
Somoclu Python Documentation

Release 1.7.2

Peter Wittek, Shi Chao Gao

Nov 24, 2016

CONTENTS

1	Introduction	1
1.1	Copyright and License	1
1.2	Acknowledgment	1
2	Download and Installation	3
2.1	Dependencies	3
3	Examples	5
3.1	Planar maps	6
3.2	Toroid topology, hexagonal grid	10
3.3	Initialization with principal component analysis and clustering the results	12
3.4	Evolving maps	15
4	Function Reference	19
	Index	23

INTRODUCTION

Somoclu is a massively parallel implementation of self-organizing maps. It relies on OpenMP for multicore execution and it can be accelerated by CUDA. The topology of map is either planar or toroid, the grid is rectangular or hexagonal. Currently a subset of the command line version is supported with this Python module.

Key features of the Python interface:

- Fast execution by parallelization: OpenMP and CUDA are supported.
- Multi-platform: Linux, OS X, and Windows are supported.
- Planar and toroid maps.
- Rectangular and hexagonal grids.
- Gaussian or bubble neighborhood functions.
- Visualization of maps, including those that were trained outside of Python.

The documentation is available online. Further details are found in the following paper:

Peter Wittek, Shi Chao Gao, Ik Soo Lim, Li Zhao (2015). Somoclu: An Efficient Parallel Library for Self-Organizing Maps. [arXiv:1305.1422](https://arxiv.org/abs/1305.1422).

1.1 Copyright and License

Somoclu is free software; you can redistribute it and/or modify it under the terms of the [GNU General Public License](#) as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

Somoclu is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the [GNU General Public License](#) for more details.

1.2 Acknowledgment

This work is supported by the European Commission Seventh Framework Programme under Grant Agreement Number FP7-601138 [PERICLES](#) and by the AWS in Education Machine Learning Grant award.

DOWNLOAD AND INSTALLATION

The package is available in the [Python Package Index](#), containing the source, documentation, and examples. The latest development version is available on [GitHub](#).

2.1 Dependencies

The module requires [Numpy](#) and [matplotlib](#). The code is compatible with both Python 2 and 3.

On Linux, you need a standard GCC compile chain. Other compiler chains, for instance, ICC, are known to work too.

On OS X, you must have `libomp` installed. See [this issue](#) if you encounter problems. If you build it from source, please refer to the section below.

On Windows, having `MSVCP90.DLL` and `VCOMP90.DLL` is usually sufficient. See [this issue](#) if you have problems.

2.1.1 Installation

The code is available on PyPI, hence it can be installed by

```
$ sudo pip install somoclu
```

If you want the latest git version, clone the repository, make the Python target, and follow the standard procedure for installing Python modules:

```
$ git clone https://github.com/peterwittek/somoclu.git
$ cd somoclu
$ ./autogen.sh
$ ./configure
$ make python
$ cd src/python
$ sudo python setup.py install
```

2.1.2 Build on Mac OS X

GCC should be installed first. As of OS X 10.9, `gcc` is just symlink to `clang`. To build `somoclu` and this extension correctly, it is recommended to install `gcc` using something like:

```
$ brew install gcc48
```

and set environment using:

```
export CC=/usr/local/bin/gcc
export CXX=/usr/local/bin/g++
export CPP=/usr/local/bin/cpp
export LD=/usr/local/bin/gcc
alias c++=/usr/local/bin/c++
alias g++=/usr/local/bin/g++
alias gcc=/usr/local/bin/gcc
alias cpp=/usr/local/bin/cpp
alias ld=/usr/local/bin/gcc
alias cc=/usr/local/bin/gcc
```

Then you can issue

```
$ sudo pip install somoclu
```

2.1.3 Build with CUDA support on Linux and OS X:

If the CUDAHOME variable is set, the usual install command will build and install the library:

```
$ sudo python setup.py install
```

2.1.4 Build with CUDA support on Windows:

You should first follow the instructions to [build the Windows binary](#) with MPI disabled with the same version Visual Studio as your Python is built with. (Since currently Python is built by VS2008 by default and CUDA v6.5 removed VS2008 support, you may use CUDA 6.0 with VS2008 or find a Python prebuilt with VS2010. And remember to install VS2010 or Windows SDK7.1 to get the option in Platform Toolset if you use VS2013.) Then you should copy the .obj files generated in the release build path to the Python/src folder.

Then modify the win_cuda_dir in setup.py to your CUDA path and run the install command

```
$ sudo python setup.py install
```

Then it should be able to build and install the module.

EXAMPLES

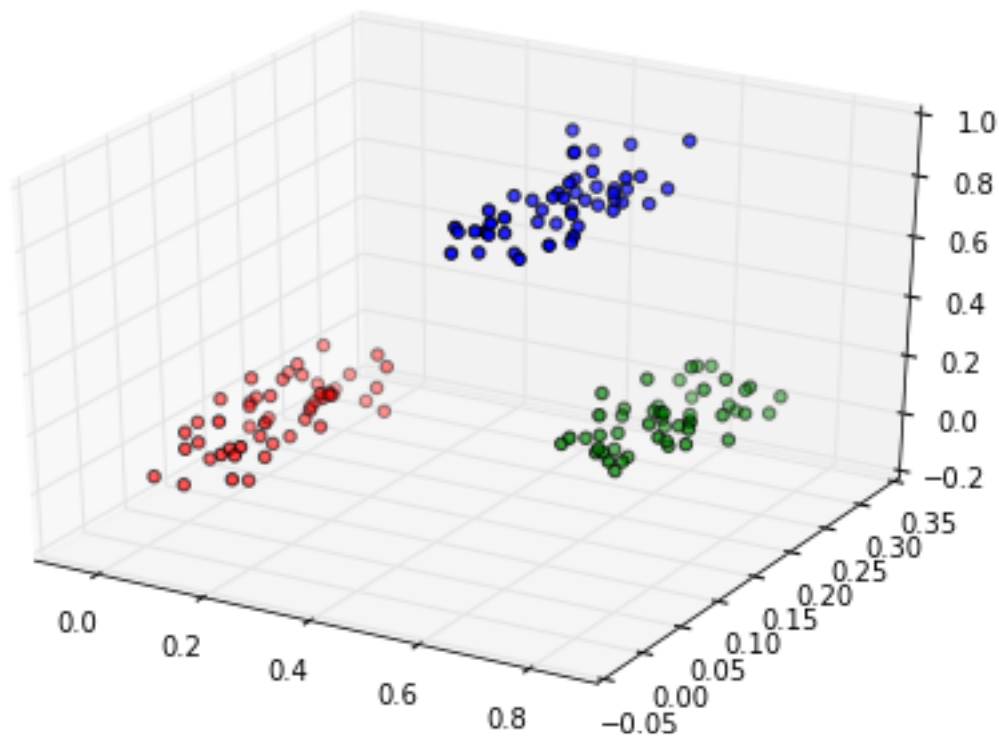
Self-organizing maps are computationally intensive to train, especially if the original space is high-dimensional or the map is large. Very large maps where the number of neurons is at least five times the number of data points are sometimes called emergent-self organizing maps – these are especially demanding to train. Somoclu is a highly efficient, parallel and distributed algorithm to train such maps, and its Python interface was recently updated. This enables fast training of self-organizing maps on multicore CPUs or a GPU from Python, albeit only on dense data, and the distributed computing capability is also not exposed. The Python interface also lets you process the output files of the command-line version, so if the data is sparse or the map was trained on a cluster, you can still use the module for visualization. Here we take a quick look at how to train and visualize a small map.

First, we import the necessary modules:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import somoclu
%matplotlib inline
```

Then we generate and plot some random data in three categories:

```
c1 = np.random.rand(50, 3)/5
c2 = (0.6, 0.1, 0.05) + np.random.rand(50, 3)/5
c3 = (0.4, 0.1, 0.7) + np.random.rand(50, 3)/5
data = np.float32(np.concatenate((c1, c2, c3)))
colors = ["red"] * 50
colors.extend(["green"] * 50)
colors.extend(["blue"] * 50)
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(data[:, 0], data[:, 1], data[:, 2], c=colors)
labels = range(150)
```



3.1 Planar maps

We train Somoclu with default parameter settings, asking for a large map that qualifies as an emergent self-organizing map for this data:

```
n_rows, n_columns = 100, 160
som = somoclu.Somoclu(n_columns, n_rows, data=data, compactsupport=False)
%time som.train()
```

```
CPU times: user 6.99 s, sys: 3.33 ms, total: 6.99 s
Wall time: 5.21 s
```

We plot the component planes of the trained codebook of the ESOM:

```
som.view_component_planes()
```

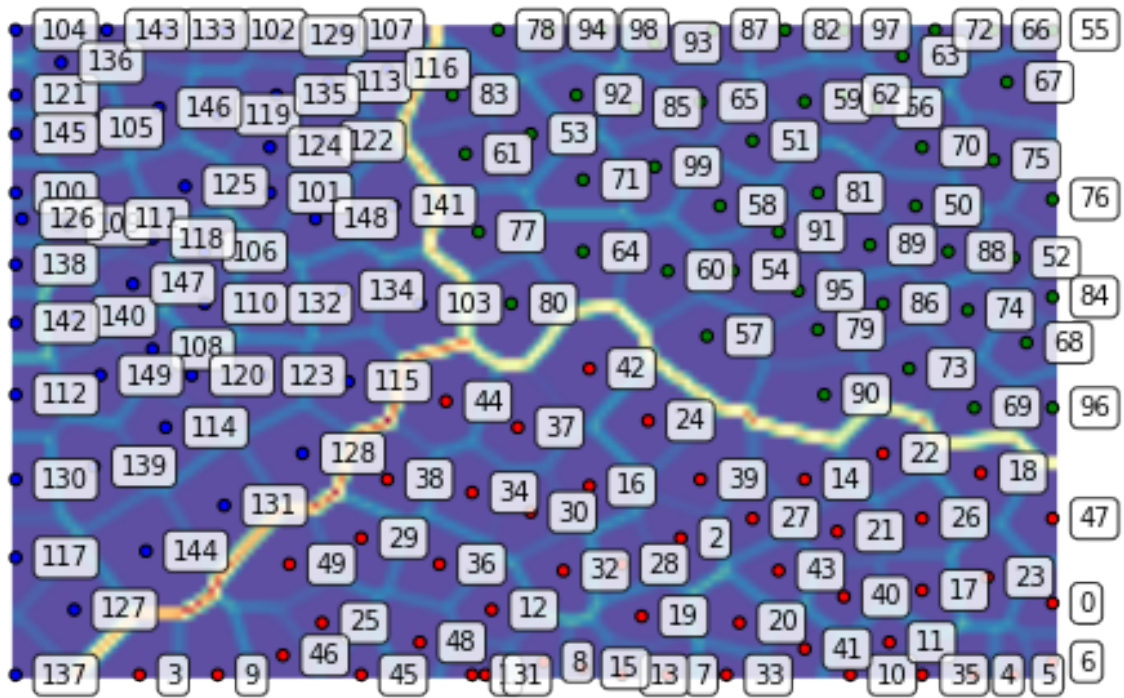




```
<module 'matplotlib.pyplot' from '/usr/lib/python3.5/site-packages/matplotlib/pyplot.py'>
```

We can plot the U-Matrix, together with the best matching units for each data point. We color code the units with the classes of the data points and also add the labels of the data points.

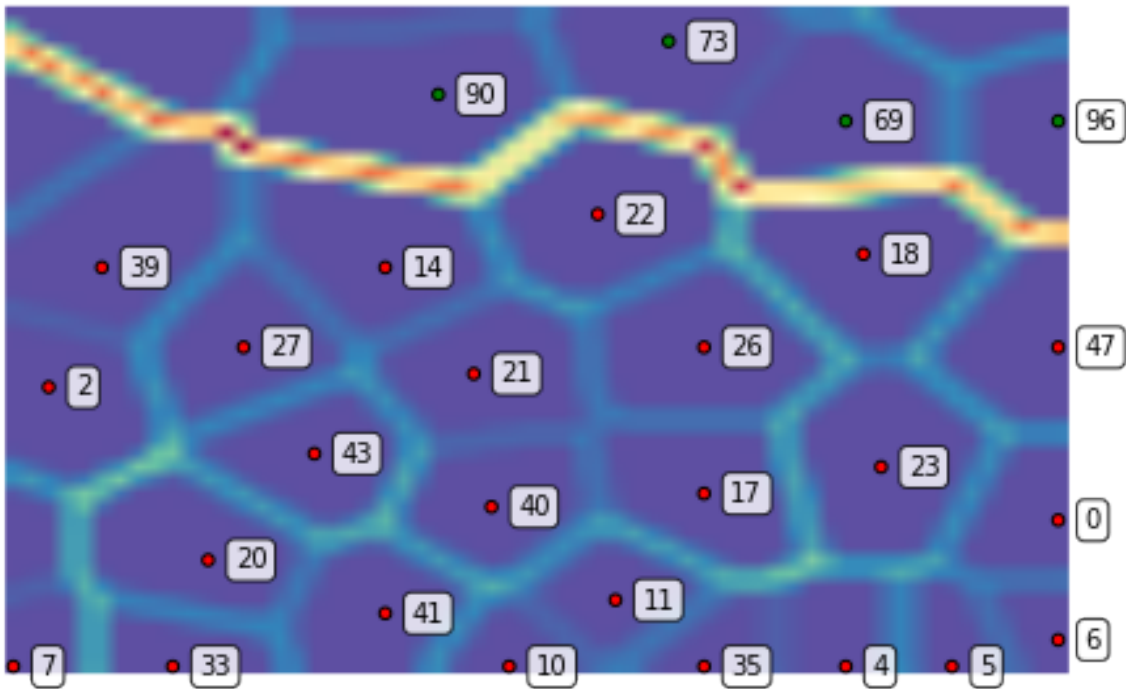
```
som.view_umatrix(bestmatches=True, bestmatchcolors=colors, labels=labels)
```



```
<module 'matplotlib.pyplot' from '/usr/lib/python3.5/site-packages/matplotlib/pyplot.
↳py'>
```

We can also zoom into a region of interest, for instance, the dense lower right corner:

```
som.view_umatrix(bestmatches=True, bestmatchcolors=colors, labels=labels,
                  zoom=((50, n_rows), (100, n_columns)))
```

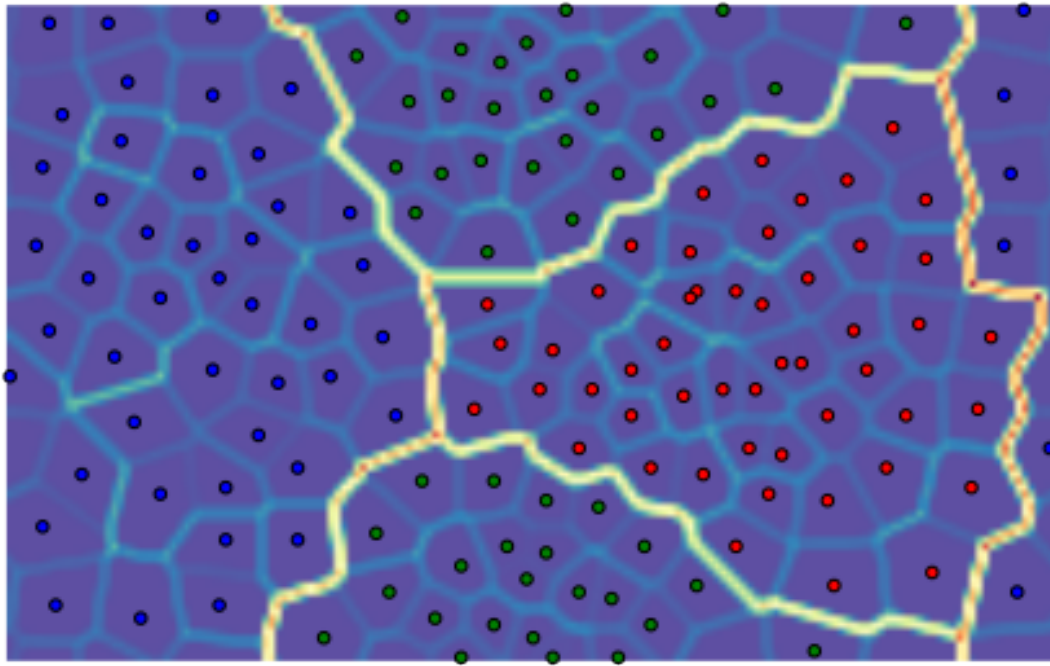


```
<module 'matplotlib.pyplot' from '/usr/lib/python3.5/site-packages/matplotlib/pyplot.  
↳py'>
```

3.2 Toroid topology, hexagonal grid

We can repeat the above with a toroid topology by specifying the map type as follows:

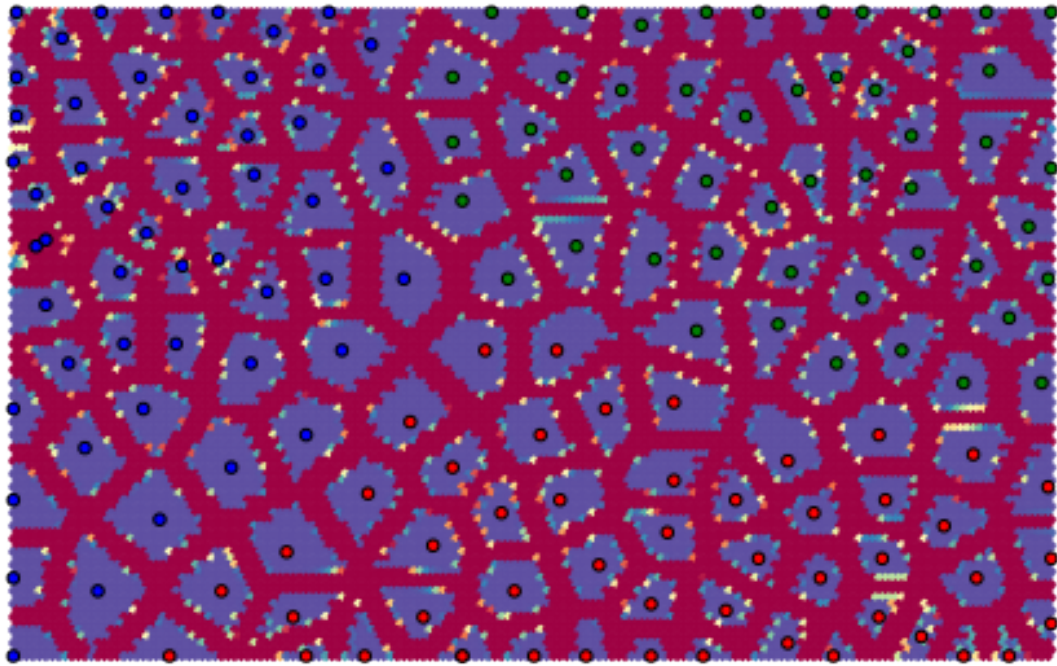
```
som = somoclu.Somoclu(n_columns, n_rows, data=data, maptype="toroid",  
                      compactsupport=False)  
som.train()  
som.view_umatrix(bestmatches=True, bestmatchcolors=colors)
```



```
<module 'matplotlib.pyplot' from '/usr/lib/python3.5/site-packages/matplotlib/pyplot.  
→py'>
```

Notice how the edges of the map connect to the other side. Hexagonal neurons are also implemented:

```
som = somoclu.Somoclu(n_columns, n_rows, data=data, gridtype="hexagonal",  
                      compactsupport=False)  
som.train()  
som.view_umatrix(bestmatches=True, bestmatchcolors=colors)
```



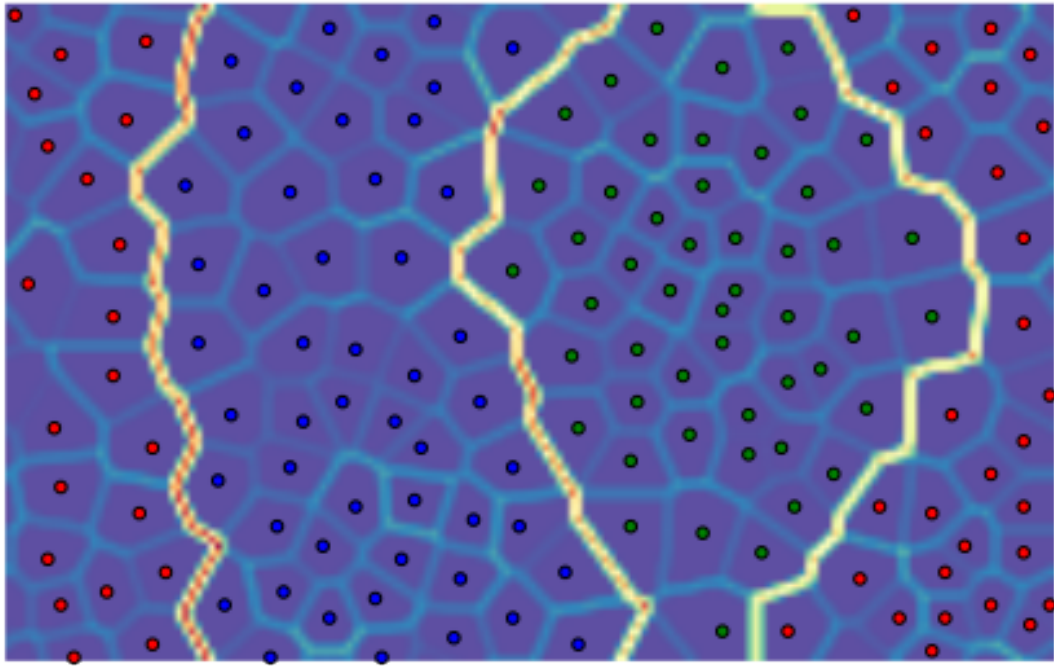
```
<module 'matplotlib.pyplot' from '/usr/lib/python3.5/site-packages/matplotlib/pyplot.py'>
```

The separation of the individual points is more marked with these neurons.

3.3 Initialization with principal component analysis and clustering the results

We can pass an initial codebook of our choice, but we can also ask Somoclu to initialize the codebook with vectors from the subspace spanned by the first two eigenvalues of the correlation matrix. To do this, we need to pass an optional argument to the constructor:

```
som = somoclu.Somoclu(n_columns, n_rows, data=data, maptype="toroid",  
                      compactsupport=False, initialization="pca")  
som.train()  
som.view_umatrix(bestmatches=True, bestmatchcolors=colors)
```

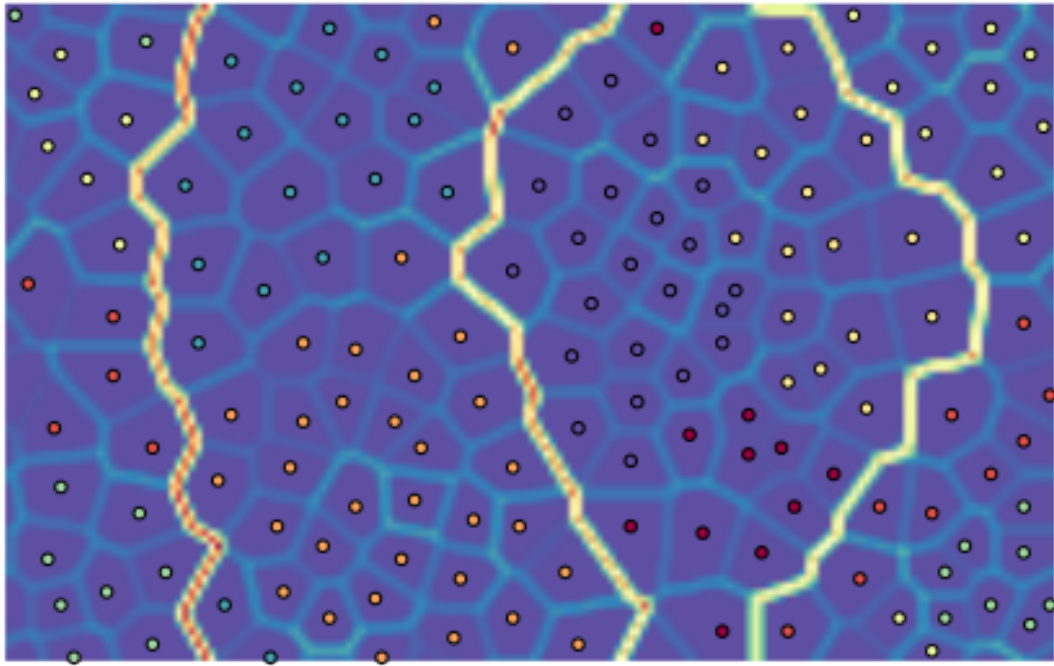



```
<module 'matplotlib.pyplot' from '/usr/lib/python3.5/site-packages/matplotlib/pyplot.py'>
```

While one would expect entirely deterministic results on repeated runs with the initialization based on PCA, this is not the case. The order in which the data instances arrive matters: since Somoclu uses multiple cores, there is no control over the order of each batch, hence the maps will show small variation even with a PCA initialization.

We can also postprocess the codebook with an arbitrary clustering algorithm that is included in [scikit-learn](#). The default algorithm is K-means with eight clusters. After clustering, the labels for each node are available in the SOM object in the `clusters` class variable. If we do not pass colors to the matrix viewing functions and clustering is already done, the plotting routines automatically color the best matching units according to the clustering structure.

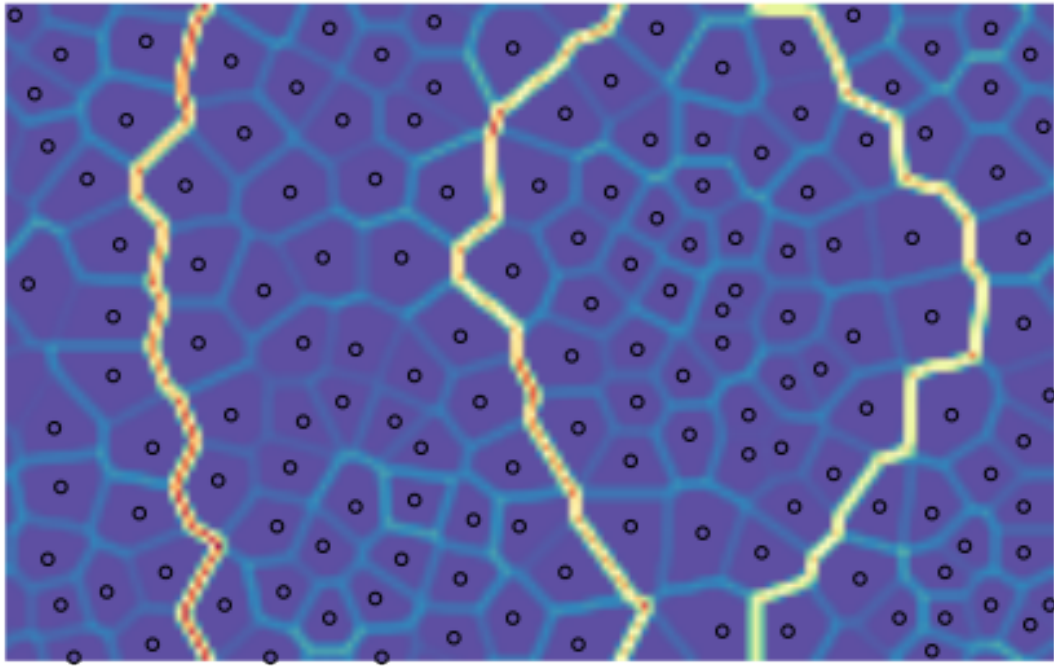
```
som.cluster()  
som.view_umatrix(bestmatches=True)
```



```
<module 'matplotlib.pyplot' from '/usr/lib/python3.5/site-packages/matplotlib/pyplot.  
→py'>
```

We can, of course, choose another clustering algorithm, but topological clustering methods will make little sense with their default parameterization. DBSCAN, for instance, will assign the same class to all nodes:

```
from sklearn.cluster import DBSCAN  
algorithm = DBSCAN()  
som.cluster(algorithm=algorithm)  
som.view_umatrix(bestmatches=True)
```



```
<module 'matplotlib.pyplot' from '/usr/lib/python3.5/site-packages/matplotlib/pyplot.
  ↳py'>
```

3.4 Evolving maps

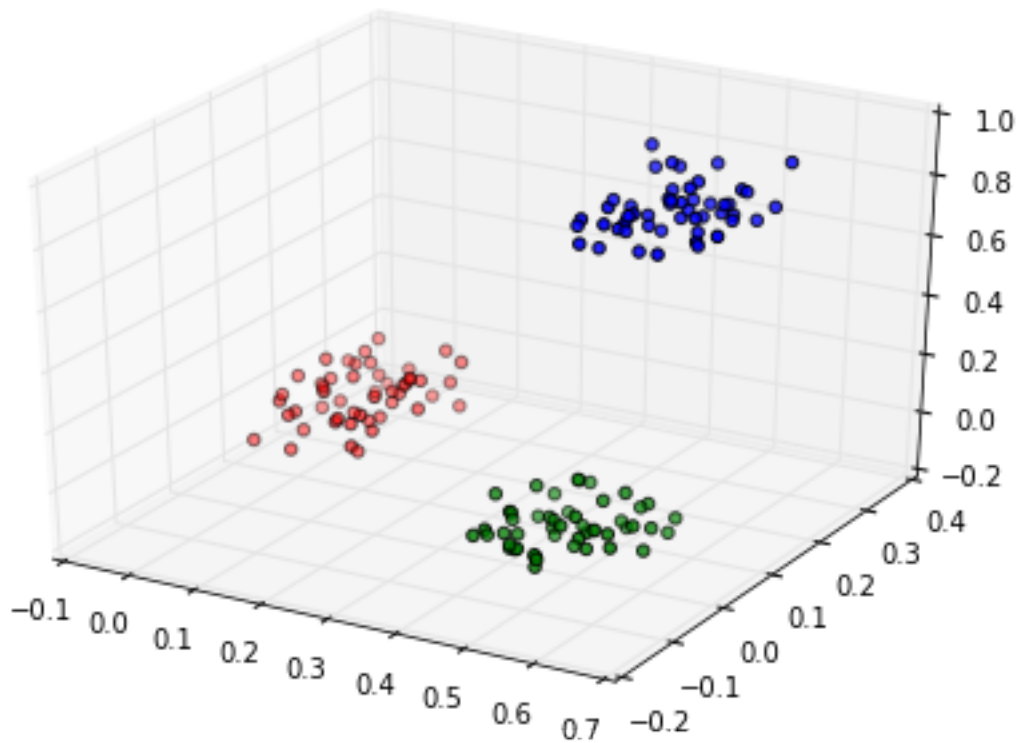
One of the great advantages of self-organizing maps is that they are incremental, they can be updated with new data. This is especially interesting if the data points retain their old label, that is, the properties of the vectors change in the high-dimensional space. Let us train again a toroid rectangular emergent map on the same data:

```
som = somoclu.Somoclu(n_columns, n_rows, data=data, maptype="toroid")
som.train()
```

Next, let us assume that the green cluster moves to the left, the other points remaining invariant:

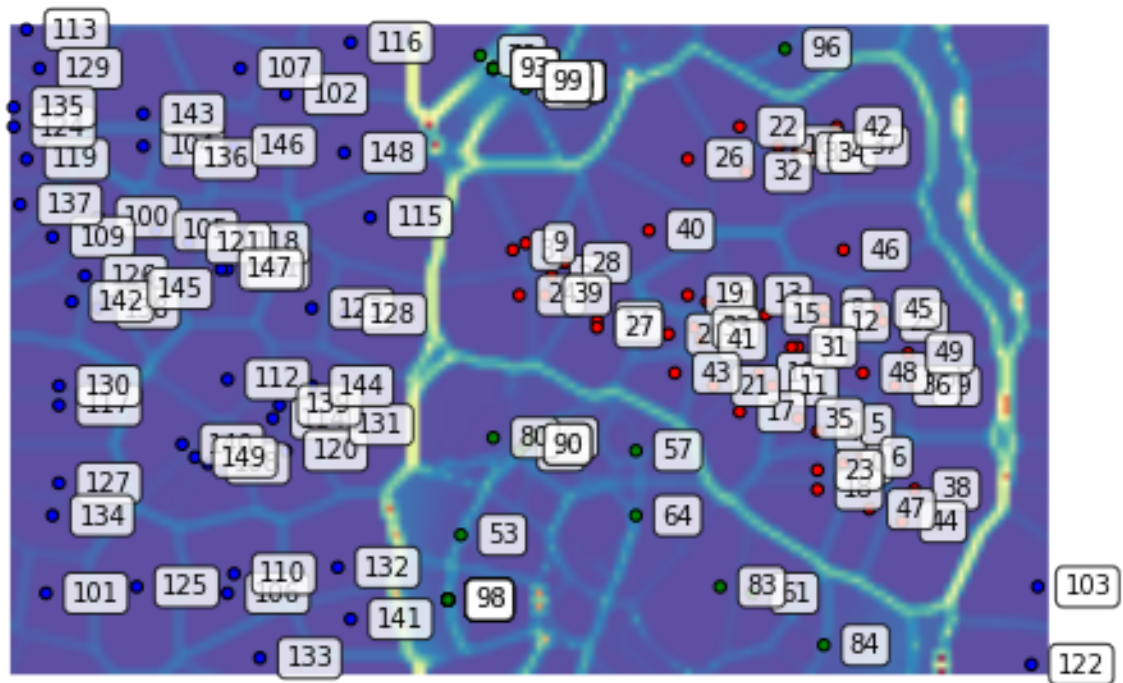
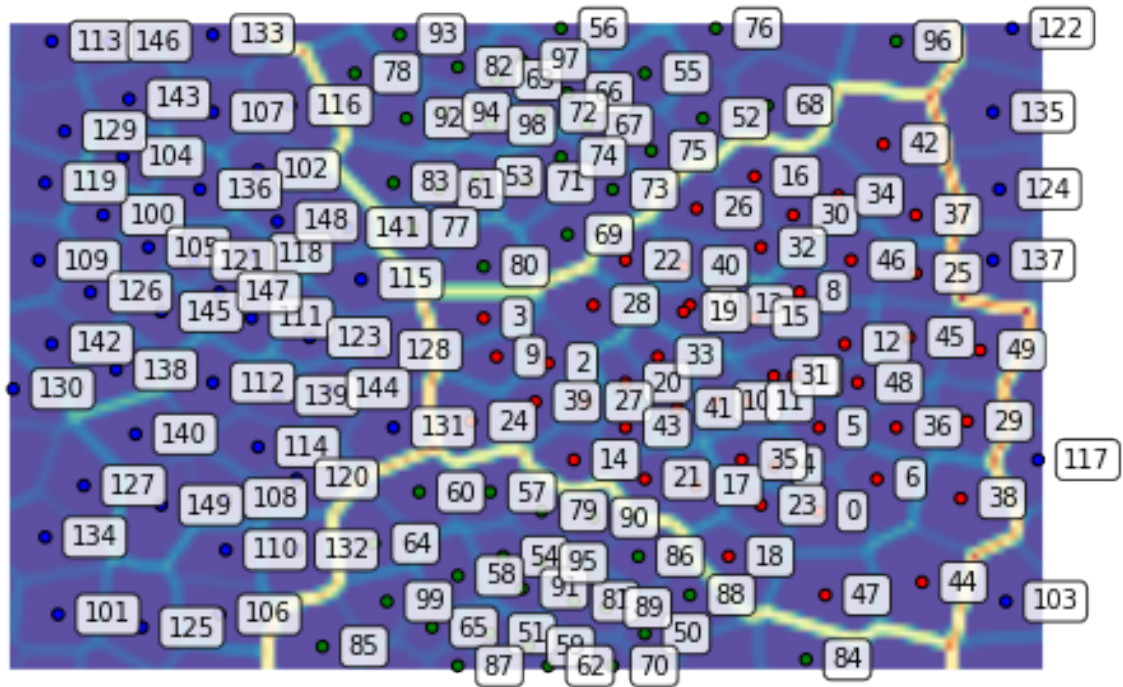
```
c2_shifted = c2 - 0.2
updated_data = np.float32(np.concatenate((c1, c2_shifted, c3)))
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(updated_data[:, 0], updated_data[:, 1], updated_data[:, 2], c=colors)
```

```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7fc9cf752470>
```



We can update the map to reflect this shift. We plot the map before and after continuing the training:

```
som.view_umatrix(bestmatches=True, bestmatchcolors=colors, labels=labels)
som.update_data(updated_data)
som.train(epochs=2, radius0=20, scale0=0.02)
som.view_umatrix(bestmatches=True, bestmatchcolors=colors, labels=labels)
```



```
<module 'matplotlib.pyplot' from '/usr/lib/python3.5/site-packages/matplotlib/pyplot.py'>
```

As a result of the shift, the blue points do not move around much. On the other hand, the relationship of the red and green clusters is being redefined as their coordinates inched closer in the original space.

FUNCTION REFERENCE

```
class Somoclu (n_columns, n_rows, data=None, initialcodebook=None, kerneltype=0, maptype='planar',
               gridtype='rectangular', compactsupport=False, neighborhood='gaussian', std_coeff=0.5,
               initialization=None)
```

Class for training and visualizing a self-organizing map.

Parameters

- **n_columns** (*int.*) – The number of columns in the map.
- **n_rows** (*int.*) – The number of rows in the map.
- **data** (*2D numpy.array of float32.*) – Optional parameter to provide training data. It is not necessary if the map is otherwise trained outside Python, e.g., on a GPU cluster.
- **initialcodebook** (*2D numpy.array of float32.*) – Optional parameter to start the training with a given codebook.
- **kerneltype** (*int.*) – Optional parameter to specify which kernel to use:
 - 0: dense CPU kernel (default)
 - 1: dense GPU kernel (if compiled with it)
- **maptype** (*str.*) – Optional parameter to specify the map topology:
 - “planar”: Planar map (default)
 - “toroid”: Toroid map
- **gridtype** (*str.*) – Optional parameter to specify the grid form of the nodes:
 - “rectangular”: rectangular neurons (default)
 - “hexagonal”: hexagonal neurons
- **compactsupport** (*bool.*) – Optional parameter to cut off map updates beyond the training radius with the Gaussian neighborhood. Default: True.
- **neighborhood** (*str.*) – Optional parameter to specify the neighborhood:
 - “gaussian”: Gaussian neighborhood (default)
 - “bubble”: bubble neighborhood function
- **std_coeff** (*float.*) – Optional parameter to set the coefficient in the Gaussian neighborhood function $\exp(-\|x-y\|^2/(2*(coeff*radius)^2))$ Default: 0.5
- **initialization** (*str.*) – Optional parameter to specify the initialization:
 - “random”: random weights in the codebook

- “pca”: codebook is initialized from the first subspace spanned by the first two eigenvectors of the correlation matrix

cluster (*algorithm=None*)

Cluster the codebook. The clusters of the data instances can be assigned based on the BMUs. The method populates the class variable `Somoclu.clusters`. If viewing methods are called after clustering, but without colors for best matching units, colors will be automatically assigned based on cluster membership.

Parameters **algorithm** – Optional parameter to specify a scikit-learn clustering algorithm. The default is K-means with eight clusters.

get_surface_state (*data=None*)

Return the dot product of the codebook and the data.

Parameters **data** (*2D numpy.array of float32.*) – Optional parameter to specify data, otherwise the data used previously to train the SOM is used.

Returns The the dot product of the codebook and the data.

Return type 2D numpy.array

get_bmus (*activation_map*)

Return Best Matching Unit indexes of the activation map.

Parameters **activation_map** (*2D numpy.array*) – Activation map computed with `self.get_surface_state()`

Returns The bmus indexes corresponding to this activation map (same as `self.bmus` for the training samples).

Return type 2D numpy.array

load_bmus (*filename*)

Load the best matching units from a file to the Somoclu object.

Parameters **filename** (*str.*) – The name of the file.

load_codebook (*filename*)

Load the codebook from a file to the Somoclu object.

Parameters **filename** (*str.*) – The name of the file.

load_umatrix (*filename*)

Load the umatrix from a file to the Somoclu object.

Parameters **filename** (*str.*) – The name of the file.

train (*epochs=10, radius0=0, radiusN=1, radiuscooling='linear', scale0=0.1, scaleN=0.01, scalecooling='linear'*)

Train the map on the current data in the Somoclu object.

Parameters

- **epochs** (*int.*) – The number of epochs to train the map for.
- **radius0** (*float.*) – The initial radius on the map where the update happens around a best matching unit. Default value of 0 will trigger a value of $\min(n_columns, n_rows)/2$.
- **radiusN** (*float.*) – The radius on the map where the update happens around a best matching unit in the final epoch. Default: 1.
- **radiuscooling** – The cooling strategy between `radius0` and `radiusN`:
 - “linear”: Linear interpolation (default)
 - “exponential”: Exponential decay

- **scale0** (*float.*) – The initial learning scale. Default value: 0.1.
- **scaleN** (*float.*) – The learning scale in the final epoch. Default: 0.01.
- **scalecooling** (*str.*) – The cooling strategy between scale0 and scaleN:
 - “linear”: Linear interpolation (default)
 - “exponential”: Exponential decay

update_data (*data*)

Change the data set in the Somoclu object. It is useful when the data is updated and the training should continue on the new data.

Parameters *data* (*2D numpy.array of float32.*) – The training data.

view_activation_map (*data_vector=None, data_index=None, activation_map=None, figsize=None, colormap=cm.Spectral_r, colorbar=False, bestmatches=False, bestmatchcolors=None, labels=None, zoom=None, filename=None*)

Plot the activation map of a given data instance or a new data vector

Parameters

- **data_vector** (*numpy.array*) – Optional parameter for a new vector
- **data_index** (*int.*) – Optional parameter for the index of the data instance
- **activation_map** (*numpy.array*) – Optional parameter to pass the an activation map
- **figsize** (*((int, int))*) – Optional parameter to specify the size of the figure.
- **colormap** (*matplotlib.colors.Colormap*) – Optional parameter to specify the color map to be used.
- **colorbar** (*bool.*) – Optional parameter to include a colormap as legend.
- **bestmatches** (*bool.*) – Optional parameter to plot best matching units.
- **bestmatchcolors** (*list of int.*) – Optional parameter to specify the color of each best matching unit.
- **labels** (*list of str.*) – Optional parameter to specify the label of each point.
- **zoom** (*((int, int), (int, int))*) – Optional parameter to zoom into a region on the map. The first two coordinates of the tuple are the row limits, the second tuple contains the column limits.
- **filename** (*str.*) – If specified, the plot will not be shown but saved to this file.

view_component_planes (*dimensions=None, figsize=None, colormap=cm.Spectral_r, colorbar=False, bestmatches=False, bestmatchcolors=None, labels=None, zoom=None, filename=None*)

Observe the component planes in the codebook of the SOM.

Parameters

- **dimensions** – Optional parameter to specify along which dimension or dimensions should the plotting happen. By default, each dimension is plotted in a sequence of plots.
- **figsize** (*((int, int))*) – Optional parameter to specify the size of the figure.
- **colormap** (*matplotlib.colors.Colormap*) – Optional parameter to specify the color map to be used.
- **colorbar** (*bool.*) – Optional parameter to include a colormap as legend.

- **bestmatches** (*bool.*) – Optional parameter to plot best matching units.
- **bestmatchcolors** (*list of int.*) – Optional parameter to specify the color of each best matching unit.
- **labels** (*list of str.*) – Optional parameter to specify the label of each point.
- **zoom** (*((int, int), (int, int))*) – Optional parameter to zoom into a region on the map. The first two coordinates of the tuple are the row limits, the second tuple contains the column limits.
- **filename** (*str.*) – If specified, the plot will not be shown but saved to this file.

view_similarity_matrix (*data=None, labels=None, figsize=None, filename=None*)

Plot the similarity map according to the activation map

Parameters

- **data** (*numpy.array*) – Optional parameter for data points to calculate the similarity with
- **figsize** (*(int, int)*) – Optional parameter to specify the size of the figure.
- **labels** (*list of str.*) – Optional parameter to specify the label of each point.
- **filename** (*str.*) – If specified, the plot will not be shown but saved to this file.

view_umatrix (*figsize=None, colormap=<Mock name=cm.Spectral_r, colorbar=False, bestmatches=False, bestmatchcolors=None, labels=None, zoom=None, filename=None*)

Plot the U-matrix of the trained map.

Parameters

- **figsize** (*(int, int)*) – Optional parameter to specify the size of the figure.
- **colormap** (*matplotlib.colors.Colormap*) – Optional parameter to specify the color map to be used.
- **colorbar** (*bool.*) – Optional parameter to include a colormap as legend.
- **bestmatches** (*bool.*) – Optional parameter to plot best matching units.
- **bestmatchcolors** (*list of int.*) – Optional parameter to specify the color of each best matching unit.
- **labels** (*list of str.*) – Optional parameter to specify the label of each point.
- **zoom** (*((int, int), (int, int))*) – Optional parameter to zoom into a region on the map. The first two coordinates of the tuple are the row limits, the second tuple contains the column limits.
- **filename** (*str.*) – If specified, the plot will not be shown but saved to this file.

C

`cluster()` (Somoclu method), [20](#)

G

`get_bmus()` (Somoclu method), [20](#)

`get_surface_state()` (Somoclu method), [20](#)

L

`load_bmus()` (Somoclu method), [20](#)

`load_codebook()` (Somoclu method), [20](#)

`load_umatrix()` (Somoclu method), [20](#)

S

Somoclu (built-in class), [19](#)

T

`train()` (Somoclu method), [20](#)

U

`update_data()` (Somoclu method), [21](#)

V

`view_activation_map()` (Somoclu method), [21](#)

`view_component_planes()` (Somoclu method), [21](#)

`view_similarity_matrix()` (Somoclu method), [22](#)

`view_umatrix()` (Somoclu method), [22](#)