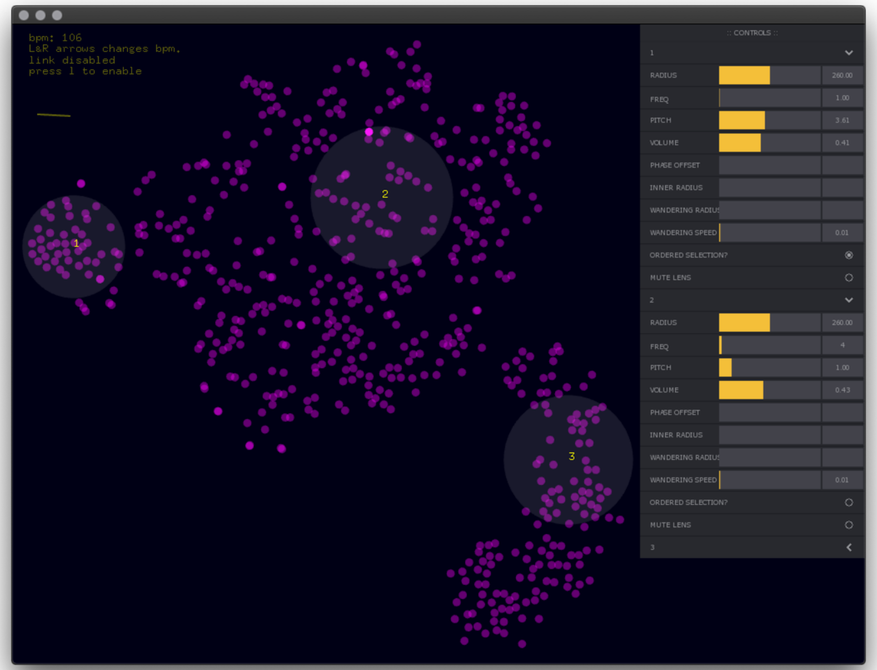
Harry Morley

t-SNE Sampler

Data and Machine Learning Final Project Write-up



```
bpm: 106
L&R arrows changes bpm.
link disabled
press l to enable
```

:: CONTROLS ::

1	
RADIUS	260.00
FREQ	1.00
PITCH	3.61
VOLUME	0.41
PHASE OFFSET	
INNER RADIUS	
WANDERING RADIUS	
WANDERING SPEED	0.01
ORDERED SELECTION?	●
MUTE LENS	○
2	
RADIUS	260.00
FREQ	4
PITCH	1.00
VOLUME	0.43
PHASE OFFSET	
INNER RADIUS	
WANDERING RADIUS	
WANDERING SPEED	0.01
ORDERED SELECTION?	○
MUTE LENS	○
3	

My project consists of 2 elements: an audio analysis and t-SNE generation part, and a sampling/sequencing portion part based on the t-SNE output. A video demonstration of my project is available to view here.

I was initially inspired by the work of Kyle McDonald and Manny Tan, The Infinite Drum Machine. This is an impressive site with over 10000 samples arranged by the t-SNE algorithm in 2D space. It is a brilliant demonstration of using web audio and machine learning in interesting ways. In addition to this, Leon Fedden's Medium article on audio analysis contains a demo similar to The Infinite Drum Machine, that compares different dimensionality reduction techniques. While mousing over sounds is fun to a degree, along with the limited 16 x 4 step-sequencing functionality (in the case of TIDM), I thought the exploration journey seemed a bit curtailed. With that number of sounds, there should be so many ways to use them to create interesting compositions.

Another artist I discovered using t-SNE in interesting musical ways is Jason Levine. He live codes a sequencer that traverses the embedding (such as this or this). Jason did an inspiring talk at MIT where he explained his process from using a t-SNE in its natural form to applying custom grid-fitting algorithms to better organise the sounds in space. I was still interested in using the classic t-SNE form, as I felt this style of 2D representation has a personal and niche property to it, making it suitable for sound selection and sequencing. For example, the shape is easily memorable and one learns where particular sounds are.

With all this in mind, I set out to create a similar project that analyses my own sounds (around 600), performs the t-SNE and then have a sampling/sequencing part offering interesting ways to traverse the 2D map, sonifying the journey of exploration.

Implementation

Audio analysis

What I learned early on in the project is that I would need to work with a large number of samples straight away, since t-SNE is more effective and purposeful with large datasets. When working with a few samples to start with, the t-SNE would produce wildly varying results, and it was tricky to predict how it would perform with more samples.

So, I took some time to collect and curate my own samples, either by recording them myself or downloading royalty-free samples from sites like LANDR Samples. I wanted this project to be of compositional benefit to myself (and others eventually!) straight away rather than it be a demo of random sounds, so I actively chose and made decisions about what sounds I would include. A dog woof did make its way in there however.

To ensure my samples worked with the system, I had to perform some batch preprocessing of the files using an Audacity macro. Firstly, the macro truncated the silences at the beginning and end of the audio files. This is for two reasons: firstly, so that there would be no delay in hearing the sample when triggered and the analysis would be more accurate (i.e. not ‘influenced’ by any silences), and secondly to save disk space and RAM usage during runtime.

For the audio analysis, I briefly explored using a C++ library such as Gist, and although there exists an openFrameworks addon for it, I found myself spending too much time trying to set it up when there was an easier way. For example, a function for computing the first and second order MFCC deltas did not exist, so rather than spending too much time on it, I decided to use Python and LibROSA, as per Gene Kogan’s Audio t-SNE demo.

The Python script analyses only the first second of each sample. The 13 Mel-Frequency Cepstrum Coefficients for this sample are computed, as are the first and second order deltas of the MFCCs, making a vector of 13 * 3 = 39 features.

Dimensionality reduction

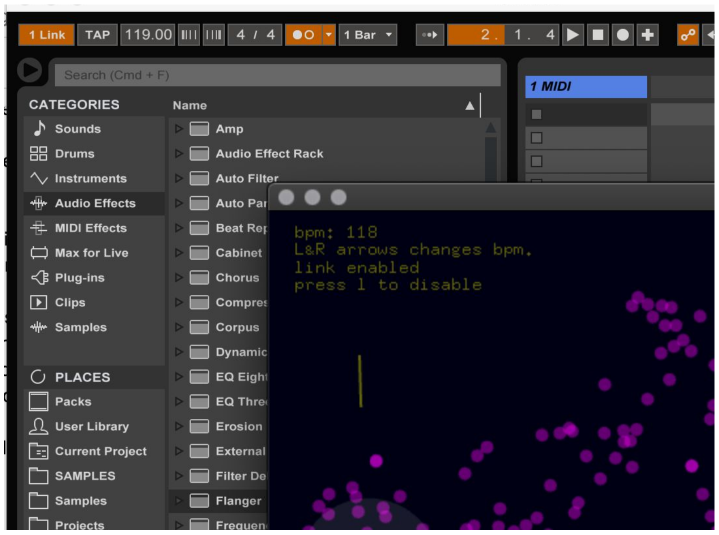
I decided to use the t-SNE algorithm (van der Maaten, Hinton, 2008) because of its effectiveness at visualising high-dimensional data and preserving the local and global structure of the data. There is also an easy to use implementation of it in Scikit-learn, with a handful of tunable parameters.

t-SNE can be very slow with a large dataset. Also, the stochastic means by which it works can cause uncertainty in how long the algorithm will take to reach a suitable endpoint. It generally took around 10-15 seconds to process my ~600 samples, which was absolutely fine as this was just in the preprocessing stage. It can also be tricky to interpret the results if the parameters are not tuned correctly, for example it can show clusters when there shouldn't be. The online article by Wattenberg, et al (2016) does a great job at explaining the importance of trying different parameters, such as for perplexity: when set too high, the output can just turn out to be a ball of points. When perplexity is set too low, local structures take over, causing too many small neighbourhoods of points.

Sampler part

For the sampler/sequencer part, I used openFrameworks with the Maximilian addon. Currently, the program only plays 1-channel, 16-bit, 44.1kHz .wav files (as per the ofxMaxim specification) which was just fine for this stage of the prototype. Using Maximilian gave me a good balance between low-level control (for example timing to a phasor oscillator, and triggering on a per-sample basis in the audio callback) and ease of use (file loading, ability to change speed/pitch, etc). I experienced erratic latency when triggering sounds using the built in ofSoundPlayer class, and it felt easier to trigger the sounds off of a phasor clock, so I used Maximilian.

I then came across an openFrameworks addon for Ableton Link, which worked straight away and was interchangeable with my master clock (a phasor that ramps from 0 to 1 every bar), so I implemented the option to enable Link.



```
bpm: 118
L&R arrows changes bpm.
link enabled
press l to disable
```

Ableton Link in action with the t-SNE Sampler

Sounds are triggered by 'Lens' objects. I liken these to sonic magnifying glasses, trawling over and uncovering the sonic artefacts. A Lens can be created by clicking and dragging to set its radius. All samples inside a Lens can be triggered. The speed and time that the triggering occurs depends on the 'freq' and 'phase offset' sliders for each Lens in the controls panel. These values are relative to the master clock, which goes from 0 to 1 every bar. For example, a frequency of 4 will make the Lens trigger a sound 4 times in a bar. Using this method, polyrhythms can be easily formed. Something that emerged which I liked was the ability to turn the frequency up super fast, creating a kind of granular synthesis / microsound aesthetic.

Each Lens triggers its samples at a set volume and relative speed/pitch, again according to the control panel. The wandering radius and speed controls make the Lens drift around its origin position, according to Perlin noise. This allows the Lens to travel into different neighbourhoods of sound, as well as just triggering sounds within a set region.

After each Lens collects all of its currently "in view" samples (this happens every frame), it sorts them by distance from the center of the Lens. A feature can be turned on which triggers the samples in order from closest to farthest. When this feature is turned off, the samples are chosen at random. I implemented this feature as an exploratory tool for sonifying the relationship between one sound and its neighbouring sounds. The 'inner radius' parameter allows you to turn the Lens into a ring, where only sounds within the ring are triggered.

Reflection

Future versions

I am interested in turning this into a more accessible system for composition - maybe inside of a DAW as a plugin, or as a standalone piece of hardware powered by a Raspberry Pi. What I dislike about the current state is that many sliders are required to interact fully with the program. This interrupts the workflow as it detracts from the visualisation - I do however like clicking and dragging the Lenses around.

Perhaps there could be another machine learning element that takes in an input from a device such as a game controller and translates gestures into parameter settings and Lens positions. Or maybe a rule-based system with simple but emergent properties for modulating parameters. With this project, I really wanted to draw out as many techniques for traversing the 2D map as possible. Although it is a simplified representation of sound, there exists countless ways for interacting with it, and it is far more intuitive than creating beats by trawling through sounds in a file browser.

With more time, I would like to investigate performing analysis on new samples in the background and adding new points to the embedding without changing the layout too much.

With regards to the sequencing part, I would like to come up with intuitive ways of creating long-form compositions using this system

In terms of visualising the sounds, I would have liked to colour-code them according to their likeness to a set of reference sounds (such as kick, snare, symbal, tone, etc). This could have been achieved by calculating the distance between every sound and the reference sounds, and sorting them.

Compiling the project

The analysis part of the project requires the Scikit-learn, Matplotlib, Numpy and Librosa libraries. I used the Anaconda distribution of Python which comes with the machine learning and maths libraries, and then I installed Librosa separately using `conda install`.

The openFrameworks/C++ part of the project requires the ofxJson, ofxDatGui, ofxAbletonLink and ofxMaxim libraries. I modified the ofxMaxim library by implementing a getter function for the MaxiSample class in order to know whether the sample has finished playing.

To do this, I simply added this to the public section of the maxiSample class (in maximilian.h):

```
bool atEnd()
{
    return (long)position>length;
}
```

To run the openFramework project, move the folder containing the Xcode project into your of_root/apps/myApps folder, or alternatively set up a new project with the ofxJson, ofxDatGui, ofxAbletonLink and ofxMaxim addons and copy over the contents of the src and bin/data folders. Ensure the maximilian.h file is edited to include the atEnd() function, as above.

The analysis.py script lives in the bin/data folder of the openFrameworks project, and will automatically look for .wav files inside the wavs folder upon running. Ensure the listed Python packages are installed and run the script from a terminal. It will output a file named tsne-data.json in the same directory. When running the openFrameworks project, this file will get loaded.

References

Third-party code

The `analysis.py` script is based on Gene Kogan's 'audio-tsne' Jupyter notebook which worked very well. I turned it into a single script, added comments and modified it to write out relative file paths to the JSON, for ease of use in openFrameworks - the rest is Gene's original code.

Libraries

Python part: Scikit-learn, Matplotlib, Numpy, LibROSA
openFrameworks part: ofxAbletonLink, ofxDatGui, ofxJSON, ofxMaxim

Bibliography

Fedden, L. (2017) *Comparative Audio Analysis With Wavenet, MFCCs, UMAP, t-SNE and PCA.* Available at: https://medium.com/@LeonFedden/comparative-audio-analysis-with-wavenet-mfccs-umap-t-sne-and-pca-cb8237bfce2f (Accessed: 07 May 2019).

Kogan, G. *Audio t-SNE* (ml4a Guide). Available at: http://ml4a.github.io/guides/AudioTSNEViewer/ (Accessed: 09 May 2019).

Kogan, G. *Audio t-SNE* (Jupyter Notebook). Available at: https://github.com/ml4a/ml4a-guides/blob/master/notebooks/audio-tsne.ipynb (Accessed: 09 May 2019).

McDonald, K. and Tan, M. (2017) *The Infinite Drum Machine.* Available at: https://experiments.withgoogle.com/drum-machine (Accessed: 08 May 2019).

van der Maaten, L. J. P. and Hinton, G. E. (2008) 'Visualizing High-Dimensional Data using t-SNE', *Journal of Machine Learning Research, 9(Nov), pp. 2579-2605.* Available at: http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf (Accessed: 09 May 2019).

Wattenberg, et al. (2016) *How to Use t-SNE Effectively*. Available at: http://doi.org/10.23915/distill.00002 (Accessed: 01 May 2019).

All audio samples included with this submission are either royalty free and public domain or created by myself using field recordings.

A video demo of my project is available to view here: https://vimeo.com/335258061