



Giuseppe Bonaccorso

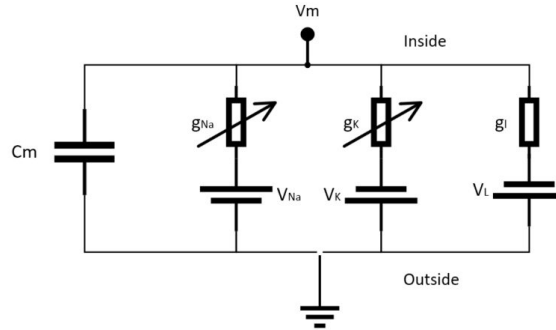
(<https://www.bonaccorso.eu/>)

ARTIFICIAL INTELLIGENCE – MACHINE LEARNING – DATA SCIENCE

Hodgkin-Huxley spiking neuron model in Python

08/19/2017 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/>) ARTIFICIAL INTELLIGENCE (<https://www.bonaccorso.eu/category/generic/artificial-intelligence/>), COMPLEX SYSTEMS (<https://www.bonaccorso.eu/category/complex-systems/>), COMPUTATIONAL NEUROSCIENCE (<https://www.bonaccorso.eu/category/computational-neuroscience/>), NEURAL NETWORKS (<https://www.bonaccorso.eu/category/machine-learning/neural-networks/>), PYTHON (<https://www.bonaccorso.eu/category/software/python/>) NO COMMENTS (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/#comments>)

The Hodgkin-Huxley model (https://en.wikipedia.org/wiki/Hodgkin%E2%80%93Huxley_model) (published on 1952 in The Journal of Physiology [1]) is the most famous spiking neuron model (also if there are simpler alternatives like the "Integrate-and-fire" model which performs quite well). It's made up of a system of four ordinary differential equations that can be easily integrated using several different tools. The main idea is based on an electrical representation of the neuron, considering only Potassium (K) and Sodium (Na) voltage-gated ion channels (even if it can be extended to include more channels). A schematic representation is shown in the following figure:



The elements are:

- C_m : a capacitance per unit area representing the membrane lipid-bilayer (adopted value: $1 \mu\text{F}/\text{cm}^2$)
- g_{Na} : voltage-controlled conductance per unit area associated with the Sodium (Na) ion-channel (adopted value: $120 \mu\text{S}/\text{cm}^2$)
- g_K : voltage-controlled conductance per unit area associated with the Potassium (K) ion-channel (adopted value: $36 \mu\text{S}/\text{cm}^2$)
- g_l : conductance per unit area associated with the leak channels (adopted value: $0.336 \mu\text{S}/\text{cm}^2$)
- V_{Na} : voltage source representing the electrochemical gradient for Sodium ions (adopted value: 115 mV)
- V_K : voltage source representing the electrochemical gradient for Potassium ions (adopted value: -12 mV)
- V_l : voltage source that determines the leakage current density together with g_l (adopted value: 10.613 mV)

In the scheme, the external stimulus current is not shown, however, we assume its presence as a current density (I) encoding the input signal. All the experimental values are the same proposed by the authors in [1] and refer to an equilibrium potential of 0 V . The system is defined through the following ODE system:

$$\begin{cases} \frac{dV_m}{dt} = \frac{I}{C_m} - \frac{\bar{g}_K n^4}{C_m} (V_m - V_K) - \frac{\bar{g}_{Na} m^3 h}{C_m} (V_m - V_{Na}) - \frac{\bar{g}_l}{C_m} (V_m - V_l) \\ \frac{dn}{dt} = \alpha_n(V_m)(1 - n) - \beta_n(V_m)n \\ \frac{dm}{dt} = \alpha_m(V_m)(1 - m) - \beta_m(V_m)m \\ \frac{dh}{dt} = \alpha_h(V_m)(1 - h) - \beta_h(V_m)h \end{cases}$$

The first equation defines the derivative of V_m considering the external stimulus (I) and the contribution of K, Na, and leakage current densities. The variables n , m , and h are associated with the probability of each channel to open and they are strictly dependent on the nature of the channel. For example, the K channel is voltage-gated and has four sub-units that must be all open in order to allow the current to flow, therefore its probability is n to the power of 4. Sodium has a slightly more complex behavior and needs two different factors (m and h) with autonomous dynamics. The last three equations describe the ion-channel kinetic model by computing the derivatives of n , m , and h as functions of the same variables and two voltage-dependent functions. The first term is the number of closed channels that are opening, while the second term is the number of open channels that are closing.

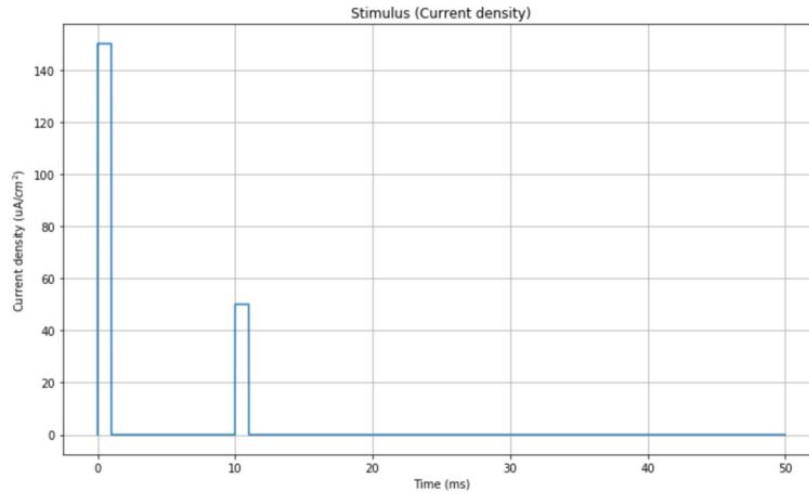
Hodgkin and Huxley suggest the following functions:

$$\begin{cases} \alpha_n(V_m) = \frac{0.01(10 - V_m)}{e^{(1.0 - 0.1V_m)} - 1} \\ \beta_n(V_m) = 0.125e^{-\frac{V_m}{80}} \end{cases}$$

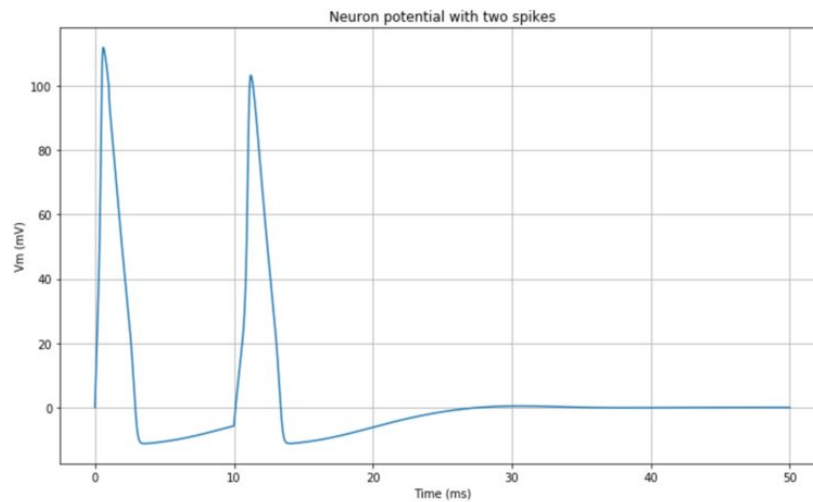
$$\begin{cases} \alpha_m(V_m) = \frac{0.1(25 - V_m)}{e^{(2.5 - 0.1V_m)} - 1} \\ \beta_m(V_m) = 4e^{-\frac{V_m}{18}} \end{cases}$$

$$\begin{cases} \alpha_h(V_m) = 0.07e^{-\frac{V_m}{20}} \\ \beta_h(V_m) = \frac{1}{e^{(3 - 0.1V_m)} + 1} \end{cases}$$

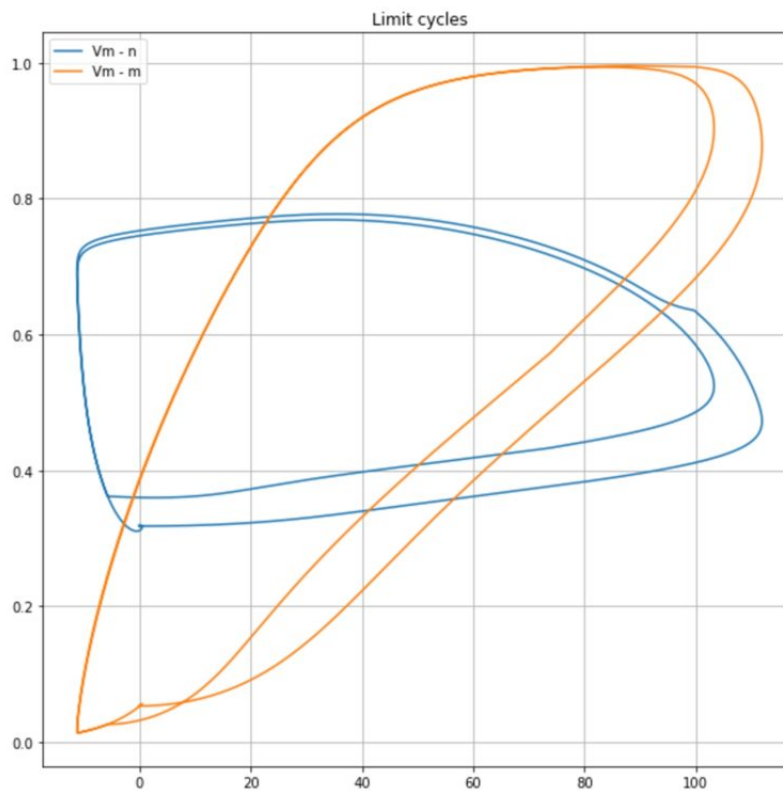
In the simulation, we're going to use a double-impulsive current density as stimulus (however, any other signal can be used to test different behaviors). The time range is [0, 50ms]:



The resulting impulsive neuron behavior is shown in the following plot:



It's also possible to observe the limit cycles present in the dynamic system. In the following plot, the trajectories $V_m - n$, and $V_m - m$ are plotted:



It's possible to observe how the probabilities (proportional to n , m , and h) for ion-channels to be opened span from a minimum to a maximum value according to n and V_m in a cyclic fashion. This allows the oscillations: steady state, potential increase, spike, potential decrease, steady state.

The complete Python code is available in this GIST (<https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef>):

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  from scipy.integrate import odeint
5
6  # Set random seed (for reproducibility)
7  np.random.seed(1000)
8
9  # Start and end time (in milliseconds)
10 tmin = 0.0
11 tmax = 50.0
12
13 # Average potassium channel conductance per unit area (mS/cm^2)
14 gK = 36.0
15
16 # Average sodium channel conductance per unit area (mS/cm^2)
17 gNa = 120.0
18
19 # Average leak channel conductance per unit area (mS/cm^2)
20 gL = 0.3
21
22 # Membrane capacitance per unit area (uF/cm^2)
23 Cm = 1.0
24
25 # Potassium potential (mV)
26 VK = -12.0
27
28 # Sodium potential (mV)
29 VNa = 115.0
30
31 # Leak potential (mV)
32 VL = 10.613
33

```

```

34 # Time values
35 T = np.linspace(tmin, tmax, 10000)
36
37 # Potassium ion-channel rate functions
38
39 def alpha_n(Vm):
40     return (0.01 * (10.0 - Vm)) / (np.exp(1.0 - (0.1 * Vm)) - 1.0)
41
42 def beta_n(Vm):
43     return 0.125 * np.exp(-Vm / 80.0)
44
45 # Sodium ion-channel rate functions
46
47 def alpha_m(Vm):
48     return (0.1 * (25.0 - Vm)) / (np.exp(2.5 - (0.1 * Vm)) - 1.0)
49
50 def beta_m(Vm):
51     return 4.0 * np.exp(-Vm / 18.0)
52
53 def alpha_h(Vm):
54     return 0.07 * np.exp(-Vm / 20.0)
55
56 def beta_h(Vm):
57     return 1.0 / (np.exp(3.0 - (0.1 * Vm)) + 1.0)
58
59 # n, m, and h steady-state values
60
61 def n_inf(Vm=0.0):
62     return alpha_n(Vm) / (alpha_n(Vm) + beta_n(Vm))
63
64 def m_inf(Vm=0.0):
65     return alpha_m(Vm) / (alpha_m(Vm) + beta_m(Vm))
66
67 def h_inf(Vm=0.0):
68     return alpha_h(Vm) / (alpha_h(Vm) + beta_h(Vm))
69
70 # Input stimulus
71 def Id(t):
72     if 0.0 < t < 1.0:
73         return 150.0
74     elif 10.0 < t < 11.0:
75         return 50.0
76     return 0.0
77
78 # Compute derivatives
79 def compute_derivatives(y, t0):
80     dy = np.zeros((4,))
81
82     Vm = y[0]
83     n = y[1]
84     m = y[2]
85     h = y[3]
86
87     # dVm/dt
88     GK = (gK / Cm) * np.power(n, 4.0)
89     GNa = (gNa / Cm) * np.power(m, 3.0) * h
90     GL = gL / Cm
91
92     dy[0] = (Id(t0) / Cm) - (GK * (Vm - VK)) - (GNa * (Vm - VNa)) - (GL * (Vm - V1))
93
94     # dn/dt
95     dy[1] = (alpha_n(Vm) * (1.0 - n)) - (beta_n(Vm) * n)
96

```

```
97     # dm/dt
98     dy[2] = (alpha_m(Vm) * (1.0 - m)) - (beta_m(Vm) * m)
99
100    # dh/dt
101    dy[3] = (alpha_h(Vm) * (1.0 - h)) - (beta_h(Vm) * h)
102
103    return dy
104
105    # State (Vm, n, m, h)
106    Y = np.array([0.0, n_inf(), m_inf(), h_inf()])
107
108    # Solve ODE system
109    # Vy = (Vm[t0:tmax], n[t0:tmax], m[t0:tmax], h[t0:tmax])
110    Vy = odeint(compute_derivatives, Y, T)
```

hodgkin- view raw (<https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef/raw/d8fc848ee1b749e4d72fe8829cf7339c566b0350/hodgkin-huxley-main.py>)
huxley-main.py (<https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef#file-hodgkin-huxley-main-py>) hosted with ❤ by GitHub (<https://github.com>)

```
1  # Input stimulus
2  Idv = [Id(t) for t in T]
3
4  fig, ax = plt.subplots(figsize=(12, 7))
5  ax.plot(T, Idv)
6  ax.set_xlabel('Time (ms)')
7  ax.set_ylabel(r'Current density (uA/$cm^2$)')
8  ax.set_title('Stimulus (Current density)')
9  plt.grid()
10
11  # Neuron potential
12  fig, ax = plt.subplots(figsize=(12, 7))
13  ax.plot(T, Vy[:, 0])
14  ax.set_xlabel('Time (ms)')
15  ax.set_ylabel('Vm (mV)')
16  ax.set_title('Neuron potential with two spikes')
17  plt.grid()
18
19  # Trajectories with limit cycles
20  fig, ax = plt.subplots(figsize=(10, 10))
21  ax.plot(Vy[:, 0], Vy[:, 1], label='Vm - n')
22  ax.plot(Vy[:, 0], Vy[:, 2], label='Vm - m')
23  ax.set_title('Limit cycles')
24  ax.legend()
25  plt.grid()
```

hodgkin- view raw (<https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef/raw/d8fc848ee1b749e4d72fe8829cf7339c566b0350/hodgkin-huxley-plots.py>)
huxley-plots.py (<https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef#file-hodgkin-huxley-plots-py>) hosted with ❤ by GitHub (<https://github.com>)

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  from scipy.integrate import odeint
5
6  # Set random seed (for reproducibility)
7  np.random.seed(1000)
8
9  # Start and end time (in milliseconds)
10 tmin = 0.0
11 tmax = 50.0
12
13 # Average potassium channel conductance per unit area (mS/cm^2)
14 gK = 36.0
15
16 # Average sodium channel conductance per unit area (mS/cm^2)
17 gNa = 120.0
18
```

```

19 # Average leak channel conductance per unit area (mS/cm^2)
20 gL = 0.3
21
22 # Membrane capacitance per unit area (uF/cm^2)
23 Cm = 1.0
24
25 # Potassium potential (mV)
26 VK = -12.0
27
28 # Sodium potential (mV)
29 VNa = 115.0
30
31 # Leak potential (mV)
32 V1 = 10.613
33
34 # Time values
35 T = np.linspace(tmin, tmax, 10000)
36
37 # Potassium ion-channel rate functions
38
39 def alpha_n(Vm):
40     return (0.01 * (10.0 - Vm)) / (np.exp(1.0 - (0.1 * Vm)) - 1.0)
41
42 def beta_n(Vm):
43     return 0.125 * np.exp(-Vm / 80.0)
44
45 # Sodium ion-channel rate functions
46
47 def alpha_m(Vm):
48     return (0.1 * (25.0 - Vm)) / (np.exp(2.5 - (0.1 * Vm)) - 1.0)
49
50 def beta_m(Vm):
51     return 4.0 * np.exp(-Vm / 18.0)
52
53 def alpha_h(Vm):
54     return 0.07 * np.exp(-Vm / 20.0)
55
56 def beta_h(Vm):
57     return 1.0 / (np.exp(3.0 - (0.1 * Vm)) + 1.0)
58
59 # n, m, and h steady-state values
60
61 def n_inf(Vm=0.0):
62     return alpha_n(Vm) / (alpha_n(Vm) + beta_n(Vm))
63
64 def m_inf(Vm=0.0):
65     return alpha_m(Vm) / (alpha_m(Vm) + beta_m(Vm))
66
67 def h_inf(Vm=0.0):
68     return alpha_h(Vm) / (alpha_h(Vm) + beta_h(Vm))
69
70 # Input stimulus
71 def Id(t):
72     if 0.0 < t < 1.0:
73         return 150.0
74     elif 10.0 < t < 11.0:
75         return 50.0
76     return 0.0
77
78 # Compute derivatives
79 def compute_derivatives(y, t0):
80     dy = np.zeros((4,))
81

```

```
82     Vm = y[0]
83     n = y[1]
84     m = y[2]
85     h = y[3]
86
87     # dVm/dt
88     GK = (gK / Cm) * np.power(n, 4.0)
89     GNa = (gNa / Cm) * np.power(m, 3.0) * h
90     GL = gL / Cm
91
92     dy[0] = (Id(t0) / Cm) - (GK * (Vm - VK)) - (GNa * (Vm - VNa)) - (GL * (Vm - V1))
93
94     # dn/dt
95     dy[1] = (alpha_n(Vm) * (1.0 - n)) - (beta_n(Vm) * n)
96
97     # dm/dt
98     dy[2] = (alpha_m(Vm) * (1.0 - m)) - (beta_m(Vm) * m)
99
100    # dh/dt
101    dy[3] = (alpha_h(Vm) * (1.0 - h)) - (beta_h(Vm) * h)
102
103    return dy
104
105    # State (Vm, n, m, h)
106    Y = np.array([0.0, n_inf(), m_inf(), h_inf()])
107
108    # Solve ODE system
109    # Vy = (Vm[t0:tmax], n[t0:tmax], m[t0:tmax], h[t0:tmax])
110    Vy = odeint(compute_derivatives, Y, T)
```

[view raw \(https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef/raw/d8fc848ee1b749e4d72fe8829cf7339c566b0350/hodgkin-huxley-main.py\)](https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef/raw/d8fc848ee1b749e4d72fe8829cf7339c566b0350/hodgkin-huxley-main.py)
[huxley-main.py \(https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef#file-hodgkin-huxley-main-py\)](https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef#file-hodgkin-huxley-main-py) hosted with ❤ by GitHub (<https://github.com/>)

```
1  # Input stimulus
2  Idv = [Id(t) for t in T]
3
4  fig, ax = plt.subplots(figsize=(12, 7))
5  ax.plot(T, Idv)
6  ax.set_xlabel('Time (ms)')
7  ax.set_ylabel(r'Current density (uA/$cm^2$)')
8  ax.set_title('Stimulus (Current density)')
9  plt.grid()
10
11  # Neuron potential
12  fig, ax = plt.subplots(figsize=(12, 7))
13  ax.plot(T, Vy[:, 0])
14  ax.set_xlabel('Time (ms)')
15  ax.set_ylabel('Vm (mV)')
16  ax.set_title('Neuron potential with two spikes')
17  plt.grid()
18
19  # Trajectories with limit cycles
20  fig, ax = plt.subplots(figsize=(10, 10))
21  ax.plot(Vy[:, 0], Vy[:, 1], label='Vm - n')
22  ax.plot(Vy[:, 0], Vy[:, 2], label='Vm - m')
23  ax.set_title('Limit cycles')
24  ax.legend()
25  plt.grid()
```

[view raw \(https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef/raw/d8fc848ee1b749e4d72fe8829cf7339c566b0350/hodgkin-huxley-plots.py\)](https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef/raw/d8fc848ee1b749e4d72fe8829cf7339c566b0350/hodgkin-huxley-plots.py)
[huxley-plots.py \(https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef#file-hodgkin-huxley-plots-py\)](https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef#file-hodgkin-huxley-plots-py) hosted with ❤ by GitHub (<https://github.com/>)

References:

1 Hodgkin, A. L., Huxley, A. F., (1952), A quantitative description of membrane current and its application to conduction and excitation in nerve.
(<http://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004764/abstract>) The Journal of Physiology, 117 doi: 10.1113/jphysiol.1952.sp004764

2. Abbott L. F., Dayan P., Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems (https://www.amazon.com/Theoretical-Neuroscience-Computational-Mathematical-Modeling-ebook/dp/B00MHAUY7U/ref=sr_1_1?s=digital-text&ie=UTF8&qid=1503135145&sr=1-1&keywords=neuroscience+abbott), The MIT Press

See also:

Share:

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=twitter&nb=1&nb=1>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=facebook&nb=1&nb=1>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=linkedin&nb=1&nb=1>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=pocket&nb=1&nb=1>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=tumblr&nb=1&nb=1>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=reddit&nb=1&nb=1>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=pinterest&nb=1&nb=1>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=skype&nb=1&nb=1>)

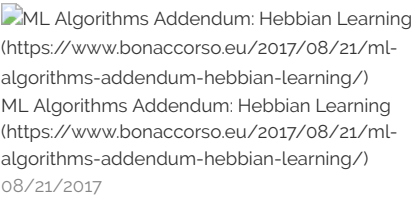
 (<https://api.whatsapp.com/send?text=Hodgkin-Huxley%20spiking%20neuron%20model%20in%20Python%20https%3A%2F%2Fwww.bonaccorso.eu%2F2017%2F08%2F19%2Fhodgkin-huxley-spiking-neuron-model-python%2F>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=telegram&nb=1&nb=1>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/?share=email&nb=1&nb=1>)

 (<https://www.bonaccorso.eu/2017/08/19/hodgkin-huxley-spiking-neuron-model-python/#print>)

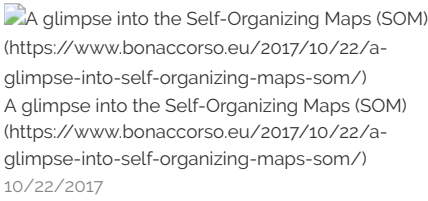
You can also be interested in these articles:



ML Algorithms Addendum: Hebbian Learning (<https://www.bonaccorso.eu/2017/08/21/ml-algorithms-addendum-hebbian-learning/>)

ML Algorithms Addendum: Hebbian Learning (<https://www.bonaccorso.eu/2017/08/21/ml-algorithms-addendum-hebbian-learning/>)

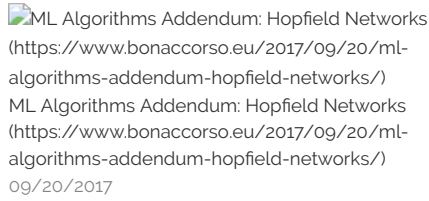
08/21/2017



A glimpse into the Self-Organizing Maps (SOM) (<https://www.bonaccorso.eu/2017/10/22/a-glimpse-into-self-organizing-maps-som/>)

A glimpse into the Self-Organizing Maps (SOM) (<https://www.bonaccorso.eu/2017/10/22/a-glimpse-into-self-organizing-maps-som/>)

10/22/2017



ML Algorithms Addendum: Hopfield Networks (<https://www.bonaccorso.eu/2017/09/20/ml-algorithms-addendum-hopfield-networks/>)

ML Algorithms Addendum: Hopfield Networks (<https://www.bonaccorso.eu/2017/09/20/ml-algorithms-addendum-hopfield-networks/>)

09/20/2017

◀ ML Algorithms addendum: Mutual information in classification tasks
(<https://www.bonaccorso.eu/2017/08/18/ml-algorithms-addendum-mutual-information-in-classification-tasks/>)

ML Algorithms Addendum: Hebbian Learning ▶
(<https://www.bonaccorso.eu/2017/08/21/ml-algorithms-addendum-hebbian-learning/>)

LEAVE A REPLY

Enter your comment here...

FOLLOW ME

[Follow me on GitHub](#) [Follow me on LinkedIn](#) [Follow me on Twitter](#) [Follow me on Facebook](#) [Follow me on YouTube](#)

SEARCH ARTICLES

Search ...

Q

LATEST BLOG POSTS

- Machine Learning Algorithms – Second Edition (<https://www.bonaccorso.eu/2018/08/28/machine-learning-algorithms-second-edition/>) 08/28/2018
- Recommendations and User-Profiling from Implicit Feedbacks (<https://www.bonaccorso.eu/2018/07/10/recommendations-user-profiling-implicit-feedbacks/>) 07/10/2018
- Are recommendations really helpful? A brief non-technical discussion (<https://www.bonaccorso.eu/2018/06/29/recommendations-really-helpful-brief-non-technical-discussion/>) 06/29/2018
- A book that every data scientist should read (<https://www.bonaccorso.eu/2018/06/22/a-book-that-every-data-scientist-should-read/>) 06/22/2018
- Mastering Machine Learning Algorithms (<https://www.bonaccorso.eu/2018/05/24/mastering-machine-learning-algorithms/>) 05/24/2018

SUBSCRIBE TO THIS BLOG

Join 2,190 other subscribers

Email

SUBSCRIBE

MACHINE LEARNING ALGORITHMS ADDENDA

(<https://www.bonaccorso.eu/category/machine-learning/machine-learning-algorithms-addenda/>)

FOLLOW ME ON TWITTER

Tweets by @GiuseppeB

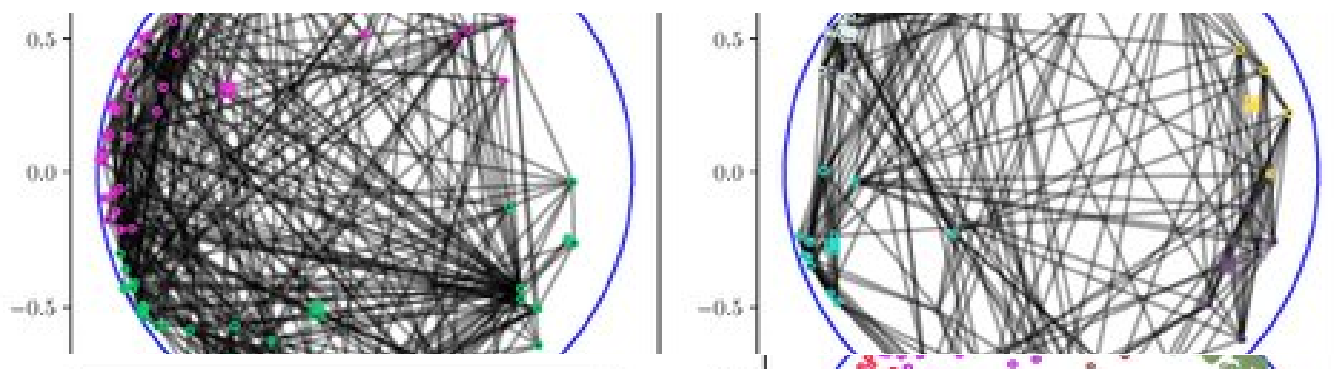
Giuseppe Bonaccorso Retweeted



arxiv

@arxiv_org

Learning graph-structured data using Poincaré embeddings and Riemannian K-means algorit... arxiv.org/abs/1907.01662



Copyright © 2019 Giuseppe Bonaccorso (<https://www.bonaccorso.eu/>). All Rights Reserved. Privacy policy (<https://www.iubenda.com/privacy-policy/331721>) - Cookie policy (<https://www.iubenda.com/privacy-policy/331721/cookie-policy>)

By using this website, you agree that we and our partners may set cookies for purposes such as analytics. Please read our privacy and cookie policies (GDPR compliant) in the disclaimer page or in the footer. Through these pages it's also possible to opt-out from specific services. I Agree