FluCoMa for Pedagogues

by Ted Moore, James Bradbury, Pierre Alexandre Trembly, and Owen Green

Contents

1 Introduction

The FluCoMa Toolkit is much more than the actual code objects that use the "Fluid" prefix. FluCoMa is also a collection of learning resources, code examples, commissioned artworks, musicological articles, interviews, podcasts, a philosophy about interface design for creative coding, a conversation about the future of computer music, a curriculum of machine listening and machine learning topics, a community of users around the world, and more. The materials included here extend this ecosystem to encompass resources for pedagogues that might be used to teach FluCoMa in various settings. While some of the ideas and resources presented below are FluCoMa-specific, many of them are toolkit-agnostic and we hope that they can be used by anyone looking to teach machine listening and machine learning for creative music making.

The extent of the FluCoMa ecosystem supports our belief that "providing the tools" is not enough to achieve FluCoMa's mission: *enable techno-fluent musicians to use machine listening and machine learning in their creative practices*. To truly enable these artists we also must provide knowledge and inspiration, all in a way that is learnable for our main user base: computer musicians. Topics in machine listening, machine learning, computational thinking, and data science are often not included in the training of electronic musicians and therefore our task is to build bridges of understanding from the knowledge of computer music training to a degree of fluency with these topics that enables creative music making.

1.1 Methodology

The learning materials included have been created through participatory, iterative, and interactive design. The first draft of learning materials were developed based on feedback from the composers commissioned to create music with early versions of the toolkit. Their feedback on what was confusing about the initial versions of the tools, resources, examples, etc. led to a second draft of learning materials which were then used in over thirty workshops around the world with computer musicians from various backgrounds. The feedback from these workshop participants further refined the learning materials and have led to the broad ecosystem of learning resources now found as part of the FluCoMa Toolkit.

1.2 Tiered Learning Resources

Because different learners will desire different degrees of fluency with these topics, we have tried to tier the learning resources accordingly. The most proximal resources (such as environment native help files) provide a working understanding of what an algorithm does alongside musical examples of how it might be used. Additional resources (such as the internet resources) provide a deeper understanding that might satisfy one's curiosity and/or build an intuition of what is happening "under the hood", both of which can enable a more informed manipulation of the Toolkit. When appropriate, we also link to the white paper that describes the algorithm from an engineering perspective, should the learner wish to pursue that amount of technical detail.

In some cases, potential paths of pursuing additional resources are more extensive and less

linear. This is especially true for the more complex tools in FluCoMa, such as neural networks which have multiple dedicated web resources with varying degrees of technical information, any one of which might come after an initial introduction, but when taken all together encompass the degree of fluency we propose for our learners and users. Learners who are eager to jump to manipulating the many neural network hyper-parameters might jump to MLP Parameters, while a user who needs a little more time absorbing how a neural network works might opt for MLP Training. (Also see MLPRegressor, MLPClassifier, and Training-Testing Split)

Tiered learning resources allow the learner to pursue knowledge as far as they deem appropriate to feed their creative practice in a given moment. Providing the learner what the *need* to know *when* they *need* to know it enables them to stay focused on a creative idea and not become overwhelmed by what could be a very large body of knowledge with a steep learning curve. This sensitivity to the relationship between creative pursuits and technical knowledge reflects earlier findings of FluCoMa outlined as " 'Techno-Fluency' and 'Divergence' ". By offering signposts and links to further resources, the user knows where to keep learning if necessary in the moment, or, in the future if they decide to continue exploring.

1.3 Music-Forward Resources

Because of the specificity of our target learner (a creative coding musician), we have always tried to keep our learning materials and examples musically oriented (as can be seen below). We aim to have the help files and example code make sound in a creative way. When possible, we offer pedagogical examples and thought experiments that will feel familiar and relevant to our learner such as instrument samples, drum hits, MIDI notes, synthesizer settings, measures of frequency and loudness, etc. We hope this not only explains an object and its interface, but also provides some copy-and-paste code to get started quickly, and generally gets the creative juices flowing while a user is engaging in the learning process.

2 Ecosystem of Learning Materials

2.1 Creative Coding Environment Materials

Each creative coding environment (CCE) supported by FluCoMa (Max, SuperCollider, and Pure Data) has a native system for offering reference materials. In Max and Pure Data a "help file" provides annotated examples, while a "reference" offers additional description and detail about parameters. In SuperCollider all of this information is contained in one "help file" document. Despite this difference in interface, we have strived to keep the CCE-based FluCoMa materials similar across all three environments. This is enabled, in part, by the shared documents used to render the reference materials for all three CCEs.

The FluCoMa materials provided natively in the CCE are often the learner's first engagement with our supporting materials. Therefore, the information provided is intended to provide the learner/user a working understanding of what an object does, how it might be used

musically through a sound-making example, and if appropriate, how it interfaces with other FluCoMa objects. This information will hopefully provide a learner some motivation for exploring an object and the amount of knowledge necessary to do so. Each resource in the CCE contains a link to the corresponding web reference if the learner wishes to pursue a deeper understanding of the object.

The example code provided has been created to be as similar as possible across the three CCEs while keeping idiomatic to each environment. One goal of this is to enable cross-environment communication and knowledge sharing. Users in different CCEs are able to discuss the technical and musical facets of a shared FluCoMa example. It is also possible that this allows referencing help files and example code to a classroom containing a diversity of coding environment users. Lastly, this keeps all three CCEs on an equal status, preventing any potential inference of CCE preference within the FluCoMa Toolkit.

2.2 Web Reference

Every object in FluCoMa (except CCE-specific objects) has a web reference found at learn.flucoma.org. Because the materials that appear natively in the CCEs link to the web, the web references are considered to be a secondary resource. The goal of the web references is to offer more detailed descriptions of how an algorithm is working "under the hood". This may be useful for satisfying curious learners, or it may build a better intuition of what a particular FluCoMa object might be used for musically. In either case, it could lead to a more informed, and therefore potentially more advanced, use of the object.

Many of the web references have interactive explanations that allow a learner to "use" the algorithm in the browser. Not only can these be used by individual learners, we have also found that these are very useful for explanations in the course of teaching. For example, when creating a KDTree for the first time during a code-along class, using the interactive page helps give learners a visual sense of what is happening (especially if the KDTree is about to be used with the plotter). The MFCC reference has an interactive explanation that invites a learner (or a teacher during demonstration) to step through a series of interactions that build intuition about MFCCs.

We imagine the web references to be used as solo-learning resources, in parallel with class assignments, teaching demonstrations, and/or useful reminders.

2.3 Learn Articles

FluCoMa's learning website (learn.flucoma.org) also contains many articles about topics that may not fit in a single web reference page. These articles may arise as a tertiary step in a learners path and are likely to be encountered after the CCE materials and web reference.

There are a few varieties of articles found in this category:

- explainers specific to a single FluCoMa object but offer a depth of knowledge about the internal algorithms that would be outside the scope of a web reference page, such as Audio Decomposition using BufNMF.

- knowledge about data science that is useful for using many of the FluCoMa objects, such as Distribution and Histograms and Why Scale? Distance as Similarity
- common workflows using the toolkit, such as Batch Processing with FluCoMa

These articles are not necessarily designed to be consumed in series as part of a sequence of learning (although some could be used this way). Instead, each article is made to be approached by a learner (or guided by a teacher) at a particular point in the learning process and revisited as necessary. The idea for many of these articles arose in direct response to questions asked by workshop participants and therefore are designed to answer or provide context to common questions asked by learners. When designing a curriculum (such as the one below), many of these articles would support student learning for different topics in a course.

2.4 Explore Articles

In addition to the web reference and learn articles, the website has many additional materials surrounding artistic uses of the toolkit. These include example artworks, interviews with creative coders, and musicological articles that offer in-depth analysis and example patches of music made with FluCoMa. All of these can be used as context, examples, and inspiration for learners. These can also be used as entry points, as many learners will find the work produced using FluCoMa or the musical ideas expressed in these articles inspiring and motivating.

2.5 Discourse

The international community of FluCoMa users primarily communicates through the Discourse online discussion forum. Any learner (or user) of FluCoMa should be a member of the Discourse as it is an excellent place to converse with like-minded artists. Learners may find the search functionality very useful to see if others have already asked and answered a question they have. The community is very positive and supportive, so it is a safe place to ask all kinds of questions. In addition to the thread for *Usage Questions*, there are also threads for *Code Sharing*, *Learning Resources*, and *Interesting Links*, making it another place for learners to browse for examples, inspiration, and knowledge.

2.6 Inter-connectivity of Resources

As described above, the FluCoMa learning resources are generally tiered to offer learners the degree of detail needed at a given moment to pursue a creative idea. One way in which our tiered approach is executed is cross referencing the different learning resources. The help files in each creative coding environment (CCE) link to their respective web reference, from which a learner can be pointed to many more resources. The web reference, learn articles, and explore articles all cross-link with each other so that a learner reading a web reference might discover an explore article about a musician's use of an object, and from there discover a learn article that might help them pursue the creative use they just learned about, etc. Different learners will need different degrees of technical specificity, inspiration, and modes of engagement at different times. We hope that setting someone "loose" on the website will

enable them to find uses of FluCoMa that are meaningful to them as well as the knowledge needed to support them.

2.6.1 CCE Specific Objects

Pedagogues should be aware that there are a few objects in FluCoMa that are CCE specific in name and/or implementation because of CCE differences. These objects diverge from FluCoMa's philosophy of cross-environment parity in order to ensure workflows in the toolkit are idiomatic and *Fluid* for beginner and expert users.

Max	Pure Data	SuperCollider	
fluid.list2buf	(native)	FluidKrToBuf	*writing control information into a buffer*
fluid.buf2list	(native)	FluidBufToKr	*reading control information out of a buffer*
fluid.plotter	fluid.plotter	FluidPlotter	*similar functionality and syntax but divergent implementation*
fluid.waveform	fluid.waveform	FluidWaveform	*similar functionality and syntax but divergent implementation*

3 "101" Tutorials

After teaching numerous workshops we have identified a few class plans that work well to get new learners excited and making sound as well as laying a foundation of facility with FluCoMa to support further activities and/or self-guided learning. A few of these lesson plans are described briefly below.

3.1 MLPRegressor

Watch a Video Tutorial of this Lesson Plan

- in Max
- in SuperCollider

Often the first activity we engage with learners is to build a neural network that performs regression to control a synthesizer. This gets learners making sound quickly and uses part of the toolkit that is often quite exciting for newcomers to machine learning to engage with (neural networks). This activity takes any where from 40-90 minutes depending on the class of learners. Here is a brief outline of the lesson plan:

1. Share a real world example (including watching a performance excerpt) of why someone might want to use a system like this.
2. Using a slides presentation step through how we will be collecting training data and training the neural network, including some intuition about how the training process works.
3. Open up the CCE of choice and demonstrate a completed version of the instrument we're about to code.

4. Code the instrument together, as a code-along, starting from a "starter patch" that has a few key items already in place:
    - a synthesizer to control
    - a 2D control space to use as input to the neural network
    - a MLPRegressor object with many arguments already specified
5. Let the learners play with the instrument (and augment it in their own way).

    The starter patch is important here so that we don't spend too much time doing CCE-specific boiler plate code but instead get right into using FluCoMa. It also ensures that learners have a synthesizer to make sound with right away when the code-along is complete. Because there are many arguments to the MLPRegressor object and each of them can require a fair amount of explanation to use well–and in coordination with each other–we've chosen to provide the arguments to the MLPRegressor programmed into the starter patch. During the lesson we tell the learners that these arguments can be explored further in a future lesson and/or on our learn article.

The extensions of this activity are to:

1. Practice training the neural network.
    - clearing the neural network and retraining
    - training the neural network to a less-low loss value to see if a less-fit model is more (or less, or differently) musically expressive
    - delete the input data and choose new input points to pair with the synthesis parameters already chosen
    - delete all the data and create a whole new training
2. Attach a different sound-making algorithm to the output of the neural network
    - granular synthesis / sample playback
    - frequency modulation
    - VST
    - we have often encouraged learners to bring a sound-making algorithm of their design to the workshop to connect as a next step to this activity
3. Attach a different type of controller to the input of the neural network. This might be something like:
    - multiple parameters on TouchOSC
    - MIDI controller
    - leap motion
    - wearable device
    - pixel information from a camera (perhaps using Jitter in Max)

Not only does this activity quickly provide learners with a machine learning instrument that is very extensible, it also introduces some of the key elements of FluCoMa:

1. DataSets
2. Buffer interfacing
    - fluid.buf2list~ and fluid.list2buf~ for Max
    - FluidBufToKr and FluidKrToBuf for SuperCollider

    - cloning fluid.multiarray as 'multichannel buffers' for Pure Data
3. Using artist-created DataSets
4. Aesthetic evaluation of results
5. Iterative trial-and-error workflows with machine learning algorithms

3.1.1 Classifier Extension

After completing the MLPRegressor activity, one common extension is to do an activity with the MLPClassifier. Depending on the group of learners and how much time is available, sometimes this would only include opening up the classifier demonstration file and doing a quick training and testing, along the way relating it to what was just done with the MLPRegressor activity. If time allowed and there is interest we performed a more involved activity similar to the MLPRegressor: we train the MLPClassifier to classify timbres, distinguishing between provided trombone and oboe recordings.

Watch a Video Tutorial of this Lesson Plan

- in SuperCollider
- in Max

1. Share a real world example (including watching a performance excerpt, this one has a before and after) of why someone might want to use a system like this.
2. Using a slides presentation step through how we will be collecting training data and training the neural network, including some intuition about how the training process works.
3. Open up the CCE of choice and demonstrate a completed version of what we're about to code.
4. Code-along, starting from a "starter patch" that has a few key items already in place:
    - the sound files containing different trombone and oboe sounds
    - a MLPClassifier object with many arguments already specified
5. Let the learners explore classifying some of their own sounds

3.2 2D Sound Browser

Watch Video Tutorials of this Sequence

- in Max
- in SuperCollider

Many of the common practices with FluCoMa involve:

1. starting from a large sound file as the corpus
2. using a slicer object to divide that file into audio slices
3. analyzing each slice using audio descriptors
4. plotting slices in a two dimensional space to perform with or build understanding about the audio that has been analyzed.

Because these steps can be applied in various workflows and branch towards many diverse projects, we've created a learning sequence that steps through each, building on them in

order. This is a longer sequence of activities that usually takes at least 2 hours, often more. Because it is a lot of information and may not be possible to get through it all with limited time, or because it would be better spread out over multiple days, we've identified a few stopping points, "off ramps", and extensions that can surround this activity to suit it to the group of learners present. The sequence of activities with these variations is outlined below.

As with the neural network "101" tutorials, this one also often begins with an example of a creative application of the workflow. This Keynote Presentation (which includes a few videos) has been used in describing this project. Some corresponding videos can also be found at the links below, as well as in the last section of the Explore Article, Levels of Translation.

- Full Work
- Show Data Processing Steps
- UMAP in 1 Dimension Path & Video
- Travelling Sales Person Path

Lesson Plan:

1. Load the sound file we begin with into a buffer ("Nicol-LoopE-M.wav", which comes with the FluCoMa Toolkit).

Slicing

2. Use the output of a real time slicer (which are audio rate impulses) to listen to where the slicer is finding slice points in the buffer. We usually begin with either OnsetSlice or NoveltySlice. If time allows, perhaps spend some time tweaking the parameters and listening to how it changes the slice points.
3. Use the non-real time version of the slicer (BufOnsetSlice or BufNoveltySlice) to collect the slice points into a buffer of sample indices.
4. Code a way to play back these slices (audio between adjacent slice points) by reading out of the slice points buffer.
    - This moment offers a potential stopping point, where learners can now use these slice points and player to make some sounds, see how different slicer parameters change the slices, and/or try out different slicer algorithms.

Analysis

5. Use BufSpectralShape and BufStats to retrieve the mean spectral centroid of each slice.
    - Sometimes we then use this analysis to sort the slices from lowest to highest mean centroid and listen to them in that order. This *mostly* works but also raises an important awareness of why some slices are perceived to be out of place: critical reflection on what a spectral centroid represents (which we mostly expect users to have some idea of), why using the mean as a statistical summary can be misleading, and how/why there could be multiple sounds in a single slice.
    - After sorting, one could create a stopping point by encouraging users to use some of their own sounds for slicing, analysis, sorting, or playback. They should experiment changing parameters in each of these steps to see how it changes

the whole process (including trying different statistical summaries or different analyses from BufSpectralShape).

6. Use BufLoudness and BufStats to retrieve the mean loudness of each slice.
7. Combine the mean spectral centroid and mean loudness into one buffer and use this to add a data point for each sound slice to a DataSet.

Plotting

7. Plot the 2D DataSet using the FluCoMa Plotter.
   - At first none of the points will be visible because Plotter defaults to displaying a range of 0-1 in each dimension. This acts as a good moment to raise some awareness around "know your data". Notice that these are in very different ranges.
8. Adjust the Plotter ranges to see the data.

KDTree

9. Notice that the Plotter outputs the mouse position *in the ranges specified* by the user.
10. Fit a KDTree to the analyses DataSet and use it to find the point nearest the mouse, highlight that point, and play back that slice.
    - When clicking and dragging on the plotter, the learner will notice that the highlighted point doesn't seem to track with the mouse as expected. This is because the visual perception of nearness in the Plotter doesn't correspond to the greatly mismatched scales in the two dimensions. This is another good opportunity to raise awareness about "know your data", particularly through the Why Scale? Distance as Similarity web page. The content on this page was first expressed as slide presentation, which could be used instead.
11. Normalize the DataSet so it it can be displayed in Plotter's default normalized space and the KDTree lookup of the mouse position tracks better.
    - This is a potential stopping point where users can, again, include their own sounds and tweak parameters. Learners could organize into small groups and give short improvised performances using idiosyncratic sounds with this performance instrument.

MFCC Analysis and UMAP Dimensionality Reduction

This next set of steps tends to work will with a more diverse corpus of sounds, so before moving on one might replace the buffer that has the drump loop with a buffer of many more sounds–perhaps all *the sounds in FluCoMa's example sounds folder.*

12. Replace the BufSpectralShape and BufLoudness analyses with one BufMFCC analysis using 13 coefficients and startCoeff = 1.
    - We find that questions like, "But what *is* an MFCC?", are very common. Depending on how much time is available, this may be a good opportunity to explain MFCCs more in depth using the web reference and/or interactive explanation.
13. Use UMAP to reduce the 13 dimensional DataSet to 2 dimensions. Use the UMAP web reference to provide some understanding of what UMAP does.

14. Normalize the output of UMAP.
15. Plot it in Plotter.
    - This moment offers a good opportunity to discuss two important points of UMAP: **numNeighbours** and **minDist**. Users should adjust these, recompute UMAP, and replot. As with many of these iterative tweaks, remind them that the goal is to make music so their assessment criteria can simply be whatever settings feel most creatively useful!

Up to this point is the main sequence of steps that we go through with many learners. The following section that adds Concatenative Synthesis is sometimes used as a follow-up. During workshops this was usually after lunch or the next day.

Concatenative Synthesis

16. Delete the Plotter and UMAP, leaving the slicing –> DataSet pipeline.
17. Analyze a sound file (the "target") that is not in the corpus and store in a separate DataSet.
18. Use a KDTree to find which slice in the corpus is closest to each slice in the target.
19. Play back the discovered corpus slices in place of the target slices, "resynthesizing" the target sound by "concatenating" slices from the corpus.
    - This offers a good moment to compare the different scalers in FluCoMa. Scale the data before fitting the KDTree (this **fit** scaler will also need to be used on the target slice data point before query). Use different scalers and see if/how it changes what the nearest neighbors are and therefore what the resulting sound is.

3.3 Introduction to NMF

The family of nonnegative matrix factorization objects in FluCoMa can be daunting to wrap one's head around. We've identified a sequence of introducing the objects and their features that seems to work well for revealing the power these objects offer without overwhelming or confusing learners. Before/in parallel with considering the sequence below it will be good to familiarize oneself with the NMF objects in FluCoMa (BufNMF, NMFFilter, NMFMatch, BufNMFSeed, BufNMFCross, and NMFMorph).

The steps outlined below pretty closely follow these two Learn articles:

- Audio Decomposition using BufNMF
- Seeding BufNMF with Bases and Activations

They can also be found in these example files:

- NMF Examples for Max
- NMF Examples for SuperCollider

1. Listen to the Nicol drump loop (that comes with the FluCoMa Toolkit). Look at the spectrogram and identify where in the spectrum and in time the different drum instruments (roughly) exist. Notice that because there is a lot of overlap both in

spectral space and in time, trying to "decompose" this buffer using a slicer or a spectral filter wouldn't be very successful. There will be many components that would contains *some* of more than one drum instrument's sound.

2. Use BufNMF to decompose the drum loop into two components and listen to what that returns. Make sure the learners know that that the extent of information that is given to the NMF algorithm is the mono source drum loop and the number "two".
3. Increase the number of components to three and ask the learners what they think will happen. Many will predict that NMF will compose the mono drum loop into three components containing the (1) snare drum sounds, (2) hi-hat sounds, and (3) kick drum sounds. It turns out they are correct. (If this is not the result, run it a few more times, it surely will be soon! This unpredictability can feed into some nuances learned later.)
4. Explain that what the algorithm is trying to do is create spectral templates that can (as well as possible) describe or summarize the different kinds of spectral frames it finds in the buffer. It will create as many spectral templates as the number of components requested. It looks for parts of the spectrum that tend to occur together in time and combines those parts into the same spectral template so that template can be used to describe lots of the parts of the buffer.
    - One good place to look for this is the partials of the resonant tone of the snare drum. These spectral peaks are a salient feature to notice always occur together.
5. At this point introduce the concept of bases. In NMF terminology this is what the spectral templates are called. Using NMFFilter play some pink noise "through" these bases to hear how it filters the noise according to spectra that we can recognize from the source drum loop.
6. Paired with each basis is an activation, which represents how present (or how "active") each one of the bases is throughout the audio buffer. Because these exist in buffers, play back the values in these buffers and use them as amplitude envelopes to control the loudness of pink noise. Notice that we recognize the rhythms of the different drum instruments.
    - One nice thing to do here is to mix and match bases and activations. Use NMFFilter with a basis from one drum instrument to filter pink noise while using an activation from a different drum instruments to control the loudness.
    - Also, pair the correct basis and activation from each drum hit to "resynthesize" the original instrument using pink noise as a source. Explain that this is kind of how BufNMF actually performs resynthesis, but using the reconstructed magnitude spectrum of the original (and the phases from the original). (Of course no pink noise is used.) Playing all three faux-pink noise resynthesized instruments at the same is surprisingly convincing and compelling to listen to.

This is one potential stopping point for this sequence. Learners will have a sense of why the might want to use BufNMF for audio decomposition as well as some intuition for how it is actually operating. In our workshops we've often gone farther than this point, as outlined below.

NMFFilter

7. Now we turn more properly towards NMFFilter, not as tool to demonstrate bases but as its own object. Using the bases derived in step 3, play the real time audio of the original drum loop through NMFFilter. Notice how what we hear is very similar to the three components we heard in step 3.
    - What's nice about getting these components in real time like this is that NMFFilter isn't just filtering through a static spectral filter. Instead it is performing 10 steps on non-negative matrix factorization on each incoming spectral frame starting from the bases provided (as seeds, which will be discussed soon). This means that basis derived in step 3 are further adjusted to match even more closely to each spectral frame as it comes in–often providing *even more* separation in the components that provided by BufNMF. (The bases begin fresh on each spectral frame, so their modifications are not cumulative.)
    - Point out that one potentially cool use of this is to use a recording of a drummer (that could be made, even during sound check) to create bases for use with NMFFilter in real time performance. Using the separate audio streams
8. Next, rather than sending the audio stream of the original drum loop through NMFFilter, now send a different audio stream that also contains some drums. In this example we'll use the "Tremblay-BeatRemember.wav" file (it is stereo, so just using one channel of it will suffice). Notice that the audio streams we get out of NMFFilter still generally correspond to the drum instruments we heard before, but also there's a lot more "bleed", where other sounds are also coming through. This is because NMFFilter needs to account for everything in the incoming spectrum–it all needs to be send *somewhere*.

NMFMatch

9. Using the bases derived in step 3, send the real time audio of the original drum loop through NMFMatch. Note that it outputs control streams of values–one for each basis. These are "real time activations", essentially indicating how present (or "active") each of the bases is for each spectral frame in the real time audio signal. Visualize these so the learners can see how they respond to the audio as they hear it. The output of NMFMatch could be used to trigger some effects processing or as a kind of quasi-classifier by putting different thresholds on these values.
10. Again use the "Tremblay-BeatRemember.wav" file, this time send the audio stream to NMFMatch and watch the control outputs. See how the instruments in this recording still somewhat correspond to the bases created from the drum loop.
11. One could imagine using these tools to find and/or decompose specific sound objects in large files such as field recordings. Explain that one caveat with BufNMF is that as the buffer gets bigger, the computation time gets *exponentially* bigger (if appropriate for the audience, one can share that the complexity is O(N^2)). This means that it is not feasible fully decompose large files using BufNMF. However, if one knows what they're looking for they can use a subsection of a file to create appropriate bases and then use NMFFilter and/or NMFMatch on a large very large file. The example here is:
    - Show the "field recording" sound file "Tremblay-BaB-SoundscapeGolcarWithDog.wav" (it is stereo, so using just one channel will be sufficient). For the purposes of

demonstration we'll imagine that it's many hours long, and therefore not feasible to provide to BufNMF.

- If we wanted to go through this "many-hours-long" sound file and find all the dog barks, what we can do is, first decompose just the first few seconds (which does contain dog barks) into 2 components. What we're hoping will happen is that we'll end up with 2 bases: (1) a spectral template of the dog bark and (2) a spectral template of "everything else" (this will work well with these examples–with other soundfiles, it might be necessary or appropriate to do some checking / tweaking, perhaps with seeding, to make sure you have a spectral template of what you're seeking).
- Providing the spectral template basis of the dog bark to NMFMatch, run the audio stream of the entire file through NMFMatch to get an activation curve of the dog-bark spectral template for the entire file! This can all be done in real-time during the demonstration.

This is another potential stopping point. In the workshops we've taught, this was a common place to end our time on NMF, or at least pause a moment for the many questions and excitement the learners will have. If there is time and interest to continue, we've also included some demonstration on seeding NMF (below).

Seeding NMF

The sequence below generally follows this Learn Article.

1. Demonstrate that running BufNMF multiple times in a row will often result in similar components (snare drum, kick drum, & hi-hat), but the order in which the occur is unpredictable.
    - This is because the non-negative matrix factorization process begins from a randomized state, so although it often finds similar solutions, the order of the components will be random. (If appropriate you can note that the algorithm uses stochastic gradient descent.)
2. In order to gain some predictability over how the NMF process will decompose a buffer, we'll seed some "hand-made" bases to BufNMF (set **basesMode** = 1) that are *roughly* in the shape of the spectra that we hope to end up with for our spectral templates (kick drum, then snare drum, then hi-hat). The result are components similar to what we got before, but now we can be much more confident that they will be in an order we can predict: the order of the rough spectra we seeded (kick drum, snare drum, and hi-hat).
3. Instead of "hand-making" bases to seed with, we might use bases that are themselves the result of a BufNMF analysis. To create these bases, we'll take them from the result of a BufNMF process with an audio buffer of eight 1-second sawtooth wave tones that make a major scale as the source.
4. Then using these bases as seed we'll use BufNMF to decompose a short melodic fragment (that uses the tones from this major scale). The result will be eight components: each one based on one of the spectral templates from the sawtooth bases. BufNMF will have one component per note of the major scale that it was seeded with.

5. One unfortunate circumstance is that the bases BufNMF returns in step 3 will not be in the order of the scale. They'll be in a random order (as explained in step 1). In order to ensure that they will be in order we will seed activations when creating the bases of the sawtooth tones. First, we'll create a seed for activations that show *where in time* (as activations do) the different synth tones occur. Then when using these seed activations on BufNMF with the eight 1-second sawtooth wave tones as the source, the resulting bases of the different scale notes will be in the order expected.

   Understanding this process is best done by also seeing the graphics on the Seeding NMF Learn Article.

6. Now using these ordered bases to seed a BufNMF decomposition of the melodic fragment will yield components that are in the same order as the bases: ordered according to the scale.

4 Common Learning Challenges & Strategies

Over the course of teaching many workshops, we observed some common challenges for FluCoMa learners. Below are a few of the challenges we found and some strategies for approaching them pedagogically.

4.1 New Ways of Using Buffers

FluCoMa uses buffers to store all kinds of data, not just audio. This may be new for learners who are used to using buffers *only* for holding audio, and may even associate the two as a single concept ("buffer == audio"). This becomes increasingly complicated when we begin to manipulate the data in buffers as arrays or matrices. A few strategies we've developed for help learners feel more comfortable with buffers include:

Initial Encounter

The first moment at which a learner is asked to view a buffer in a new way is often when we allocate a buffer to hold a data point. If the data point has just 2 dimensions (such as with the MLPRegressor activity), we will allocate the buffer with only 2 frames (in Max: @samps 2; in SuperCollider: Buffer.alloc(s,2)). At this point we try to offer something like the following: "The way we most often use buffers in [this CCE] is to hold audio samples which very commonly have 44,100 samples per second, so our buffers could have many tens or hundreds of thousands of values in them. Because I know this buffer is only going to need to hold two samples, I'll just allocate it to have two samples."

Holding Analyses

Once we start writing audio analyses into buffers (with the feature argument), learners often have a hard time keep track of the structure of the buffers (What do the channels represent? How many are there? What do the frames represent? How many are there?). We found that offering CCE-agnostic charts of the "shape" of the buffer is very helpful for giving learners a mental model to hold in their mind.

It's also useful to point out that for buffers that hold audio analyses, the frames (or what we sometimes refer to as the "x axis" in reference to the charts above) is still a time series, just like audio is, but now it's not a time series of voltages (as in audio), it's a time series of descriptors (such as spectral centroid).

Because each frame represents an FFT frame from the STFT analysis the sample rate would not be a usual 44,100 samples per second, but a much lower rate of frames per second. FluCoMa buffer processors (such as the audio descriptor analyzers) set the *sample rate* of these buffers appropriately. If an audio buffer is analyzed with a **hopSize** of 512 samples, the **features** buffer that the analyses get written into will have a sample rate of 86.1328125 frames per second. If the values in that buffer are read back at that rate, they will correspond in time (be synchronized with) to the audio on which the analysis was based.

Manipulating & Copying Data

Often it is necessary to manipulate the data in a buffer, such as pick out values from certain channels and/or frames and copy them to another buffer. In order to provide some test and check interactivity to build fluency with these operations, the appropriate web references have interactive GUIs for practicing: - BufSelect - BufSelectEvery - BufFlatten - BufCompose - BufScale

Why Buffers

It also may be of interest to learners to hear the explanation of *why* buffers are used in this way. FluCoMa users buffers in this way for a few reasons:

- The notion of "buffer" is shared across all three CCEs that FluCoMa supports. This allows for shared syntax and usage of objects across all three environments.
- In all three CCEs, buffers are accessed at the lower levels of code allowing for:
  - Faster processing by interfacing directly with the C++ code.
  - Simpler implementation of functions across all three CCEs because all three environments share the same C++ code base for all of the audio analysis, buffer processing, and algorithmic computing.
- Having data in buffers allows for it to be more flexibly accessed and used in other parts of the CCE. For example because the MLPRegressor writes predictions into a buffer, it's possible to be predicting wavetable shapes directly into a buffer that is simultaneously being read out of (thanks to user Timo Hoogland for this example!).

4.2 Advanced Neural Networks

Learns often follow up the neural network 101 activity with questions about the hyper-parameters (which we call parameters or arguments) of the MLP. In shorter workshops (two days or fewer) we have felt that it's not enough time to delve into this with enough depth to make it understood and *useable* for the participants, so we've directed them towards our web resources on the topic. In longer workshops (3 or more days) we have take time later in the week (day 3 or 4) after the "101" activity to unpack many more ideas and strategies about the MLP objects.

Web Resources

- Neural Network Training is an overview of how neural networks learn. It is intended to be a resource for the slightly more curious or for those who would benefit from gaining a little more intuition about what is going on "under the hood". Much of this article is expressed in the "101" activity.
- Neural Network Parameters goes through each parameter or argument in the MLP objects and gives a more thorough description of what it is, why one might adjust it, and what a generally reasonable starting place is. This is often where we direct learners who ask about these parameters when we don't have time to unpack them during a workshop.
- Training and Testing Data describes why it can be important to validate the results of a trained MLP. It explains why one would go about validating a model, what to look out for, what certain results might mean, and what would might do to improve a model.

Building Intuition about How Neural Networks Learn

The sequence of explanation that we've used for both the MLPRegressor and MLPClassifier seems to work quite well for giving learners intuition about the training process of an MLP. These explanations can be seen at the beginning of the the MLPRegressor and MLPClassifier tutorial videos.

MLP Parameters

In addition to pointing at the Neural Network Parameters article, when we have had time in workshops, we've also allocated some time to explaining the parameters in more detail. A very useful site for explaining and playing with the momentum parameter can be found on distill.

Visualizing a MLP

One might notice in many of the resources above that there are node-and-edge graphs of MLP architectures. We found that these are very useful for learners to concretize a few facets of MLPs: (1) "feed-forward", (2) "back-propagation", (3) "fully-connected layers", (4) numbers of hidden layers and nodes, (5) total number of parameters in an architecture, and more. We also learned that it is important to have a visual representation of the architecture that is *actually used* in the activity. We use a very basic graphviz script to generate these graphs.

4.3 Threading

4.3.1 SC

In SuperCollider, many FluCoMa workflows rely heavily on the server's OSC queue. It will be important for FluCoMa learners to become familiar with how the server processes the OSC messages it receives.

processBlocking

In particular, when performing many buffer operations, a common workflow is to use the processBlocking call for all of the operations which will place each operation in the server OSC queue. This allows for very fast buffer processing because the the operations are all lined up in order and the server moves on to the next one as soon as it completes with the previous (all without any unnecessary sync with the language). One way in which this often confuses learners is that if these operations are inside of a loop (that is iterating over slice points perhaps), and the code is printing *which* slice is currently being analyzed (see below), it will report that it has "completed analyzing" all the slices, when in fact all this suggests is that all of the processBlocking operations have been sent to the server.

```
fa.doAdjacentPairs{
 arg start, end, i;
 var num = end - start;

 start.postln;
 end.postln;
 i.postln;

 // analyze a sound slice for spectral centroid
 FluidBufSpectralShape.processBlocking(s,~src,start,num,features:spec,select:[\centroid]);

 // get the mean centroid for this sounds slice
 FluidBufStats.processBlocking(s,spec,stats:stats,select:[\mean]);

 // copy the mean spectral centroid to the appropriate index (destStartFrame) of
 // the buffer called meancentroids
 FluidBufCompose.processBlocking(s,stats,destination:meancentroids,destStartFrame:i);

 "completed analyzing slice %".format(i).postln;
};
```

One good way of clarifying this for learners is to enforce a sync between the language and the server every 100 slices or so using a line of code like if((i % 100) == 99){ s.sync }. This will allow the code to post "completed" for every 100 slices, but wait until those slices are completed before posting anything else. This clarifies the speed at which the analyses are actually happening, while adding a trivial amount of processing time to the overall analysis.

4.4 Stateful Objects

Many of the FluCoMa data objects hold some state. For example after calling fit (or fitTransform) on a Normalize object, it holds the minimum and maximum value of each dimension in the fitted DataSet so that it can scale future transform calls appropriately. This interface design was based on many of the data processing objects in the Python sci-kit learn package. We found that some FluCoMa learners find it challenging to conceptualize or remember that certain objects are holding a state that will they will need to call upon later. One added feature that may help with conceptualizing objects in this way is the option to

name objects in Max and Pure Data (SuperCollider nominally uses variable names to identify objects).

Named objects may help learners remember that certain objects hold state because they have a sense of it being a non-generic, task-specific object, such as a Normalize object called "norm-pre-pca". This gives it a special sense of purpose and an indicator of what state it holds and where in data processing one would call upon that state.

4.5 Fourier Transform & STFT

As with many audio tasks, the STFT is central to much of how a learner interacts with FluComa. Many times learners can do exciting things, learn a lot about the toolkit, and make great music without reflecting on the FFT processes happening "under the hood". We have found however, that for many of the algorithms in FluCoMa (such as AudioTransport, Sines, and many more), adjusting STFT settings (**windowSize**, **hopSize**, and **fftSize**) has an important aesthetic impact on the results, and therefore we suggest that it is important to understand what impact these parameters have.

There exists a wealth of resources on the internet for learners to build fluency with the Fourier Transform, so we didn't feel the need to recreate many of these learning tools. We have however, curated a small set of ideas that will be important for FluCoMa learners to consider as their approaching the toolkit, which exist on these two pages:

- Fourier Transform
- BufSTFT

4.6 Navigating Human & Machine Assumptions

One of the most challenging conceptual hurdles for FluCoMa learners is to reconcile the differences between the way machines and humans listen. What perceptually might seem obvious to a human listener can be very challenging for a machine to discern. Newcomers to machine listening often set out to perform a task making many assumptions about how a system will work, what data they will use, and how they will compute a result, not realizing that what they *think* they're telling the machine or asking it to compute is quite different from what it will give back in return.

Machine Listening: Pitch

One activity that has been quite successful is a simple "listening test". We ask a class of learners to sing the pitch of this sound file. Most listeners will sing the right pitch class but down a few octaves from the actual frequency. The first thing to point out is that they were only singing the pitch from the part of the sound file that was *most* pitched. They didn't even attempt to sing the "pitch" during the scratchy parts. This chart shows the result of a Pitch analysis on the buffer. One can see where the pitch is stable (the parts that the listeners sung), but also that there is a lot more "pitch" analysis there. The machine listens to all of it (and reports back on all of it). This is because as humans we're constantly engaged in multi-modal listening and switching between different ways of perceiving sound,

depending on which seems appropriate or useful at a given moment. When the sound file is making scratchy sounds, we as humans don't even register it as a "pitch" to sing, but a machine does.

Another useful outcome of this activity can be acknowledged when discussing Distance as Similarity. Most listeners will sing the correct pitch class but down a few octaves. For these listeners, being 12 half steps lower (a distance of 12) is closer than being 1 half step lower (a distance of one). This is again a recognition that what humans might assume to be *similar* a machine might not (in this case using Chroma analysis may help align human and machine listening).

Statistics

Many workflows in FluCoMa require the use of statical summaries of sound slices, real time audio analyses, or whole DataSets. It is important for learners to develop a sense of what tools are available and why one might reach for one statistical summary rather than another.

BufStats is perhaps the most commonly used object in FluCoMa and therefore has a somewhat involved reference page. Many of the statistics available might be familiar to learners (mean, median, minimum, and maximum), while others might be new (standard deviation, skewness, kurtosis, derivatives, etc.). The "101" tutorials and many of the other workflows don't use these more advanced statistical tools and therefore in teaching FluCoMa, we've opted to only focus on a few. During the "101" tutorials, many moments arise that are useful for reflecting on the statistical analyses being used and how they affect the sonic results being heard. A few examples are reflecting on what it means to have an average spectral centroid of a sound slice and using the maximum value of an analysis rather than the mean.

BufStats has many more features that are somewhat less explored and probably not appropriate for learners just getting acquainted with FluCoMa. There are few Learn Articles that cover these topics in musicianly ways including Weighting Stats and Outliers. It is also important to convey to learners, as is stated on the BufStats page,

> *While it can be difficult to discern how to use some of these analyses practically (i.e., what does the kurtosis of the first derivative of spectral centroid indicate musically?), these statistical summaries can sometimes represent differences between analyses that dimensionality reduction and machine learning algorithms can pick up on. Including these statistical descriptions in training or analysis may provide better distinction between data points.*

"Know Your Data"

As with all data science and machine learning, understanding what data represents, what it can tell you (and more importantly what it *can't*), and what transformations *do* to data is essential. It is continuously important for FluCoMa learners to reflect on their data. There are a few visualization tools that are very important for users to get comfortable with including the Waveform object (fluid.waveform~ in Max and FluidWaveform in SuperCollider). While teaching, these tools should be used whenever possible to help learners understand

the data processing that is happening and get them in the habit of visually checking on their data regularly to build understanding of what they're data represents.

Many machine learning algorithms make various assumptions about data (similar to now humans make assumptions about sound and machine listening). One of these assumptions is that the dimensions in a DataSet are identically distributed, often Gaussian distributed. It can be important to know how one's data is distributed and relatively easy to check on using a histogram. Our Distribution and Histograms page gives some examples of different kinds of distributions, what they mean, and some example code to check on a distribution using a histogram.

Scalers & Distance as Similarity

Another important concept for learners to understand is how measures of distance impact perceptions and assumptions about similarity. Once the machine has "listened" and a statistical summary has been computed, a next step is often to compare data points by computing the distance between them (most of the FluCoMa tools use Euclidean Distance). Computing distance very quickly makes questions about ranges and scaling relevant, such as how a mismatch of scale will overly weight the importance of dimensions that have larger ranges.

This is a great opportunity to compare scalers available in FluCoMa. One way of clearly demonstrating that different scalers will have different sonic results (and that those sonic results are not always *predictable*) is to choose a single point in a DataSet (such as one sound slice), and find what the nearest neighbour is with (1) no scaling, (2) Normalize, (3) Standardize, and (4) RobustScale. This is essentially what a sequence of images does in the Comparing Scalers page. Doing in real time while hearing the sonic differences can help concretize the importance for learnings. (The Concatenative Synthesis step of the 2D Sound Browser tutorial is a good moment for this.)

It can often be important for learners to keep track of which dimensions might be logarithmic or linear and know how those differences could affect measures of distance. One concrete example we provide is on our scaling page under the heading Linear vs Logarithmic Scales where we state:

> *For example, frequency analyses might be provided in hertz (which is on a linear scale), however this doesn't reflect how humans actually perceive pitch distance. For a more perceptually relevant scale it is displayed logarithmically in pitch space (perhaps labeled as MIDI notes or semitones). If measuring in hertz, the distance from the A4 down one octave to A3 is 220 hertz, while the distance from A4 up one octave to A5 is 440 hertz—twice as far even though we perceive them to both be one octave! Measuring these distances in semitones however will reflect they way we perceive them: both a distance of 12.*

Human vs. Machine Assumptions

One more concrete example of how human and machine assumptions differ comes from a learner who has having trouble getting KMeans to cluster data points in they way they thought it should. The learner wanted KMeans to cluster this 2D plot according to the

clusters that are easily visually identifiable by a human. As one can see, KMeans was clustering it much differently, including leaving many clusters empty. This was solved by demonstrating (including with the original data) how the learner could seed KMeans as a "human in the loop" approach to direct its processing with the information also inferred by a human.

4.7 De-Myth-ifying Machine Learning

Sometimes we have questions from learners that sound something like, "I want X to do Y. How can FluCoMa do this?". At this point, our pedagogical step is to break down the goal into smaller and more specific questions and tasks that we can start approaching together with the learner. This process often reveals the assumptions that the learner might be making about how audio analyses work, or what a machine will be able to perceive, or how long an analysis or algorithm might take, and all of the tradeoffs involved in making decisions about the process. Sometimes the question transforms from "How can FluCoMa do this?" to "Can FluCoMa do this?" at which point perhaps there's a different tool that we can point them to or help them realize that their goal is too lofty–that it stems from a belief that "AI can do anything" or "throw it at a neural network and it'll figure it out". Usually this process enriches the learners ideas about what is possible with FluCoMa (even if it's not necessarily what they hoped) and provides a lot of possibilities for investigation.

Another de-myth-ification that is sometimes engaged is when learners will assume that the machine learning *is* performing some magic. This is often in the form of learners not *validating* or testing the machine learning model or the results of their algorithm. The first disclaimer to make is that, as artists, we're interested in artistically compelling experiences, so regardless of what the algorithm is or isn't doing, if it sounds good, keep it. It is also important to test the systems that we build and see if they are doing what we think they're doing. This is important for a few reasons:

- It can reveal our assumptions and/or misunderstandings about *how* things work, providing opportunities to deeper our knowledge and skills.
- It can offer ways to improve our system to get even closer to our desired outcome.
- It can reveal nuances in the system that might offer more paths of exploration and creativity.

Framing validation with these benefits in mind can help encourage learners to put in the extra work that it takes.

5 Advanced Activities & Examples

During longer workshops, we were able to demonstrate and code-along more advanced topics. All of these have occurred *after* doing all of the 101 activities.

5.1 UMAP 1D

The creative example often shown as part of the 2D Sound Browser activity sparked questions about recreating that result, so we made some simplified demo code to demonstrate "UMAP 1D". After completing the 2D Sound Browser, this example is not particularly more advanced, but offers a fast and compelling way for learners to organize their own sounds through time. It also shows a less common use of one of the tools (using UMAP to reduce to 1 dimension, rather than 2), which may spark more creative and divergent thinking with the toolkit.

1. Iterate over the length of a buffer analyzing each 100 milliseconds as a sound slice using BufMFCC, then use the average coefficients of that 100 milliseconds to represent each slice, adding them all to a DataSet. In this example, the sample position of where the 100 milliseconds begins is used for the identifier, so for later playback, that information can be used directly. (startCoeff is set to 1, but could be experimented with.)
2. Choose whether or not to use a scaler (and if so which).
3. Perform UMAP indicating the desired number of dimensions is 1.
4. Dump the DataSet written by UMAP and sort according to the 1 dimensional position determined by UMAP. The identifier (starting sample position) needs to remain associated with the UMAP position through the sorting so that the, now ordered, sample positions can be used to playback the 100 millisecond sections of the buffer.
5. Iterate over the 100 millisecond sections of the buffer, playing each one. In our code this is done with a sine shaped window and an overlap of 4 (so the result will be 1/4 the duration of the original).

Completed Code Example

- UMAP 1D example for SuperCollider
- UMAP 1D example for Max

5.2 MLPRegressor with Audio Descriptors as Input

A "201" activity that we've done using MLPRegressor involves using real time audio analyses as input to the neural network. The completed code can be found at the links below. Similar to other activities, we have used a slides presentation to explain in the abstract what we will be coding before creating the code together with the learners as a code along. This activity offers a good opportunity to extend the learners' knowledge of MLPs in a few ways:

- The process of making a DataSet is somewhat more automated and the nature of the data demands more thoughtful consideration of the DataSet. Generally speaking, completely randomizing the synthesis parameters used can lead to poorer results than selecting a handful, or more likely, placing bounds or conditions on the randomness to provide parameters that are more "likely".
- This is a harder task for the neural network to learn than the tasks in the "101" examples so training will be more difficult. This makes it an excellent opportunity to cover the advanced neural network topics.
- The result is quite interesting, although not completely compelling for a lot of newcomers. This offers the opportunity to reflect on *how* the neural network might

be able to (or not) learn from this data, what data we are showing it and what it means, and how it might be improved: more data? less data? different data? different statistics? different MLP parameters? different algorithm altogether?

Completed Code Examples:

- MLPRegressor with Descriptors example for SuperCollider
- MLPRegressor with Descriptors example for Max

5.3 Wavetable Autoencoder

This idea is based on an insight from a learner in an early workshop. The insight was that since the MLPRegressor writes predictions directly into a buffer, to also use that buffer to read out of as a wavetable. The initial idea was very similar to the 101 MLPRegressor activity but rather than predicting synthesis parameters, it predicted wavetables that then could be interpolated between while also being heard.

This lead to the idea to train an autoencoder to create a latent space of a set of wavetables. Navigating through the latent space would morph the wavetable shape in a musically and navigable way. As you can see in the steps below, this introduces many more concepts that are advanced but build on knowledge gained in previous tutorials.

1. The "wavetables" used in the example are excerpts 2048 samples long taken from the audio file of trombone tones that is included in the FluCoMa toolkit ("Olencki-TenTromboneLongTones-M.wav").
2. Because each one of these is a data point with 2048 dimensions, PCA is first used to reduce the number of dimensions to around 150 (something like around 95% of the variance can be maintained).
3. The autoencoder is then trained with this DataSet as the input and output with a latent space of 2 (**hiddenLayers** is something like [80,40,2,40,80]).
4. Then when making predictions, the **tapIn** and **tapOut** arguments need to be set and used accordingly, as well as the **inverseTransformPoint** method for PCA.

Completed Code Examples:

- Wavetable Autoencoder example for SuperCollider
- Wavetable Autoencoder example for Max

6 Workshops Formats

Over the course of about one year, we conducted numerous workshops in various formats. For each of these workshops the only tech required is a projector and stereo audio. Some of these workshops were conducted online (because of Covid-19). Below are some reflections on the formats conducted, should they serve useful for future FluCoMa workshop settings.

We did many of these class period sessions, the quantity of which was very helpful for us to get feedback on the lesson plans we had designed. Sometimes we did multiple of these sessions in one day, allowing for quick iteration on our lesson plans. The questions asked by

learners directly impacted how we taught topics in the future, as well as led to many Learn Articles, examples, and clarifications in many of the learning resources.

One Class Period

Some of the sessions we offered were just a single class period (usually about 90 minutes) embedded in a semester long curriculum on music technology. For these sessions we presented an introductory slides presentation and then continued with one neural network activity, usually either the MLPRegressor session, just it's MLPClassifier version, or a combination of both. These seemed to work well, the activities ended in learners successfully completing the code and having a working version. We had some follow up from learners after who wanted to dig in more.

One Day (2-6 Hours)

Our one day workshops were similar to the *one class period* workshops above, but allowed for one additional activity to be offered. This was often adding the 2D Sound Browser activity after the neural network activity.

Two Days

One common setting we had was a two day workshop where one day would be a presentation at an academic seminar and following that would be a classroom presentation or extra curricular workshop. The following workshop followed our *one day* format above, but we were able to skip any introductory material and therefore a little more content. This was either in the form of taking a little more time with the activities (showing some more possibilities with the code, diving deeper into some concepts, etc.), or adding an additional activity, such as the Intro to NMF, or demonstrating some of the other decomposition objects (HPSS, Sines, and Transients). Another popular object to add to these workshops when we had time was AudioTransport.

Five Days

On three occasions we offered five day workshops (about six hours each day) that allowed for a much deeper dive into FluCoMa. Because we had so much time, we tended to allow these sessions to expand in length and content as learners' questions offered more directions to explore. This was very beneficial to use to see what kinds of questions, deep dives, and idiosyncratic directions were interesting and challenging to learners.

It also allowed us to allocate more time for learners to play around with code after an activity, modifying it to their own artistic uses and inviting us to help them understand it further. We also found a constant tension between some learners wanting more individual work time and some learners wanting to use all the time absorbing more content (saving the individual work time for the evenings or after the workshop). We also found that after a few days (or sometimes just one day) many learners were overwhelmed with the breadth and depth of the content. We didn't necessarily have a solution to this, however, if enough of the learners were feeling overwhelmed we often took that moment to have a pause in content and provide some individual work time.

An example outline of a 5 day workshop:

- Day 1
  - AM
    - Intro
    - MLPClassifier
    - What are MFCCs?
  - PM
    - MLPRegressor
    - Individual work time
- Day 2
  - AM
    - 2D Sound Browser
  - PM
    - 2D Sound Browser (continued)
    - Hinges
    - UMAP
- Day 3
  - AM
    - MLPRegressor with Audio Descriptors
    - Advanced MLP Parameters
  - PM
    - Decomposition Tools (HPSS, Sines, and Transients)
    - Intro to NMF
    - Individual work time
- Day 4
  - AM
    - Project brainstorming with facilitators
    - Individual work time
  - PM: Advanced Topics
    - Grid
    - KMeans
    - Individual work time
- Day 5
  - AM: Advanced Topics
    - UMAP 1D
    - Autoencoders
    - Individual work time
  - PM
    - Participants share some of their work with the group, either code that is making sound and working or plans for what their working on.

7 Example Semester-Long Syllabus

Machine Listening and Machine Learning for Music Making

This is an example syllabus designed to be used in coordination with the FluCoMa Toolkit.

SUGGESTED COURSE PREREQUISITE

At least one semester of introduction to creative coding for music (Max, SuperCollider, or Pure Data). The creative coding environment should be the same one they will use with FluCoMa.

REQUIRED MATERIALS

- **FluCoMa Toolkit (free)** Installation instructions at https://learn.flucoma.org/installation/
- **A creative coding environment of choice:** Max, SuperCollider, or Pure Data (at the instructor's discretion, the course could be taught in multiple languages simultaneously). SuperCollider and Pure Data are both free. Many educational institutions have Max licences for students to use for free.
- **Access to FluCoMa's learning resources (free)** Found at http://learn.flucoma.org.

SWBAT

At the end of this semester, **students will be able to**:

1. Use computational thinking and data science methods to approach and solve problems based in sound.
2. Think creatively about and apply machine listening and machine learning to design compositional and performance systems.
3. Create idiosyncratic musical expressions that use machine listening and machine learning.
4. Understand the limitations of machine listening and machine learning. Be able to assess machine accuracy and analyze human assumptions.
5. Express intuitions about selecting algorithms and approaches for given tasks.
6. Relate knowledge and intuition about machine learning and machine automation to broader applications of these technologies in society.

REPERTOIRE

Although this document does not contain repertoire examples, we encourage the reader to visit https://learn.flucoma.org/explore/ which offers many examples of creative projects using machine listening and machine learning with FluCoMa as well as other toolkits. Many of these examples are accompanied by documentation, interviews, and articles about the thinking behind and implementation of the work.

SCHEDULE

This course schedule is designed for a 14 week semester with two 90-minute class meetings per week. We think the schedule outlined below is quite ambitious and may be most appropriate for a class apt to move quickly. A question mark (*?*) after a topic indicates that this maybe won't fit in the class or the week and could be considered optional.

Week 1: Intro to Neural Networks

- Using a Neural Network to Control a Synthesizer

  - DataSet
  - Buffer interface
  - MLPRegressor
- Classifying sound based on MFCCs
  - LabelSet
  - MLPClassifier
  - Training-Testing Split
  - JSON I/O

Week 2: Audio and Statistical Analysis

- Sorting Sounds Based on Audio Analysis
  - NoveltySlice
  - NRT buffer analyses
    - BufSpectralShape
    - BufPitch
    - BufLoudness
    - FluCoMa Waveform Object
  - BufStats
    - mean, standard deviation, skewness, kurtosis, order statistics
- Browsing Sounds in 2 Dimensional Space (Loudness & Centroid)
  - Batch Processing
  - FluCoMa Plotter Object
  - KDTree for nearest neighbour lookup

Week 3: Slicing Audio & Finding "Novelty"

- Slicers
  - AmpSlice
  - OnsetSlice
  - TransientSlice
  - AmpGate
- NoveltySlice & Slicer Feature Curves
  - What is "Novelty"
  - NoveltyFeature
  - AmpFeature
  - OnsetFeature

Week 4: Why Machines Listen and Learn Differently from Humans

- Comparing Human & Machine Assumptions of Machine Listening
  - Why Scale
  - Linear vs. Logarithmic Scaling
  - Log Centroid
  - MelBands
  - MFCC
  - Chroma
- The Musical Implications of Scaling (Comparing Scalers)

  - Normalize
  - Standardize
  - Robust Scaler

Week 5: Dimensionality Reduction

- Reducing Large Analyses to a 2 Dimensional Performance Space
  - Proximity as Similarity
  - Other Measures of Distance
  - Principal Component Analysis
- Uniform Manifold Approximation and Projection (UMAP)

Week 6: Concatenative Synthesis

- NRT Concatenative Synthesis
  - The Curse of Dimensionality
  - The Hinges / The Butterfly Effect: With a big system, small parameter changes can have a big effect
- Real-Time Concatenative Synthesis
  - Inline Learning
  - Real-Time considerations with FluCoMa

Week 7: Clustering

- KMeans
  - Seeding
- SKMeans
  - Cosine Distance

Week 8: Concert & Grid?

- Grid?
- Midterm Concert

Week 9: Outliers & Other Data Considerations

- What is an outlier
  - Weighting Stats
  - Removing Outliers with BufStats
- SQL-type Queries
  - DataSetQuery

Week 10: Signal Decomposition

- Signal Decomposition
  - Sines
  - Transients
- Harmonic-Percussive Source Separation
  - HPSS

Week 11: Non-negative Matrix Factorization

- Decomposing Audio into Sound Object Components
  - BufNMF (Bases and Activations)
  - NMFFilter
  - NMFMatch
  - BufNMFCross?
- Seeding NMF & Over-Decomposing
  - BufNMFSeed?
  - NMFMorph?

Week 12: Advanced Neural Networks

- Predicting Synthesis Parameters from Real-Time Audio Analyses
  - Advanced MLP Parameters
- Autoencoders
  - Latent Space
  - Noise Reduction

Week 13: Bundling Time

- Stats
  - Leaky Integrators
- Sonically Informed Analysis
  - Analyzing different envelope sections of a sound

Week 14: Concert & AudioTransport?

- AudioTransport?
- Final Concert

8 Relevance to Contemporary Society

We believe that learning about data, data science, and machine learning through FluCoMa can be be used as a lens to consider how these tools operate in contemporary society, in particular to uphold inequalities, injustices, and hegemonies. By gaining fluency and understanding with these algorithms, one can come to understand what these algorithms are good for, what they are not good for, how they go wrong, and the relationship between data, algorithms, and the humans that use them. The skills and knowledge, both explicit and tacit, that working with FluCoMa and foster can be used to reflect on many of the AI events and concerns that are constantly appearing in the news. Pedagogues might draw on these events to use as discussion topics where a classroom of learners can collectively reflect on contemporary topics using the experience and understand built through FluCoMa.

Below is a list of books (in no particular order) that provide many examples of contemporary technologies negative impacting marginalized communities. Many additionally offer directions for how to approach and use data science ethically. We recommend selecting a book or readings from these books to augment learning. Each of these is written for different audiences, so selecting which is best is at the instructor/learner's discretion.

- *Data Feminism* by Catherine D'Ignazio and Lauren F. Klein.
- *Weapons of Math Destruction* by Cathy O'Neil
- *Hello World* by Hannah Fry
- *Revolutionary Mathematics* by Justin Joque
- *Blockchain Chicken Farm: And Other Stories of Tech in China's Countryside* by Xiaowei Wang
- *The Alignment Problem* by Brian Christian