Oscillatory Neurocomputers with Dynamic Connectivity

Alexandre Garcia-Duran Castilla

May 2023

This project is based on the work by Frank C. Hoppensteadt and Eugene M. Izhikevich [1].

1 Introduction

A new generation of computers employs principles of the human brain. Such a computer, often referred to as a *neurocomputer*, consists of many interconnected units (referred to here as neurons) performing simple nonlinear transformations in parallel. There are many neural network models that can be used as a theoretical basis for a neurocomputer, and the most promising are those with oscillatory components, because they consider the rhythmic behavior of the brain.

Whether oscillatory or not, a neurocomputer consisting of n neurons needs n^2 programmable connections, which is a problem in terms of computational power. A possible way to cope with this problem was suggested by Hoppensteadt and Izhikevich [1] (in a study of thalamo-cortical systems): they treated the cortex as a network of weakly connected autonomous oscillators forced by the thalamic input, as shown in Fig. 1.

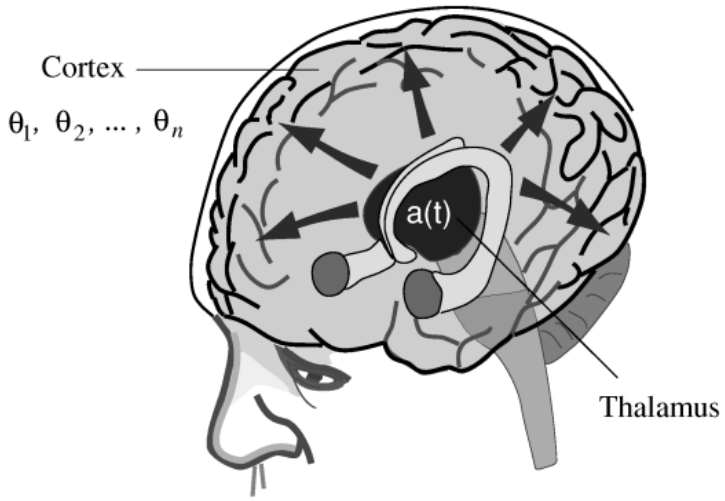

Figure 1: Cortex as a network of weakly connected autonomous oscillators $\vartheta_1, \ldots, \vartheta_n$ forced by the thalamic input $a(t)$.

This suggests the following design of a *neurocomputer*: oscillators having different frequencies

and connected homogeneously and weakly to a common medium, as shown in Fig. 2, right. On the left, we see the structure of the conventional neurocomputer.

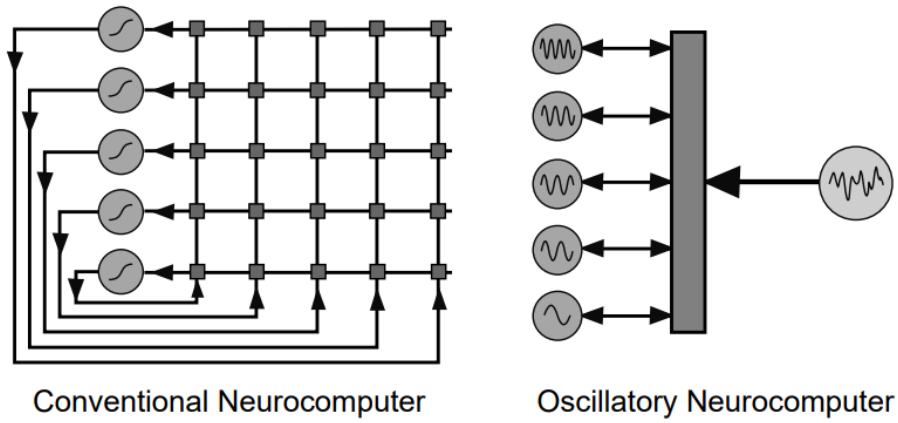

Figure 2: Left: A conventional neurocomputer having n neurons (circles) would have n^2 connections (squares). Right: An oscillatory neurocomputer with dynamic connectivity imposed by the external input (large circle) needs only n connections: between each neuron (circle) and a common medium (rectangle).

2 Conventional neurocomputer

The conventional neurocomputer is modelled with the Hopfield model [2], which with some basic assumptions it can reproduce some characteristics of biological neural networks, such as learning.

2.1 Hopfield Model

This model consists on a network of n neurons (connected as in Fig. 2 left), with only two possible states, active and silent. Let $S_i(t)$ be the state variable of a neuron at time t, such that

$$S_i(t) = \begin{cases} 1 & \text{if neuron is ON} \\ -1 & \text{if neuron is OFF} \end{cases}$$

with $i = 1, \ldots, n$. We can interpret $S_i(t) = +1$ as an action potential of neuron i at time t. Let $W = (w_{ij})$ be the matrix of weights representing neurons' interactions with each other. Then, the input potential of neuron i, influenced by the activity of other neurons is:

$$h_i(t) = \sum_{j=1}^{n} w_{ij}\, S_j(t)\ .$$

We can express the probability of $S_i(t + \Delta t)$ being at state $S_i = 1$, as a function of the given an input $h_i(t)$:

$$\text{Prob}\{S_i(t + \Delta t) = +1 | h_i(t)\} = g(h_i(t)) = g\left(\sum_{j=1}^{n} w_{ij}\ S_j(t)\right)$$

where g is a function with values between zero and one. A common choice is $g(h) = 0.5[1+\tanh(\beta h)]$ parameterized by β. If we compute the limit of $g(h)$ as $\beta \to \infty$

$$\lim_{\beta\to\infty} g(h) = \lim_{\beta\to\infty} 0.5[1+\tanh(\beta h)] = \begin{cases} 1 & \text{if } h > 0 \\ 0 & \text{otherwise} \end{cases}$$

we see that the probability $\text{Prob}\{S_i(t+\Delta t) = +1|h_i(t)\}$ is 1 ($S_i(t+\Delta t) = 1$) when $h > 0$ and 0 otherwise ($S_i(t+\Delta t) = -1$). Therefore, $S_i(t+\Delta t)$ will take the same values as the sign of $h_i(t)$, so the dynamics are deterministic and summarized by the update rule

$$S_i(t+\Delta t) = \text{sgn}[h_i(t)] \ . \tag{1}$$

2.2 Hebbian Learning

The memory capacity of the system is introduced by the Hebbian Learning. The basis of the theory is when our brains learn something new, neurons are activated and connected with other neurons, forming a neural network. These connections start off weak, but each time the stimulus is repeated, the connections grow stronger and stronger, and the action becomes more intuitive. Hebb's principle can be described as a method of determining how to alter the weights between model neurons. The weight between two neurons increases if the two neurons activate simultaneously, and reduces if they activate separately. Indeed, those neurons which are either both positive or both negative at the same time have strong positive weights, while if they have opposite sign, they have strong negative weights. For our case, suppose we have m vectors of length n and values ± 1 to be memorized:

$$\xi^k = \{\xi_1^k, \xi_2^k, \ldots, \xi_n^k\}, \quad \xi_i^k = \pm 1, \ \ k = 1, \ldots, m$$

Then, if $\xi_i^k = \xi_j^k$, the neurons are in-phase, they activate simultaneously. Whereas if $\xi_i^k = -\xi_j^k$, they are anti phase. A Hebbian rule of the form:

$$S = (s_{ij}) = \frac{1}{m}\sum_{k=1}^{m} \xi_i^k \xi_j^k$$

is used to provide learning capacity to the system. For a better intuition, this matrix can be expressed as the average interaction of neurons i and j over all m training patterns:

$$s_{ij} \propto < \xi_i^k \xi_j^k > \ ,$$

where $< \cdot >$ is the average. And therefore if two neurons are frequently being activated among all patterns, they will have a greater value (weight) in the matrix. If the average is zero, it would mean that the neurons do not interact at all. Finally, if the average is negative, it would mean that the neurons are usually anti-phase.

Using this matrix in the n-dimensional system (1), different attractors arise, corresponding to the learnt patterns.

2.3 Pattern recognition algorithm

With this simple model, using n^2 computations per timestep, one can recognize patterns. The code is structured in two main steps: initialization and pattern recognition. The first part is done to reach a value near the basin of attraction of the attractors encoding the patterns to be recognized, which will be used as a seed for the pattern recognition step. If the first step was not done, the system would converge to any attractor depending on the initial conditions. The pseudo-code is the following:

1. Initialize the network:

   (a) Let ξ^0 be the pattern that has to be recognized. We add noise to this pattern and we define ξ^0_* as the pattern with noise (in the basin of attraction of ξ^0). Then, we define a matrix of connections $C = (c_{ij}) = (\xi^0_{*,i} \cdot \xi^0_{*,j})$.

   (b) Define $S^{init.}(0)$ as n random points between -1 and 1.

   (c) Simulate $S_i^{init.}(t + \Delta t) = \text{sgn}[h_i(t)]$, with $h_i(t) = \sum_j c_{ij} \cdot S_j(t)$, until $t = t_{final}$.

   This way, we force $S(t = t_{final})$ to be close to the basin of attraction of the pattern to be recognized (ξ^0), since with only one pattern present in the connection matrix C, the initialization will converge to ξ^0_*.

2. Pattern recognition

   (a) Use the last point of the initialization as a starting point (with random normal noise η): $S_i^{Rec.}(0) = S_i^{init.}(t_{final}) + \eta$.

   (b) Simulate $S_i^{rec.}(t + \Delta t) = \text{sgn}[h_i(t)]$, with $h_i(t) = \sum_j s_{ij} \cdot S_j(t)$, until $t = t_{final}$.

To do so, I used $\Delta t = 50$ ms, $t_{final} = 10$ s. The system is not explicitly dependent on time, therefore it is the same as simulating the system $10/0.05 = 200$ times. With this, one can adapt the model into different frameworks: biological neurons fire in the range of miliseconds, whereas other systems may fire in the range of days or weeks.

2.4 Pattern recognition example

Suppose we have the following patterns representing numbers:

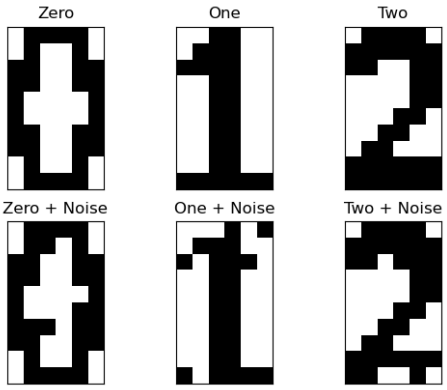

Figure 3: Top: Patterns representing numbers. Below: same patterns with added random noise.

For our example, we initialize the network with ξ^0 being the vector expressing the One + Noise pattern, as shown in Fig. 4. And then the output of the network (state at $t = t_{final}$) will be used as a seed for pattern recognition (with noise), as shown in Fig. 5. We can see that both processes converge rapidly: in a single step of the process (note that the timestep does not matter).

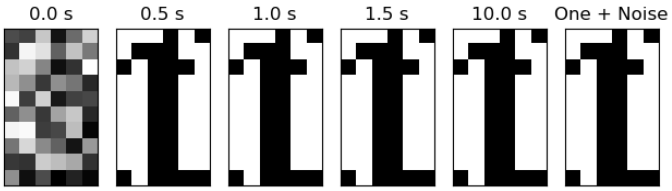

Figure 4: Initialization of the Hopfield model. Note the rapid convergence of the model.

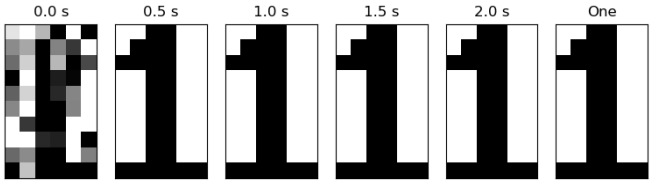

Figure 5: Pattern recognition of the Hopfield model. Note also the rapid convergence of the model and the accuracy of the recognition.

3 Oscillatory Neurocomputer

In this section we will consider Oscillatory Neurocomputers, which are different from the Hopfield model, since they are based in the Kuramoto phase model [3, 4]. Consider the esmented model:

$$\dot{\vartheta}_i = \Omega_i + \varepsilon a(t) \sum_{j=1}^{n} \sin(\vartheta_j - \vartheta_i) \tag{2}$$

where $\vartheta_i \in \mathcal{S}^1$ is the phase of the i_{th} oscillator, Ω_i is its respective free-running frequency, $a(t)$ is the external input, and $\varepsilon << 1$ is the strength of the connections. We can see that each oscillator receives identical input ($a(t)$ is the same for all i).

3.1 Dynamic Connectivity

Now, let $\vartheta_i = \Omega_i t + \varphi_i$. Then,

$$\dot{\varphi}_i = \varepsilon a(t) \sum_{j=1}^{n} \sin(\{\Omega_j - \Omega_i\}t + \varphi_j - \varphi_i)$$

Averaging: If ε is suff. small, we can express the system as

$$\dot{\varphi}_i = \varepsilon \underbrace{H_i(\varphi_1, \ldots, \varphi_n)}_{\text{Average}} + \underbrace{\mathcal{O}(\varepsilon)}_{\text{Small Variations}}$$

where

$$H_i = \lim_{T\to\infty} \frac{1}{T} \int_0^T a(t) \sum_{j=1}^{n} \sin(\{\Omega_j - \Omega_i\}t + \varphi_j - \varphi_i))\, dt$$

Therefore, depending on the input $a(t)$ we can have some connectivity dynamics or not. For example, if we let $a(t) = a_o = \text{ctant.}$, the integral goes to zero as $T \to \infty$, since it's just a cosine series upper and lower bounded by n and divided by T at the limit. Now suppose we are given a matrix of connections $W = (w_{ij})$. Let

$$a(t) = a_0 + \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cos(\{\Omega_j - \Omega_i\}t)$$

be a time dependant external input, a quasiperiodic function on t. Integrating, we find

$$H_i = \sum_{j=1}^{n} \frac{w_{ij} + w_{ji}}{2} \sin(\varphi_j - \varphi_i)$$

If we denote $s_{ij} = (w_{ij} + w_{ji})/2$, we have:

$$\dot{\varphi}_i = \varepsilon \sum_{j=1}^{n} s_{ij} \sin(\varphi_j - \varphi_i) + \mathcal{O}(\varepsilon)$$

if we use the slow time $\tau = \varepsilon t$, we have:

$$\frac{d\varphi}{d\tau} = \frac{d\varphi}{dt} \cdot \frac{1}{\varepsilon}$$

So, if we disregard the small-order term $\mathcal{O}(\varepsilon)$, we get

$$\frac{d\varphi_i}{d\tau} = \varphi_i' = \sum_{j=1}^{n} s_{ij} \sin(\varphi_j - \varphi_i) \tag{3}$$

Therefore, we see that the external input of this form can dynamically connect any two oscillators provided that the corresponding c_{ij} is not zero. To sum up, we have found a convenient expression for $a(t)$ such that the system is provided with dynamic connectivity and its phase evolution is defined by Eq. 3.

3.2 Pattern recognition algorithm

Once the system is defined, and follows a rule as the following:

$$\varphi_i' = f(W, \varphi_i)$$

with W being a connections matrix and f a function, we can proceed as in the Hopfield model. In this case, we use **only** n computations per timestep. The code is structured in two parts, as before: initialization and pattern recognition. In this case, the algorithm is defined by this pseudo-code:

1. Initialization of the network.
   (a) Let ξ^0 be the pattern that has to be recognized. We add noise to this pattern and we define ξ^0_* as the pattern with noise (in the basin of attraction of ξ^0). Then, we define a matrix of connections $C = (c_{ij}) = (\xi^0_{*,i} \cdot \xi^0_{*,j})$.
   (b) We impose that the common input $a(t)$ has c_{ij} as defined before. Then, we see that if $(\xi^0_{*,i} \cdot \xi^0_{*,j}) = 1$, since neurons are in-phase (i.e. they have the same phase), $\varphi_i(t) - \varphi_j(t) \to 0$. On the other hand,if $(\xi^0_{*,i} \cdot \xi^0_{*,j}) = -1$, since neurons are anti-phase, $\varphi_i(t) - \varphi_j(t) \to \pi$.

(c) Define $\varphi^{init.}(0)$ as random points between $-\pi$ and π.

(d) Simulate $\varphi_i^{init.'} = f(C, \varphi_i)$ up to $t = t_{final}$.

Now, we have forced the system to be in the basin of attraction of ξ^0, by ending up at ξ_*^0.

2. Pattern recognition.

   (a) Use the last point of the initialization as a starting point (with random normal noise η): $\varphi_i^{Rec.}(0) = \varphi_i^{init.}(t_{final}) + \eta$.

   (b) Simulate $\varphi_i^{rec.,'} = f(S, \varphi_i)$ up to $t = t_{final}$. With S being the Hebbian Learning matrix.

To do so, I used Runge-Kutta (RK45) to numerically integrate Eq. 3, with $t_{final} = 10$ s and a timestep of 1 ms.

3.2.1 Information retrieval

To retrieve information from the phase coordinates, we can do some basic Fourier analysis of the Mean Field:

$$M(t) = \sum_{j=1}^{n} e^{i\vartheta_j}$$

Since $\vartheta_i = \Omega_i t + \varphi_i$, we can express the mean field as

$$M(t) = \sum_{j=1}^{n} e^{i\Omega_j t} \cdot \underbrace{e^{i\varphi_j}}_{\text{Coeffs.}}$$

which is an expression of $M(t)$ as a Fourier series whose coefficients depend on φ_j. Remember that when φ_j reaches an attractor, its difference to φ_i tends to π or 0. Therefore, we would have coefficients weighing each neuron's free-running contribution (Ω_i), considering the different interaction states of neurons (in-phase or anti-phase). Since the difference in phases can be π or 0, we can read the output of the network by taking S_j =cos(φ_j), since they are separated this way (for the phase difference). This way, we would get positive and negative valued neurons (whose absolute value is upper bounded by 1). As a final step, we could take the sign of each value so we just have -1 and 1, as for the Hopfield model. However, this last step is not done because it breaks the smooth dynamics of the system, so it is not convenient for visualization.

3.3 Pattern recognition example

In this section, we will proceed with the same example as in section 2.4, but using the oscillatory neurocomputer. First the initialization step, which shows rapid convergence, as we can see in Fig. 6, 7. The attractors are located at $\varphi = 0,\ \pi$.

Then, for the pattern recognition step, we can see in Fig. 8 that the network does not converge at exactly the same value of pixels, but the shape is pretty accurate. The main thing to highlight here is that the neurons behave according to the Hebbian Learning matrix S, so if they are anti-phase and they interact, they will have a difference in phase of π. And when they are in phase their difference in the cosine of the phase (the real part of the Fourier coefficients) will be zero or close to. Therefore, if we just took the sign of the value of each pixel we could get exactly the pattern to be recognized, since as seen in Fig. 9 all points do not converge in two single attractors as before, but in 6, separated into positive and negative cosine values.

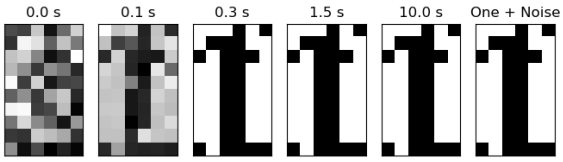

Figure 6: Initialization process of the oscillatory network.

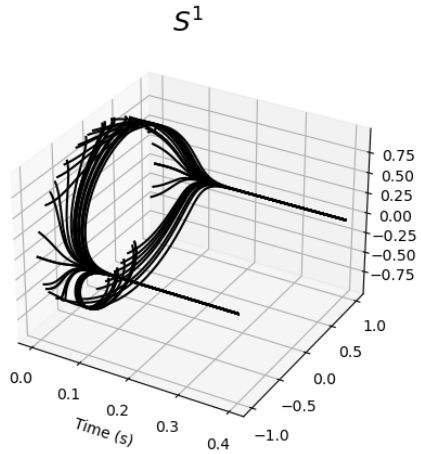

Figure 7: Evolution in time of the initialization process of the oscillatory network in the phase space ($\mathcal{S}^1$).

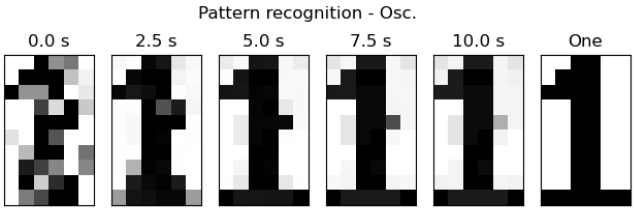

Figure 8: Pattern recognition process of the oscillatory network.

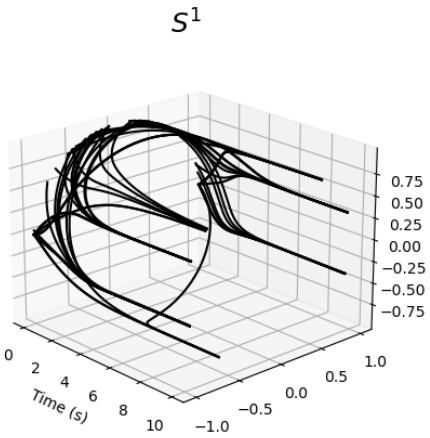

Figure 9: Evolution in time of the pattern recognition process of the oscillatory network in the phase space ($\mathcal{S}^1$)

4 Comparison

We define the overlap $\mu(t)$ as:

$$\mu(t) = \frac{1}{n}\sum_{i=1}^{n} \xi_i^* \cdot S_i(t) \ , \quad \mu \in [-1, 1]$$

Which is a scalar metric that shows how close a n-dimensional pattern $S(t)$ (at time t) is to the objective pattern ξ^* (clearly also n-dimensional). If the value is 1, it would mean that all pixels have exactly the same value, whereas if it is -1, it would mean that all pixels have the opposite value. Therefore, it can be used as a metric for the accuracy. It can be seen in Fig. 10 that the oscillatory

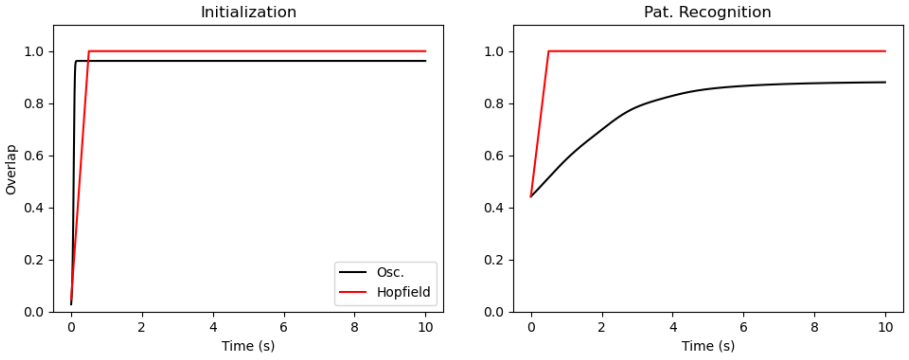

Figure 10: Evolution in time of the overlap in the two processes, for the two models.

neurocomputer converges really fast in the initialization step, whereas for the pattern recognition

it converges much slower, since the matrix involved has much more information than the one in the first step. If we took the sign of the value of each pixel, the oscillatory neurocomputer would converge to an overlap of 1.

To compare the step effect of both models, I simulated the oscillatory network with a timestep of 500 ms, and for the initialization, I got that with 1 step, the oscillatory neurocomputer converged to the pattern, and the same for the Hopfield model. For the pattern recognition step, the neurocomputer converged at $t = 3.5$ s, which are 7 steps, whereas the Hopfield model converged after the first step. All this can be seen in Fig. 11, where the sign of the cosine of the phase of each neuron was taken to see that the overlap saturated at 1 too.

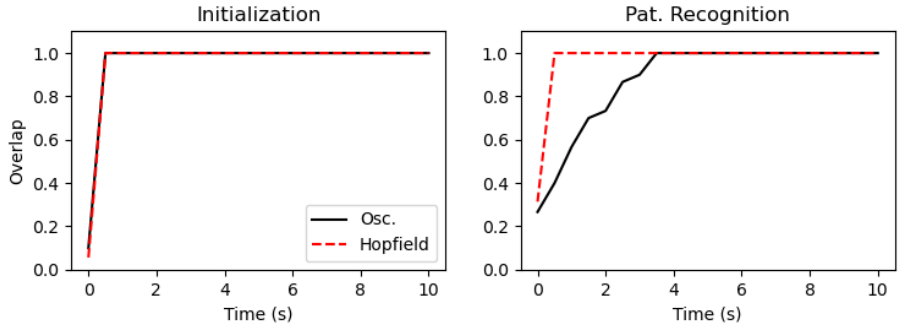

Figure 11: Evolution in time of the overlap in the two processes, for the two models. The oscillatory model is simulated using $\Delta t = 500$ ms. The sign of the value of each neuron is extracted in order to get an overlap of 1. Note that the overlap is not smooth.

5 Conclusions

- An oscillatory neurocomputer consisting of n neurons does not need n^2 programmable connections to emulate an associative neural network having a fully connected synaptic matrix.
- The oscillators can be put together in an arbitrary, random fashion, yet the network can establish a desired configuration via dynamic connectivity.
- Further analysis should be done to study the memory capacity of this kind of systems.
- For a suitable timestep, the oscillatory network can perform as good as the Hopfield model with a small computational effort.

References

[1] Frank C. Hoppensteadt and Eugene M. Izhikevich. Oscillatory neurocomputers with dynamic connectivity. *Physical Review Letters*, 82(14):2983–2986, apr 1999.

[2] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics*. Cambridge University Press, 2014.

[3] Eugene M. Izhikevich. *Dynamical Systems in Neuroscience*. The MIT Press, 2006.

[4] Steven H. Strogatz. From kuramoto to crawford: exploring the onset of synchronization in populations of coupled oscillators. *Physica D: Nonlinear Phenomena*, 143(1-4):1–20, sep 2000.