Contents

1 Getting Started 1
  1.1 Installation ................................................................. 1
  1.2 Self-Organizing Maps ..................................................... 1
  1.3 Neural Gas .................................................................... 1
  1.4 Growing Neural Gas ....................................................... 1

2 Reference 3
  2.1 kohonen.kohonen.Timeseries ............................................ 4
  2.2 kohonen.kohonen.ConstantTimeseries ................................. 4
  2.3 kohonen.kohonen.ExponentialTimeseries ............................. 5
  2.4 kohonen.kohonen.Parameters ............................................ 5
  2.5 kohonen.kohonen.GrowingGasParameters ............................ 6
  2.6 kohonen.kohonen.Map ..................................................... 6
  2.7 kohonen.kohonen.Gas ...................................................... 8
  2.8 kohonen.kohonen.GrowingGas .......................................... 10
  2.9 kohonen.kohonen.Filter .................................................. 11

Python Module Index 13
The `kohonen` package is a set vector quantizers in the style of the Kohonen Self-Organizing Map.

### 1.1 Installation

It's easiest to install the `kohonen` package using `pip`:

```
pip install kohonen
```

The command will automatically install all dependencies.

### 1.2 Self-Organizing Maps

### 1.3 Neural Gas

### 1.4 Growing Neural Gas
Basic self-organizing map implementation.

This module contains the following Kohonen map implementations:

- Map. A standard rectangular N-dimensional Kohonen map.
  - Gas. A vector quantizer that does not have a fixed topology. Neurons in a gas are sorted for updates based on their distance from the cue, with the sort order defining a topology for each cue presentation.
    - GrowingGas. A Gas-based quantizer that can add neurons dynamically to explain high-error areas of the input space.
- Filter. A wrapper over an underlying Map instance that maintains an explicit estimate of the likelihood of each neuron.

These are tested using the kohonen_test.py file in this source distribution.

Because they have a grid topology, Map objects have some cool visualization options, including Map.neuron_colormap and Map.distance_heatmap. These require the Python Image Library.

There is also a collection of distance metrics:

- cosine_metric. A callable that calculates the cosine distance between a cue and each neuron in a Kohonen Map.
- euclidean_metric. A callable that calculates the Euclidean distance between a cue and each neuron in a Kohonen Map.
- manhattan_metric. A callable that calculates the Manhattan distance between a cue and each neuron in a Kohonen Map.

There are also some small utility classes for modeling time series values:

- TIMESERIES. A callable that takes no arguments and returns a value that might vary over time. Each call to the function will generally return a unique value (though this is not necessary).
  - ExponentialTimeseries. A callable that takes no arguments and returns an exponentially decreasing (or increasing) series of values, dependent on the parameters passed in at construction time.
  - etc.

These distance functions and time series objects are generally used to regulate the learning parameters in Kohonen Map objects.

```python
kohonen.kohonen.Timeseries() Represents some sort of value that changes over time.
kohonen.kohonen.ConstantTimeseries([k]) This timeseries just returns a constant value.
kohonen.kohonen.LinearTimeseries
```

Continued on next page
2.1 kohonen.kohonen.Timeseries

class kohonen.kohonen.Timeseries
    Represents some sort of value that changes over time.

Methods

    __call__()  Call this timeseries.
    reset()     Reset the time for this series.

    __init__()
        Set up this timeseries.

Methods

    __init__()  Set up this timeseries.
    reset()     Reset the time for this series.

reset()
    Reset the time for this series.

2.2 kohonen.kohonen.ConstantTimeseries

class kohonen.kohonen.ConstantTimeseries (k=1)
    This timeseries just returns a constant value.

Methods

    __call__()  Return the constant.
    reset()     Reset the time for this series.

    __init__(k=1)
        Set up this series with a constant value.
Methods

```python
__init__(k)  # Set up this series with a constant value.
reset()      # Reset the time for this series.
```

2.3 kohonen.kohonen.ExponentialTimeseries

class kohonen.kohonen.ExponentialTimeseries(rate=-1, initial=1, final=0)

Represents an exponential decay process.

Methods

```python
__call__()  # Return an exponentially-decreasing series of values.
reset()     # Reset the time for this series.
```

```python
__init__(rate=-1, initial=1, final=0)

Create a new exponential timeseries object.
```

2.4 kohonen.kohonen.Parameters

class kohonen.kohonen.Parameters(dimension=None, shape=None, metric=None,
                                 learning_rate=None, neighborhood_size=None,
                                 noise_variance=None)

We are plain old data holding self-organizing map parameters.

```python
__init__(dimension=None, shape=None, metric=None, learning_rate=None, neighborhood_size=None,
         noise_variance=None)

This class holds standard parameters for self-organizing maps.
```

- **dimension:** The length of a neuron vector in a Map or a Gas.
- **shape:** The shape of the neuron topology in whatever Map or Gas we are building.
- **metric:** The distance metric to use when comparing cues to neurons in the map. Defaults to euclidean_metric.
- **learning_rate:** This parameter determines the time course of the learning rate for a Map. This parameter should be a callable that takes no arguments and returns a floating point value for the learning rate.

If this parameter is None, a default learning rate series will be used, equivalent to ExponentialTimeseries(-1e-3, 1, 0.2).

If this parameter is a numeric value, it will be used as the constant value for the learning rate: ConstantTimeseries(value).
**neighborhood_size**: Like the learning rate, this parameter determines the time course of the neighborhood size parameter. It should be a callable that takes no arguments and returns a neighborhood size for storing each cue.

If this is None, a default neighborhood size series will be used. The initial size will be the maximum of the dimensions given in shape, and the decay will be \(-1e-3\): `ExponentialTimeSeries(-1e-3, max(shape), 1)`.

If this is a floating point value, it will be used as a constant neighborhood size: `ConstantTimeSeries(value)`.

**noise_variance**: Like the learning rate and neighborhood size, this should be a factory for creating a callable that creates noise variance values.

If this is None, no noise will be included in the created Maps.

If this parameter is a number, it will be used as a constant noise variance.

**Methods**

```python
__init__([dimension, shape, metric, ...])  This class holds standard parameters for self-organizing maps.
```

## 2.5 kohonen.kohonen.GrowingGasParameters

class kohonen.kohonen.GrowingGasParameters(growth_interval=2, max_connection_age=5, error_decay=0.99, neighbor_error_decay=0.99, **kwargs)

Parameters for Growing Neural Gases.

```python
__init__(growth_interval=2, max_connection_age=5, error_decay=0.99, neighbor_error_decay=0.99, **kwargs)
```

**Methods**

```python
__init__(growth_interval, ...]
```

## 2.6 kohonen.kohonen.Map

class kohonen.kohonen.Map(params)

Basic implementation of a rectangular N-dimensional self-organizing map.

A Self-Organizing or Kohonen Map (henceforth just Map) is a group of lightweight processing units called neurons, which are here implemented as vectors of real numbers. Neurons in a Map are arranged in a specific topology, so that a given neuron is connected to a small, specific subset of the overall neurons in the Map. In addition, the Map uses a distance metric (e.g., Euclidean distance) for computing similarity between neurons and cue vectors, as described below.

The Map accepts cues—vectors of real numbers—as inputs. In standard Map usage, cues represent some data point of interest. Normally applications of Maps use input vectors like the activation patterns for an array of sensors, term frequency vectors for a document, etc. Cues are stored in the Map as follows: First, a “winner” neuron \(w\) is chosen from the Map, and, second, the neurons in the Map topologically near \(w\) are altered so that they become closer to the cue. Each of these steps is described briefly below.
For the first step, the Map computes the distance between the cue and each of the Map neurons using its metric. The neuron closest to the cue under this metric is declared the “winner” w. Alternatively, the winner can be selected probabilistically based on the overall distance landscape.

Next, the Map alters the neurons in the neighborhood of w, normally using some function of the difference between the cue and the neuron being modified. The weight of the alteration decreases exponentially as the topological distance from w increases. The learning rule for a neuron n is

\[ n += \eta \exp\left(-\frac{d^2}{\sigma^2}\right) (c - n) \]

where \( \eta \) is the learning rate, \( \sigma \) is called the neighborhood size, \( d \) is the topological distance between n and w, and \( c \) is the cue vector being stored in the map. \( \eta \) and \( \sigma \) normally decrease in value over time, to take advantage of the empirical machine learning benefits of simulated annealing.

The storage mechanism in a Map has the effect of grouping cues with similar characteristics into similar areas of the Map. Because the winner—and its neighborhood—are altered to look more like the cues that they capture, the winner for a given cue will tend to win similar inputs in the future. This tends to cluster similar Map inputs, and can lead to interesting data organization patterns.

Attributes

| shape |

Methods

| distance_heatmap(cue[, axes, lower, upper]) | Return an image representation of the distance to a cue. |
| distances(cue) | Get the distance of each neuron in the Map to a particular cue. |
| flat_to_coords(i) | Given a flattened index, convert it to a coordinate tuple. |
| learn(cue[, weights, distances]) | Add a new cue vector to the Map, moving neurons as needed. |
| neuron(coords) | Get the current state of a specific neuron. |
| neuron_heatmap([axes, lower, upper]) | Return an image representation of this Map. |
| reset([f]) | Reset the neurons and timeseries in the Map. |
| sample(n) | Get a sample of n neuron coordinates from the map. |
| smallest(distances) | Get the index of the smallest element in the given distances array. |
| weights(distances) | Get an array of learning weights to use for storing a cue. |
| winner(cue) | Get the coordinates of the most similar neuron to the given cue. |

**__init__(params)**

Initialize this Map.

Methods

| __init__(params) | Initialize this Map. |
| distance_heatmap(cue[, axes, lower, upper]) | Return an image representation of the distance to a cue. |
| distances(cue) | Get the distance of each neuron in the Map to a particular cue. |
| flat_to_coords(i) | Given a flattened index, convert it to a coordinate tuple. |
| learn(cue[, weights, distances]) | Add a new cue vector to the Map, moving neurons as needed. |
| neuron(coords) | Get the current state of a specific neuron. |
| neuron_heatmap([axes, lower, upper]) | Return an image representation of this Map. |
| reset([f]) | Reset the neurons and timeseries in the Map. |
Table 2.12 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample(n)</td>
<td>Get a sample of n neuron coordinates from the map.</td>
</tr>
<tr>
<td>smallest(distances)</td>
<td>Get the index of the smallest element in the given distances array.</td>
</tr>
<tr>
<td>weights(distances)</td>
<td>Get an array of learning weights to use for storing a cue.</td>
</tr>
<tr>
<td>winner(cue)</td>
<td>Get the coordinates of the most similar neuron to the given cue.</td>
</tr>
</tbody>
</table>

**Attributes**

```
shape
```

```
distance_heatmap(cue, axes=(0, 1), lower=None, upper=None)
```

Return an image representation of the distance to a cue.

```
distances(cue)
```

Get the distance of each neuron in the Map to a particular cue.

```
flat_to_coords(i)
```

Given a flattened index, convert it to a coordinate tuple.

```
learn(cue, weights=None, distances=None)
```

Add a new cue vector to the Map, moving neurons as needed.

```neuron(coords)
```

Get the current state of a specific neuron.

```neuron_heatmap(axes=(0, 1), lower=None, upper=None)
```

Return an image representation of this Map.

```
reset(f=None)
```

Reset the neurons and timeseries in the Map.

```
f: A callable that takes a neuron coordinate and returns a value for that neuron. Defaults to random values from the standard normal.
```

```
sample(n)
```

Get a sample of n neuron coordinates from the map.

```
The returned values will be flat indices; use flat_to_coords to convert them to neuron indices.
```

```
smallest(distances)
```

Get the index of the smallest element in the given distances array.

```
Returns a flat index; use flat_to_coords to convert this to a neuron index.
```

```
weights(distances)
```

Get an array of learning weights to use for storing a cue.

```
winner(cue)
```

Get the coordinates of the most similar neuron to the given cue.

```
Returns a flat index; use flat_to_coords to convert this to a neuron index.
```

**2.7 kohonen.kohonen.Gas**

```python
class kohonen.kohonen.Gas(params)
```

A neural Gas is a topologically unordered collection of neurons.
Learning takes place in the Gas by ordering the neurons according to their distance from each cue that is presented. Neurons are updated using this sorted order, with an exponentially decreasing weight for neurons that are further (in sort order) from the cue.

Attributes

shape

Methods

distance_heatmap(cue[, axes, lower, upper])
distances(cue)
flat_to_coords(i)
learn(cue[, weights, distances])
neuron(coords)
neuron_heatmap([axes, lower, upper])
reset([f])
sample(n)
smallest(distances)
weights(distances)
winner(cue)

distance_heatmap(cue[, axes, lower, upper])

Return an image representation of the distance to a cue.

distances(cue)

Get the distance of each neuron in the Map to a particular cue.

flat_to_coords(i)

Given a flattened index, convert it to a coordinate tuple.

learn(cue[, weights, distances])

Add a new cue vector to the Map, moving neurons as needed.

neuron(coords)

Get the current state of a specific neuron.

neuron_heatmap([axes, lower, upper])

Return an image representation of this Map.

reset([f])

Reset the neurons and timeseries in the Map.

sample(n)

Get a sample of n neuron coordinates from the map.

smallest(distances)

Get the index of the smallest element in the given distances array.

weights(distances)

Get the coordinates of the most similar neuron to the given cue.

winner(cue)

Get the coordinates of the most similar neuron to the given cue.

__init__(params)

Initialize this Gas. A Gas must have a 1D shape.

Methods

__init__(params)

Initialize this Gas.

distance_heatmap(cue[, axes, lower, upper])

distances(cue)

Return an image representation of the distance to a cue.

given a flattened index, convert it to a coordinate tuple.

learn(cue[, weights, distances])

Add a new cue vector to the Map, moving neurons as needed.

neuron(coords)

Get the current state of a specific neuron.

neuron_heatmap([axes, lower, upper])

Return an image representation of this Map.

reset([f])

Reset the neurons and timeseries in the Map.

sample(n)

Get a sample of n neuron coordinates from the map.

smallest(distances)

Get the index of the smallest element in the given distances array.

weights(distances)

Get the coordinates of the most similar neuron to the given cue.

winner(cue)

Get the coordinates of the most similar neuron to the given cue.
# 2.8 kohonen.kohonen.GrowingGas

A Growing Neural Gas uses a variable number of variable-topology neurons.

In essence, a GNG is similar to a standard Gas, but there is additional logic in this class for adding new neurons to better explain areas of the sample space that currently have large error.

## Attributes

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>shape</code></td>
<td>Return the shape of the Growing Gas.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>distance_heatmap</code></td>
<td>Return an image representation of the distance to a cue.</td>
</tr>
<tr>
<td><code>distances</code></td>
<td>Get the distance of each neuron in the Map to a particular cue.</td>
</tr>
<tr>
<td><code>flat_to_coords</code></td>
<td>Given a flattened index, convert it to a coordinate tuple.</td>
</tr>
<tr>
<td><code>learn</code></td>
<td>Store a cue in the gas.</td>
</tr>
<tr>
<td><code>neighbors</code></td>
<td>Get the current state of a specific neuron.</td>
</tr>
<tr>
<td><code>neuron_heatmap</code></td>
<td>Return an image representation of this Map.</td>
</tr>
<tr>
<td><code>reset</code></td>
<td>Reset the neurons and timeseries in the Map.</td>
</tr>
<tr>
<td><code>sample</code></td>
<td>Get a sample of n neuron coordinates from the map.</td>
</tr>
<tr>
<td><code>smallest</code></td>
<td>Get the index of the smallest element in the given distances array.</td>
</tr>
<tr>
<td><code>weights</code></td>
<td>Get the coordinates of the most similar neuron to the given cue.</td>
</tr>
</tbody>
</table>

### `__init__` (params)

Initialize a new Growing Gas with parameters.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code> (params)</td>
<td>Initialize a new Growing Gas with parameters.</td>
</tr>
<tr>
<td><code>distance_heatmap</code></td>
<td>Return an image representation of the distance to a cue.</td>
</tr>
<tr>
<td><code>distances</code></td>
<td>Get the distance of each neuron in the Map to a particular cue.</td>
</tr>
<tr>
<td><code>flat_to_coords</code></td>
<td>Given a flattened index, convert it to a coordinate tuple.</td>
</tr>
<tr>
<td><code>learn</code></td>
<td>Store a cue in the gas.</td>
</tr>
<tr>
<td><code>neighbors</code></td>
<td>Get the current state of a specific neuron.</td>
</tr>
<tr>
<td><code>neuron_heatmap</code></td>
<td>Return an image representation of this Map.</td>
</tr>
<tr>
<td><code>reset</code></td>
<td>Reset the neurons and timeseries in the Map.</td>
</tr>
<tr>
<td><code>sample</code></td>
<td>Get a sample of n neuron coordinates from the map.</td>
</tr>
<tr>
<td><code>smallest</code></td>
<td>Get the index of the smallest element in the given distances array.</td>
</tr>
<tr>
<td><code>weights</code></td>
<td>Get the coordinates of the most similar neuron to the given cue.</td>
</tr>
</tbody>
</table>

## Attributes
learn(cue, weights=None, distances=None)
    Store a cue in the gas.

2.9 kohonen.kohonen.Filter

class kohonen.kohonen.Filter(map, history=None)
    A Filter is an estimate of the probability density of the inputs.

    Attributes

    shape

    Methods

    distances(cue)
    flat_to_coords(i)
    learn(cue, **kwags)
    neuron(coords)
    reset([f])
    sample(n)
    smallest(distances)
    weights(distances)
    winner(cue)

    __init__(map[, history])  
    Initialize this Filter with an underlying Map implementation.

    history: A callable that returns values in the open interval (0, 1). These values determine how much new cues influence the activation state of the Filter.

    A 0 value would mean that no history is preserved (i.e. each new cue stored in the Filter completely determines the activity of the Filter) while a 1 value would mean that new cues have no impact on the activity of the Filter (i.e. the initial activity is the only activity that is ever used).

    Methods

    __init__(map, history)  
    Initialize this Filter with an underlying Map implementation.

    distances(cue)
    flat_to_coords(i)
    learn(cue, **kwags)
    neuron(coords)
    reset([f])
    sample(n)
Table 2.24 – continued from previous page

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>shape</td>
</tr>
</tbody>
</table>

smallest(distances)  
weights(distances)  
winner(cue)
k
kohonen.kohonen, 3
Symbols

__init__() (kohonen.kohonen.ConstantTimeseries method), 4
__init__() (kohonen.kohonen.ExponentialTimeseries method), 5
__init__() (kohonen.kohonen.Filter method), 11
__init__() (kohonen.kohonen.Gas method), 9
__init__() (kohonen.kohonen.GrowingGas method), 10
__init__() (kohonen.kohonen.GrowingGasParameters method), 6
__init__() (kohonen.kohonen.Map method), 7
__init__() (kohonen.kohonen.Parameters method), 5
__init__() (kohonen.kohonen.Timeseries method), 4

C
ConstantTimeseries (class in kohonen.kohonen), 4

D
distance_heatmap() (kohonen.kohonen.Map method), 8
distances() (kohonen.kohonen.Map method), 8

E
ExponentialTimeseries (class in kohonen.kohonen), 5

F
Filter (class in kohonen.kohonen), 11
flat_to_coords() (kohonen.kohonen.Map method), 8

G
Gas (class in kohonen.kohonen), 8
GrowingGas (class in kohonen.kohonen), 10
GrowingGasParameters (class in kohonen.kohonen), 6

K
kohonen.kohonen (module), 3

L
learn() (kohonen.kohonen.GrowingGas method), 11
learn() (kohonen.kohonen.Map method), 8

M
Map (class in kohonen.kohonen), 6

N
neuron() (kohonen.kohonen.Map method), 8
neuron_heatmap() (kohonen.kohonen.Map method), 8

P
Parameters (class in kohonen.kohonen), 5

R
reset() (kohonen.kohonen.Map method), 8
reset() (kohonen.kohonen.Timeseries method), 4

S
sample() (kohonen.kohonen.Map method), 8
smallest() (kohonen.kohonen.Map method), 8

T
Timeseries (class in kohonen.kohonen), 4

W
weights() (kohonen.kohonen.Map method), 8
winner() (kohonen.kohonen.Map method), 8