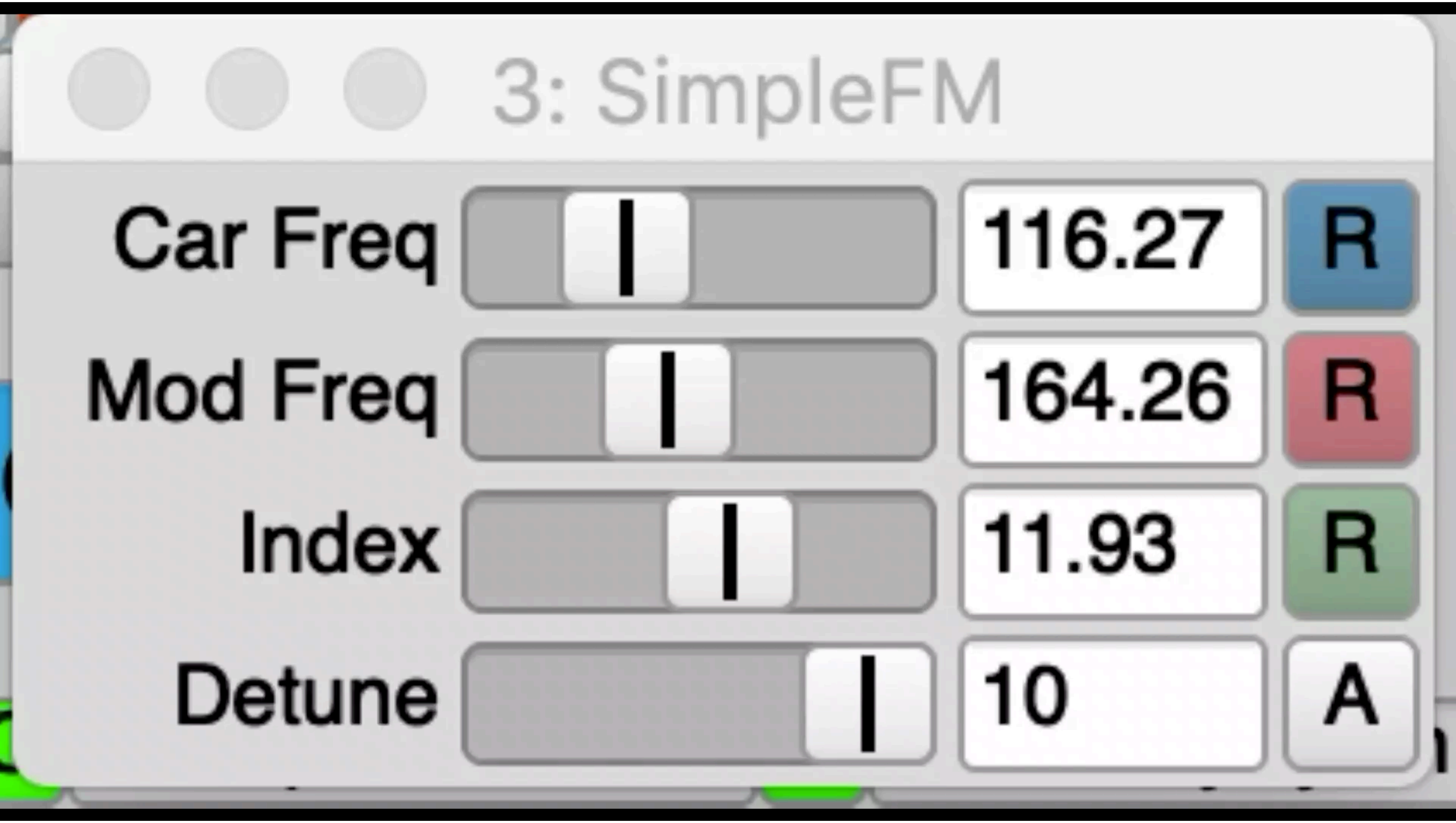
```
feedLights = FeedLightMaster([

    // distorted noise
    FeedLightMode(nLights,[
        FeedLightGroup([
            \amplitude,\myAmp,\v,ControlSpec(0.01,1,\exp),
            \specCentroid,ControlSpec(50,5000,\exp),\h,ControlSpec(0.5,0.7),
            \specFlatness,nil.asSpec,\s,ControlSpec(1,0.3)
        ]),
        FeedLightGroup([
            \amplitude,\myAmp,\v,ControlSpec(0.01,1,\exp),
            \specCentroid,ControlSpec(50,5000,\exp),\h,ControlSpec(0.4,0.6),
            \specFlatness,nil.asSpec,\s,ControlSpec(1,0.3)
        ])
    ]),

    // high squeal
    FeedLightMode(nLights,[
        FeedLightGroup([
            \amplitude,\myAmp,\s,ControlSpec(0.5,0.9),
            \zeroCrossing,ControlSpec(3000,10000,\exp),\h,ControlSpec(0,0.25),
            \constant,1,\v,nil
            //\specFlatness,nil.asSpec,\w,ControlSpec(0,255),
            //\zeroCrossing,ControlSpec(50,6000,\exp),\r,ControlSpec(0,255)
        ]),
        FeedLightGroup([
            \amplitude,\myAmp,\s,ControlSpec(0.5,0.9),
```

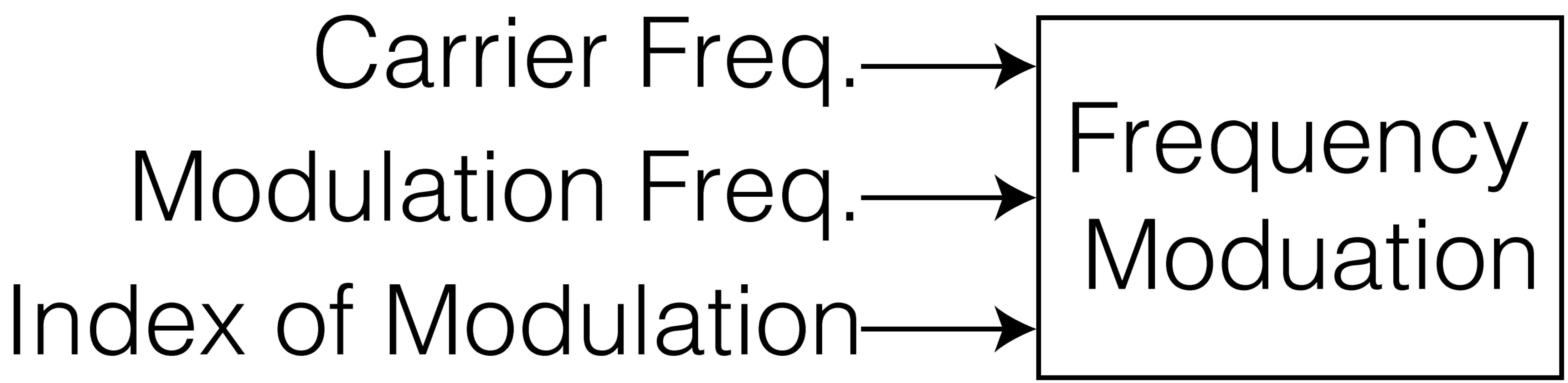
2. Analysis / Resynthesis of Frequency Modulation Spectra

The Goal

Live audio processing module

using sounds of frequency modulation synthesis


3: SimpleFM
Car Freq
116.27
R
Mod Freq
164.26
R
Index
11.93
R
Detune
10
A

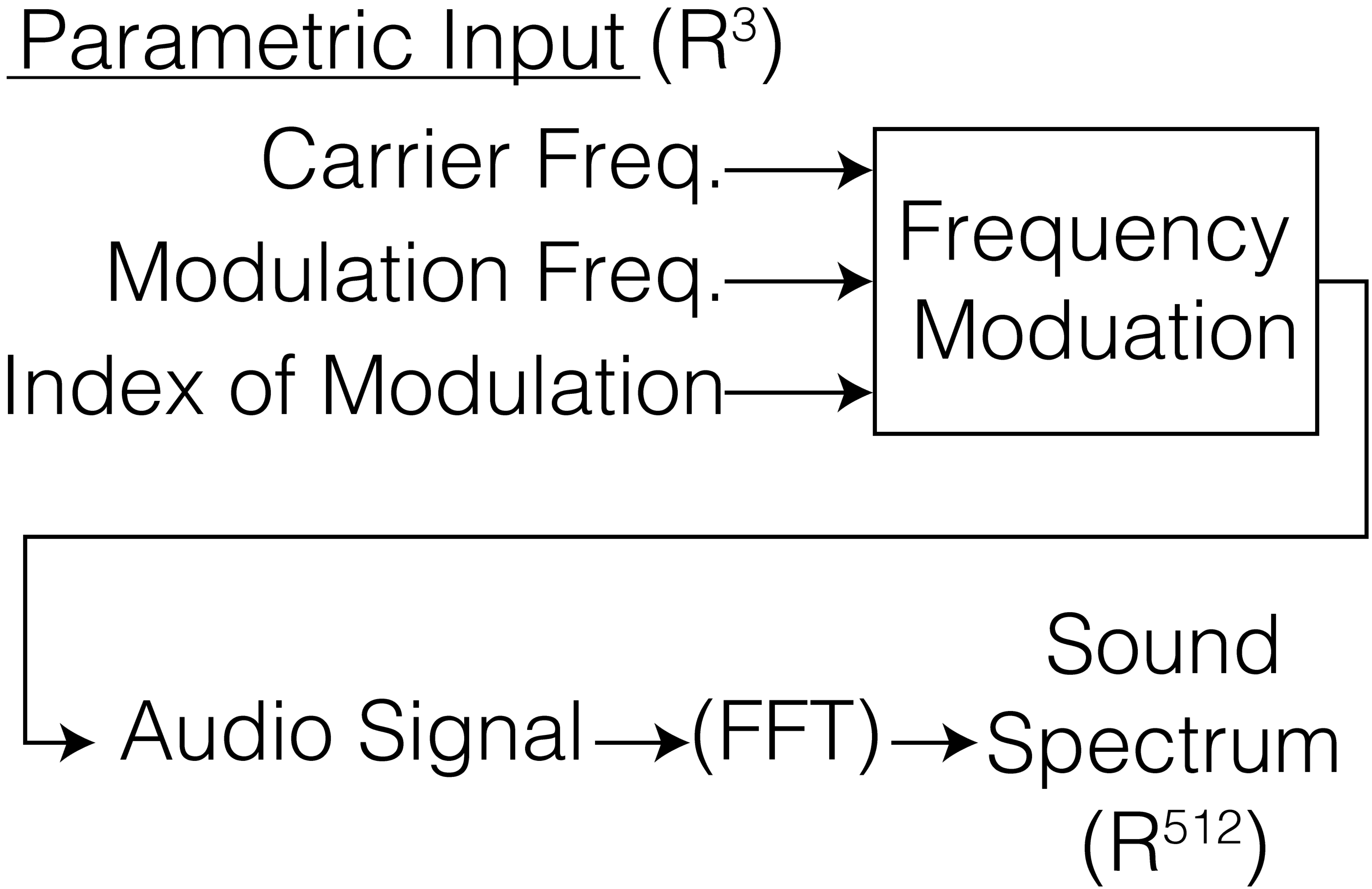

Carrier Freq.
Modulation Freq.
Index of Modulation
Frequency
Moduation

Carrier Freq.: base frequency of tone
Modulation Freq.: smoothness or roughness of tone
Index of Mod.: brightness of tone

Creating Training Set

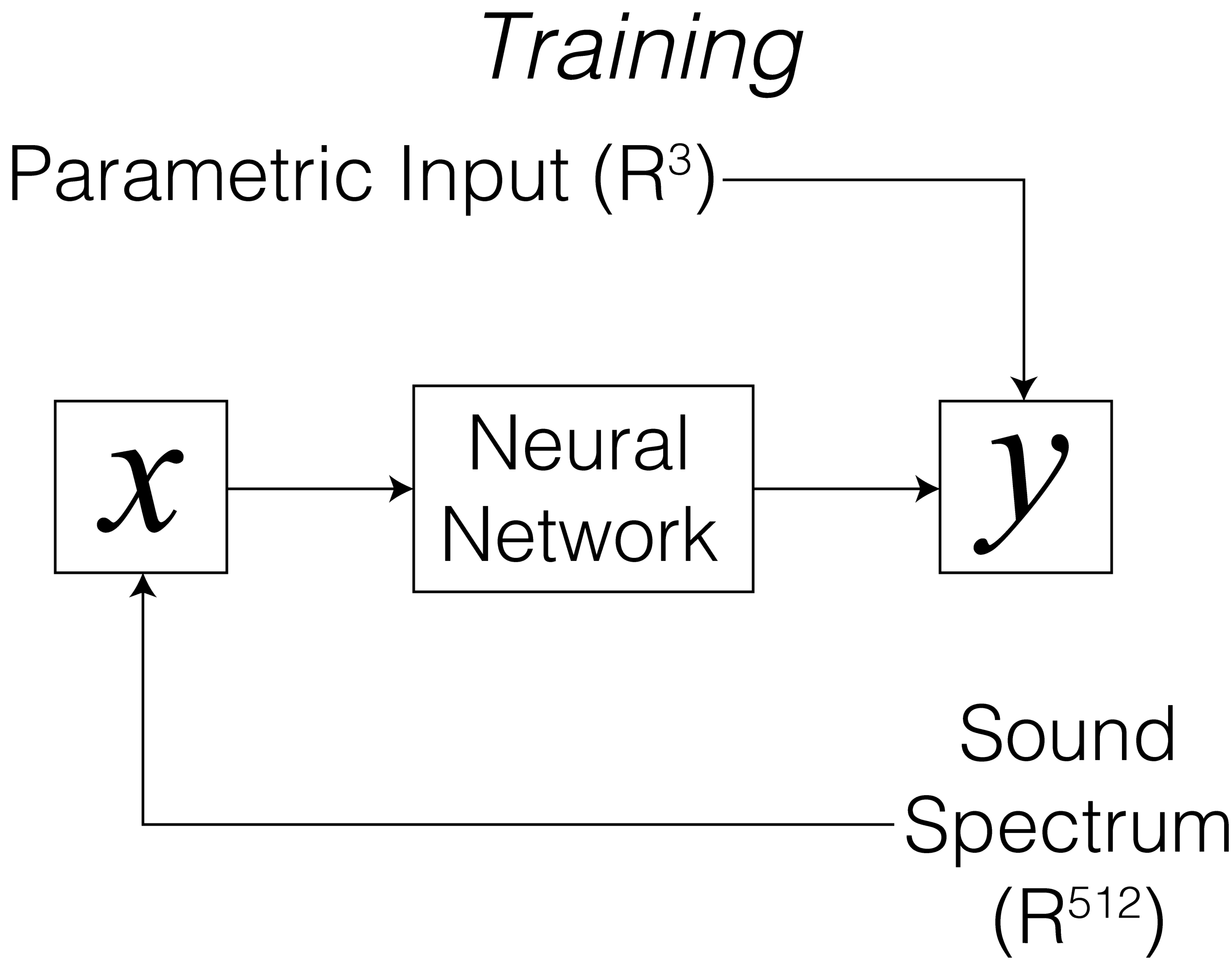
Parametric Input (R^3)

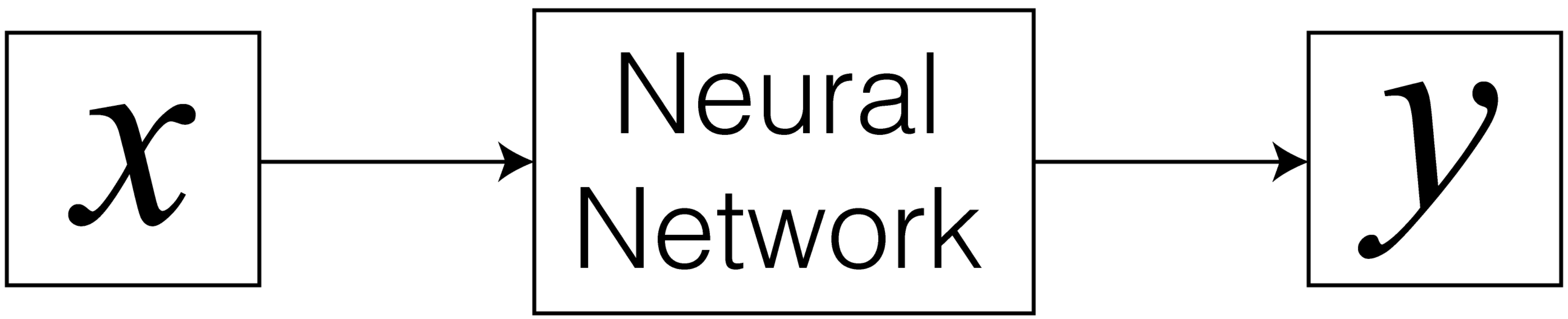
Creating Training Set



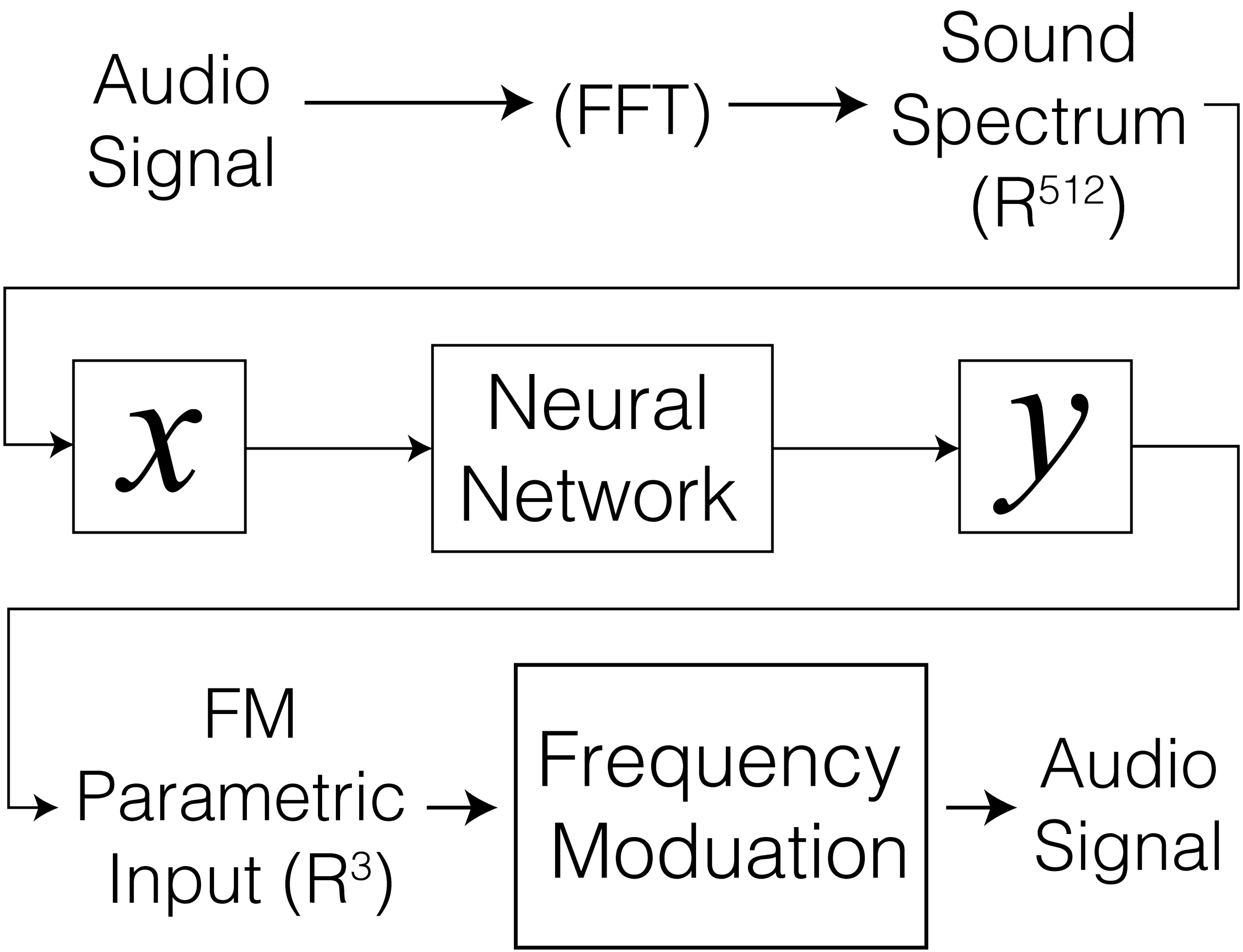
Parametric Input (R^3)

Sound
Spectrum
(R^{512})


Training
Parametric Input (R3)
x
Neural Network
y
Sound Spectrum (R512)


x
Neural
Network
y

Analysis / Synthesis



live demo

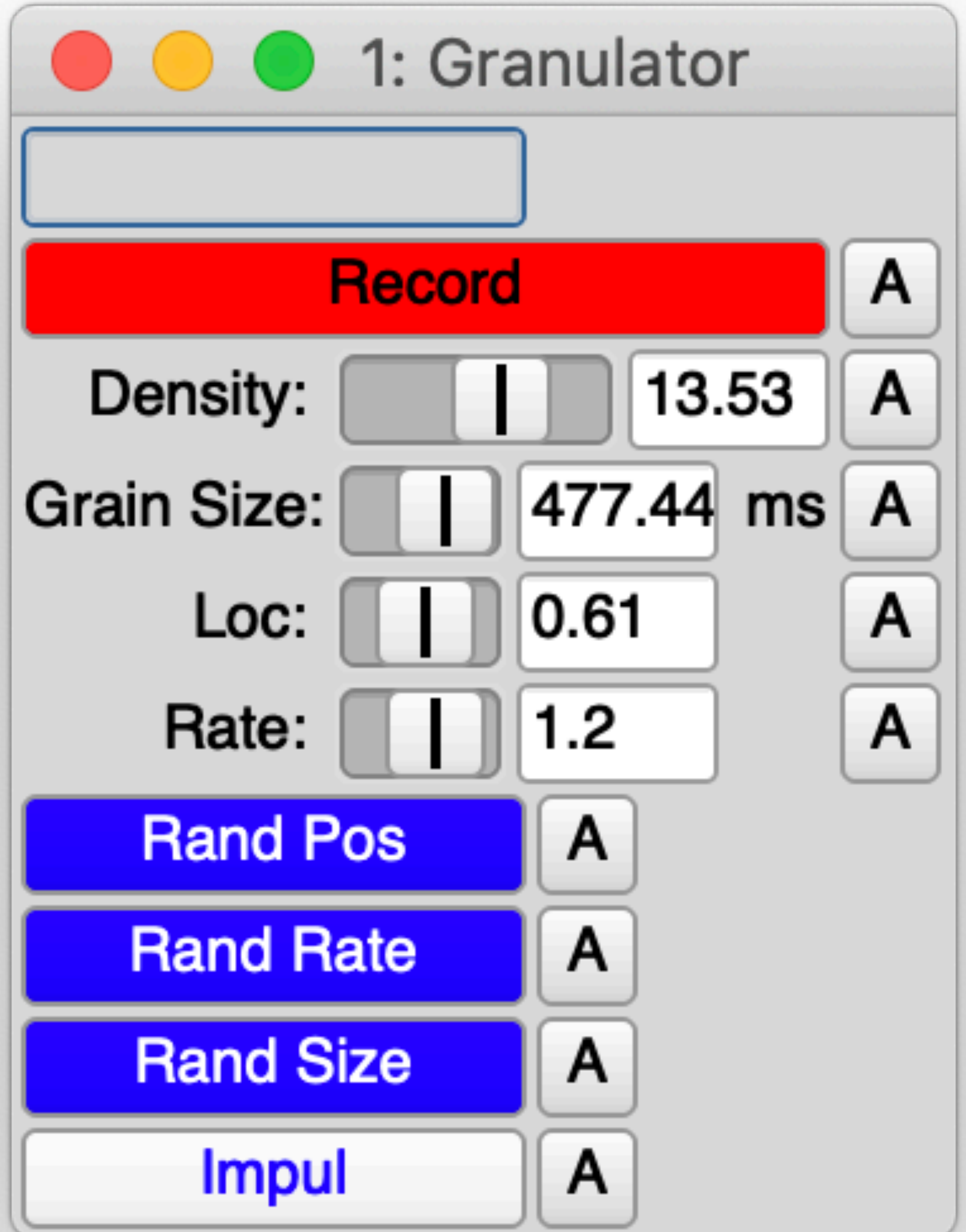
3. Collapsing user-defined expressivity into lower dimensions

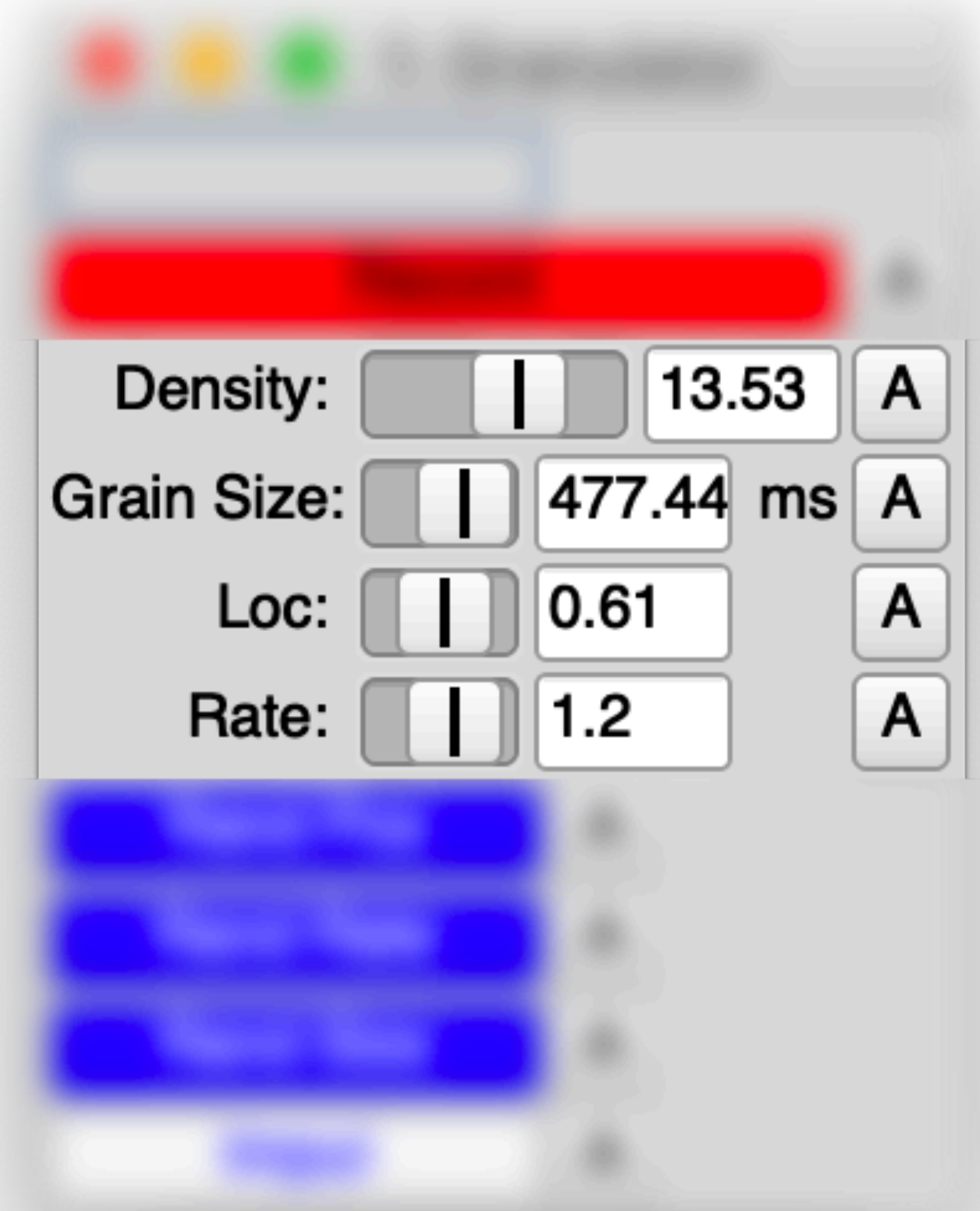
The Goal

Using sound generators that have a high dimension of control inputs,

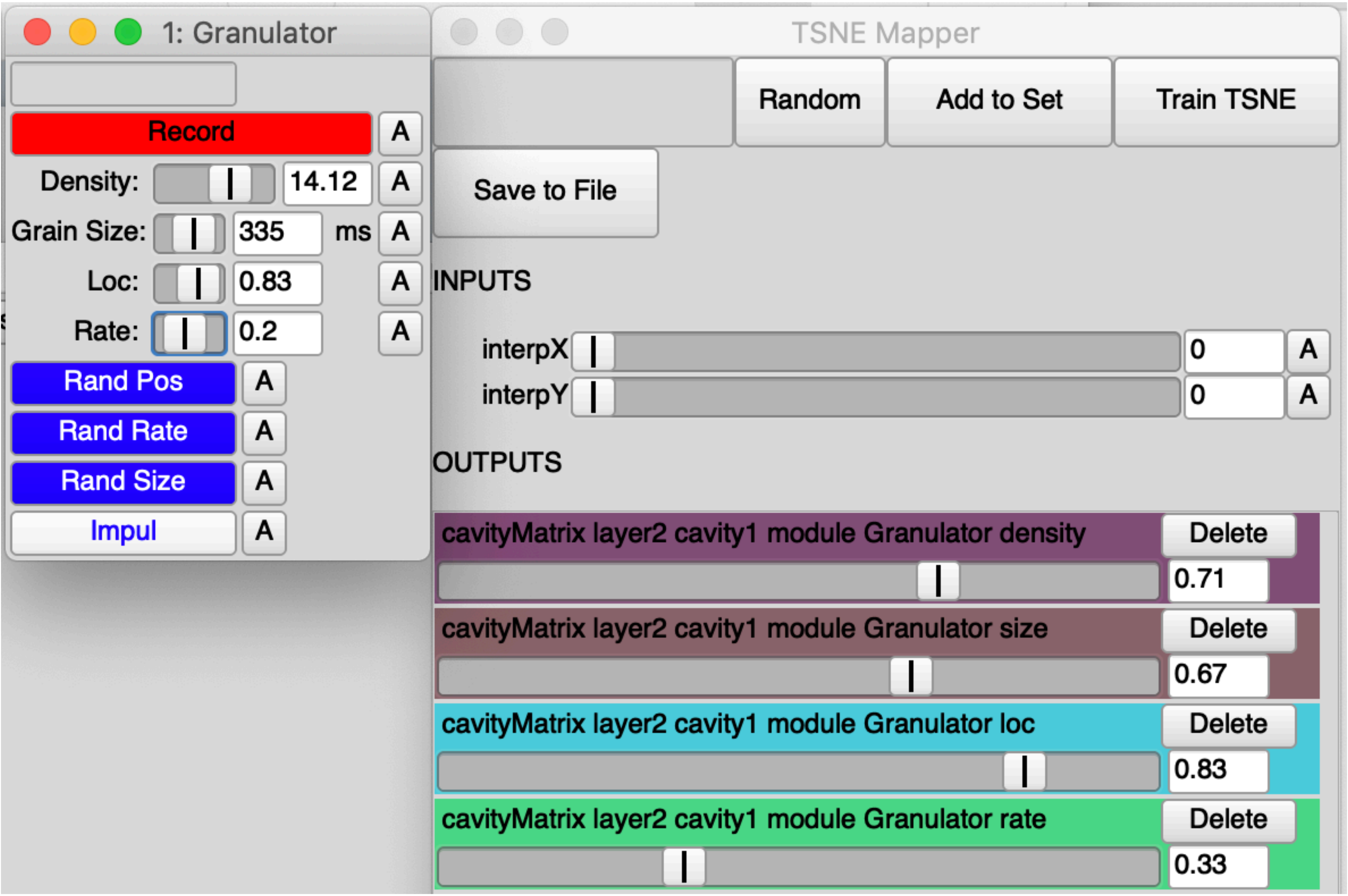
find expressively meaningful combinations of input settings,

then intelligently organized those settings in two dimensions.

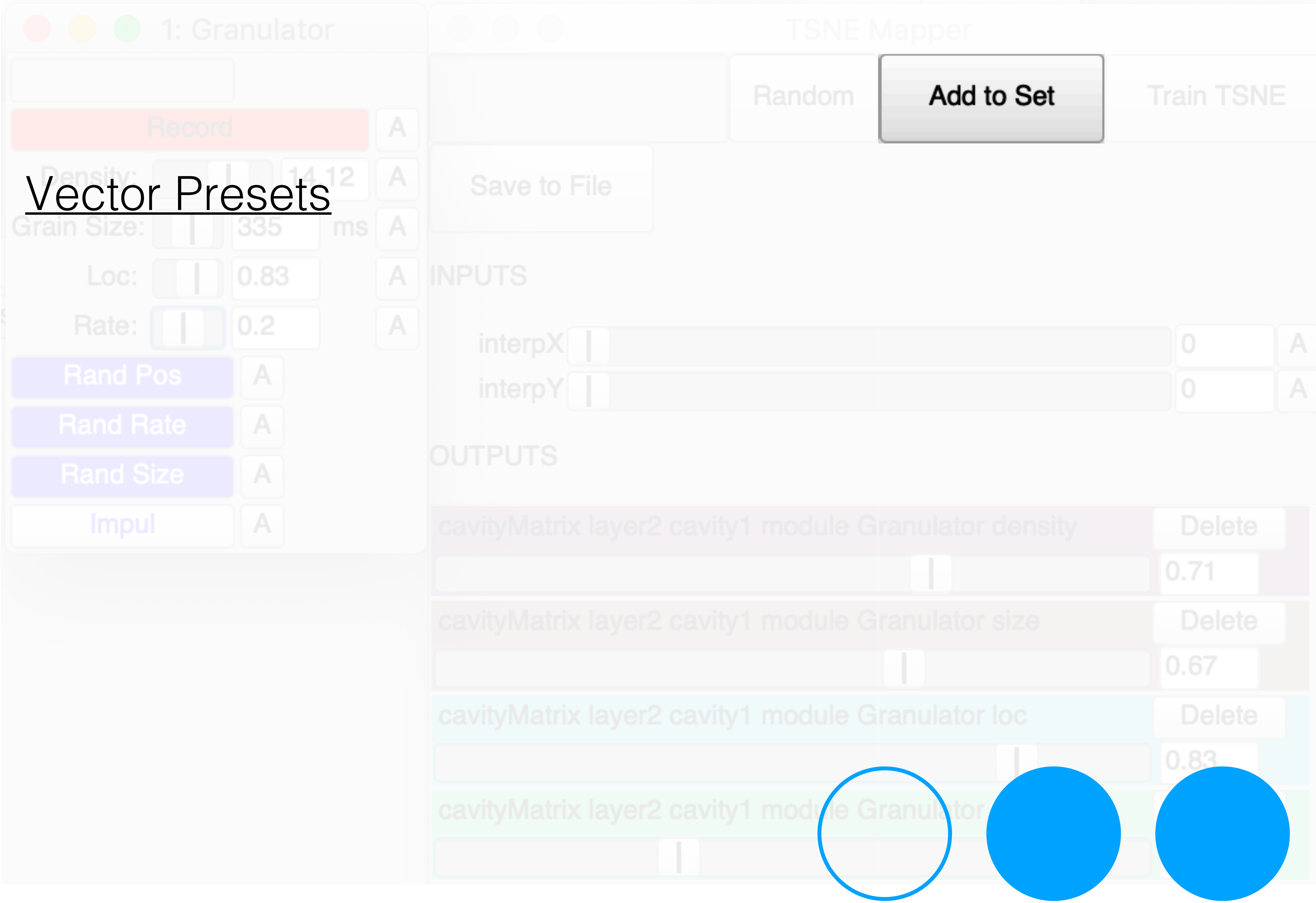

1: Granulator
Record
A
Density:
13.53
A
Grain Size:
477.44
ms
A
Loc:
0.61
A
Rate:
1.2
A
Rand Pos
A
Rand Rate
A
Rand Size
A
Impul
A



Density:
13.53
A
Grain Size:
477.44
ms
A
Loc:
0.61
A
Rate:
1.2
A

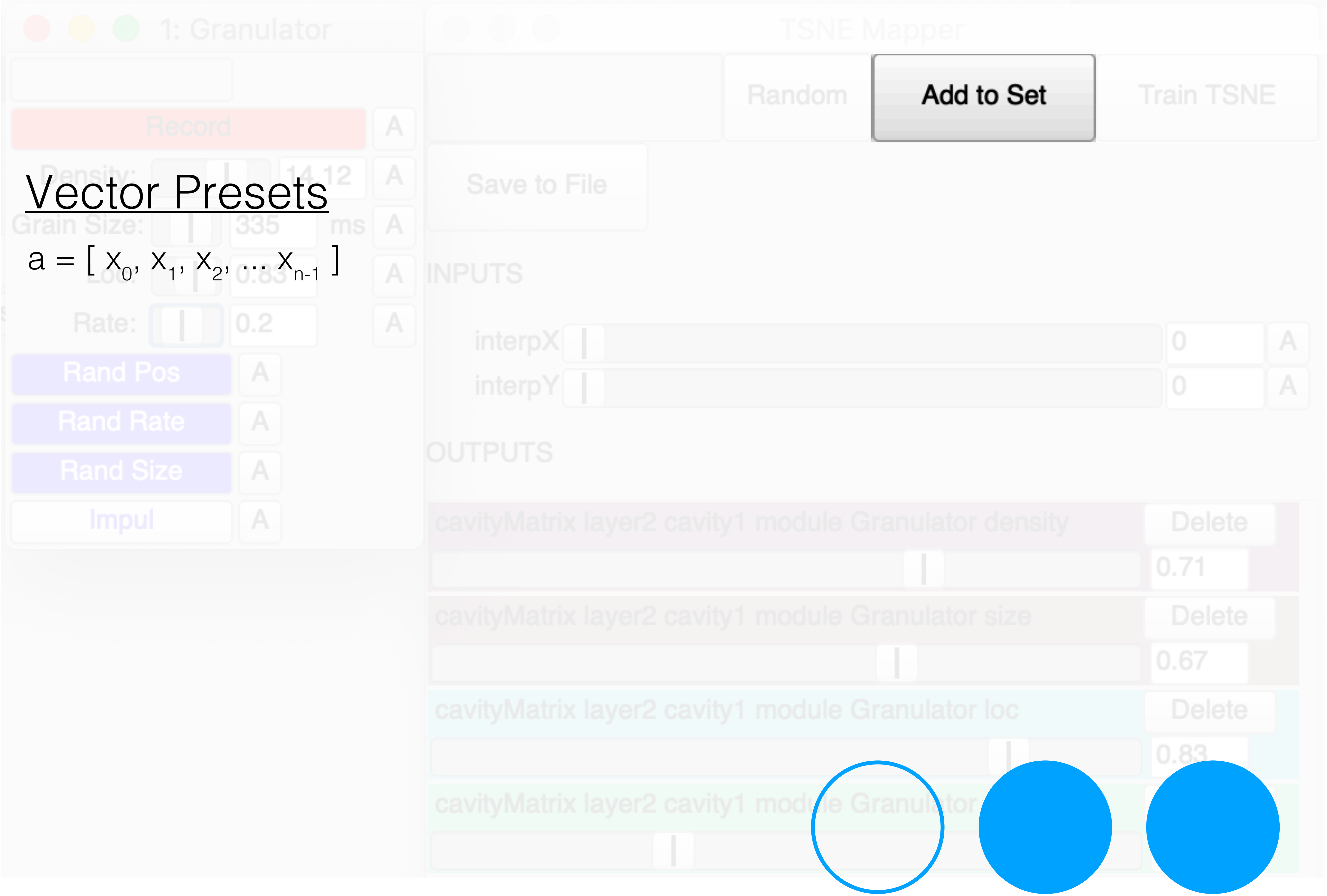

1: Granulator
Record
A
Density:
14.12
A
Grain Size:
335
ms
A
Loc:
0.83
A
Rate:
0.2
A
Rand Pos
A
Rand Rate
A
Rand Size
A
Impul
A
TSNE Mapper
Random
Add to Set
Train TSNE
Save to File
INPUTS
interpX
0
A
interpY
0
A
OUTPUTS
cavityMatrix layer2 cavity1 module Granulator density
Delete
0.71
cavityMatrix layer2 cavity1 module Granulator size
Delete
0.67
cavityMatrix layer2 cavity1 module Granulator loc
Delete
0.83
cavityMatrix layer2 cavity1 module Granulator rate
Delete
0.33

Vector Presets



Vector Presets

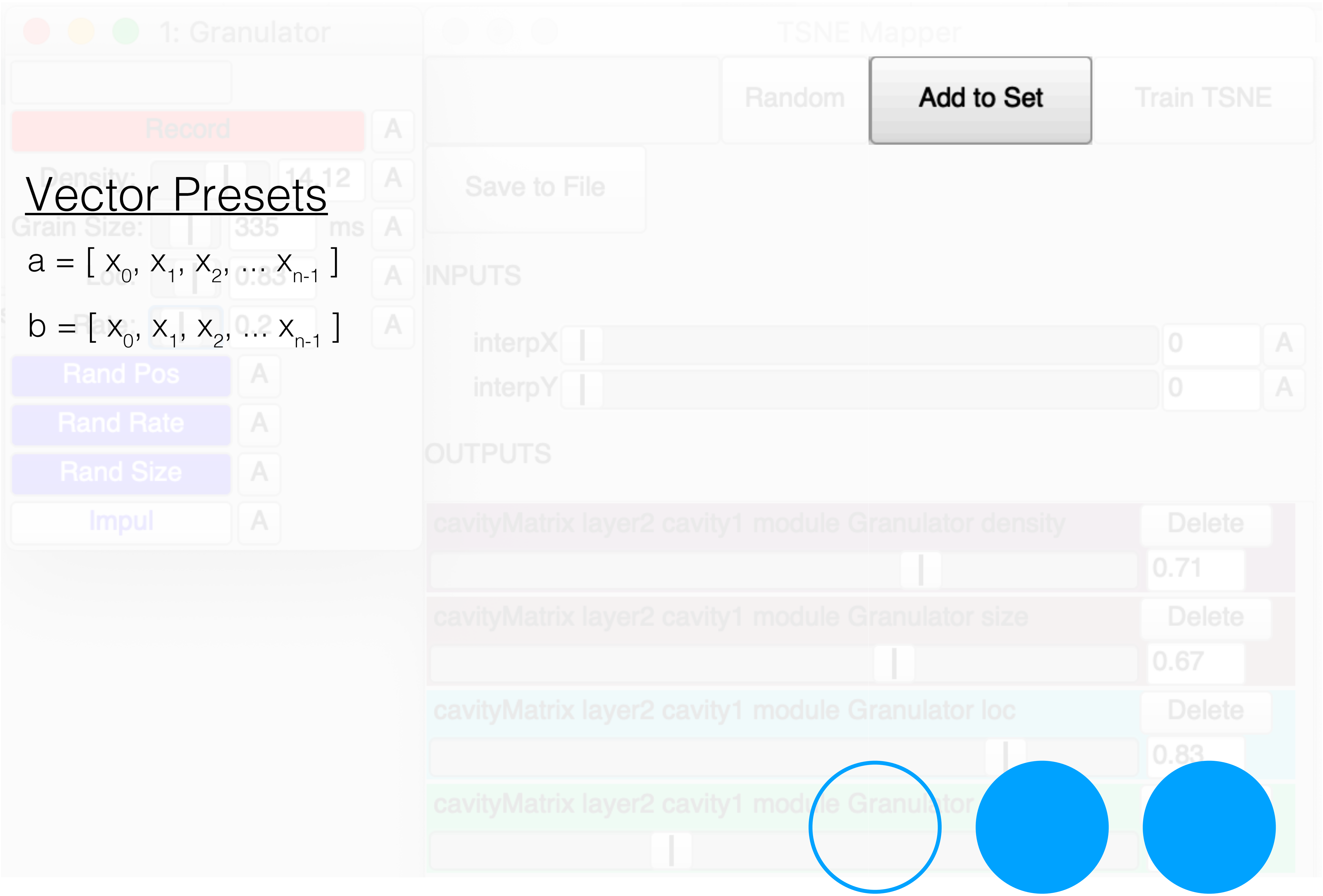
$$a = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$$



Vector Presets

$$a = [\ x_0,\ x_1,\ x_2,\ \dots\ x_{n-1}\ ]$$

$$b = [\ x_0,\ x_1,\ x_2,\ \dots\ x_{n-1}\ ]$$



Vector Presets

$a = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$b = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$c = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

Vector Presets

$a = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$b = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$c = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$d = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

Vector Presets

$a = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

$b = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

$c = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

$d = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

$e = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

$\vdots$

Vector Presets

$a = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

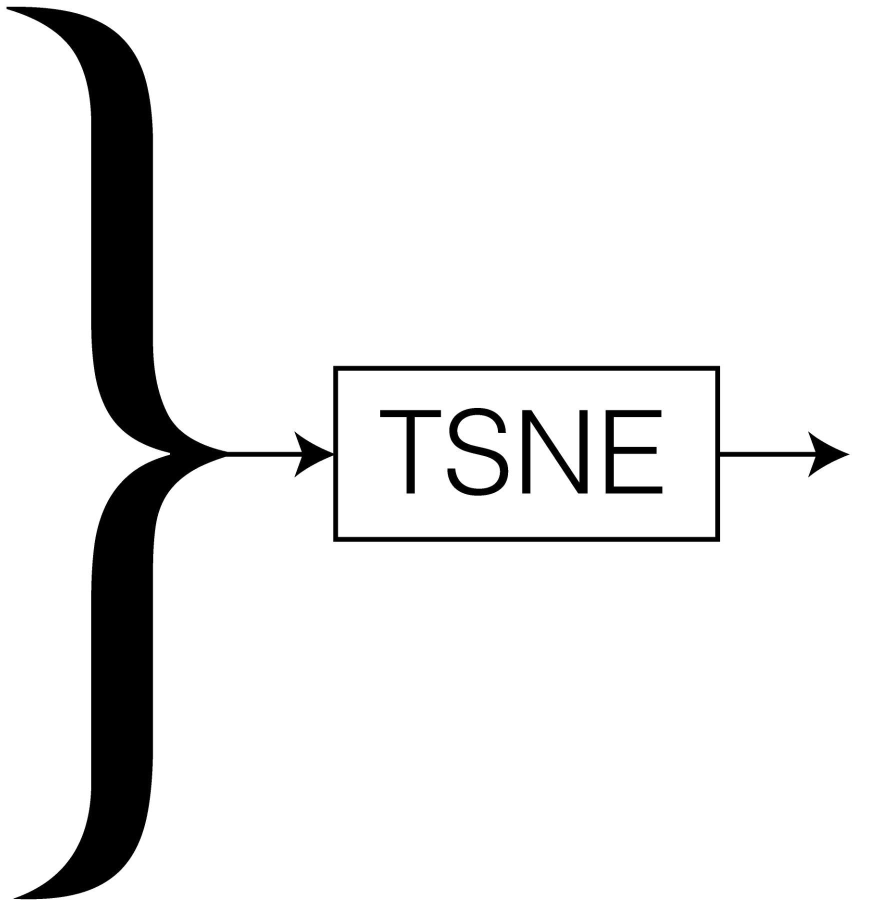
$b = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

$c = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

$d = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

$e = [\ x_0,\ x_1,\ x_2,\ \ldots\ x_{n-1}\ ]$

$\vdots$



TSNE

- t-Distributed Stochastic Neighbor Embedding
- Dimensionality Reduction Algorithm (taking data in a high number of dimensions and reorganizing it into 2 or 3 dimensions, such that it preserves it's structure)
- Vectors that are similar in high dimensional space are embedded near each other, while vectors dissimilar in high dimensional space are embedded far away

Vector Presets

$a = [\ x_0, x_1, x_2, \dots x_{n-1}\ ]$

$b = [\ x_0, x_1, x_2, \dots x_{n-1}\ ]$

$c = [\ x_0, x_1, x_2, \dots x_{n-1}\ ]$

$d = [\ x_0, x_1, x_2, \dots x_{n-1}\ ]$

$e = [\ x_0, x_1, x_2, \dots x_{n-1}\ ]$

$\vdots$

TSNE

Vector Presets

$a = [\ x_0,\ x_1,\ x_2,\ \dots\ x_{n-1}\ ]$

$b = [\ x_0,\ x_1,\ x_2,\ \dots\ x_{n-1}\ ]$

$c = [\ x_0,\ x_1,\ x_2,\ \dots\ x_{n-1}\ ]$

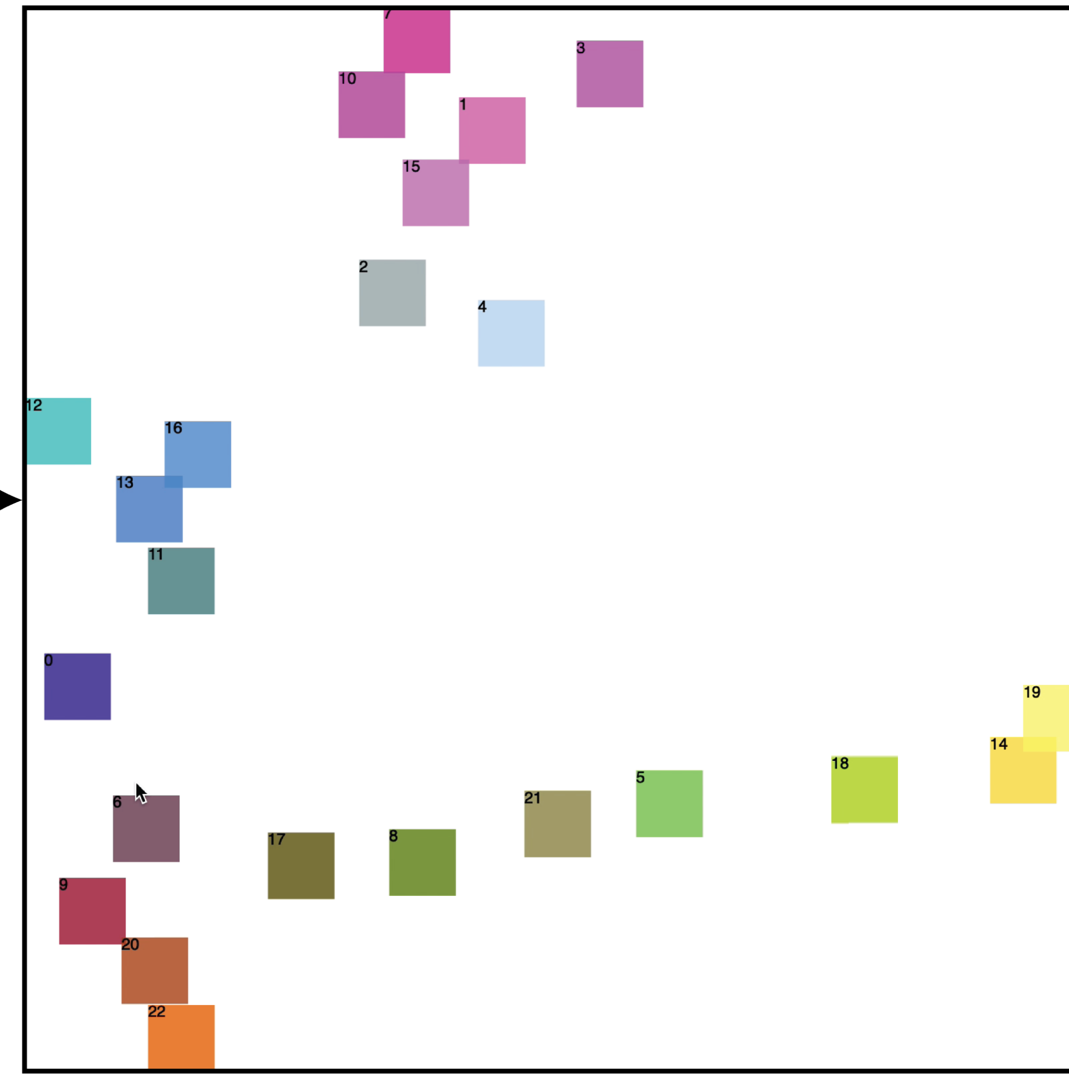
$d = [\ x_0,\ x_1,\ x_2,\ \dots\ x_{n-1}\ ]$

$e = [\ x_0,\ x_1,\ x_2,\ \dots\ x_{n-1}\ ]$

$\vdots$

TSNE

TSNE Solution



Vector Presets

$a = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$b = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

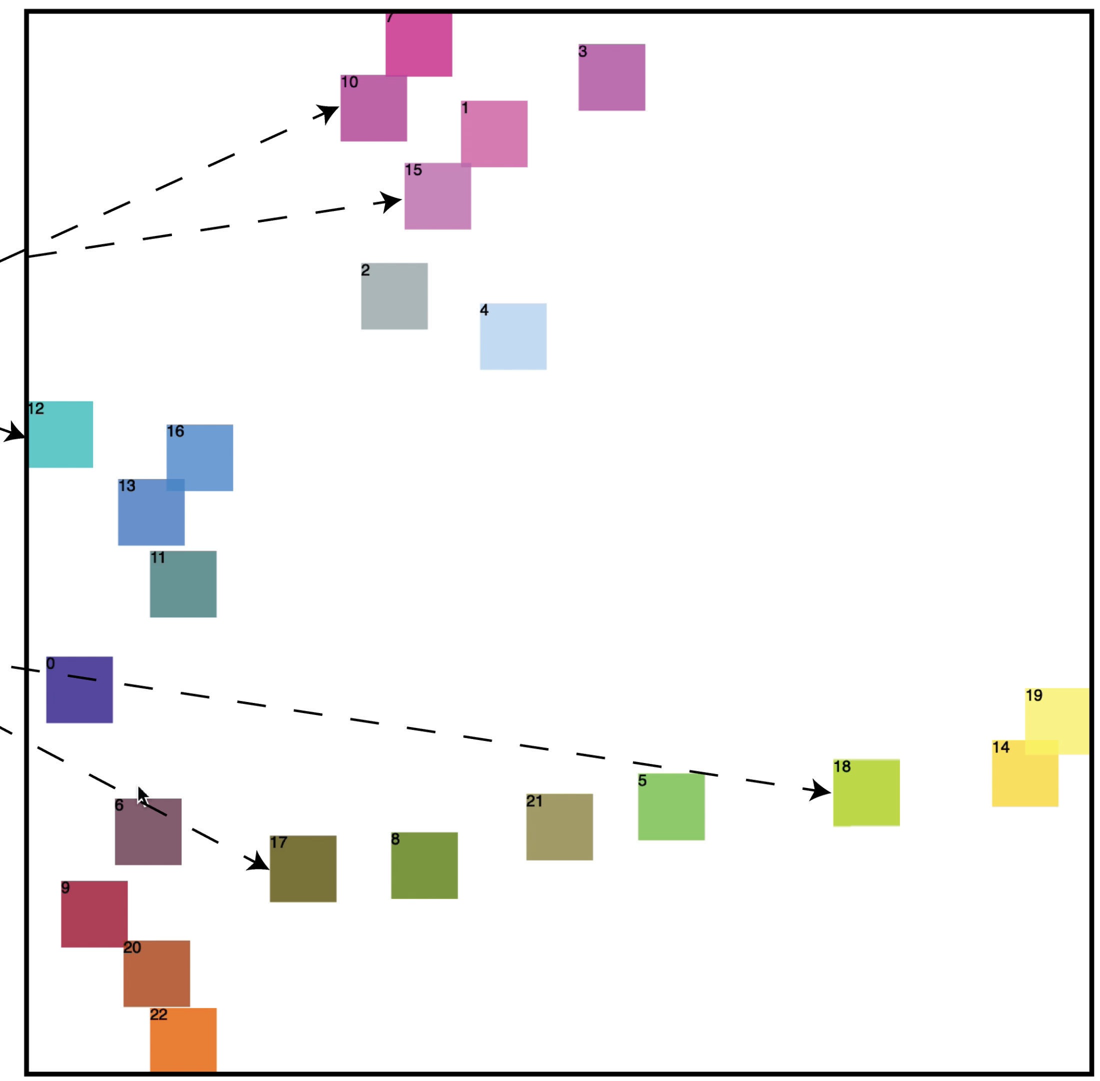
$c = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$d = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$e = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

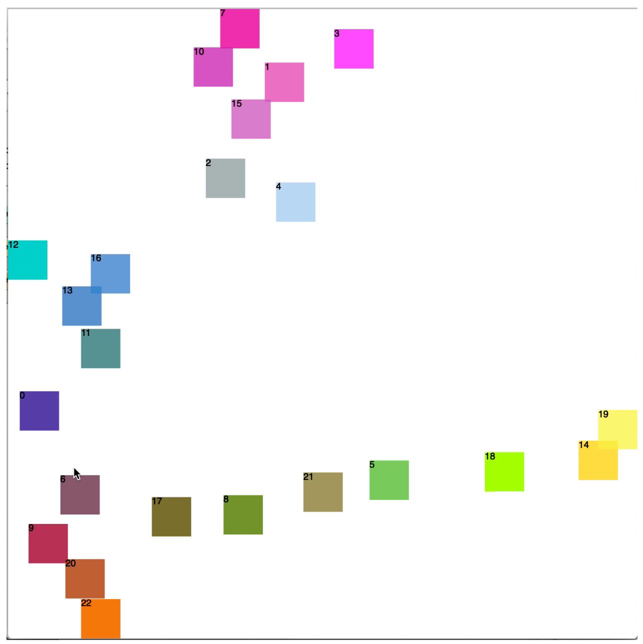
$\vdots$

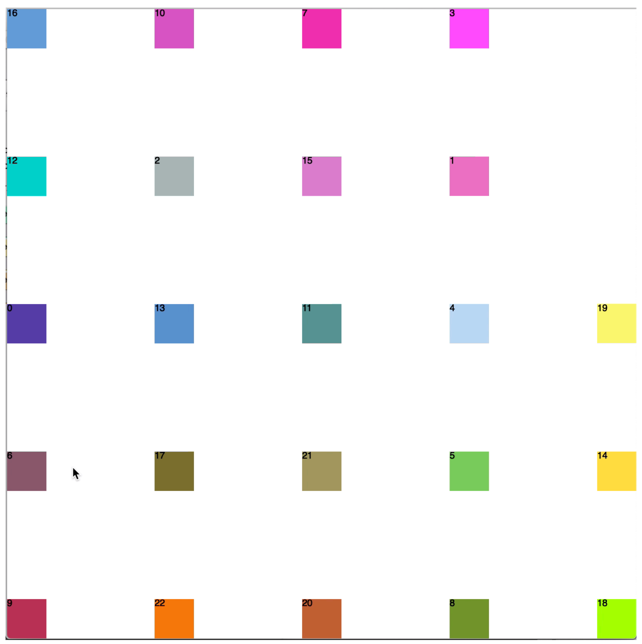
TSNE Solution



Munkres Algorithm

- aka "Hungarian Algorithm" or "Kuhn-Munkres Algorithm"
- Optimal solution to linear assignment problem
- Every element in set A (2D TSNE embedding locations) must be assigned to one unique element in set B (grid of locations in 2D space)





Vector Presets

$a = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$b = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

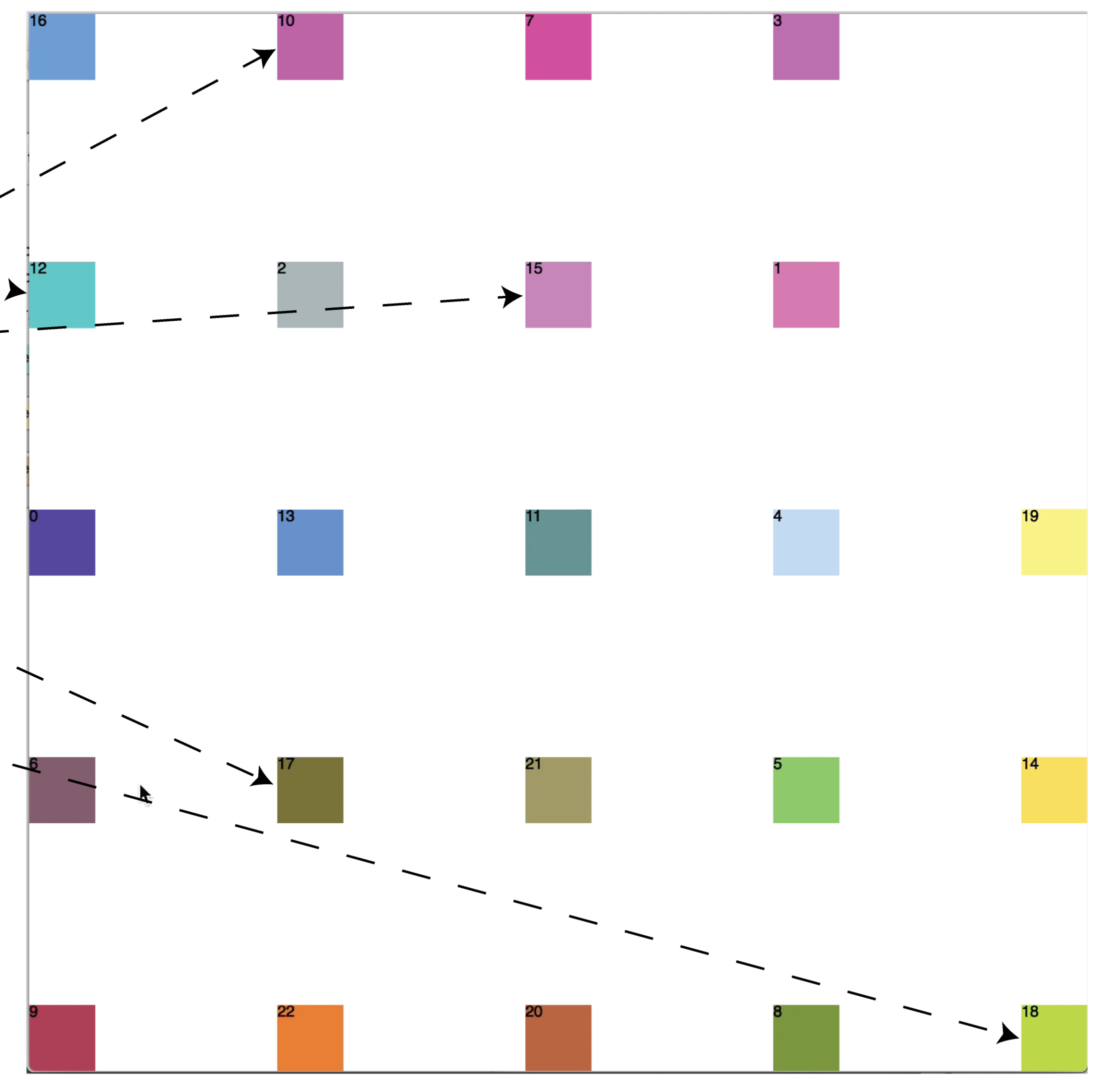
$c = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

$d = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

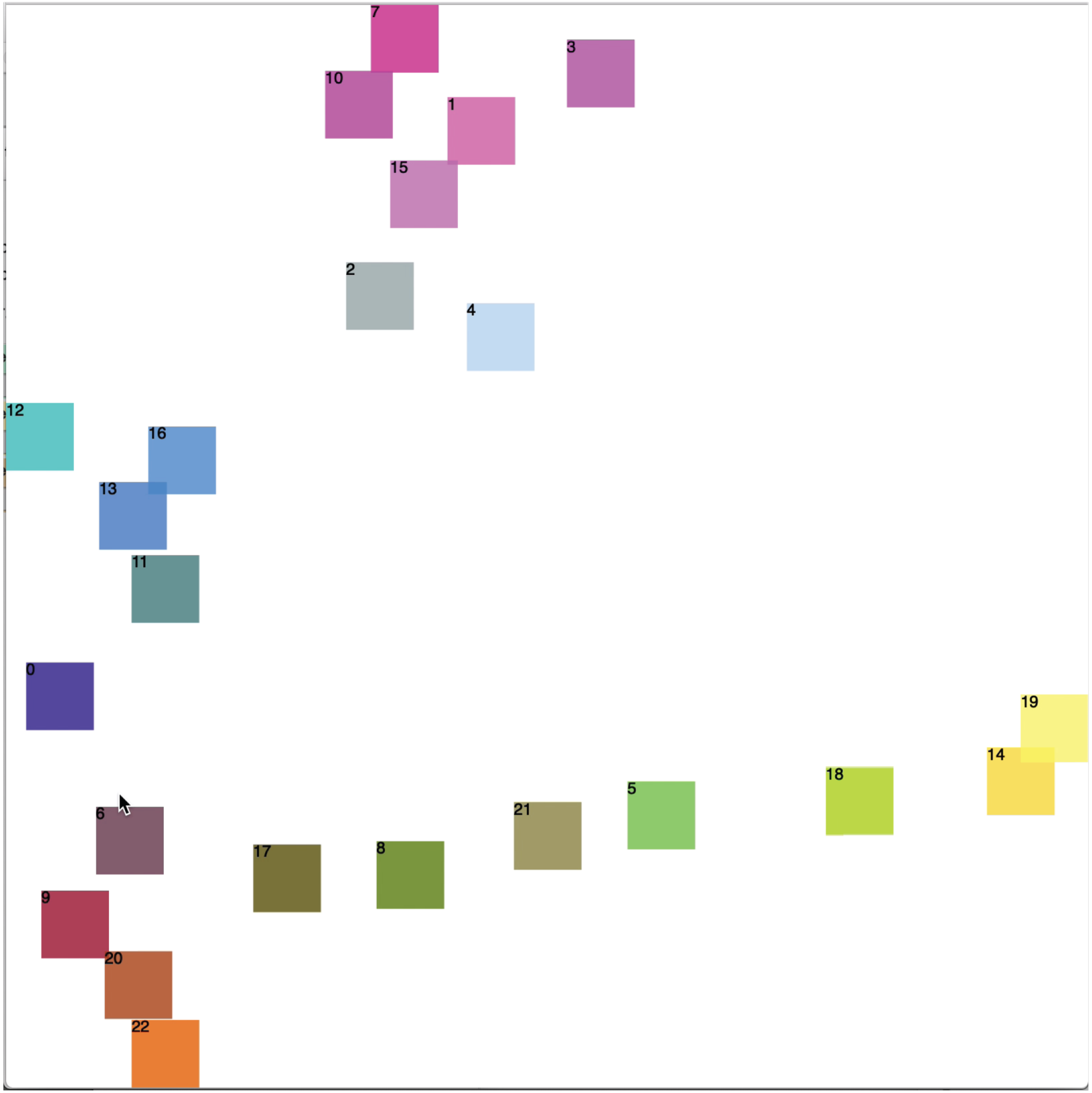
$e = [\ x_0, x_1, x_2, \ldots\ x_{n-1}\ ]$

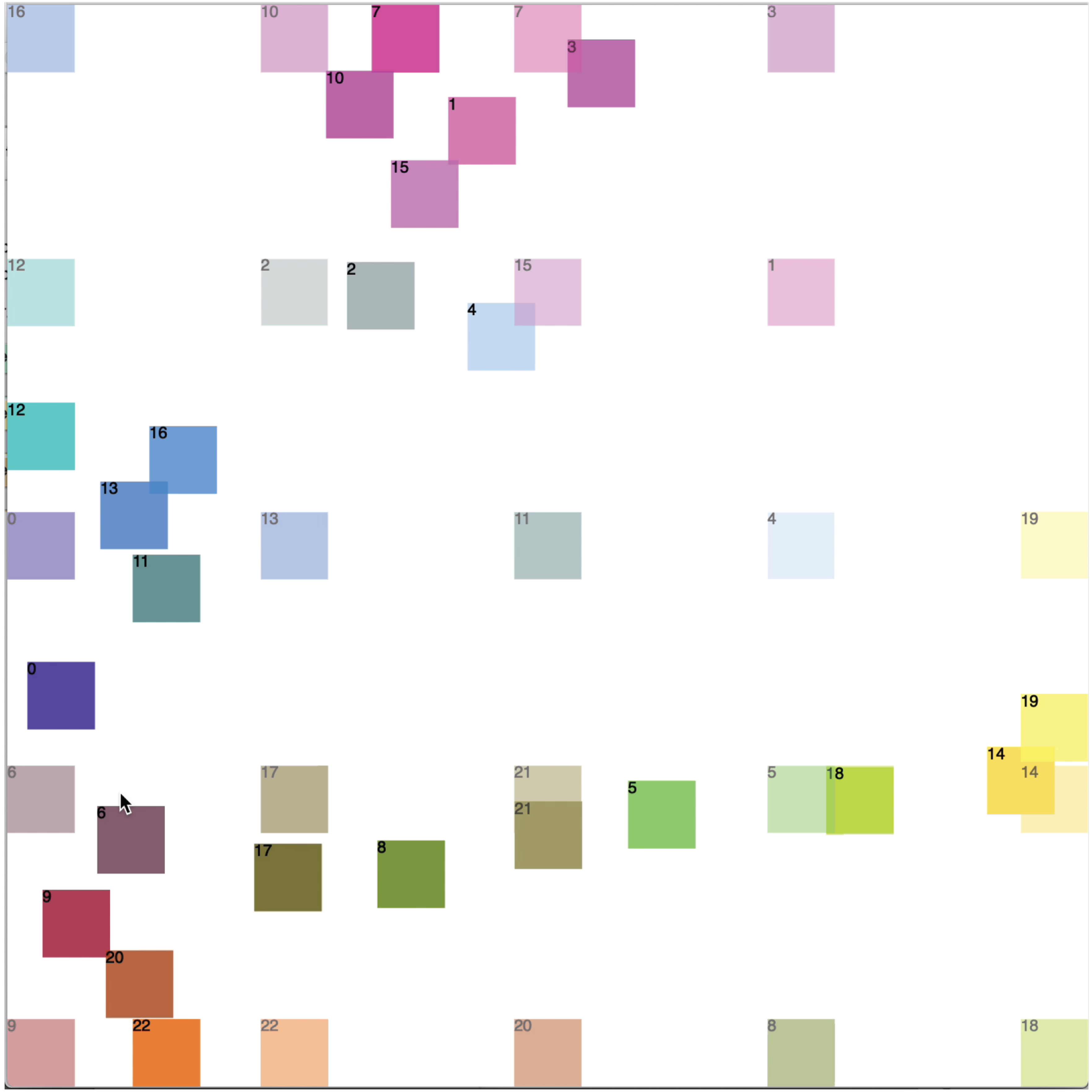
$\vdots$

Munkres Solution

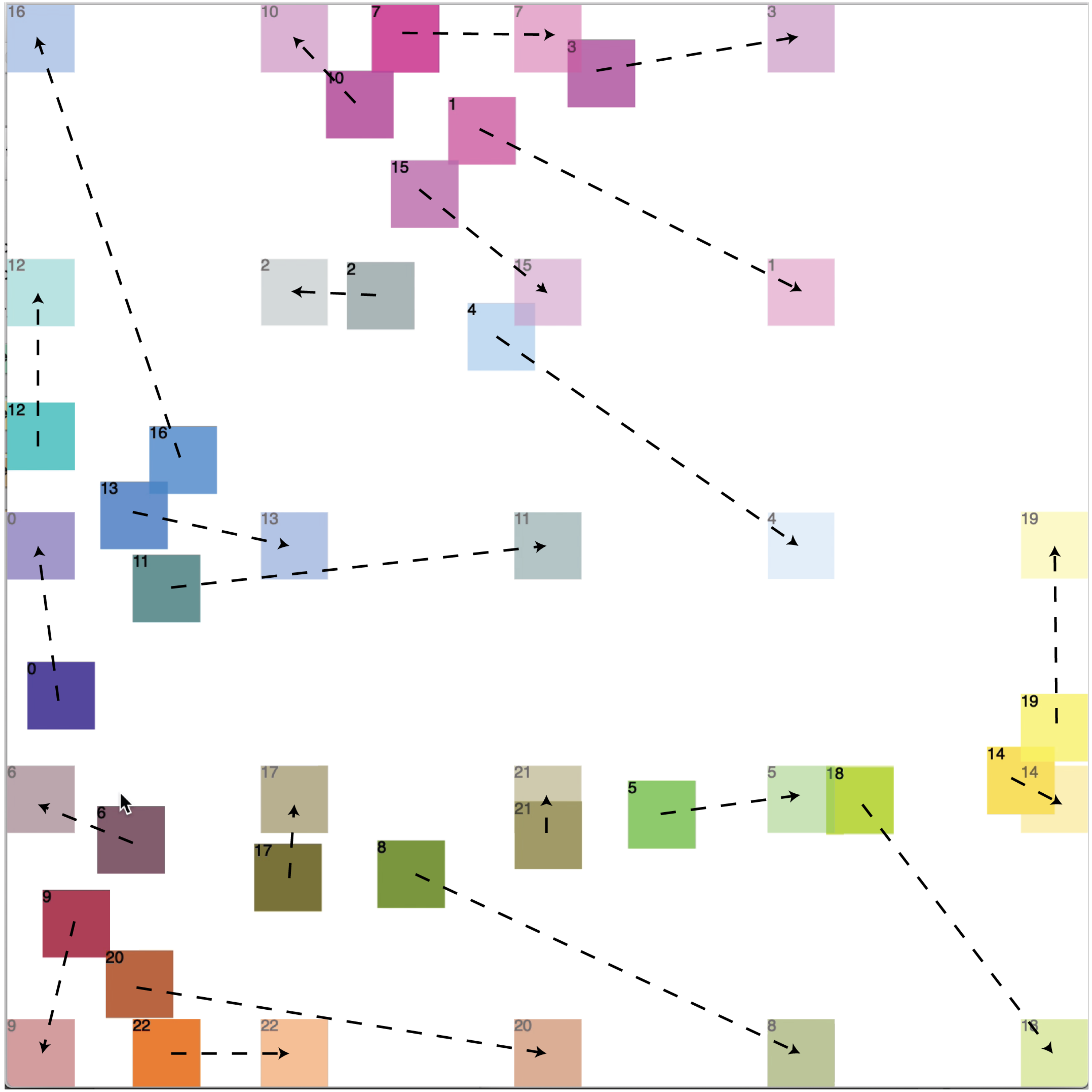


TSNE Solution





16
10
7
7
3
3
10
1
15
12
2
2
15
1
4
12
16
13
0
13
11
11
4
19
0
19
14
6
17
21
5
18
14
21
6
17
8
9
20
9
22
22
20
8
18


16
10
7
7
3
3
10
1
15
12
2
2
15
1
4
12
16
13
0
13
11
11
4
19
0
19
14
6
17
21
21
5
5
18
14
6
17
8
9
20
9
22
22
20
8
18

```
            if(b.value == 0,{
                "% OFF".format(name).postln;
            },{
                "% ON".format(name).postln;
            });
        })
        .addToggleRequestNew(
            path,
            Rect(0,0,20,20),
            testSaver,
            testWin
        );

        if(addToMapper,{
            tsne.makeOutput(path);
        });
    });*/

    testWin.front;


andData = {
    arg tsne;
    23.do({ // works with 7 or greater...
        tsne.randomize;
        tsne.addToSet;
    });
    tsne.trainTSNE(true);




DManager_New.initIfNot;
ndow.closeAll;
sne = TSNEMapper();
akeWin.(~ranges,~tsne);
andData.(~tsne);



alog.savePanel({
    arg path;
    ~tsne.save.writeArchive(path);
;
```

live demo

granulator blue
synth green
granulator red?
lots of params yellow

Benefits of TSNE / Munkres approach

- Preserves user-defined presets
- TSNE recognized as superior dimensionality reduction
- Munkres finds optimal solution
- Non-linear 2D layout requires practice to learn

Rejected Alternatives

- Neural Network - supervised learning requires knowing the desired 2D structure before training
- Self-Organizing Maps - doesn't guarantee that exact user-defined presets are preserved

Future Improvements

- Force-Directed Spreading out of data (able to handle larger datasets)
- Self Organizing Map (reduce redundancy in control vectors)
- Embeddings based on audio descriptions of outputs (instead of parameter inputs)
- Changes in 2D space prompted by machine listening and heuristics (improvising computer)

“The view according to which the novelty of a work guarantees its quality is often expressed in electroacoustic music circles, and for some it is the only criterion of worthiness.”

–Francis Dhomont, *For classicism*

Thank you. Questions?