Somoclu Python Documentation

Release 1.5

Peter Wittek, Shi Chao Gao

September 30, 2015

CONTENTS

CHAPTER

ONE

INTRODUCTION

Somoclu is a massively parallel implementation of self-organizing maps. It relies on OpenMP for multicore execution, MPI for distributing the workload, and it can be accelerated by CUDA. A sparse kernel is also included, which is useful for training maps on vector spaces generated in text mining processes. 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.
- 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.

1.1 Copyright and License

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.

CHAPTER TWO

DOWNLOAD AND INSTALLATION

The entire package for is available as a gzipped tar file from 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.

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, follow the standard procedure for installing Python modules:

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

2.1.2 Build on Mac OS X

Before installing using pip, 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 your CUDA is installed elsewhere than /usr/local/cuda, you cannot directly install the module from PyPI. Please download the source distribution from PyPI. Open the setup.py file in an editor and modify the path to your CUDA installation directory:

```
cuda_dir = /path/to/cuda
```

Then run the install command

```
$ 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.

CHAPTER

THREE

EXAMPLES

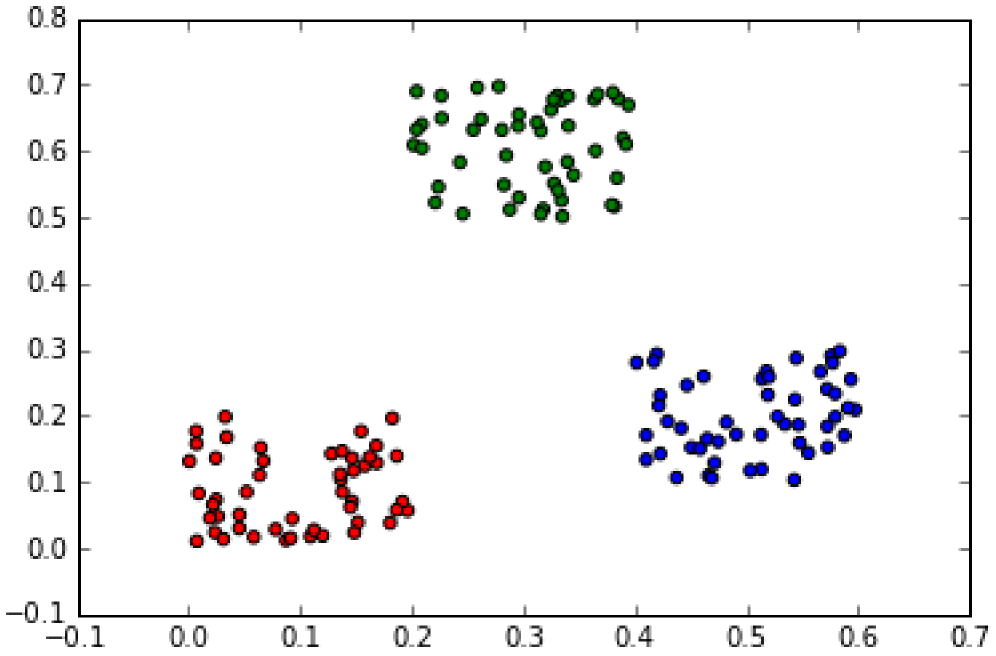
Self-organizing maps are computationally intensive to train, especially if the original space is high-dimensional or the map is large. Very large maps where the number of neurons is at least five times the number of data points are sometimes called emergent-self organizing maps – these are especially 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
import somoclu
%matplotlib inline
```

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

```
c1 = np.random.rand(50, 2)/5
c2 = (0.2, 0.5) + np.random.rand(50, 2)/5
c3 = (0.4, 0.1) + np.random.rand(50, 2)/5
data = np.float32(np.concatenate((c1, c2, c3)))
colors = ["red"] * 50
colors.extend(["green"] * 50)
colors.extend(["blue"] * 50)
plt.scatter(data[:, 0], data[:, 1], c=colors)
labels = range(150)
```



3.1 Planar maps

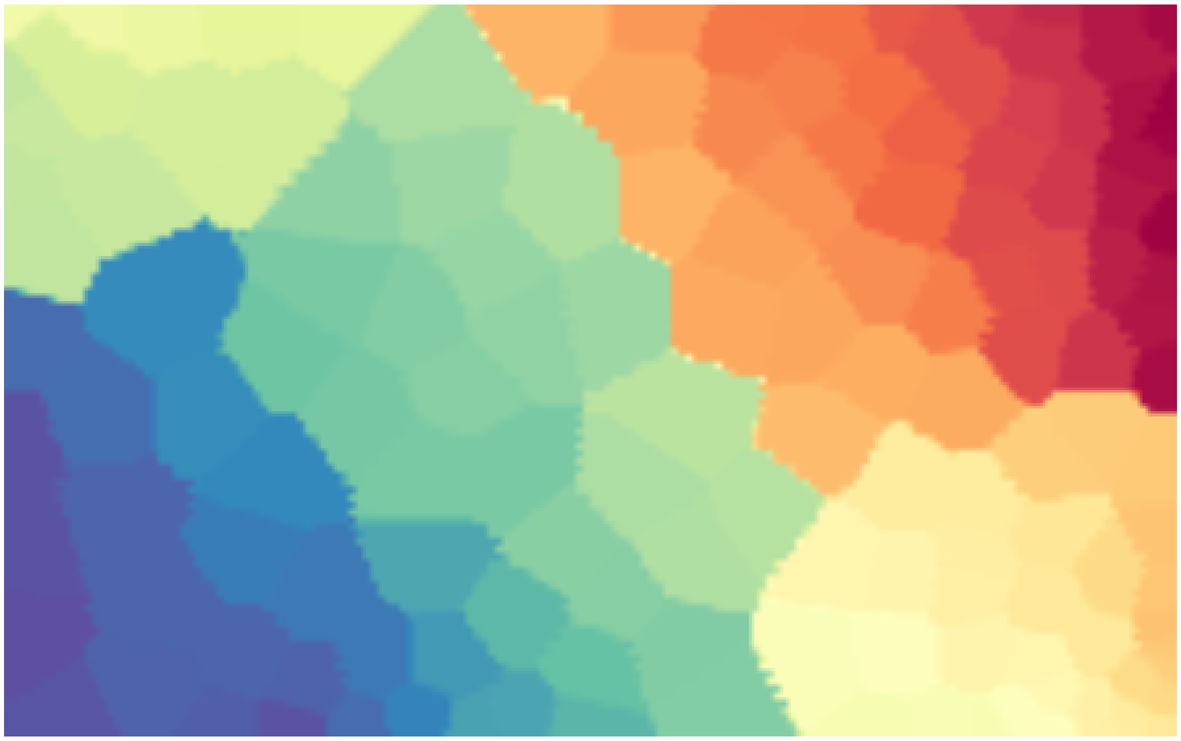
We train Somoclu with default parameter settings, asking for a large map that qualifies as an emergent self-organizing map for this data:

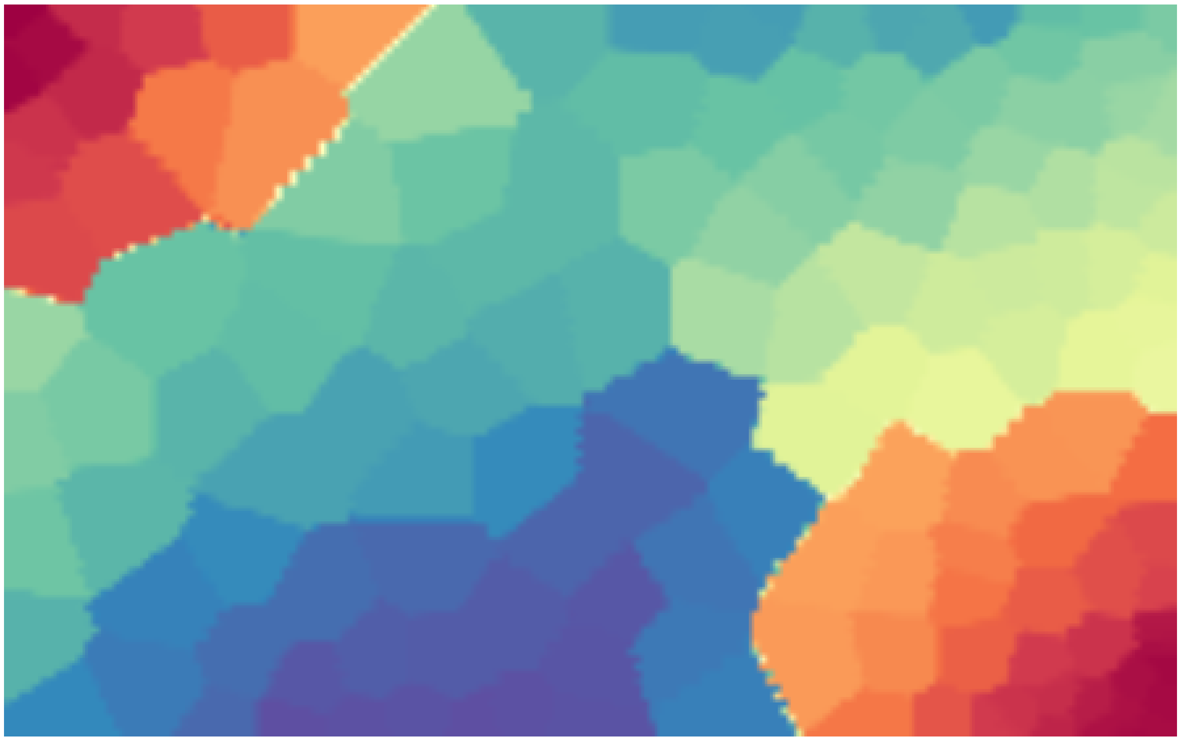
```
n_rows, n_columns = 100, 160
som = somoclu.Somoclu(n_columns, n_rows, data=data)
%time som.train()
```

```
CPU times: user 6.62 s, sys: 10 ms, total: 6.63 s
Wall time: 4.76 s
```

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

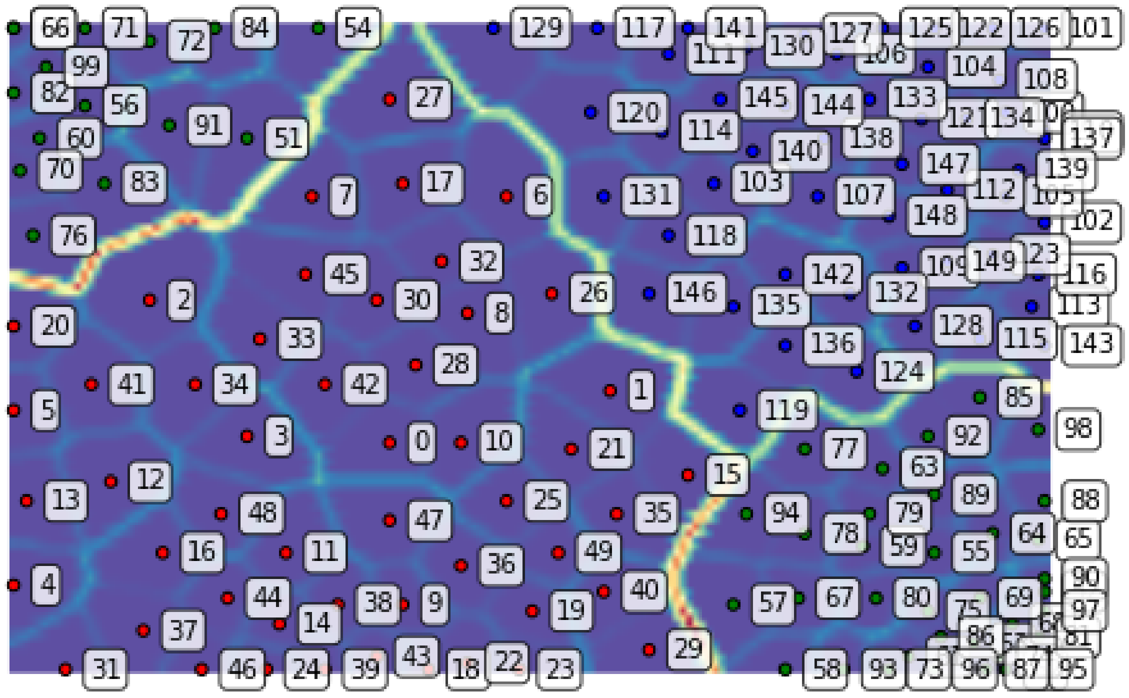
```
som.view_component_planes()
```





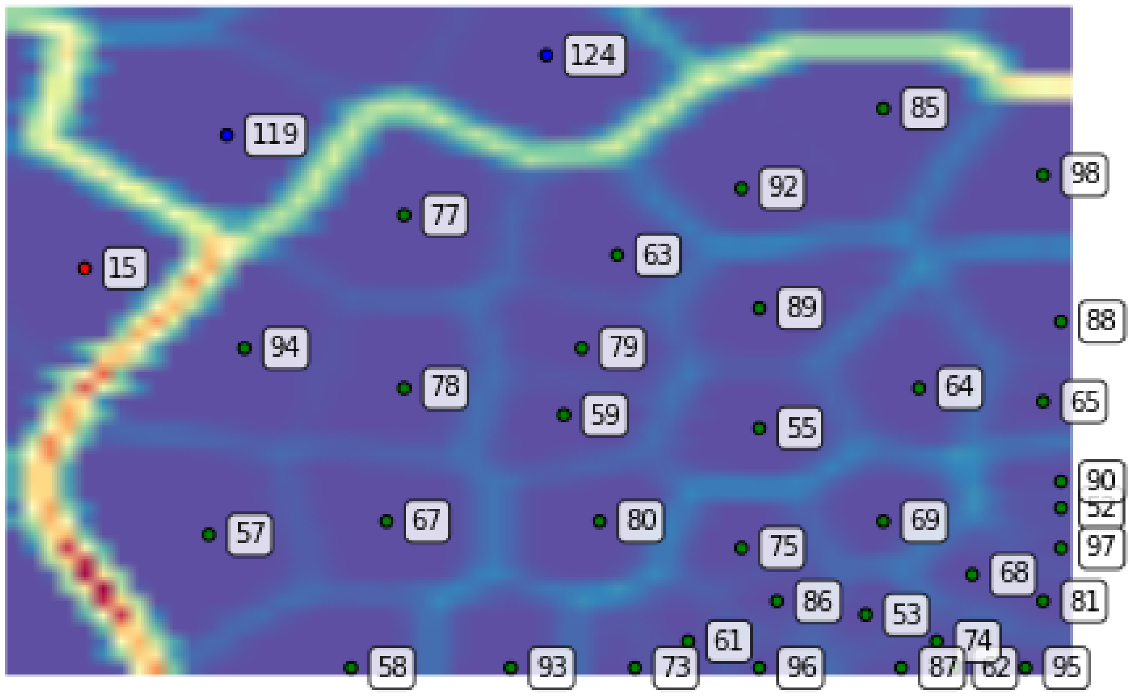
We can plot the U-Matrix, together with the best matching units for each data point. We color code the units with the classes of the data points and also add the labels of the data points.

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



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

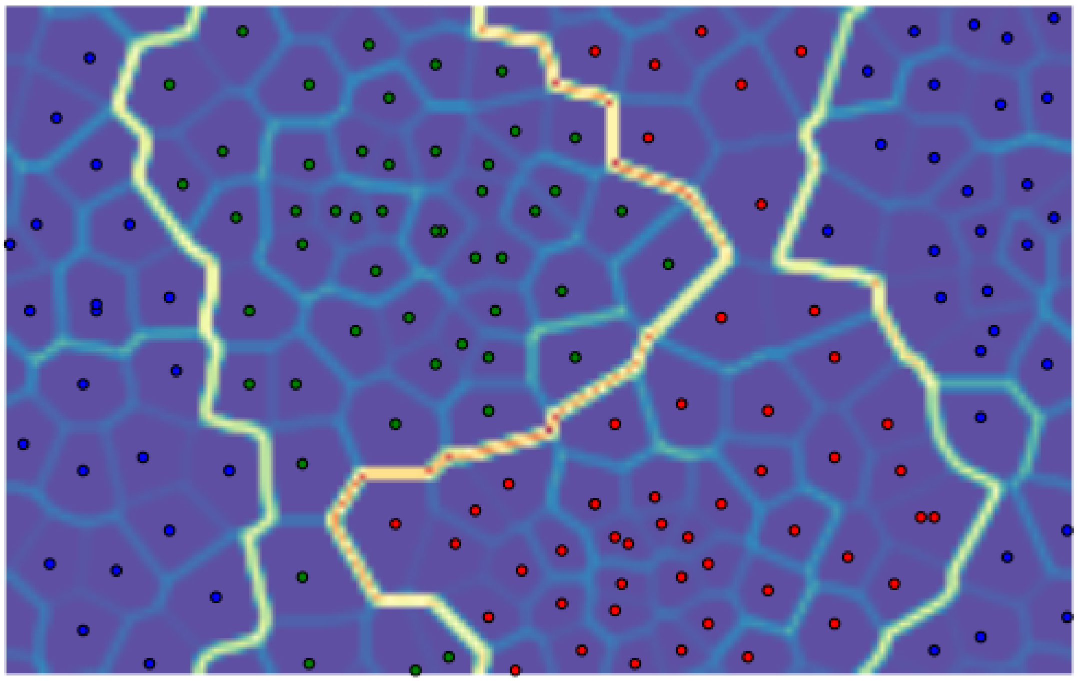
```
som.view_umatrix(bestmatches=True, bestmatchcolors=colors, labels=labels,
                 zoom=((50, n_rows), (100, n_columns)))
```



3.2 Toroid topology, hexagonal grid

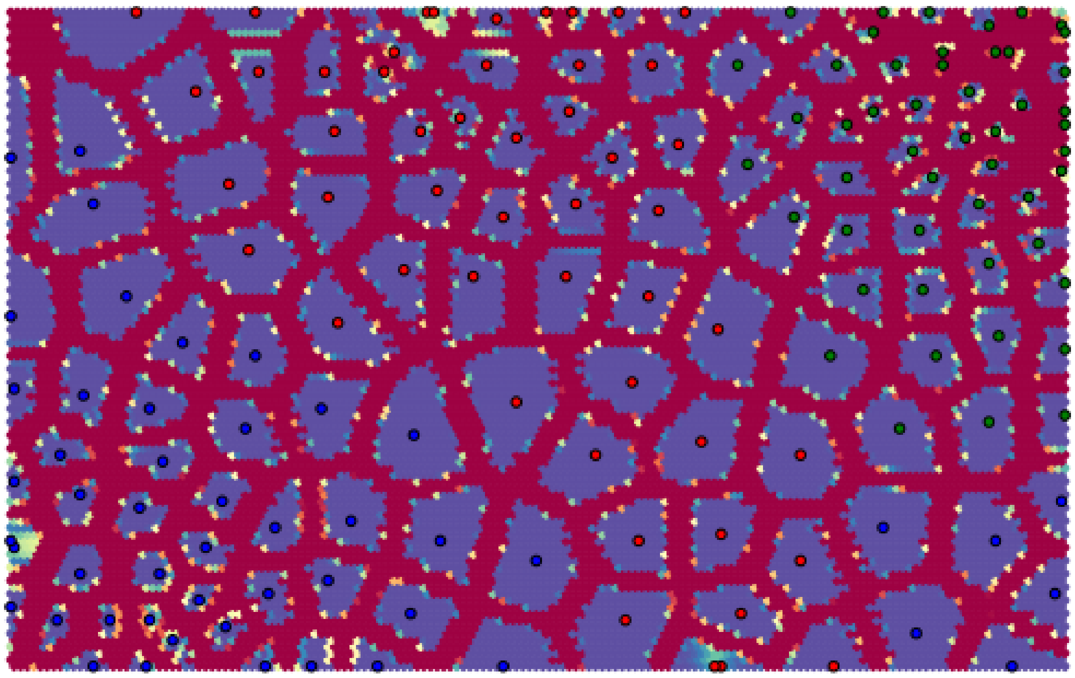
We can repeat the above with a toroid topology by specifying the map type as follows:

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



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



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

3.3 Evolving maps

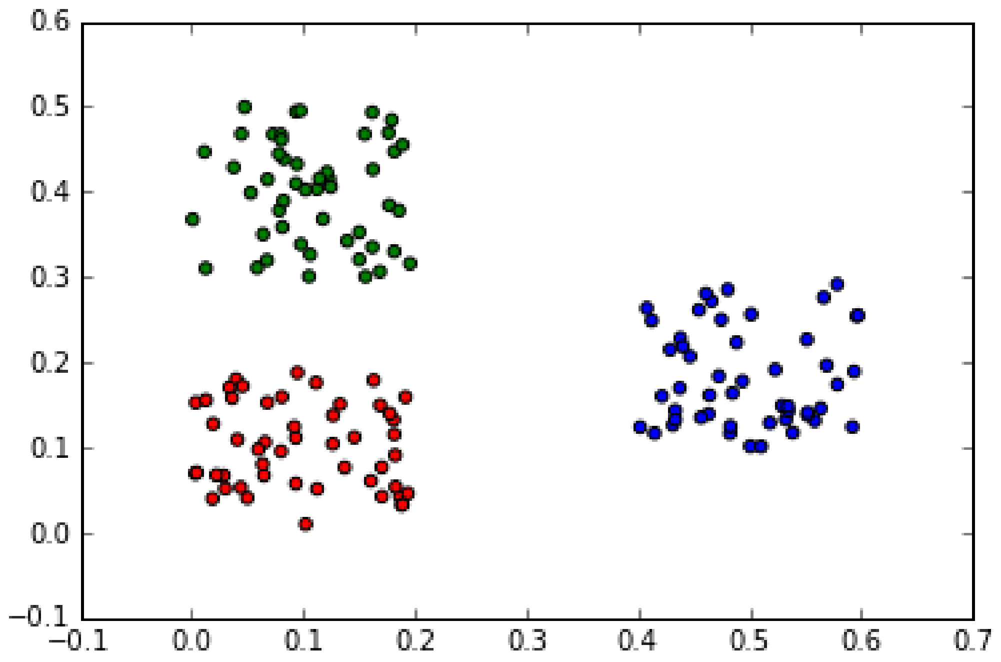
One of the great advantages of self-organizing maps is that they are incremental, they can be updated with new data. This is especially interesting if the data points retain their old label, that is, the properties of the vectors change in the high-dimensional space. Let us train again a toroid rectangular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

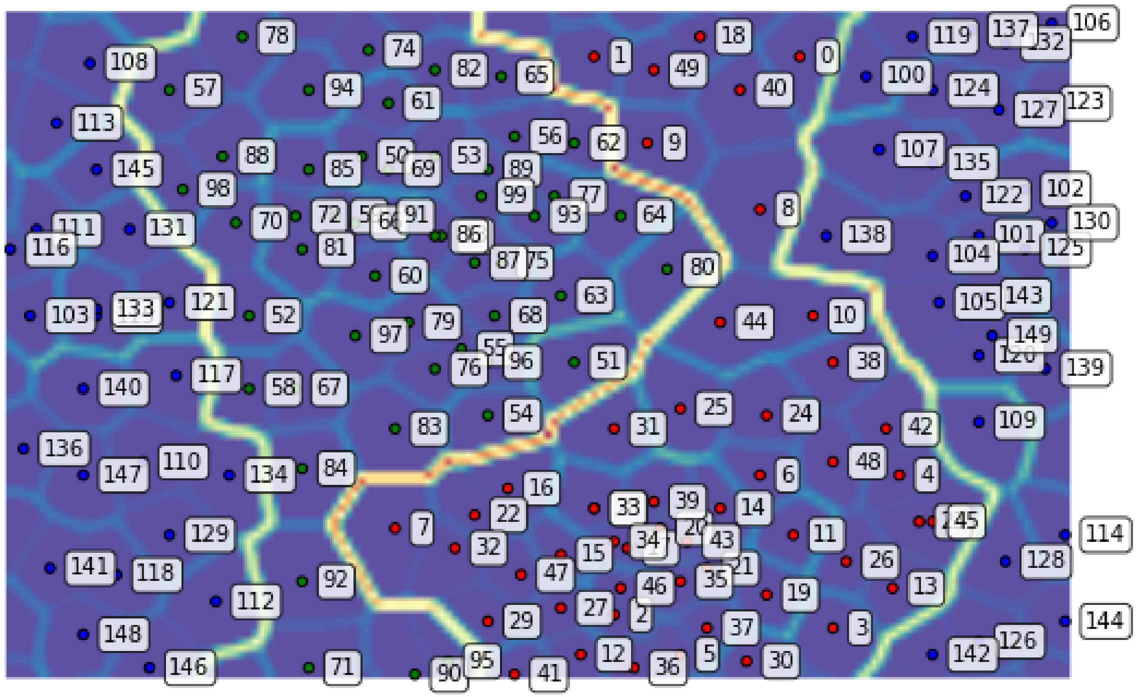
```
c2_shifted = c2 - 0.2
updated_data = np.float32(np.concatenate((c1, c2_shifted, c3)))
plt.scatter(updated_data[:,0], updated_data[:,1], c=colors)
```

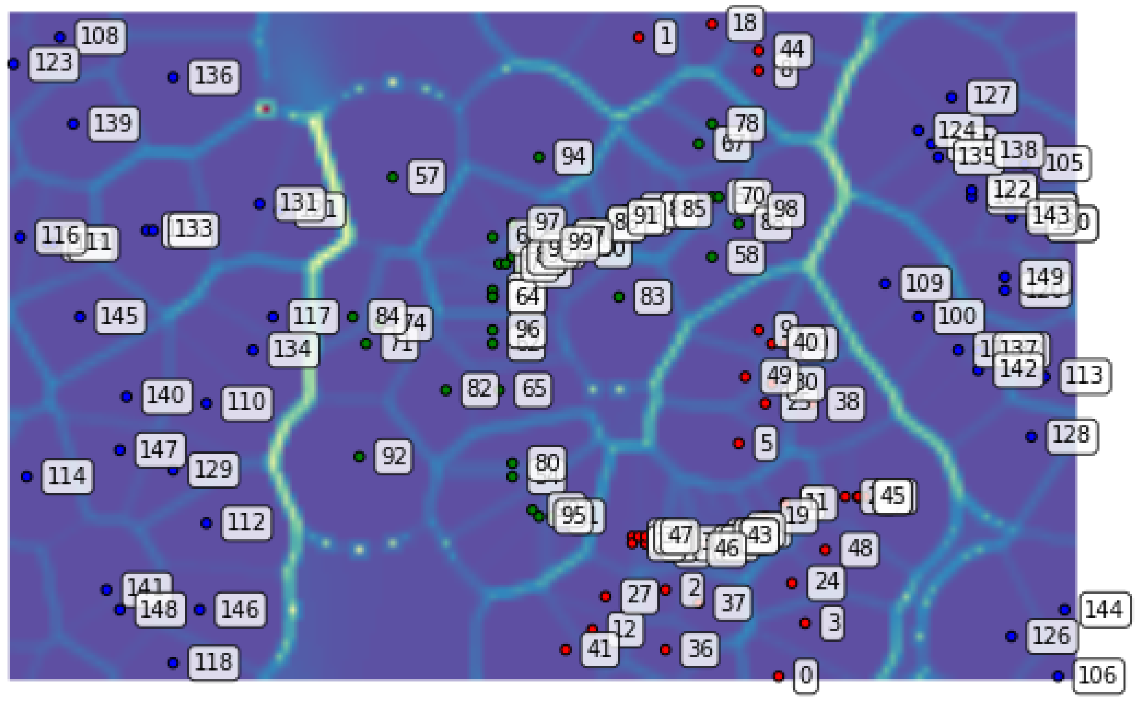
```
<matplotlib.collections.PathCollection at 0x7fb962be9908>
```



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)
```





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.

CHAPTER

FOUR

FUNCTION REFERENCE

4.1 Somoclu Class

class `somoclu.`**`Somoclu`**(*n_columns*, *n_rows*, *data=None*, *initialcodebook=None*, *kerneltype=0*, *maptype='planar'*, *gridtype='rectangular'*, *compactsupport=False*)

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. Default: False.

`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`** (*int.*) – 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`** (*int.*) – 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`** (*int.*) – The initial learning scale. Default value: 0.1.
- **`scaleN`** (*int.*) – 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_component_planes`(*dimensions=None*, *figsize=None*, *colormap=<matplotlib.colors.LinearSegmentedColormap object>*, *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_umatrix`(*figsize=None*, *colormap=<matplotlib.colors.LinearSegmentedColormap object>*, *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.