Neural Networks

Ciprian Bangu

April 26 2024

Contents

1 Introduction

In this report, we will study how the activity of many neurons influence each other in neural networks. We will begin by examining the simplest type of neural network, containing only a single neuron. We will then progress to a two neuron network, and finally to N neuron networks. In this, we will examine a specific type of neural network, a Hoppfield Network, to explore how networks of neurons can store 'memories'. Finally, we will investigate how Principal Component Analysis can be used to reduce the dimensionality of high dimensional neural networks to help us understand their dynamics.

2 The Self-Connected Neuron

The simplest form of a neural network is a single neuron that is self-connected. Biologically speaking, this would be an example of a neuron with an autapse: a synapse that connects the neuron with itself. In this section, we will first examine how neurons process information recieved from their peers (in this case, from themselves), and then examine the dynamics of a self-connected neuron.

2.1 Signal Integration and Activation Functions

Neurons recieve input in the form of electircal current from the neurons they are connected to. The recieving neuron is called the *post-synaptic* neuron, and the neuron sending the signal is called the *pre-synaptic* neuron. When the pre-synaptic neurons generate action potentials, these spikes are propogated down their respective axons to their synapeses, which transmit the signal to the dendrites of the post-synaptic neuron. The biological process by which the post-synaptic neuron judges the recieved signal can be modeled as a weighted linear sum of the recieved input:

$$I(t) = \sum_j w_j u_j(t) \tag{1}$$

Where $I(t)$ is the total input current to the post-synaptic neuron at time t, w_j is the weight of the connection between the pre-synaptic neuron j and the post-synaptic neuron, and $u_j(t)$ is the output of the pre-synaptic neuron j at time t. Note that neurons recieve both excitatory and inhibitory signals, meaning that the result of this sum can be both negative (inhibitory) or positive (excitatory).

To model the resultant neuronal firing rate given said input, we also need to introudce an *activation function*. This function takes as input the total input current to the neuron, and outputs the firing rate of the neuron. The use of the activation function is meant to reflect the biological fact that neurons do not fire at a constant rate, but in a non-linear fashion. Thus, a non-linear function, like the *Sigmoid* or *tanh* function is generally used.

2.2 The Dynamics of a Self-Connected Neuron

For our neuron, we will use the following activation function:

$$f(s) = 60(1 + \tanh(s)) \tag{2}$$

Where s is the total input to the neuron. Figure 1 below illustrates the behavior of this activation function for a range of input values.

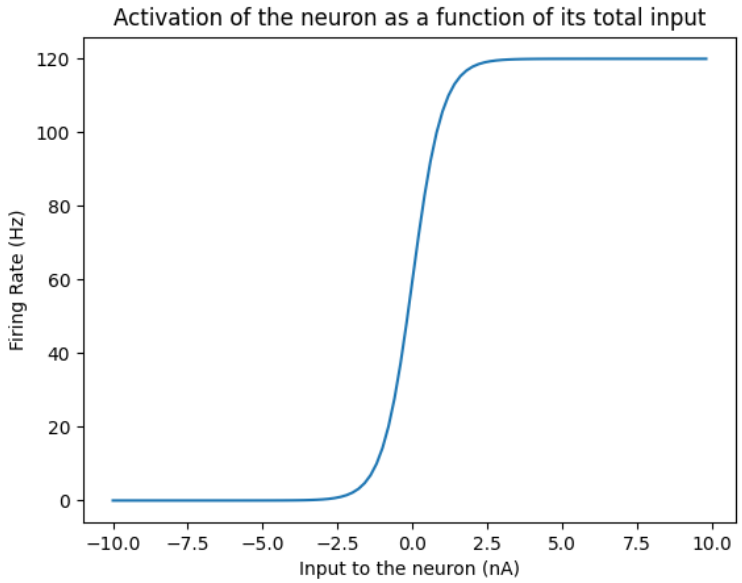

Figure 1: Resultant firing rate of the neuron as a function of the total input current, given the activation function in equation 2.

As we can see from Figure 1, this activation function would result in a neuron with a firing rate of 0 Hz (inhibited) for negative input values far from 0, but that begins to fire (becomes excited) once the input current approaches 0 from the left (around -2.5nA). When the input current is 0, it has a firing rate of 60Hz. As the input increases, the firing rate also increases, but in a non-linear fashion: the maximum firing rate is 120Hz for all values of input current greater than about 2nA.

2.3 Dynamics Over Time

Having established the activation function for our neuron, we can now examine how the neuron's firing rate changes over time. Equation 3 below describes these dynamics:

$$\frac{dr(t)}{dt} = -r(t) + f(wr(t) + I) \tag{3}$$

where $r(t)$ is the firing rate of the neuron at time t, f is the activation function, w is the weight of the connection between the neuron and itself, and I is the background input to the neuron. For our model, we will set $w = 0.05$ and I = -3. Figure 2 below shows the evolution of our neuron's firing rate over time, given different initial conditions r_0, where $dt = 0.1$.

From Figure 2, we see that, when the initial firing rate is less than 60, the neuron's activity tends towards 0. This is because $\frac{dr(t)}{dt}$ is negative when $r(t) < 60$, and approaches 0 as $r(t)$ approaches 0. Conversely, when the initial firing rate is greater than 60, the neuron's activity tends towards 120, since $\frac{dr(t)}{dt}$ is positive when $r(t) > 60$, and approaches 0 as $r(t)$ approaches 120. Finally, when the initial firing rate is 60, the neuron's activity remains constant at 60, since $\frac{dr(t)}{dt} = 0$ when $r(t) = 60$.

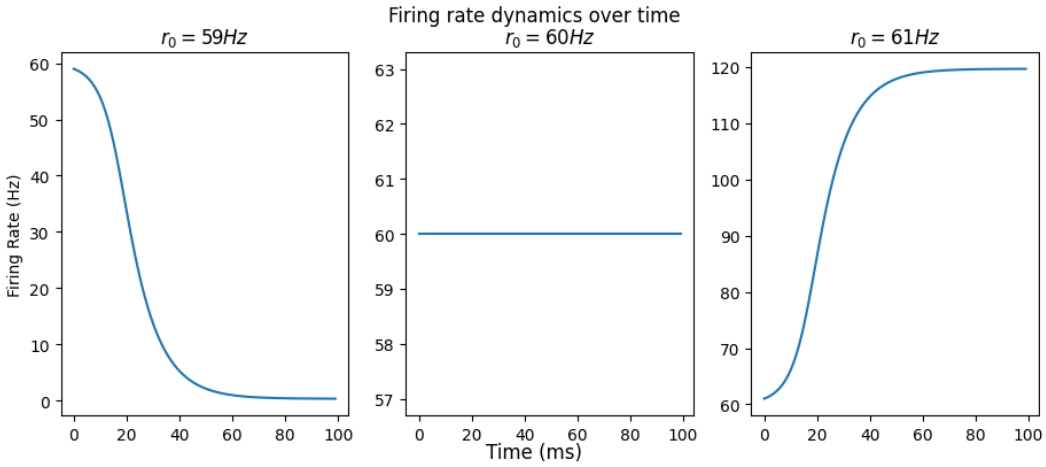

Figure 2: Firing rate date of the neuron over time, with varying initial conditions r_0. $dt = 0.1$.

To make this model more realistic, we can add gaussian noise to mimic the noisy behavior of biological neurons. The dynamics of our noisy neuron model are given by the following equation:

$$\frac{dr(t)}{dt} = -r(t) + f(wr(t) + I) + \sigma\eta(t) \tag{4}$$

Where $\eta(t)$ is a gaussian white noise process, and σ is the magnitude of the noise. Figure 3 below illustrates the dynamics of our noisy neuron model, with varying values of σ, for the three different initial conditions.

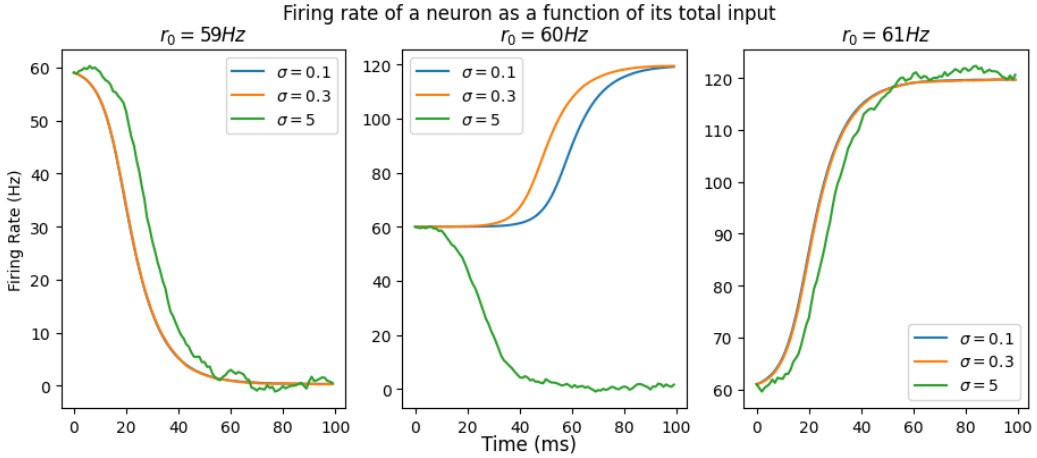

Figure 3: Firing rate dynamics of the noisy neuron over time for varying σ magnitudes, with different r_0.

From Figure 3, we see that adding noise to the neuron's dynamics does not change the firing rate the neuron evolves to in all cases except $r_0 = 60$. When $r_0 = 60Hz$, noise in the activity of the neuron can cause it to evolve away from the previous stable point of $60Hz$, and towards either 0Hz or 120H, because $\frac{dr(t)}{dt} = 0$ only at 60.

2.4 Attractor States and 0 Crossings

We have repeatedly noted that the neuron's activity evolves to a discrete number of states given different initial conditions. These states are the *attractor states* of our neruon, given our initial conditions I, w and $f(s)$. As seen from Fig 1, they are 0Hz, 60Hz, and 120Hz. These are the values at which there will be no change in the neuron's firing rate over time. That is, these are the values for which $\frac{dr(t)}{dt} = 0$. Thus, these values correspond to the *0 crossings* of the neuronal dynamics. Figure 4. below illustrates this by showing how $\frac{dr(t)}{dt}$ evolves over different values of $r(t)$.

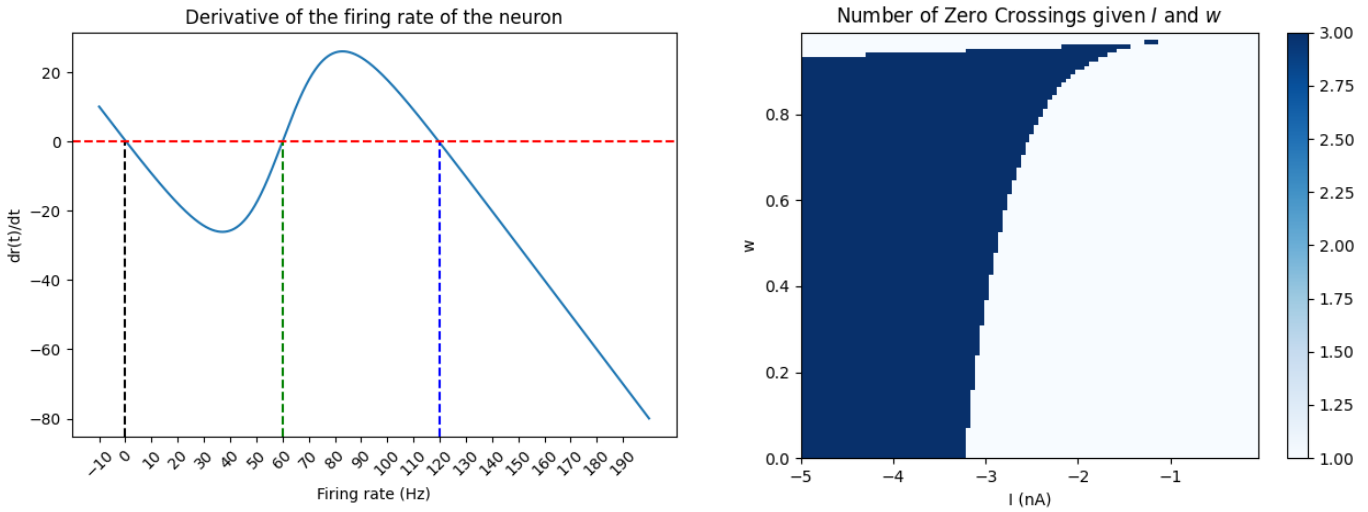

Figure 4: The left hand plot illustrates the derivative of the neuron's firing rate over time, given different values of the firing rate. $w = 0.05$ and I = -3. As indicated by the dashed lines, the 0 crossings of the neuron's dynamics are at 0Hz, 60Hz, and 120. The right hand plot is the bifurcation diagram showing the number of zero crossings (and thus attractor states) as a function of the weight w and the background current I for our neuron model. The colors indicate the number of zero crossings.

As we can see from Figure 4, the 0 crossings of the neuron's dynamics are exactly at 0Hz, 60Hz, and 120Hz. However, this is not neccesarly the case: the number of attractor states can vary depending on the parameters of the system. Figure 5 below illustrates how the number of attractor states changes as we vary the weight w of the connection between the neuron and itself, as well as the backgorund current I.

From Figure 4, we also see that the number of solutions to $\frac{dr(t)}{dt} = 0$ varies from 3 to 1, as I varies from -5 to 0, and w varies from 0 to 1. Likewise, this implies that the number of attractor states varies between 3 and 1, depending on the values of I and w. As I grows, the number of solutions drops to 1, except for high values of w. Once I is sufficiently large, there is only one solution regardles of the value of w. Moreover, this is an abrupt change: the number of attractor states is either 3 or 1, with no in-between values. This is an example of a *bifurcation* in the system's dynamics.

3 A Two Neuron Network

We will now move on to a more complex neural network, consisting of two interconnected neurons. We will thus need two different equations to characterize their dynamics, one for each neuron. Since the neurons are connected to each other, they influence each other's dynamics. Thus, the equations for the two neurons must be coupled. This can be done by passing the other neuron's activity as an argument to the activation function of each neuron. The coupled equations for the two neurons are thus as follows:

$$\begin{aligned} \frac{dx(t)}{dt} &= -x(t) + f(w_2 y(t) + I) \\ \frac{dy(t)}{dt} &= -y(t) + f(w_1 x(t) + I) \end{aligned} \tag{5}$$

Where, $x(t)$ and $y(t)$ are the firing rates of neurons x and y at time t, respectively, f is the activation function, w_1 and w_2 are the weights of the connections between the neurons, and I is the background input to the neurons. For our model, we will set $w_1 = w_2 = 0.4, I = -10$nA. Moreover, our activation funciton will be the sigmoid function:

$$f(s) = 50\sigma(s) = 50\frac{1}{1 + e^{-s}} \tag{6}$$

Where s is the total input to the neuron. Figure 5 below illustrates the dynamics of the two neurons over time, given different initial conditions x_0 and y_0.

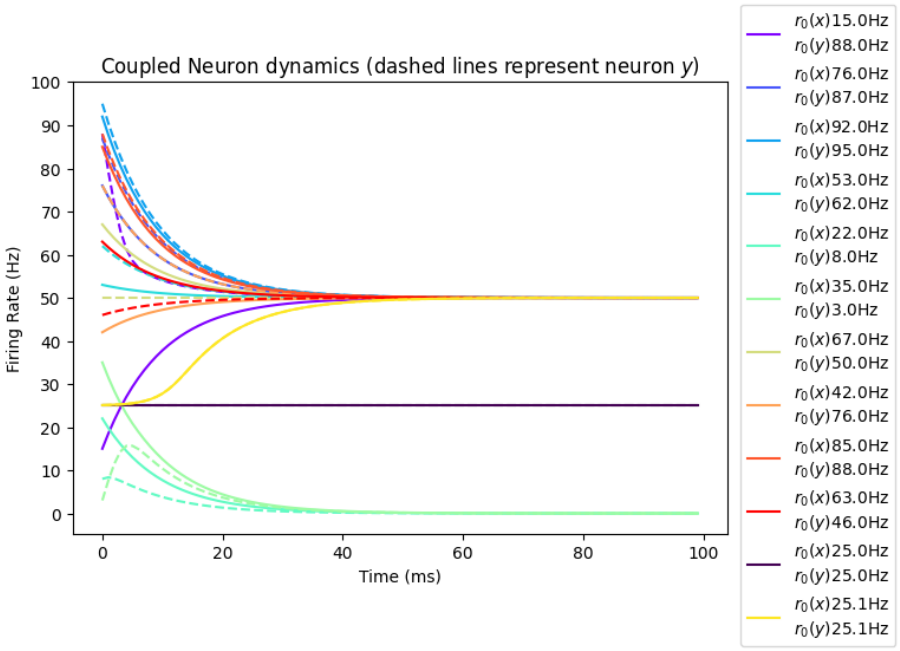

Figure 5: Dynamics of two coupled neurons over time. The similarly colored curves correspond to the same simulation. The dashed line curves indicate the behavior of neuron y throughout.

From Figure 6, we can see that this network has three attractor states: namely, when the inital firing rate of at leaste one of the neurons is $\geq 50Hz$, each neuron evolves to a fixed point of 50Hz. However, if the initial firing rate of both neurons is less than 50Hz, the neurons evolve to a fixed point of 0Hz. Finally, if the initial conditions of both neurons is 25Hz, the neurons will remain at 25Hz.

3.1 Nullclines and Fixed Points

We can get a better understanding of the dynamics of this system by looking at the trajectories of the dynamics in the state space for different initial conditions, as well as the *nullclines* of the system. The nullclines are the points at which the derivative of the firing rate of the neurons is 0, i.e., the points at which the firing rate of the neurons does not change over time. We can find the nullclines of our system by setting $\frac{dx(t)}{dt} = 0$ and $\frac{dy(t)}{dt} = 0$ from equation 5, and solving for the respective x and y values.

Taking neuron x as an example, first solving for x:

$$\begin{aligned} 0 &= -x + f(w_2 y + I) \\ x &= f(w_2 y + I) \\ x &= 50\frac{1}{1 + e^{-(w_2 y + I)}} \end{aligned}$$

and solving for y:

$$\begin{aligned} 0 &= -x + 50\frac{1}{1 + e^{-(w_2 y + I)}} \\ x &= 50\frac{1}{1 + e^{-(w_2 y + I)}} \\ 50 &= x(1 + e^{-(w_2 y + I)}) \\ \frac{50}{x} &= 1 + e^{-(w_2 y + I)} \\ \frac{50}{x} - 1 &= e^{-(w_2 y + I)} \\ \ln(\frac{50}{x} - 1) &= -(w_2 y + I) \\ \frac{\ln(\frac{50}{x} - 1) + I}{-w_2} &= y \end{aligned}$$

The same process can be repeated for neuron y, but with w_1 instead of w_2. However, the curves produced by both of these equations are identical (where $\ln(\frac{50}{x} - 1)$ is defined). Figure 7 below illustrates the nullclines of the system, as well as the trajectories of the system in the state space for different initial conditions.

Solving these equations numerically, we find that the fixed points are: (0,0), (50,50), and (25, 25). Figure 6 below confirms this as these are exaclty the points where the nullclines cross. However, from the figure, we can see that these fixed points are not all of the same nature. The dynamics seem to converge to either (0,0) or (50,50), while they seem to diverge from (25,25) (unless they start there).

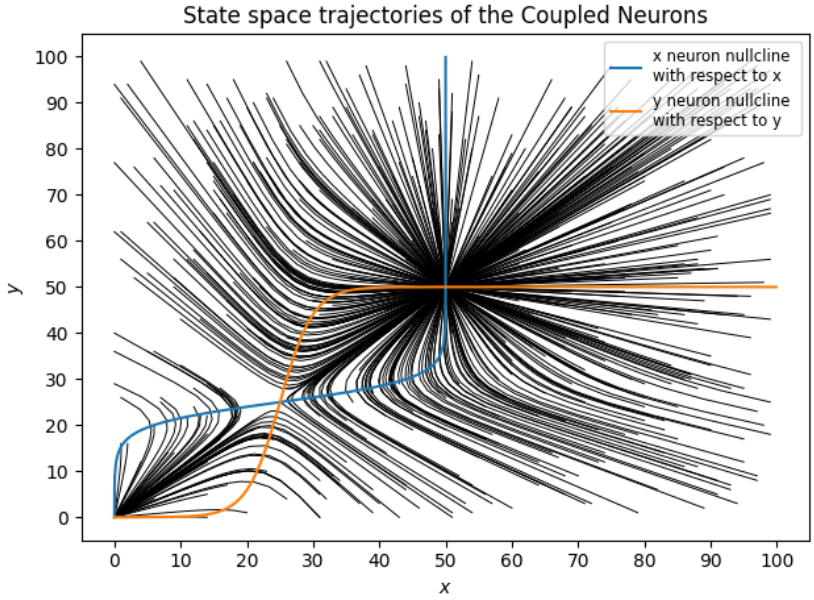

Figure 6: The trajectories in the state space for different initial conditions of the two neurons.

3.2 The Jacobian

We can get a better understanding of the nature of the fixed points of the system by studying the Jacobian matrix of the system at the fixed points. The determinant and trace of the Jacobian at these points can tell us whether the dynamics move toward or away from the point, and whether it is stable or unstable (respectively).

The Jacobian matrix of a two unit system at a fixed point (x^*, y^*) is given by:

$$J = \begin{pmatrix} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial y} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial y} \end{pmatrix}\Bigg|_{(x^*,y^*)} \tag{7}$$

where $\frac{\partial F_1}{\partial x}$ is differential equation for unit 1 with respect to x, and $\frac{\partial F_1}{\partial y}$ is the differential equation for unit 1 with respect to y, and similarly for unit 2. For our system of neurons, the Jacobian matrix at a fixed point (x^*, y^*) is given by:

$$J = \begin{pmatrix} \frac{\partial}{\partial x}(-x + 50\sigma(w_2 y + I)) & \frac{\partial}{\partial y}(-x + 50\sigma(w_2 y + I)) \\ \frac{\partial}{\partial x}(-y + 50\sigma(w_1 x + I)) & \frac{\partial}{\partial y}(-y + 50\sigma(w_1 x + I)) \end{pmatrix}\Bigg|_{(x^*,y^*)} \tag{8}$$

Taking the fixed point $(0, 0)$ as an example, the Jacobian evaluated at this point is given by:

$$\begin{aligned} \frac{\partial F_1}{\partial x} &= \frac{\partial}{\partial x}(-x + 50\sigma(w_2 y + I)) \\ &= -\frac{\partial}{\partial x}x + 50\frac{\partial}{\partial x}\sigma(w_2 y + I) && \text{sum rule} \\ &= -1 + 50\frac{\partial}{\partial x}\sigma(w_2 y + I) && \text{derivative of x} \\ &= -1 + 50 * 0 && \text{treating } y \text{ as a constant} \\ &= -1 \end{aligned}$$

Similarly,

$$\frac{\partial F_2}{\partial y} = \frac{\partial}{\partial y}(-y + 50\sigma(w_2 x + I)) = -1$$

Furthermore,

$$\begin{aligned}
\frac{\partial F_1}{\partial y} &= \frac{\partial}{\partial y}(-x + 50\sigma(w_2 y + I)) \\
&= -\frac{\partial}{\partial y}x + 50\frac{\partial}{\partial y}\sigma(w_2 y + I) && \text{sum rule} \\
&= 0 + 50\frac{\partial}{\partial y}\sigma(w_2 y + I) && \text{derivative of x as a constant} \\
&= 50\sigma'(w_2 y + I) * w_2 && \text{chain rule} \\
&= 50\sigma(w_2 y + I)(1 - \sigma(w_2 y + I)) * w_2 && \text{derivative of sigmoid}
\end{aligned}$$

Evaluated at $(0, 0)$, this becomes:

$$\begin{aligned}
\frac{\partial F_1}{\partial y} &= 50\sigma(0.4 * 0 - 10)(1 - \sigma(0.4 * 0 - 10)) * 0.4 \\
&= 50\sigma(-10)(1 - \sigma(-10)) * 0.4 \\
&= 0.000907
\end{aligned}$$

Similarly,

$$\frac{\partial F_2}{\partial x} = 50\sigma(w_1 x + I)(1 - \sigma(w_1 x + I)) * w_1 = 0.000907$$

Therefore, the Jacobian evaluate at the fixed point (0,0) is:

$$J = \begin{pmatrix} -1 & 0.000907 \\ 0.000907 & -1 \end{pmatrix} \tag{9}$$

We can then compute the trace and determinant of the Jacobian to determine the nature of the fixed point. If $\det(J_{xy}) > 0$ and $\text{tr}(J_{xy}) < 0$ the point is a stable node, meaning states will evolve to this point. If the $\det(J_{xy})$ is negative, the point is a saddle point, meaning states will move away from this point, except for states that start at this point. Finally, if $\det(J_{xy})$ and $\text{tr}(J_{xy})$ are positive, the point is an unstable node, meaning states will move away from this point.

The determinant at point (0,0) is given by $\det(J) = -1^2 - 0.000907^2 = 0.999999187$. The trace at point (0,0) is given by $\text{tr}(J) = -1 - 1 = -2$. Therefore, (0,0) is a stable node in the system.

We can do similar caclulations with the other points we have identified. Regarding point (50, 50): $\det(J_{50,50}) = -1^2 - 0.000907^2 = 0.999999187$, and $\text{tr}(J_{50,50}) = -1 - 1 = -2$. Therefore (50, 50) is a stable node in the system.

Regarding point (25, 25): $\det(J_{25,25}) = -1^2 - 5^2 = -26$, and $\text{tr}(J_{25,25}) = -1 - 1 = -2$. Therefore (25, 25) is a saddle point in the system: initial conditions starting at this point will stay at it, whereas initial conditions close to it will move away from it.

3.3 Regimes of the System

We can also study the behavior of the system as the background input current I varies. Figure 7 below illustrates how the nullclines and the number of fixed points varies as I changes from -10 to -20. As we can see from Figure 8 below, as I decreases, the amount of fixed points, and thus the number of solutions to the equations, drops from 3 to 1. However, there is a small region where there are two solutions, around 16nA. This is expected as the nullcline plot shows the nullclines moving away from each other as I decreases. However, there should be a value for I where they touch in exaclty 2 places. This type of bifurcation is known as a sadlle-node bifurcation

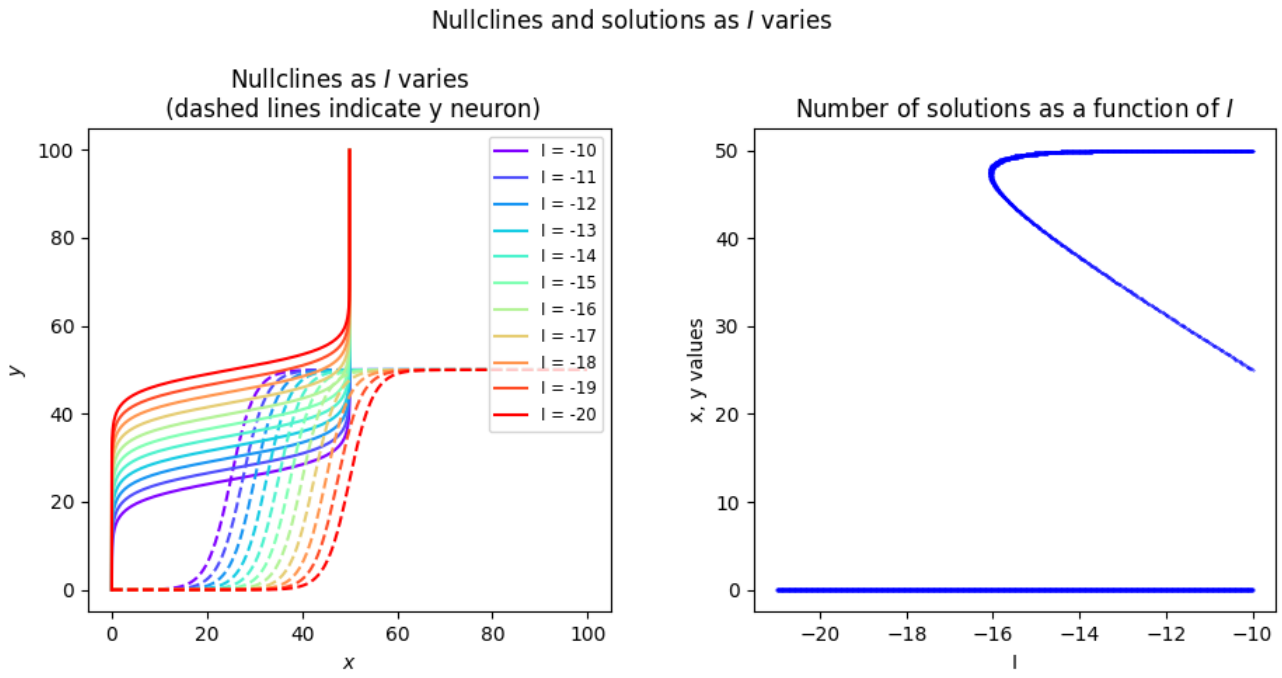

Figure 7: The nullclines and number of solutions for the system as the background input current I varies. The number of solutions is indicated by summing the number of points at each index on the x-axis.

4 Hoppfield Networks

We will now move on to a type of network called a Reccurent Neural Network (RNN). It is 'Recurrent' because all the neurons are connected to each other, as well as themselves. In fact, the first network we looked at was a simple example of an RNN. However, the network we will examine now consists of 64 inter-and-self connected neurons. Specifically, we will be looking at a Hoppfield Network. These types of network are designed to mimic associative memory in humans: they can store patterns as attractor states in the network, so as to 'recall' them when given a noisy input similar to the patterns.

The dynamics of this network are given by:

$$\frac{d\vec{x}(t)}{dt} = -\vec{x}(t) + f(W\vec{x}(t)) + \sigma\vec{\eta}(t) \tag{10}$$

Where $\vec{x}$ is a 64-dimensional vector storing the activation of the 64 neuorons in our network at time t, W is a 64x64 connectivity matrix, storing the strenghts of the connections between the neurons, $\vec{\eta}(t)$ is a noise term, and σ is the magnitude of the noise. Moreover, the activation funciton $f(s)$ we will use is the sign funciton, defined as:

$$f(s) = \text{sign}(s) = \begin{cases} 1 & \text{if } s > 0 \\ 0 & \text{if } s = 0 \\ -1 & \text{if } s \leq 0 \end{cases} \tag{11}$$

The weight matrix W is defined as follows:

$$W = \frac{1}{N}\vec{p}\,\vec{p}^{T} \tag{12}$$

where N is the number of neurons in the network, and $\vec{p}$ is the pattern vector we want to store.

4.1 Smiley

Let's create a pattern to store with the network. We will use the following 8x8 pattern, called "Smiley". Moreover, applying the formula descirbed in Equation 12, we can generate the weight matrix for Smiley. Smiley and his weight matrix is shown in Figure 8 below.

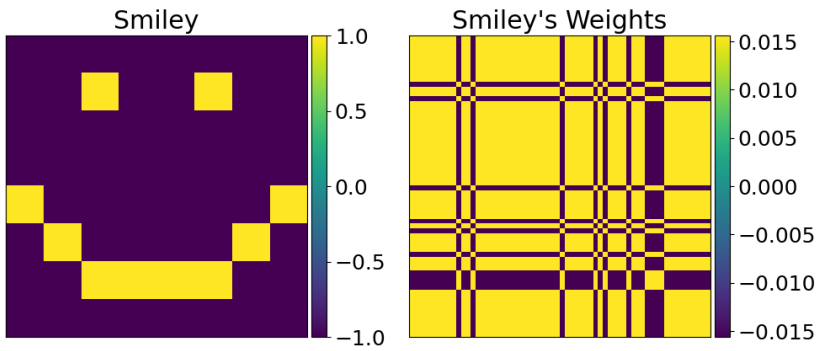

Figure 8: Smiley pattern as an 8x8 representation of our 64 element vector. Each element is either 1 or -1.

Now, we can simulate the network given the dynamics outlined in Equation 10, using $\sigma = 0.1$, where η is normally distirbuted random number with $\mu = 0$ and and $\sigma = 1$. Figure 9 below illustrates the evolution of the network over time, given a random initial condition.

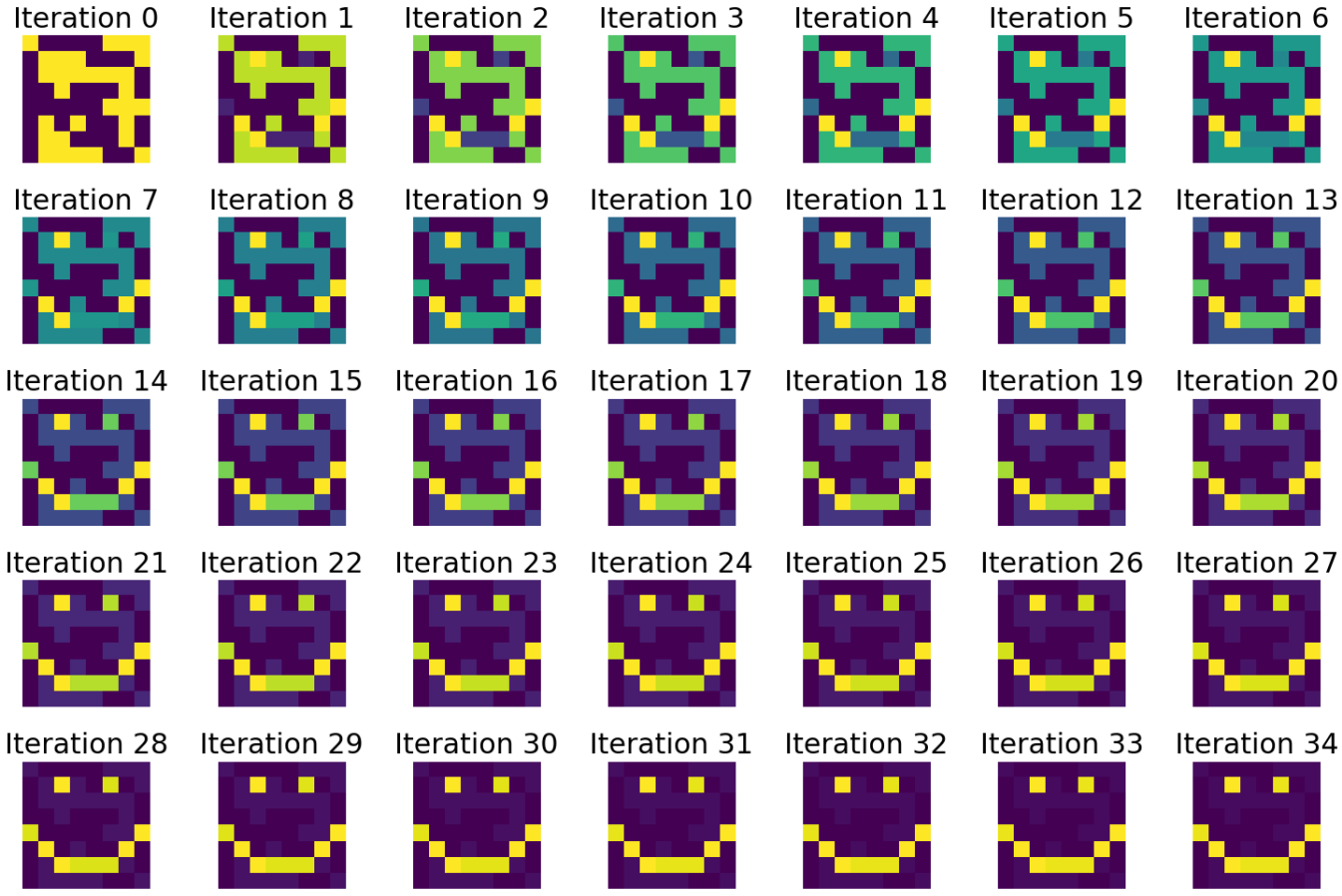

Figure 9: Evolution of the network over time, with 'Smiley' as the stored pattern.

Repeating the simulation with different initial conditions has the same result: the network always evolves to the stored pattern. This is because the pattern stored in the weights acts as the sole attractor state for the network.

4.2 Smiley and Frowny

Let's introduce another pattern into the network, called "Frowny". This pattern is shown in Figure 10 below. Moreover, we can store both Smiley and Frowny in the network by generating the weight matrix using both patterns, i.e., generating

the weight matrix as follows:

$$W = \frac{1}{N}(\vec{p}\,\vec{p}^{T} + \vec{q}\vec{q}^{T}) \tag{13}$$

This updated weight matrix is also shown in Figure 10 below.

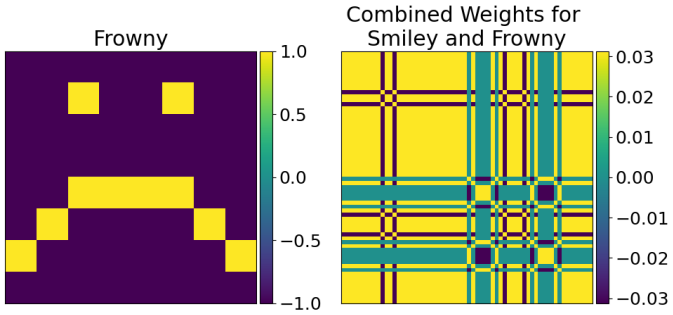

Figure 10: The pattern "Frowny" and the weight matrix generated from Equation 13, such that both frowny and smiley are stored in the network.

Having generated a combined weight matrix, we have effecitvely managed to store both patterns in the network. We can see this in Figure 12 below, which displays samples from two simulations of the updated network. The network evolves to either the Smiley or Frowny pattern, depending on the initial conditions. When the initial conditions are random, it evolves to either pattern, depending on which pattern is closer to the initial condition.

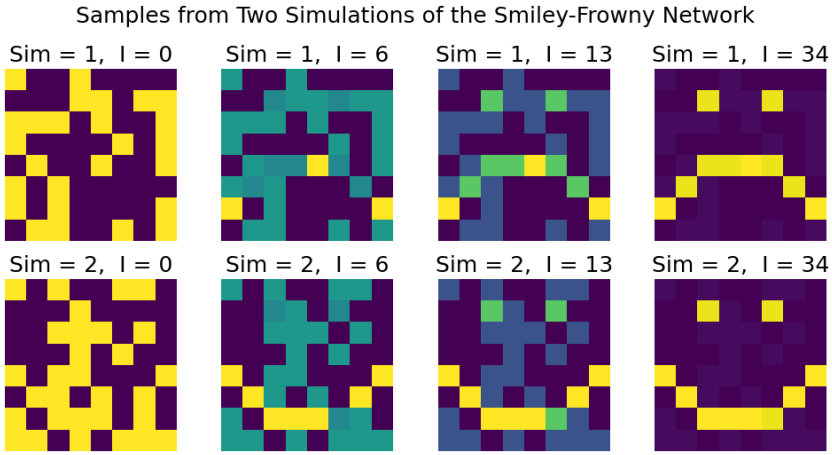

Figure 11: Samples of iterations from two differnet simulations of the Smiley-Frowny network. Each simulation starts with a random initial condition. 'Sim' indicates which simulation the frame belongs to, while I indicates the iteration number within the simulation.

However, when we pass a 'corrupted' version of the Smiley pattern as the initial condition, i.e., the Smiley pattern with some guassian noise added to each element, the network evolves to the uncorrupted version of the pattern. This is shown in Figure 12 below.

Figure 12: Simley-Frowny network given a corrupted Smiley pattern as an initial condition. The network evolves to the un-corrupted version of the pattern.

4.3 The Emotions

We can, of course, attempt to store more than two patterns in the network. Take, for example, the family of patterns in Figure 13 below.

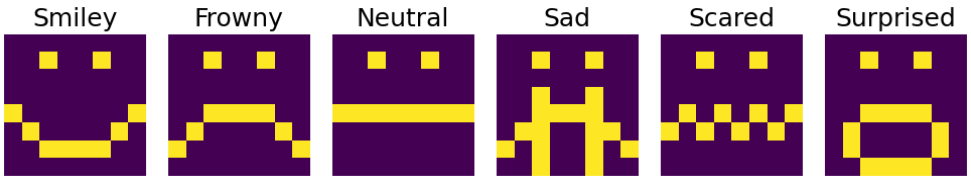

Figure 13: A famillt of patterns to store in the network.

We can store all of these patterns in the network by generating the weight matrix as follows:

$$W = \frac{1}{N} \sum_i \vec{p_i} \vec{p_i}^T \tag{14}$$

However, such a pattern poses problems, and highlights a limitation of this type of network. Namely, since they patterns are very similar, it is difficult to store all of them in the network. In this case, the network can only store any 2 of the 6 patterns displayed in figure 13, when the initial conditions are close to the patterns. On the other hand, if our patterns are completely random, and thus dissimilar, and we start the network near the patterns, it can store a maximum of around 17 patterns. This is demonstrated in Figure 14 below.

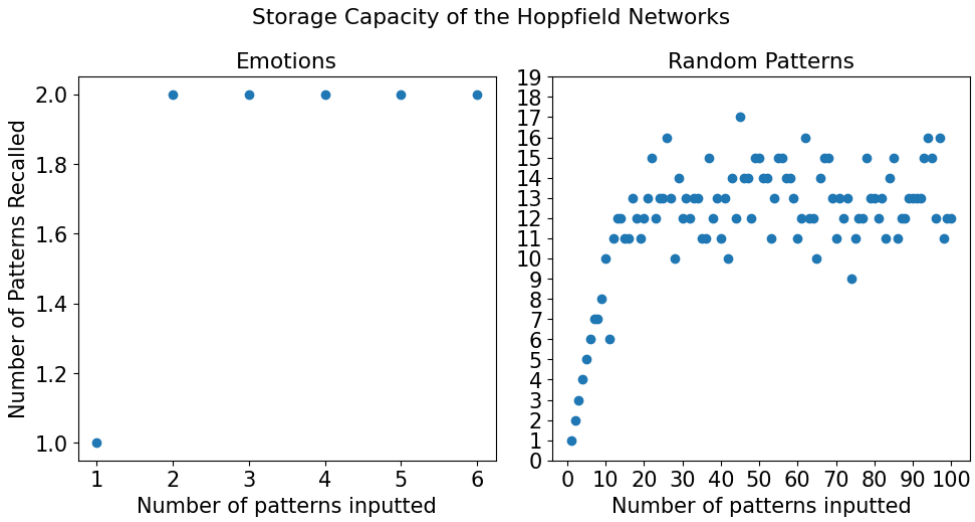

Figure 14: Storage Capacity of a hoppfield network, when attempting to store different kinds of patterns.

5 Principal Component Analysis and Ring Attractors

5.1 A 100 Neuron network

Unfortunately, the brain consists of billions of neurons. Thus, the dimensionality of their networks is far greater than even the 64 neuron network we have worked with thus far. This makes it difficult to visualize, and therefore get a sense of the dynamics of this network. Take, for example, the following network, composed of 100 neurons and described by:

$$\frac{d\vec{x}(t)}{dt} = -\vec{x}(t) + J\phi(\vec{x}(t)) \tag{15}$$

where $\vec{x}$ is a 100-dimensional vector storing the firing rates of the 100 neurons in our network at time t, J is a 100x100 connectivity matrix, given by:

$$J_{i,j} = 2\cos(\theta_i - \theta_j) \tag{16}$$

where $\theta \in [0, 2\pi)$, and ϕ, the activation function is given by:

$$\phi(s) = \tanh(s) \tag{17}$$

where s is the total input to the neuron.

Figure 15. below illustrates the connectivity matrix, as well as the network dynamics resulting from this setup.

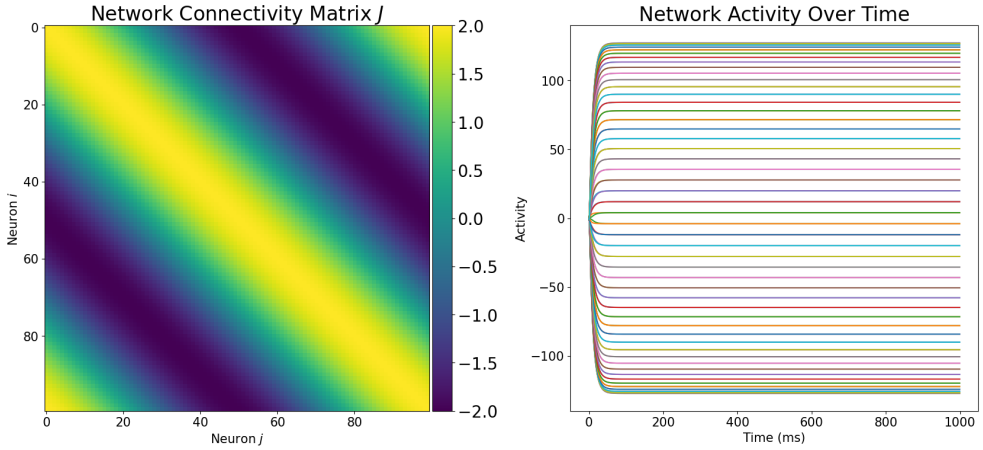

Figure 15: The dynamics of a 100 neuron network over time, and it's connectivtiy matrix, given the connectivity matrix in Equation 16 and the activation function in Equation 17. The initial conditions are random numbers between -1 and 1.

As we can see from Figure 15, unlike our previous simple networks, the 100 Neuron network evolves into a multitude of fixed points given different initial conditions. Moreover, we can see that the connectivity matrix appears to be symmetrical, with a periodically repeating pattern. This is because the connectivity matrix is generated by the cosine function, which is periodic.

5.2 Principle Component Analysis

We can gain more insight if we reduce the dimensionality of the network, using a dimensionality reduciton technique such as Principal Component Analysis (PCA). PCA allows us to reduce the dimensionality of the network by projecting the data onto the eigenvectors of the covariance matrix of the data. This allows us to isolate the features of the data that are the most explanatory, i.e., that have the most variance. We can then plot these features to get a better understanding of the dynamics.

First, we have to collect the the dynamics of the system into a matrix X, which has dimensions $N_{neurons}$ x $T_{timesteps}$, where each column is a time-step, and each row is an individual neuron. Then, we center the data by subtracting the mean of each row from each element in the row:

$$X' = X - ((X \cdot O)\frac{1}{T}) \tag{18}$$

where O is a matrix of ones with dimensions T x 1, and T is the number of time steps. Next, we compute Σ, the covariance matrix of the centered data. This is given by:

$$\Sigma = \frac{1}{T}X'X'^{t} \tag{19}$$

where X' is the centered data matrix, and X'^t is the transpose of X'.

Finally, we compute the eigen values and eigenvectors of Σ. Figure 16 below illustrates the covariance matrix of X', as well as the eigenvalues of the covaraince matrix. By plotting the eigenvalues, we can see how much variance each eigenvector explains, and importantly, how many eigenvectors we need to capture most of the variance in the data.

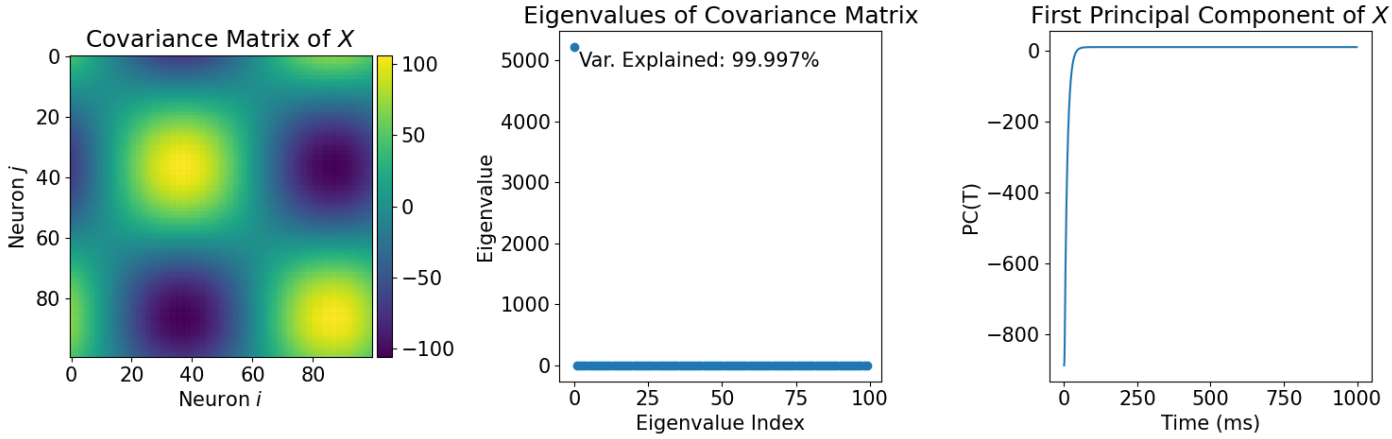

Figure 16: The Covariance matrix of the centered data X', and the eigenvalues of the covariance matrix. The eigenvalues indicate how much variance each eigenvector explains. Given our PCA transformation, only the first eigenvector is needed to capture most of the variance in the data.

This first principal component is the only one that explains a signifcant portion of the variance, and indicates that the dynamics of the network over time are unidimensional. We can then project the data onto this eigenvector, demonstrated in equation 20 below:

$$X_{PC} = X'^t V_d \tag{20}$$

where X_{PC} is a one column matrix of the data projected onto the first principal component, and V_d is the eigenvector corresponding to the first eigenvalue of the covariance matrix. From Fig. 16, we see that the activity initially changes rapidly, but then settles into a fixed state. This reflects the network activity dynamics we observed in Figure 15.

5.3 PCA on Attractor States

However, we can also apply PCA to the attractor states of the network. This way, we can gain a better understading of the nature of the attractor states of the network. To do this, we simulate the network 500 times, and collect the last column of each simulation into a matrix Q. We then follow the procedure outlined above in equations 18 and 19 to compute the covariance matrix of the attractor states, and the eigenvalues of the covariance matrix (note, $T = 500$ in this case, since Q has 500 columns instead of 1000).

Figure 17 shows the resulting covariance matrix and eigen values of the attractor states of the network. Interestingly, the covariance matrix is idential to the connectivity matrix. This makes sense, since the covariance matrix contains measures of the linear relations between variables accross time. At the end of the simulation, the neurons have settled into their fixed points as defined by the weights between them. Moreover, plotting the eigenvalues of the covariance matrix now indicates that both the first and second ranked eigenvectors are needed to capture most of the variance in the data, as they capture 52.086% and 47.914% of the variance, respectively. Thus, we should project the attractor states of the network projected onto the first two principal components.

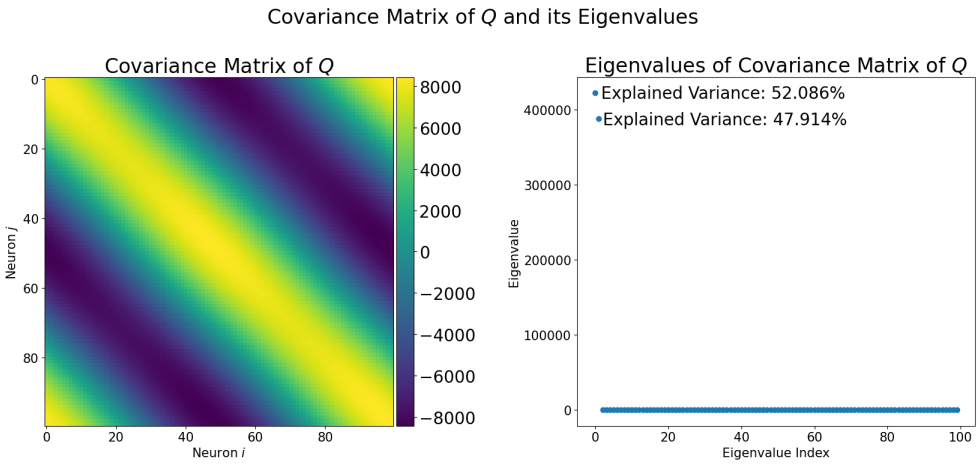

Figure 17: The Covarinace Matrix of the attractor states of the network, and the eigenvalues of the covariance matrix. The top eigenvalue explains 52.086% of the variance, while the penultimate eigenvalue explains 47.914% of the variance. Together, they explain 100% of the variance in the network's attractor states.

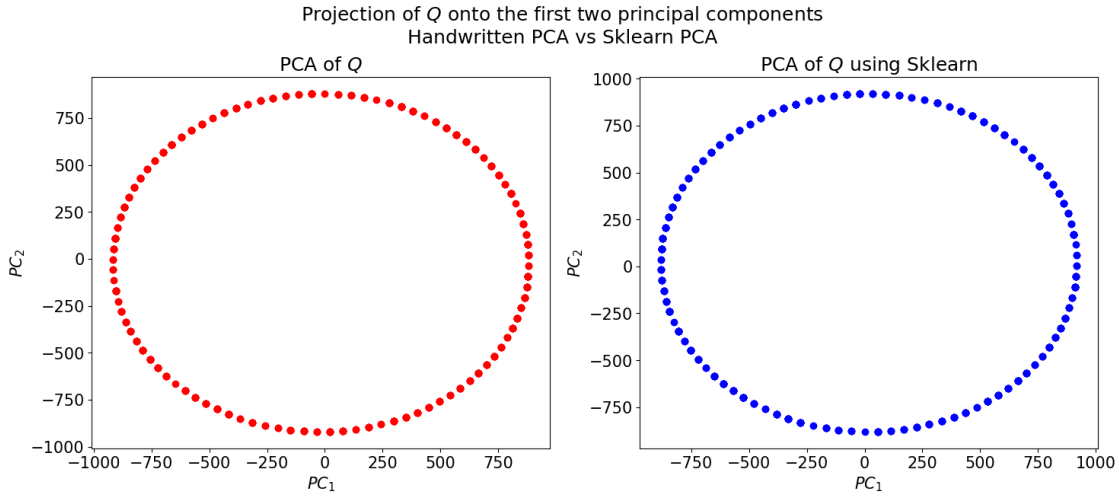

Figure 18: The attractor states of the network projected onto the first two principal components. The projection indicates that this is a 'ring' attractor network

Figure 18 shows the centered attractor states projected onto the first two principal components (done both manually and in Sklearn). The projection reveals that the attractor states of the network form a *ring attractor*. Namely, the attractor states of the network form a ring-like pattern. Thus, regardles of the initial conditions, the network will always evolve to one of the states on the ring. Biologically, this type of attractor state is commonly linked to animal spatial awareness and navigation. For example, the neural circuit that supports the head-direction of fruit flies is a ring attractor network.

6 Conclusion

In this report we have examined neural networks of increasing complexity. We have seen that, while the complexity of these networks differ, we can describe them in terms of *fixed points* and *0 crossings*. Moreover, we have seen that networks of neurons can be used to perform tasks, such as storing memories. Finally, we have seen how PCA can be used to gain a better understanding of the dynamics of high dimensional networks. Since brains are a high dimensional network, PCA is a key tool in understanding the dynamics of biological neural networks.