# Contents

1 First steps 3

2 Reference 5

3 Licensing 7
   3.1 Getting started ................................................................. 7
   3.2 Advanced Topics ............................................................... 12
   3.3 Auto-generated reference .................................................. 12

Python Module Index 19
Neurons is a simple simulation tool for neurons.

It currently supports the SRM model for calculating spike trains, and the STDP model for learning synaptic weights. It originated from a student project at the Chair for Theoretical Biophysics at TU München in summer term 2014.
Read this if you are new to the project.

- **Installation:** Linux | Windows
- **Tutorials:** Your first SRM network | Visualizing the network
- **Topics:** Project Overview | Spiking | Learning
- **Contribute:** Write Bug Reports | Request new features | Extend the Code
We have a *auto-generated reference*.
CHAPTER 3

Licensing

This project is licensed under the BSD-2-Clause License.
The source code is freely available at Github.
Used open-source projects:

    python 3 | pytest | numpy | matplotlib | sphinx

3.1 Getting started

Get started using the installation guides and the tutorials.

3.1.1 Ubuntu Installation guide

This is a step-by-step installation tutorial for Ubuntu.

Standard installation

First install the requirements:

$ sudo apt-get install python3 python3-numpy python3-matplotlib

Next, download and unzip the zip-package.
Change into the directory, and run the install script

$ cd neurons-master
$ python3 setup.py install --user

Note: This copies all needed data into .local in your home directory.

Now you can test if you can import the neurons package:

$ python3
>>> import neurons

This shouldn’t lead to any error messages. Congratulations, you are done.
Installation using git

First install the requirements:

$ sudo apt-get install python3 python3-numpy python3-matplotlib

Next, clone the git repository:

$ git clone https://github.com/johannesmik/neurons.git

Change into the directory, and run the install script

$ cd neurons
$ python3 setup.py install --user

**Note:** This copies all needed data into .local in your home directory.

Now you can test if you can import the `neurons` package:

$ python3
>>> import neurons

This shouldn’t lead to any error messages. Congratulations, you are done.

**Feedback**

The installation does not work? Please write a bug report.

### 3.1.2 Windows Installation Guide

First install the requirements:

1. Python3
2. Numpy
3. Matplotlib

You can also use pre-packaged distributions like Anaconda.

Next, download and unzip the zip-package into a convenient location.

Open a terminal (Shortcut: Windows + R). Change into the directory you just unzipped and run the install script

$ cd neurons-master
$ python3 setup.py install --user

**Note:** This copies all needed data into in your home directory.

Now you can test if you can import the `neurons` package:

$ python3
>>> import neurons

This shouldn’t lead to any error messages. Congratulations, you are done.
Feedback

The installation does not work? Please write a bug report.

3.1.3 Tutorial: Your first SRM network

In this first tutorial we let three SRM neurons spike.
There are two input neurons with predefined spikes, and they will excite one output neuron.

What do neurons do?

Simply said, neurons send out short peaks of current, called spikes, and collect spikes from other neurons.

A neuron sends out spikes on it’s axon and it collects the spikes from other neurons using it’s many dendrites. The axon is connected to the dendrites by a synapse, which transmits the spike chemically. Not every synapse transmits the spike equally good (depending on many factors), some strengthen. We will call this effect the weight of a synapse. Later we will see that neural networks can learn. This is mainly because of the different synaptic weights!

Importing the needed libraries

First thing we have to do is to import all the libraries we need:

```python
import numpy as np
import neurons.spiking as spiking
```

Setting up the SRM model

The SRM (Spike-response model) is a model for neurons. New spikes are generated if the incoming current exceeds a certain threshold.

```python
model = spiking.SRM(neurons=3, threshold=1, t_current=0.3, t_membrane=20, nu_reset=5)
```

This sets up a model of 3 neurons. A neuron generates a new spike if it exceeds a current of 1mV. The variables t_current, t_membrane are time constants used in the SRM model, and nu_reset resets the current after a spike (repolarization).

Note: We can say here, that the values are not ‘realistic.’ This is because our SRM implementation assumes a resting potential of 0mV! Real neurons have a resting potential of around -73mV (and a threshold of -54mV, and an action potential of 40mV).

But this should not further disturb us, because we assume that our model linearly shifts those real values. After all, it makes the computations easier if we assume a resting potential of 0mV.

Inter-neural weights

We want to connect our three neurons together like this:
Both, the first and the second neuron, are connected to the third neuron with a weight of 1. Note that the connections are directed. Neuron 1 is connected to Neuron 3 but not vice versa!

We can express this connection in a symmetric matrix: \( w = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \), where the entry \( w_{ij} \) represents the weight that neuron \( i \) is connected to neuron \( j \).

Therefore, we define a suitable Numpy array:

```python
weights = np.array([[0, 0, 1],
                    [0, 0, 1],
                    [0, 0, 0]])
```

Preparing a spiketrain

Without any initial spikes, our model wouldn’t do nothing!

So let’s define some spikes for our neurons:

```python
spiketrain = np.array([[0, 0, 1, 0, 0, 0, 1, 1, 0, 0],
                       [1, 0, 0, 0, 0, 0, 1, 1, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

This matrix means that the first neuron spikes at times of 2ms, 6ms and 7ms, and that the second neuron spikes at times of 0ms, 6ms and 7ms.

For the third neuron, we didn’t define any spikes at all. We expect that it will spike during the simulation.

Simulate the network

We prepared the SRM neurons, a spiketrain, and the inter-neural weights, so we are ready to simulate the net!

```python
for time in range(10):
    total_current = model.simulate(spiketrain, weights, time)
```

Simulate(spiketrain, weights, time) calculates the current at a time \( t \). It checks if any spikes occurred, and accordingly changes the spiketrain array in-place.
In the for-loop we calculate the current and maybe generate new spikes for every time from 0ms – 9ms.

**Enjoy the result**

We are nearly finished, now all that we want to do is to enjoy our result:

```python
print("Spiketrain:")
print(spiketrain)
```

Which gives us:

```
Spiketrain:
[[0 0 1 0 0 0 1 1 0 0]
 [1 0 0 0 0 1 1 0 0]
 [0 0 0 1 0 0 0 0 1 0]]
```

As we expected, our third neuron spiked (at times 4ms and 9ms), because it collected the spikes of the other two neurons.

**Conclusion**

As you see it didn’t take much to simulate our first SRM network: just under 10 lines of Python code.

In the next section we’ll see how we can visualize our results.

**Sourcecode**

Here you can see the whole source code for our little SRM network:

```python
import numpy as np
import neurons.spiking as spiking

model = spiking.SRM(neurons=3, threshold=1, t_current=0.3, t_membrane=20, nu_reset=5)
weights = np.array([[0, 0, 1], [0, 0, 1], [0, 0, 0]])
spiketrain = np.array([[0, 0, 1, 0, 0, 0, 1, 1, 0, 0],
                       [1, 0, 0, 0, 0, 1, 1, 0, 0],
                       [0, 0, 0, 0, 0, 0, 1, 1, 0, 0]])

for time in range(10):
    total_current = model.simulate(spiketrain, weights, time)

print("Spiketrain")
print(spiketrain)
```

**Questions**

Why don’t we define the weights at the initialization, but at every call of spiking?

**3.1.4 Tutorial: Visualizing the network**

Some words
3.2 Advanced Topics

3.2.1 Project overview

3.2.2 Spiking models

3.2.3 Learning Models

3.2.4 Write a bug report

If you have a Github account, please open an Issue at GithubIssues. Otherwise, please write an email to mikulas@in.tum.de

Please include following details to the Issue:

- Operating system
- Error messages

3.2.5 Write a feature request

test

3.2.6 Extend the source

3.3 Auto-generated reference

This reference is automatically generated from the source code.

3.3.1 spiking.py

SRM 0 model

class spiking.SRM(neurons, threshold, t_current, t_membrane, nu_reset, simulation_window_size=100,
                    verbose=False)

SRM_0 (Spike Response Model)

Neurons can have different t_current, t_membrane and nu_resets: Set those variables to 1D np.arrays of all the same size.

Parameters

- **neurons** – Number of neurons
- **threshold** – Spiking threshold
- **t_current** (*Float or Numpy Float Array*) – Current-time-constant (t_s)
- **t_membrane** – Membrane-time-constant (t_m)
- **nu_reset** – Reset constant
- **simulation_window_size** – Only look at the n last spikes
- **verbose** –
Returns None

**eps**(s)
Evaluate the Epsilon function:

\[
\epsilon(s) = \frac{1}{1 - \frac{s}{\tau_m}} \left( \exp\left(\frac{s}{\tau_m}\right) - \exp\left(-\frac{s}{\tau_c}\right) \right)
\]  

(3.1)

Returns a single Float Value if the time constants (current, membrane) are the same for each neuron. Returns a Float Vector with eps(s) for each neuron, if the time constants are different for each neuron.

Parameters s – Time s
Returns Function eps(s) at time s
Return type Float or Vector of Floats

**eps_matrix**(k, size)
Examples: Test
Parameters
• k –
• size –

Returns

**simulate**(spikes, weights, t)
Simulate one time step at time t. Changes the spiketrain in place at time t! Return the total current of all neurons.

Parameters
• spikes – Spiketrain (Time indexing begins with 0)
• weights – Weights
• t – Evaluation time

Returns total current of all neurons at time step t (vector), spikes at time t

**Izhikevich model**

class spiking.Izhikevich(neurons, a, b, c, d, v0=-65, threshold=30, verbose=False)

Izhikevich model

http://www.izhikevich.org/publications/spikes.htm

Parameters
• neurons – total number of neurons
• a –
• b –
• c –
• d –
• v0 – Initial voltage
• threshold –
• verbose –
Returns

3.3.2 learning.py

This module contains classes that can learn the weights.

3.3.3 plotting.py

This module contains functions do create nice plots such as:

Example_plot 1
Example_plot 2

```python
class plotting.CurrentsHeatmapAnimation (fps=30)

    Example
    cha = CurrentsHeatmapAnimation() for i in range(300):
        cha.add(np.random.randn(1, 20))
    cha.show_animation() plt.show()

    Initialize the Currents Animation
    :param fps: How many frames-per-second the animation should be played.
    Default: 30
    :return: None

def add (currents)
    Add currents to the end of the animation
    :param currents: None

class plotting.PSTH (spiketrain, binsize=20)

    Peri-Stimulus Time Histogram
```
Example

The image above was created by following code:

```python
spikes = np.random.randint(0, 2, size=(2, 200))
a = PSTH(spikes)
a.show_plot()
a.save_plot()
plt.show()
```

Initialize the PSTH plot with a spiketrain

**Parameters**

- **spiketrain** *(Numpy array)* – The spiketrain to be plotted
- **binsize** – The size of the bins

**Returns** None

**save_plot** *(filename=None)*

Saves the plot.

**Parameters** **filename** *(String)* – Name of the file. Default: ‘plot.png’

**show_plot** *(neuron_indices=None)*

Shows the PSTH plot.

**Parameters** **neuron_indices** *(List or Tuple)* – The indices of the neurons to be plotted
class `plotting.WeightHeatmapAnimation`(fps=30)

Show an animation of the weights (as a heatmap).

**Example**

```python
w = WeightHeatmapAnimation()
for i in range(300):
    w.add(np.array([[i/300, 1], [1, 3]]))
w.plot_weights()
plt.show()
```

Initialize the Weight Animation

**Parameters**

- **fps** – How many frames-per-second the animation should be played. Default: 30

**Returns** None

**add** *(weights)*

Add weights to the end of animation.

**Parameters**

- **weights** *(Symmetric Numpy array)* – The weight matrix to be shown.

**Returns** None

### 3.3.4 tools.py

In this module, we can find assorted functions that help us with creating neuronal models. This module consists of some helper functions for creating neuronal networks.

**tools.poisson_homogenous**(lam, timesteps)

Generate a poisson spike train for a single neuron using a homogenous poisson distribution.

**Parameters**

- **lam** *(Float)* – lambda value
- **timesteps** *(Int)* – total length of spike train

**tools.poisson_inhomogenous**(lamdas, timesteps)

Generate a poisson spike train for a single neuron using an inhomogenous poisson distribution.
Example

The spike train of the image above was generated by following function:

```python
>>> poisson_inhomogenous((0.5, 0.25, 0, 0, 1, 0.5, 0, 0, 0.25, 0.5), 200)
```

Parameters

- **lambdas** (*List or Tuple*) – Lambda values
- **timesteps** (*Int*) – total length of the spike train

`tools.sound(timesteps, midpoint, maximum, variance)`

Generates a spike train with a peak at `midpoint`.

Example

The spike train of the image above was generated by following function:

```python
>>> sound(280, 150, 0.4, 50)
```

Parameters

- **timesteps** –
- **midpoint** – central peak
- **maximum** – Lambda value at peak
- **variance** – Variance around peak

Returns
p
plotting, 14

T
tools, 16
Index

A
add() (plotting.CurrentsHeatmapAnimation method), 14
add() (plotting.WeightHeatmapAnimation method), 16

C
CurrentsHeatmapAnimation (class in plotting), 14

E
eps() (spiking.SRM method), 13
eps_matrix() (spiking.SRM method), 13

I
Izhikevich (class in spiking), 13

P
plotting (module), 14
poisson_homogenous() (in module tools), 16
poisson_inhomogenous() (in module tools), 16
PSTH (class in plotting), 14

S
save_plot() (plotting.PSTH method), 15
show_plot() (plotting.PSTH method), 15
simulate() (spiking.SRM method), 13
sound() (in module tools), 17
SRM (class in spiking), 12

T
tools (module), 16

W
WeightHeatmapAnimation (class in plotting), 15