Order from Chaos by FORCE

Ciprian Bangu

9 June 2024

Contents

1 Introduction

Amazingly, biological brains manage to organize spontaneous chaotic activity into coherent patterns. Reccurent Neural Networks (RNNs), which aim to emulate the brain's activity, have historically struggled to bridge this gap. However, this report will present a network architecture, and learning procedure, developed by Sussillo and Abbott (2009) that is able to make exactly this type of transition: from chaotic activity to the approximation of a target function. This report will first outline the network architecture, and describe the role of chaotic spontaneous activity in it. It will then detail the learning procedure, FORCE, and provide examples of the networks performance following this training ofr several target funcitons. Finally, it will conuct a Principle Component Analysis (PCA) of the network activity, and discuss the biological plausibility of the network.

2 Network Architecture

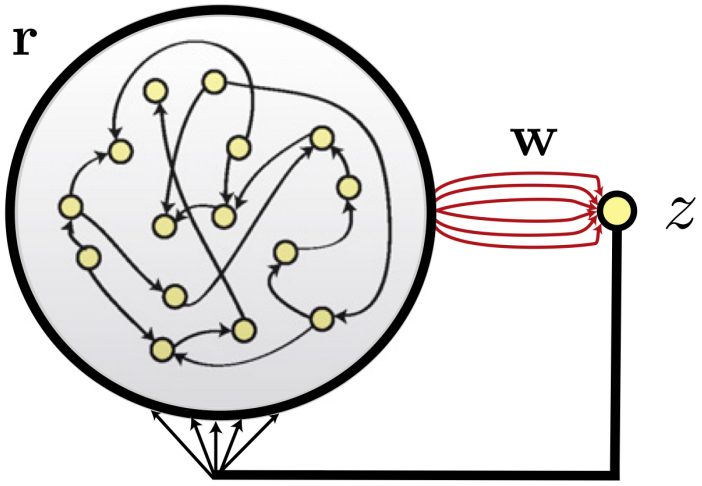

Figure 1: Overivew of the network architecture.

The network described by Sussilo and Abbott, and used throughout this report, is a general purpose recurrent neural network, composed of $N = 1000$ neurons (Sussilo and Abbott, 2009). Initialized randomly, these neurons are sparsely interconnected by an N x N activty matrix, $\mathbf{M}$, with a density $p = 0.1$, which is randomly generated at the beginning of each simulation. Furthermore, the weights between these connections is given by an N x 1 vector $\mathbf{w_f}$, which is also randomly initialized at the beginning of each simulation.

For this network, the activation function of each neuron is given by the tanh function (Sussillo & Abbott, 2009), i.e.,

$$\phi(x) = \tanh(x) \tag{1}$$

The output weights, $\mathbf{w}$, are randomly initialized (between 0 and 1) at the beginning of each simulation, to be updated by the network to better approximate the target funciton z at each timestep. They are organized into a N x 1 vector, and scaled by a factor of $1/\sqrt{N}$ (Sussillo & Abbott, 2009).

Thus, at each timestep, $\Delta t = 0.1$, the network activity is:

$$\mathbf{x}(t) = (1 - \Delta t)\mathbf{x}(t-1) + \mathbf{M} \cdot (\phi(\mathbf{x}(t-1))\Delta t) + \mathbf{w_f}(z(t)\Delta t) \tag{2}$$

where $\mathbf{x}(t-1)$ is the neuronal activtiy at time $t-1$, and $\mathbf{z}(t)$ is the network output, given by:

$$z(t) = \mathbf{w}^T\mathbf{r}(t) \tag{3}$$

where $\mathbf{r}(t)$ is the network activity passed through the activation function at time t.

This output is then fed back into the network, and the weights are updated using the FORCE method, which will be discussed in the next section.

2.1 Chaotic Spontaneous Activity

A crucial, and interesting aspect of the network is the presence of chaotic activity in the network prior to learning. From a modelling perspective, models that produce realistic brain activity tend to behave chaotically (Sussillo & Abbott, 2009). Thus, RNNs that aim to model brain activity should be able to transition from a chaotic state, to a controlled one. With this in mind, a gain factor g is introduced into the network activity, which controls the type of activity produced by the network (Sussillo & Abbott, 2009). Specifically, this is a hyperparamter that scales the aforementioned network activity matrix. When $g < 1$, there is no network activtiy before training; when $g > 1$ the network activity becomes spontaneously chaotic (Sussillo & Abbott, 2009). As g increases past 1, the chaos becomes more pronounced. Figure 2 below illustrates this dynamic.

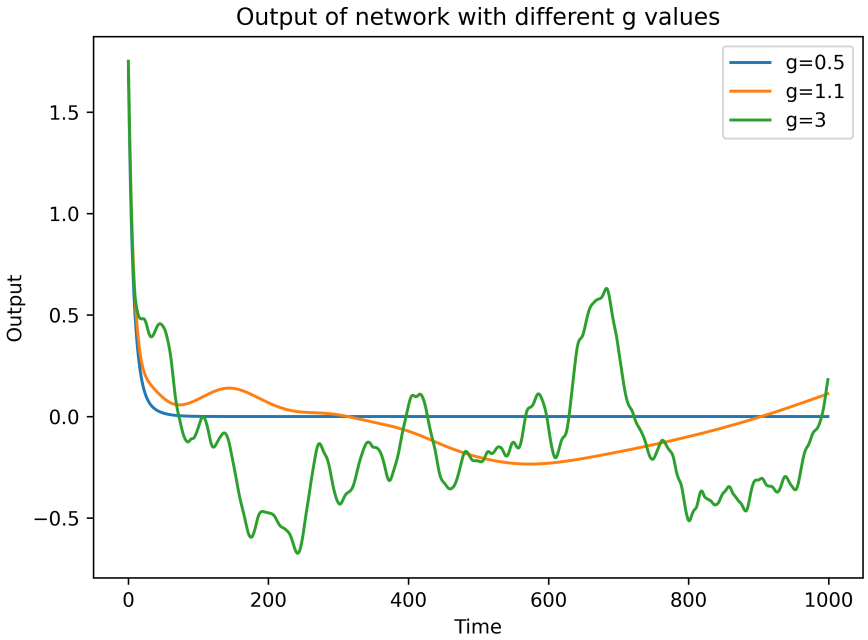

Figure 2: Spontaneous network activity given different values of g. When $g < 1$, there is no network activity before training; when $g > 1$ the network activity becomes spontaneously chaotic. As g gets farther from 1, the chaos becomes more pronounced.

For this report, we will set $g = 1.5$, as in the original paper (Sussillo & Abbott, 2009). They note that, while the network is able to learn with $g = 1$, the learning process is slower, and does not happen when $g < 0.75$ (Sussillo & Abbott, 2009). Moreover, there is also an upper limit: if $g > 1.56$, learning no longer converges (Sussillo & Abbott, 2009).

3 FORCE Learning

Core to the functioning of the introduced architecture is the learning method used in training the network. Sussillo and Abbott refer to this method as "first order reduced and controlled error" (2009), or FORCE Learning. FORCE learning updates the readout weights with the aim of minimizing the error between the network output $z(t)$ and the target funciton $f(t)$. However, unlike previous methods, FORCE learning updates network weights rapidly with small output errors, avoiding instabilities introduced by providing large errors to the network feedback mechanism (Sussillo & Abbott, 2009). This method solves one of the main challenges in using a feedback loop to train the kind of RNN described in the previous section. Specifically, a delay is introduced when the change in output that results from modifying $\mathbf{w}$ goes through the feedback pathway. This delay can result in a situation where the change made to $\mathbf{w}$ is actually counterproductive, even though it initally seemed promising (Sussillo & Abbott, 2009). FORCE learning avoids this issue by updating $\mathbf{w}$ rapidly with small errors, i.e., by keeping the actual output of the newtork close to the target output throughout trainig (Sussillo & Abbott, 2009).

3.1 Reccursive Least Squares

While a range of learning algorithms are suitable, the architecture developed by Sussilo and Abbott employs the Recursive Least Squares (RLS) algorithm. RLS is a form of adaptive filter for updating weights during

learning. Specifically, RLS aims to update the weights in order to minimize the following cost function:

$$J(t) = \sum_{i=1}^{T} \lambda^{T-i} e^2(i) \tag{4}$$

where $e(i)$ is the error at time i, and λ is a constant that determines the relative importance of past errors. For our network, we set $\lambda = 1$. To implement the algorithm, we first need to initialize the inverse correlation matrix $\mathbf{P}(0)$ (along with the network itself). The inverse correlation matrix is an N x N matrix, where N is the number of neurons in the network. $\mathbf{P}(t)$ represents the confidence in the weights $\mathbf{w}$ at time t. At the first step, $\mathbf{P}(0)$ is initialized as the identity matrix scaled by a constant α, i.e.,

$$\mathbf{P}(0) = \frac{\mathbf{I}}{\alpha} \tag{5}$$

Note, α is a constant hyperparameter that can be considered a 'learning rate' of the algorithm. Sussillo and Abbott note that smaller values for α can speed up learning, but lead to instability in the algorithm (Sussillo & Abbott, 2009). Conversely, α that are too large can result in the network not keeping the output close to the target function (Sussillo & Abbott, 2009).

The algorithm proceeds by calculating the gain vector $\mathbf{k}(t)$, which is used to guide the updating of the weights. Specifically, the direction of the gain vector indicates the direction that minimizes the error, while the magnitude determines the size of the update. The gain vector is given by:

$$\mathbf{k}(t) = \frac{\mathbf{P}(t)\mathbf{r}(t)}{1 + \mathbf{r}^T(t)\mathbf{P}(t-1)\mathbf{r}(t)} \tag{6}$$

We then calculate the prediction error $e(t)$, which is given by:

$$e(t) = \mathbf{w}^T(t - \Delta t)\mathbf{r}(t) - f(t) \tag{7}$$

then update the weights using a delta type rule,

$$\mathbf{w}(t) = \mathbf{w}(t - \Delta t) - e(t)\mathbf{P}(t)\mathbf{r}(t) \tag{8}$$

Finally, we update the inverse correlation matrix $\mathbf{P}(t)$:

$$\mathbf{P}(t) = \frac{1}{\lambda}(\mathbf{P}(t - \Delta t) - \mathbf{k}(t)\mathbf{r}^T(t)\mathbf{P}(t - \Delta t)) \tag{9}$$

$$\mathbf{P}(t) = \mathbf{P}(t - \Delta t) - \frac{\mathbf{P}(t - \Delta t)\mathbf{r}(t)\mathbf{r}^T\mathbf{P}(t - \Delta t)}{1 + \mathbf{r}^T(t)\mathbf{P}(t - \Delta t)\mathbf{r}(t)} \qquad \text{in terms of } \mathbf{P}(t - \Delta t) \tag{10}$$

In other delta-rule scenarios, the learning rate is a scaler. In RLS, however, $\mathbf{P}(t)$ takes this role and provides many learning rates for different parts of the network (Sussillo & Abbott, 2009). Throughout this report, this process will be used to train the network to generate the approximations found in the following sections.

4 Function Approximation

Having defined the network and the FORCE learning approach, we can now observe it's performance in approximating several target funcitons. Firstly, we will test the networks' performance on a trianglular wave function. This target function is defined as:

$$f(2\pi \cdot 5 \cdot t, 0.5) = -2\left(5t - \left\lfloor 5t + \frac{1}{2} \right\rfloor\right) \tag{11}$$

where 0.5 is the width of the rising amplitude (symetrical wave), and the freqeuncey is 5hz. In each of the figures, the network is initialized randomly following the architecture laid out in section 2. The network is first run to produce chaotic spontaneous activity, after which it is trained to match the target function. Figure 3 below shows this process for the triangular wave function.

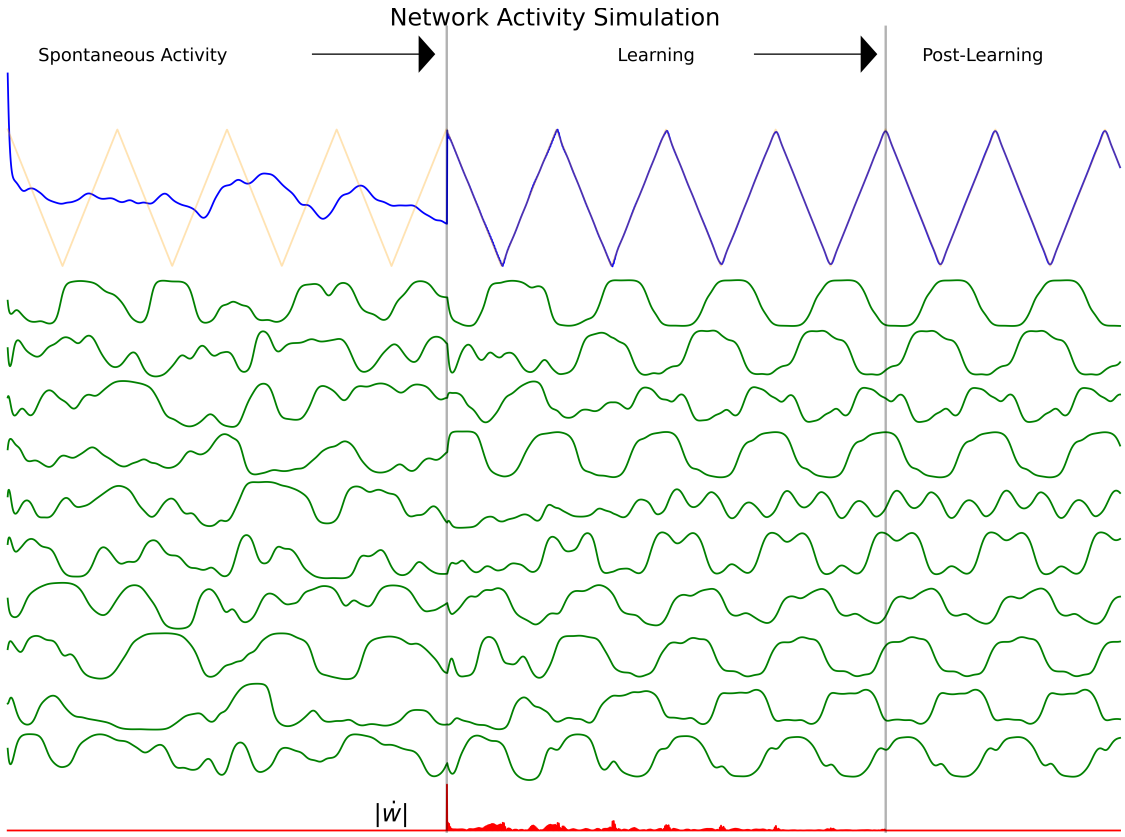

Figure 3: Illustration of the network activty (blue), a sample of neuronal activity (green), and magnitude of the time derivative of the weight readout vector (red) during before, during, and after training on the triangular wave function. The target funciton is also traced in orange, to illustrate that the network is able to approximate the target function. The grey bars demarcate the transition between the chaotic spontaneous activity and the training period.

As we can see from Fig 3. when first initialized, the network activity, in blue, is chaotic. As noted in the architecture, this is by desing as it will facilitate learning. The sample of 10 neurons during this time (green traces) reflect this chaotic behavior, as their activity is random and aperiodic. Moreover, we can also see the magnitude of the time derivative of the weight readout vector (red trace) is stable at 0, since the weights are not changing during this time. Once learning begins, we see that the network activity begins to approximate the target function. This is evidenced by the fact that the orange target trace is completely covered by the blue network trace. Similarly, we see that the neurons sampled begin to exhibit periodic behavior as their acitvity is forced to match the target function. Finally, we see that the time derivative of the weight readout vector is no longer stable at 0, but rather fluctuates as the weights are updated. As expected, we see strong changes in the beginning of training, that diminish over time. After the learning period is over, we see that the network output continues to approximate the target, as the network has settled into that state. Likewise, the neuron sample continues the pattern adopted during training, as the weights are no longer changing (evidenced by the 0-constant time derivative of the weight readout vector).

4.1 A Variety of Functions

The network preforms well on other periodic functions with varying levels of complexity as well. Figure 4 below shows the network's performance after training on a combination of four sinosoids defined as:

$$f(t) = \frac{2.6}{1.5}\left[\sin(\pi f_0 t) + \frac{1}{2}\sin(2\pi f_0 t) + \frac{1}{6}\sin(3\pi f_0 t) + \frac{1}{3}\sin(4\pi f_0 t)\right] \tag{12}$$

where $f_0 = \frac{1}{60}$ is the fundamental frequency.

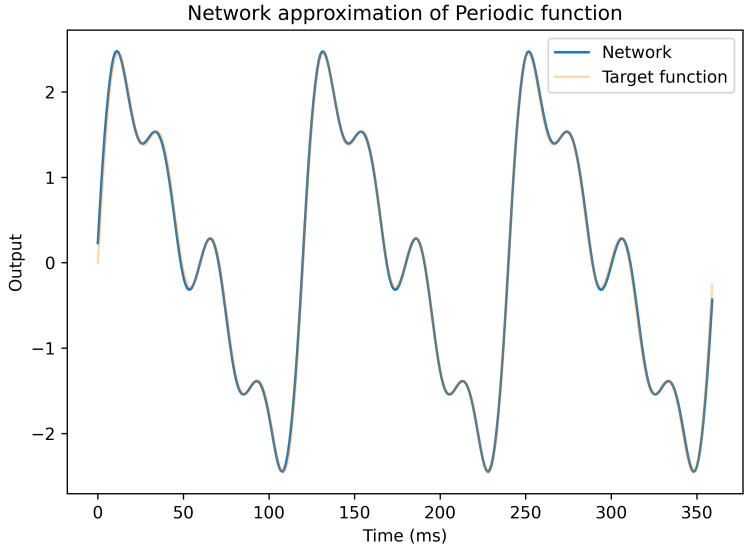

Figure 4: Post-training network activity projected (blue), as well as target function (orange) for the combination of four sinusoids. The network activity after training is stable, and able to continue to approximate the target function.

As we can see from Fig 4, for this type of target function the network can continue to approximate it even after training has ended.

The same behavior can be seen in the case of a compicated 16 sinosoids, defined as:

$$f(t) = \frac{1}{1.5}\left(\sum_{n=1}^{16} \frac{A}{k_n}\sin(n\pi f_0 t)\right) + \frac{1}{1.5}\left(\sum_{n=1}^{16} \frac{A}{k_n}\sin(n\pi f_0 t)\right) \tag{13}$$

where $f_0 = \frac{1}{60}$ is the fundamental frequency, A is the amplitude, and k_n is the nth harmonic in the series $k = 1, 2, 6, 3, 1, 2, 6, 3, 1, 2, 6, 3, 1, 2, 6, 3$. Figure 5 below shows the network's performance on this target function.

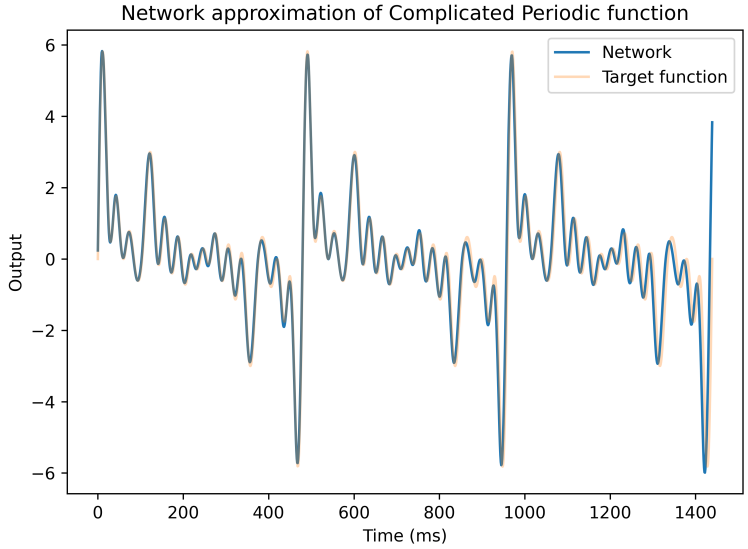

Figure 5: Network performance after training on the complicated 16 sinosoids function. Network dynamics are depicted in blue, while the target function is depicted in orange.

Likewise, we see that the network is able to handle discontinous targets relateively well. For example, Figure 6 below shows the network's performance on a square wave function, defined as:

$$sq(t) = \text{sign}(\sin(2\pi f_0 t)) \tag{14}$$

where $f_0 = \frac{1}{120}$ is the fundamental frequency.

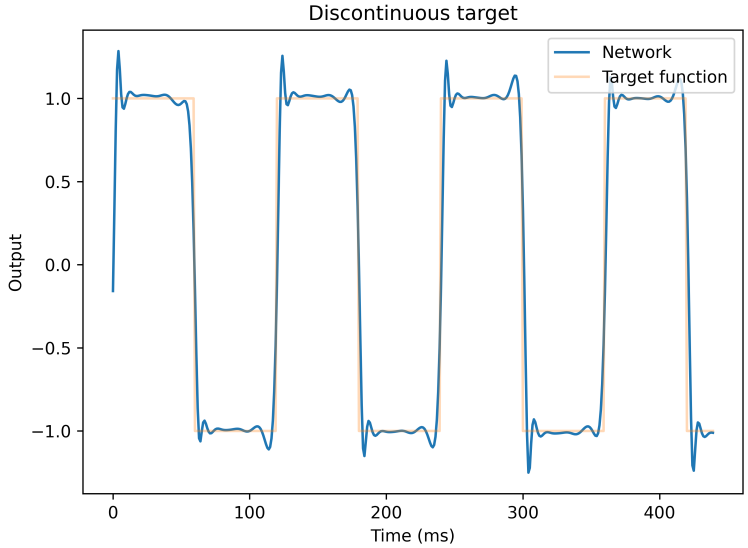

Figure 6: Post-training network activity (blue), as well as target function (orange) for the square wave function.

Morever, the network is also able to handle regular sinosoidal functions with a variety of periods. Figure 7 below shows the network's performance on a sinosoid with a period of 60 time units, as well as 8000 time units.

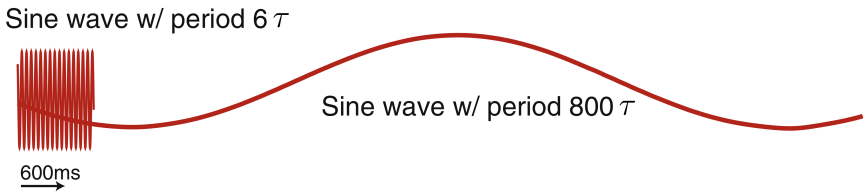

Figure 7: Post-training network activity (red), of a network trained on a sionsoid with a period of 6τ, as well as a period with 800τ. The target function is traced in green, but is not visible as it is covered by the network activity. NOTE: see annex for more infomation on this plot, and why it is copied from the article.

Finally, the network can even reproduce functions that have been corrupted by noise. Take for example the following function:

$$f(t) = \frac{1.3}{1.5}\left(\sin\left(\frac{\pi t}{60}\right) + \frac{1}{2}\sin\left(\frac{2\pi t}{60}\right) + \frac{1}{3}\sin\left(\frac{4\pi t}{60}\right) + \frac{1}{6}\sin\left(\frac{3\pi t}{60}\right)\right) + 0.3\eta(t) \tag{15}$$

where $\eta(t)$ represents Guassian noise with mean 0 and standard devation 1. Figure 8 below shows this function, as well as the networks performance after training.

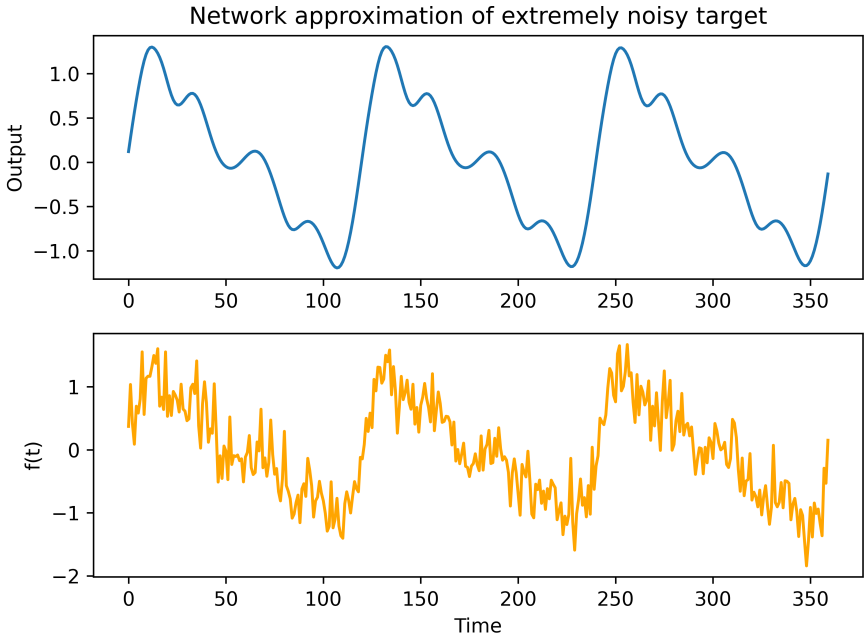

Figure 8: Post-training network activity (blue), as well as target function (orange) for the noisy sinosoid function. In this case, the network manages to approximate a de-noised version of the target function after training.

5 Principle Component Analysis

In the previous seciton we saw the flexibility of the FORCE network's ability to aproximate functions by testing it on different flavours of targets. In this section, we will delve into the workings of the network using PCA. Figure 9 below shows the distribution of the retrieved eigenvalues of the network which generated the output from Figure 4.

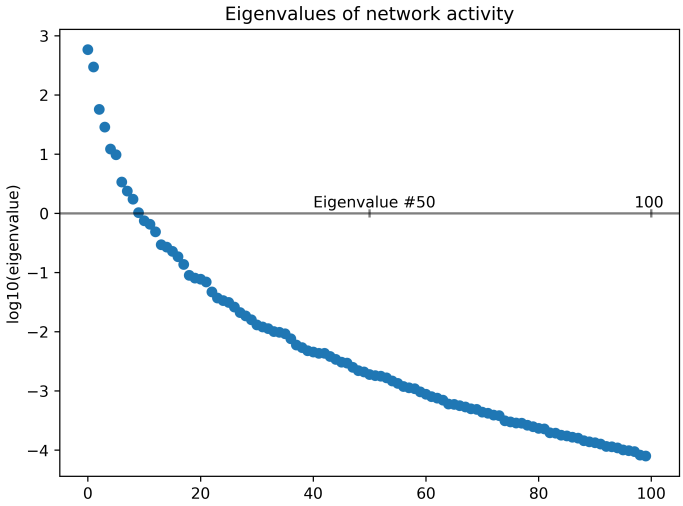

Figure 9: The distribution of the retrieved eigenvalues from the network that generated the output in Figure 3, on a log10 sclale. The first 100 eigenvalues are shown, with the 50th and 100th demarcated on the x-axis.

5.1 Activity Reconstruction

While the network used containes 1000 neurons, from Figure 9 we can see that the trajectory of the network activity is confined to a lower dimensional subspace. The exponential decrease in the eigenvalues indicates that only the first few vectors are needed to approximate the network's activity. Moreover, if we project the network activity onto the PC vectors, we can reconstruct the basis functions that generate the target funcition (Sussillo & Abbott, 2009). Figure 10 below shows the result of this process for the top 8 principle components.

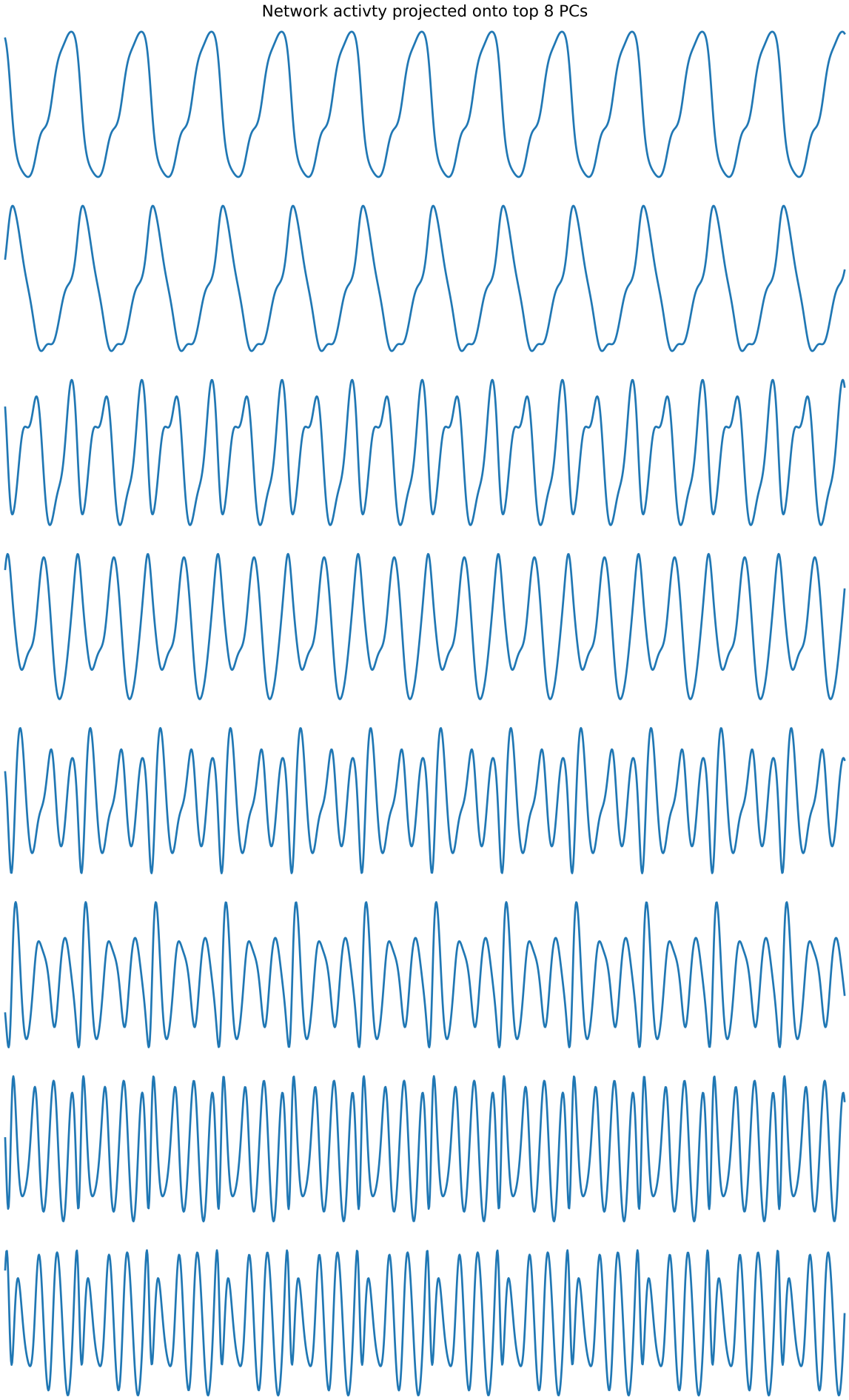

Figure 10: Network activity $\mathbf{r}(t)$ projected on to the top 8 principle components, in descending order top to bottom.

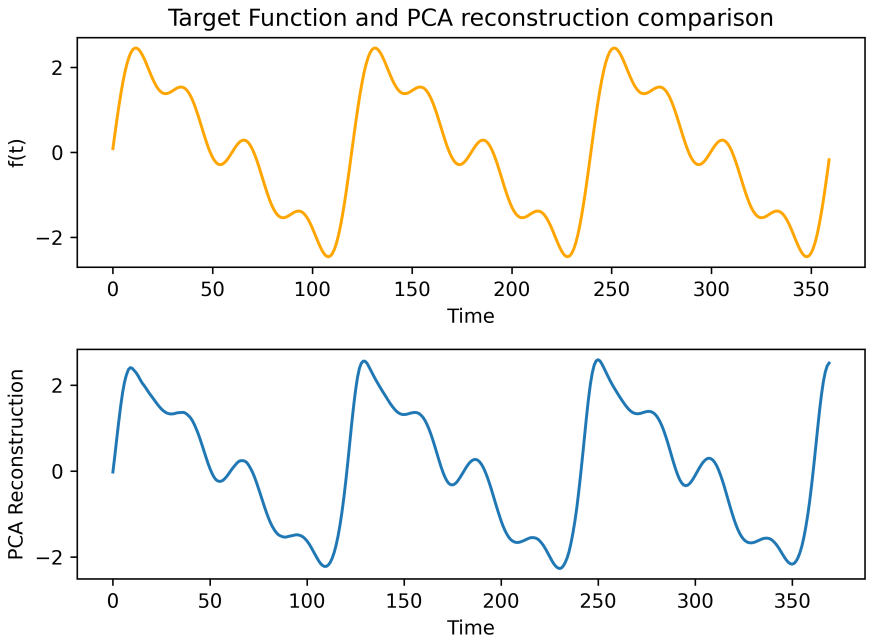

Figure 11: Combination of the projection of the network activity on the top 8 principle components, and the reconstruction of the network activity using only these components. The target function is shown in orange, while the reconstruciton is shown in blue.

From Figure 10, we see that these top 8 components exhibit the periodicity that characterized the neurons during and after the learning seen in Figure 4. In fact, we can reconstruct the network activity using only these first 8 principle components, as demonstrated in Figure 11 above.

Moreover, we can visualize the convergence by the network to a stable state by projecting the readout weight vector $\mathbf{w}$ over training onto the top 2 principle components. Figure 12 demonstrates the convergence onto a stable state by the network over the course of training, over 5 different trials with different initial conditions. Regradless of the initial conditions, the FORCE algorithm pushes the weights to the same static solution.

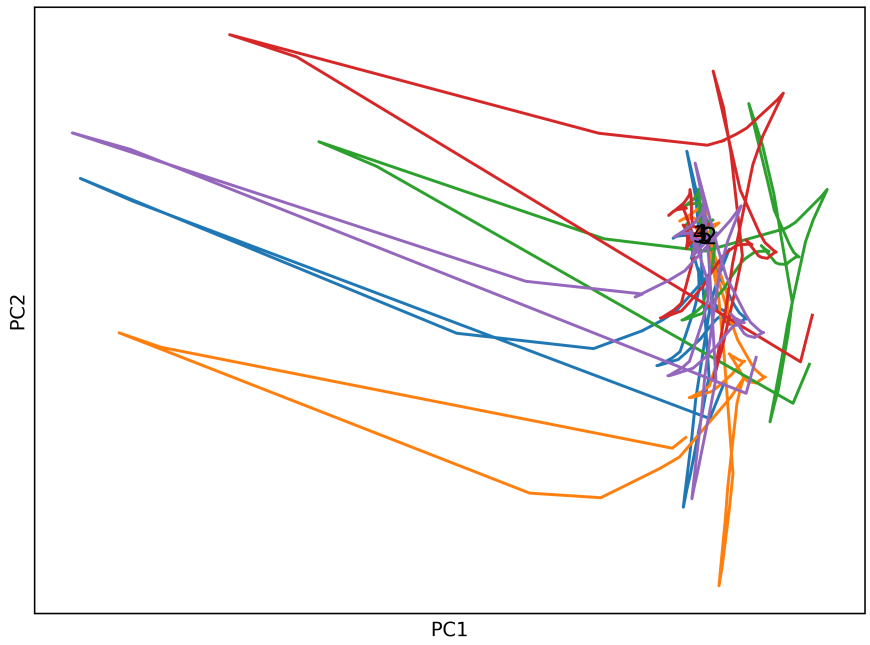

Figure 12: Projection of the readout weight vector throughout training, given 5 different initial conditions. While behavior is initially different, all trajectories eventually converge to the same point, as shown by the overlapping labels, indicating the discovery of a solution

5.2 Learning and Control Phases

Analysis of the principle components also sheds light on the characteristics of different parts of the network during the training process. Specifcally, we can describe two phases in the activity of the components of $\mathbf{w}$: a learning phase and a control phase. Recall that $\mathbf{P}$ corresponds to the RLS learning rates (Sussillo & Abbott, 2009). Morevover, projecting $\mathbf{w}$ and $\mathbf{r}$ onto the principle component vectors of $\mathbf{P}$ changes the basis of $\mathbf{P}$ to be diagonal. In this basis, the gain vector for the ath component can be expressed as:

$$k_a(t) = \frac{1/\lambda_a \cdot r'_a(t)}{\lambda + \sum i \frac{(r'_i(t))^2}{\lambda_i}} \tag{16}$$

where $r'_a(t)$ is the ath component of $\mathbf{r}$ in the basis of $\mathbf{P}$, and λ_i are the eigenvalues of $\mathbf{P}$. Assuming that the network has succesfully learned the target function, and the $\mathbf{P}$ has converged to the inverse correlation matrix, the contributions from $r'_i(t)$ should average out, and the gain vector becomes:

$$k_a(t) = \frac{1/\lambda_a}{\lambda + 1/\lambda_a} \tag{17}$$

$$k_a(t) = \frac{1}{\lambda\lambda_a + 1} \qquad \text{simplifying} \tag{18}$$

Since this process takes M updates, and k_a converges to a constant value in the steady state, the total learning rate is given by:

$$\text{Learning Rate} \approx M \cdot k_a = M \cdot \frac{1}{\lambda\lambda_a + 1} \tag{19}$$

Recalling that $\lambda = 1$, and we included a regularization parameter α, we have that the total learning rate is given by:

$$\text{Learning Rate} = \frac{1}{M\lambda_a + \alpha} \tag{20}$$

where λ_a is the ath eigenvalue of $\mathbf{P}$, and α is the regularization parameter.

Thus, we have the following two phases for each eigenvalue during RLS: a phase in which $M > \alpha/\lambda_a$, and a phase in which $M < \alpha/\lambda_a$ (Sussillo & Abbott, 2009). In the former, the task of the weight modification is to find the optimal weights, while in the latter, the task is to bring the output close to the target function (Sussillo & Abbott, 2009). Note that, larger components will cross this threshold faster, and thus begin learning earlier. Thus, most of the components will be devoted to the control phase, allowing the network to learn weights projected in dimensions with large eigenvalues (Sussillo & Abbott, 2009).

6 Biologically Plausible?

There remains, however, the question of biological plausibility. Sussillo and Abbott remark that this type of network can considered a model of "training-induced modification" of neural activity (2009). Moreover, it does so without requiring any changes to the structure of the network, other than diffrences in synaptic weighting (Sussillo & Abbott, 2009). In this regard, it makes progress in approximating the state of the brain before learning to after learning. While it succeeds in this dimension, it does so with the help of certain non-biological features. For example, it still relies on an error mechanism to update the weights, for which a biological implementation is unclear (Sussillo & Abbott, 2009). Futhermore, the output of the network is fed directly back into the network. In biological systems, this process is generally more dispersed, coming from diverse sources within the brain.

The training method poses certain challenges as well. Firstly, as previously remarked, FORCE learning requires changes to happen quickly, the plasticity that would be required of the synapses challenge experimental results (Sussillo & Abbott, 2009). Finally, as noted by Nicola and Clopath (2017), the $\mathbf{P}$ matrix used in the RLS algorithm is not biologically plausible since it provides non-local inforamtion to the network . Thus, while an interesting proof of concept, this particular verison of a FORCE learning network does not quite meet the criteria for biological plausibility.

7 Conclusion

In this report we have explored the FORCE learning network developed by Sussillo and Abbott (2009). We have seen how this network harnesses chaotic spontaneous activity to approximate target functions, and how it

can be used to learn a variety of functions. We have also seen how the network can be analyzed using PCA, and how the network activity can be reconstructed using the top principle components. Finally, we have discussed the biological plausibility of the network, and found that while it is a promising model, it contains certain features that precludes it from being considered biologically plausible. While this type of network represents an advancement over previous attempts at RNNs, it unfortunately still falls short of being an appropriate model of neural dynamics.

8 References

Nicola, W., & Clopath, C. (2017). Supervised learning in spiking neural networks with FORCE training. Nature Communications, 8(1), 2208. https://doi.org/10.1038/s41467-017-01827-3

Sussillo, D., & Abbott, L. F. (2009). Generating Coherent Patterns of Activity from Chaotic Neural Networks. Neuron, 63(4), 544–557. https://doi.org/10.1016/j.neuron.2009.07.018

9 Annex

Unfortunately, I could not get the network to continue to produce very long period sinosoid for Figure 7. This might because I did not properly define this function for the network to learn, since it was succesful in its other attempts. Interestingly, however, my network does in fact manage to approximate this function during the training period - it fails only once training has stopped. Below, you may find the figure generated by my network during the training and testing phases, compared to the target functions.

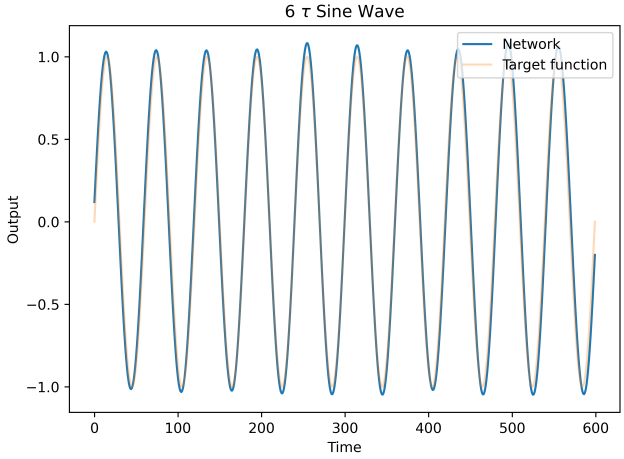

Figure 13: Post-training approximation of the 6τ sinosoid function. The network activity is in blue, while the target function is in orange.

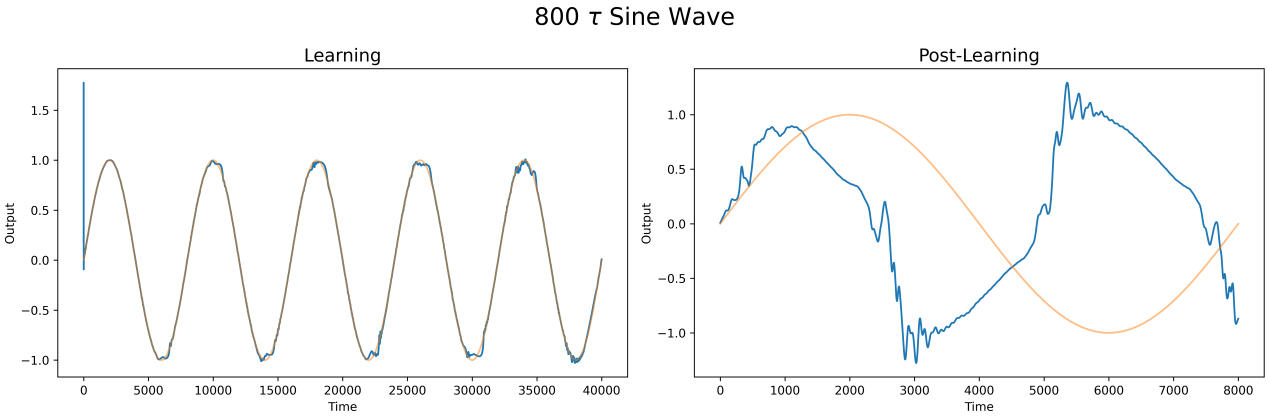

Figure 14: Training and Post-Training approximation of the 800τ sinosoid function. The network activity is in blue, while the target function is in orange.