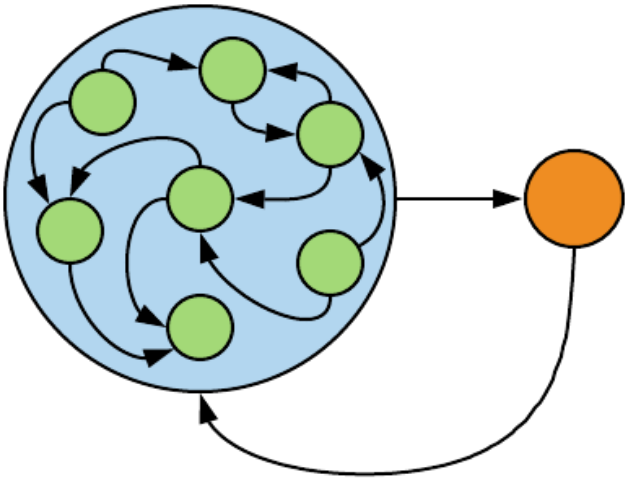


University of Colorado at Boulder

APPM 4370

Computational Neuroscience

Fundamental FORCEs: An Analysis of the FORCE Training Regime

Authors:

Cooper Simpson

Cole Sturza

Zach Berriman-Rozen

April 29, 2020

Contents

1 Disclaimer:

This paper is a partial recreation of David Sussillo and L.F Abbott's work in [SL09]. Although all work was done by the authors it should be noted that most of the ideas and inspiration came from said source. Sussillo and Abbott have been cited (and others as necessary) where applicable throughout the paper.

2 Abstract:

We recreate and extend portions of the work accomplished in *Generating Coherent Patterns of Activity from Chaotic Neural Networks* by Sussillo and Abbott – which details a training regime, FORCE, for highly chaotic recurrent neural networks (RNNs). We began by discussing the FORCE learning process both from a high level and technical point of view. We examined the use of FORCE learning, why it is effective, and the variations of it that may be employed. Then, we moved on to reproducing some of the networks that generate a variety of relatively simple learned functions and showed that the same network structure is capable of learning a wide variety of functions quickly and accurately. The range of functions a network is capable of learning were investigated and the learning process employed by the chaotic network was then further examined. Having successfully trained a network to generate some basic functions we then recreated the principal component analysis (PCA) accomplished in Sussillo and Abbott's paper. Through this analysis we will investigate the network activity of our trained models – helping to identify the different functional aspects of these models. Our findings support Sussillo and Abbott's claim that only a small number of the principal components are needed to produce the desired output, but that the remaining components are necessary for the stability of the network. Finally, we expanded on Sussillo and Abbott's paper by further examining the principle components of a multi-readout network as well as investigating a FORCE trained RNN's reliance on different network parameters.

3 Introduction:

Sussillo and Abbott begin their work by noting that neurological structures exhibit chaotic spontaneous activity, but are equally capable of complex output patterns. They ask the question of how these seemingly disparate outputs are related? In order to answer this question the authors sought to study these processes in an artificial neural circuit setting (i.e. a neural network). By building a network that models the brain and exhibits random chaotic activity one can attempt to reorganize this activity into distinct learned outputs – investigating whether this is a valid biological process. This in turn further motivates Sussillo and Abbott's work as there is a distinct lack of effective algorithms for training the highly chaotic recurrent networks that are inspired by neurobiological

systems. Whether from a machine learning or neurological point of view, an efficient method for training these types of networks is advantageous.

Thus, Sussillo and Abbott introduce FORCE learning, a training regime for highly recurrent chaotic neural networks. Three different architectures are considered for the FORCE procedure: a network with a feedback pathway where only the output weights are trained, a network with a supplementary generator network, and a standard recurrent neural network where all weights are trained. For a more in depth discussion of the FORCE learning procedure and the model architectures please see section 4.

Initially considering a feedback architecture, Sussillo and Abbott display the extensive capabilities of such a network trained with FORCE by showcasing the large variety of learn-able target functions. This includes periodic functions, noisy functions, and aperiodic targets. At this point one important aspect of FORCE to note is that in order for training to succeed the network must transition out of chaos during learning. This is achieved by sending a strong enough signal through the feedback loop which requires the feedback weights to be large and for the learned function to be of sufficient amplitude. This is not an issue when considering a network architecture other than that with a feedback loop.

Still considering this feedback network, Principal Component Analysis (PCA) is used to analyze a model trained on a periodic function. Through PCA, the authors show that the activity of the network is relatively low dimensional – especially when compared to the size of the network – and only a limited number of the principal components (PCs) are needed to reproduce the trained output. However, we see that not only does the network need to produce the correct output, but it also needs to be stable. We see that a larger subset than that needed to approximate the output converge to unique values (regardless of initial conditions) indicating that those extra PC's are needed for stability.

A fundamental aspect of FORCE learning is that the feedback signal is not fixed to the target function, but is output of the network. Earlier researchers had fixed this feedback to the target function with zero error [JH04], which can lead to stability issues and error in prediction. Instead, Sussillo and Abbott show how it is beneficial to feedback the unchanged network output by looking at variable compositions of the network output and the target function. It is clearly shown that as the feedback moves towards the target function the network is less likely to stabilize and increases in its mean average error in prediction. We see that FORCE is effective in part because it allows the network to sample state fluctuations during learning.

The authors next give us a distinct reason why chaotic networks are effective apart from their neurobiological realism. By changing the initial level of chaos inside the network we can see a sizable advantageous effect on both training and prediction: increasing the chaos decreases the learning time, decreases the prediction error, and decreases the magnitude of the internal weights. These first two points have clear positive impacts, and the third point indicates that the network

is more stable. This follows because larger magnitude weights indicate that there are both large positive and large negative values which causes the network to be more unstable. The benefits of chaos, although promising, have a limit. Once the chaos becomes too large, the feedback loop cannot suppress it during learning and the network fails to converge to a solution. This is a point stressed by the authors previously, and their conclusion is that the initial chaotic network must be in some way suppressed during learning in order to succeed.

Up to this point Sussillo and Abbott have mostly considered generative networks and periodic functions. A generative network is one that infinitely produces an output with no external input. The authors point out that this is actually a more difficult task, but a neural network learning algorithm would not be complete without considering input-output mapping. From a biological point of view these sort of neural interactions are very important. The authors show the ability of FORCE to train such a network by considering a few functional memory examples. Different inputs are used to select from a number of learned outputs for which the network is able to reproduce. Finally, the authors consider training a network to reproduce human motor output. This is accomplished with the use of motion capture data, and multiple readouts that act as joint angles spread across the body. Furthermore, the network can be trained to produce multiple motions (i.e running and walking) using variable inputs to switch between the two. We see that such a network is capable of simulating these movements very realistically.

Sussillo and Abbott conclude their work by noting that FORCE learning is a big step towards biologically plausible networks and network learning, but that it still has its own shortcomings. Notably FORCE is capable of learning in random and chaotic environments much like the brain, and has a wide variation in the types of functions it can learn. Furthermore, FORCE acts quickly and maintains a small error even while learning which is far more biologically realistic (especially in motor tasks) than the traditional method of reducing a large error slowly over time. FORCE also only requires the altering of weights (i.e synaptic strengths) in order to learn as opposed to changing any fundamental structure. This again is biologically realistic as synaptic connections are the main area of modification in biological circuits. However, FORCE still relies on computing an error and having this inform modifications which does not yet have an understood biological analogue. In other words this error computation is a stand in for processes we do not fully understand and therefore may be inaccurate. As well, like other neural networks it is difficult to see what individual pieces or processes directly result in the output. The connections in the network are random and there is not necessarily a one-to-one mapping for what happens in the network and what that means for the outputs. Lastly, it is important to note that FORCE learning can be implemented a number of ways and there may be more effective or biologically realistic methods for doing so.

4 FORCE Learning:

What exactly is the FORCE learning regime? In short it is an approach to effectively training chaotic and highly recurrent neural networks. FORCE learning is a training paradigm; there are various methods and algorithms that can be used to implement it, but they all follow the same overarching theme. For our discussion we will restrict ourselves to one specific implementation and network architecture (outlined later), although we will indicate were deviations could occur. We will discuss in depth what FORCE learning is at a high level, how it is implemented mathematically, and the variations of it given different architectures.

4.1 High Level:

When FORCE was first conceived in [SL09] existing neural network training regimes had many problems when it came to chaotic RNN's like those we consider. Those existing regimes were either intended for feed forward networks (no recurrent connections), or were intended for recurrent networks but failed to converge when presented with chaotic activity. While there were regimes capable of training a type of network that we are considering, they were usually slower, computationally expensive, and unpredictable. Since then, more progress has been made, and much of this is attributable to the initial work on FORCE [DeP+18]. There are three major issues that need to be considered when developing a training procedure for RNN's. The first comes from how the output is fed back into the network. Removing all errors in this signal can cause problems for the network by failing to accumulate a generalized sample of fluctuations; however, too much error can cause stability issues for the network and prevent it from converging. The second problem arises from the fact that the RNN is highly chaotic, which can cause problems during training where the network is unable to learn due to too much noise. Both of these problems are solved simultaneously by FORCE in the way it modifies network weights. The third problem is that of assigning proper credit for error in the output to specific weights in the network. This is solved for us by considering the architecture shown below in figure 1.

FORCE stands for first-order reduced and controlled error – this gets at the fundamental trait of FORCE. During learning output errors are fed back into the network, but to avoid the issues described above the errors are kept small by making weight modifications efficiently (i.e fast and effective). This allows the network to sample fluctuations during learning making it more robust, and prevents the chaotic activity from disrupting learning.

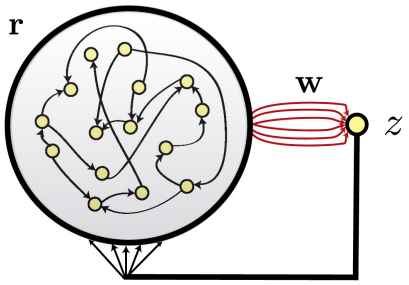


Figure 1: Network architecture (Edited from Sussillo and Abbott)

The figure above shows the network architecture we will consider and use throughout this paper. We see that the internal neural population $\mathbf{r}$ is recurrently connected, and that it is connected to the readout z by the weights $\mathbf{w}$. The output from the readout is then fed back into the network with connections to all internal neurons in $\mathbf{r}$. The readout weights are highlighted in red to indicate that these are the only weights that will be modified during learning. Given the architecture in figure 1 the main equation we are concerned with is as follows:

$$z(t) = \mathbf{W}^T \mathbf{r}(t) \tag{1}$$

In the above equation we see the output at time t ($z(t)$) is given by a weighted ($\mathbf{W}$) sum of the activity of the neurons connected to the output ($\mathbf{r}(t)$). This is a standard process for neural networks, but is important in our discussion as only this weight vector $\mathbf{W}$ will be modified during training. It is also important to consider the case where multiple readouts are used. This corresponds to $z(t)$ becoming a vector $\mathbf{z}(t)$ and $\mathbf{W}$ now representing a matrix. In other words there are multiple sets of weights connecting our output neurons to the various actual outputs. As well, each one of these readouts would have its own feedback loop into the network.

When instantiating a network to train, all network weights are chosen randomly including those of the feedback loop. However, the feedback loop weights are of a magnitude larger than those of the network itself. This is done so as to make the effects of the feedback loop impactful. As partially discussed earlier when considering figure 1 we will only modify $\mathbf{W}$. The task of FORCE learning (as it is with any network) is that given a defined goal function $f(t)$ (or $\mathbf{f}(t)$ in the multiple readout case) we want to set $z(t) = f(t)$. As learning proceeds the efficient weight modifications will keep the error small, so the goal of FORCE is to reach a solution where the weights are unchanging. Thus, the implementation of FORCE must be done in a way that quickly alters the output weights to keep the error small and continues to reduce it while learning.

4.2 Algorithm:

We will discuss the technical and mathematical details around FORCE learning, but before doing so it is important to note that we will only be considering a generative network. We are assuming that we are training a network to reproduce some specified pattern, as opposed to operating on some kind of input. This does not change the discussion too much, and this pattern generation task is actually harder as noted in section 3.

We begin by assuming that our network has been established – defining the size, and randomly assigning connections and weights. For a network of size N neurons the internal connections are chosen from a Gaussian distribution with $\mu = 0$ and $\sigma^2 = \frac{1}{pN}$ where p determines the sparsity of the connections. This p parameter is essentially a probability of being a nonzero connection. The readout weights are chosen from a Gaussian distribution with $\mu = 0$ and $\sigma^2 = \frac{1}{N}$, while the feedback weights are distributed uniformly from $[-1, 1]$. As well, r is considered to be the post-activation output of the neurons (i.e $r = h(x)$) where in our case $h(x) = tanh(x)$, but it could be any desired activation function. So in order for the network to begin operating it needs some initial condition for x which we take to be a Gaussian with $\mu = 0$ and $\sigma^2 = \frac{1}{\sqrt{2}}$.

Every Δt time interval we will update our weights, so starting at some arbitrary time t our base network output is given as $z(t) = \mathbf{W}^T(t - \Delta t)\mathbf{r}(t)$. Our error before and after modification are then given as follows:

$$e_-(t) = z(t) - f(t) = \mathbf{W}^T(t - \Delta t)\mathbf{r}(t) - f(t) \tag{2}$$

$$e_+(t) = z(t) - f(t) = \mathbf{W}^T(t)\mathbf{r}(t) - f(t) \tag{3}$$

Where the e_- represents the error before modification, and e_+ the error post modification. We also note that this assumes that weight updates are happening quickly which is why f and z are seen evaluated at t regardless of the weights. In the FORCE regime we want to make $|e_+|$ small immediately after the first update and keep $|e_-|$ small while slowly taking $\frac{|e_+|}{|e_-|} \to 1$ over the course of the training period.

In order to accomplish this goal we consider the Recursive Least Squares (RLS) algorithm for making this fast and effective weight modifications. Note that this is the specific implementation we are considering although others exist [SL09]. With this we take $\mathbf{P}(t)$ which is an NxN matrix (where N is the number of neurons in the network). $\mathbf{P}(t)$ is initially given as:

$$\mathbf{P}(0) = \frac{\mathbf{I}}{\alpha} \tag{4}$$

and is updated at the same time that the weights are updated according to the following equation:

$$\mathbf{P}(t) = \mathbf{P}(t - \Delta t) - \frac{\mathbf{P}(t - \Delta t)\mathbf{r}(t)\mathbf{r}^T(t)\mathbf{P}(t - \Delta t)}{1 + \mathbf{r}^T(t)\mathbf{P}(t - \Delta t)\mathbf{r}(t)} \tag{5}$$

And the weights are updated according to:

$$\mathbf{w}(t) = \mathbf{w}(t - \Delta t) - e_-(t)\mathbf{P}(t)\mathbf{r}(t) \tag{6}$$

The P matrix is essentially a matrix of learning rates and can also be viewed as an estimate of the inverse correlation matrix of the output r. The fact that learning rates are not the same for all neurons increases the versatility of FORCE and helps learning. The α parameter initially acts as the inverse global learning rate, so small values will speed up training and large values will slow it down. As training progresses the learning rates for individual connections deviate away from this. We will not prove that RLS works as a learning algorithm although this is done in [SL09]. We will however note that the equation governing the forward in time progress of the network is given as follows:

$$\tau \frac{dx_i}{dt} = -x_i + g \sum_{j=1}^{N} W_{ij} r_j + B_i z \tag{7}$$

In the above equation τ is the basic time constant of the network, x_i is the pre-activation output of neuron i, W are the internal weight connections, and B_i are the feedback weights. This equation governs how the network activity progresses in time and is used to simulate said network.

4.3 Differences In Network Architecture

While we will only focus on a network architecture of the form in figure 1, it is not the only sort of network that FORCE can be used on. It is useful to be able to train such a wide array of networks both from a neurobiological standpoint and from a general neural network view. Two other network architectures are given below:

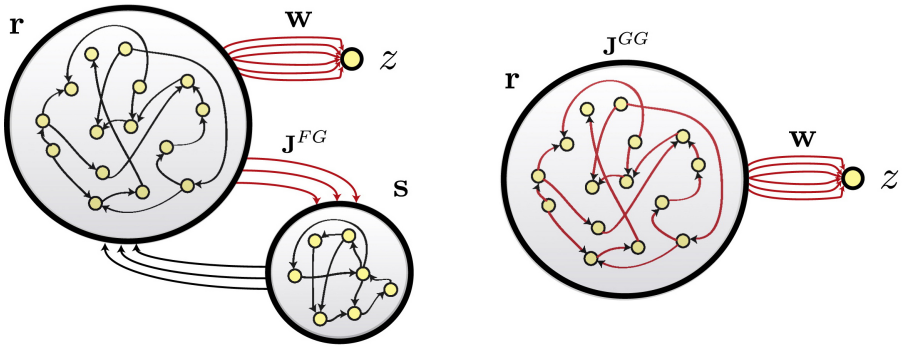


Figure 2: Alternate network architectures (edited from Sussillo and Abbott)

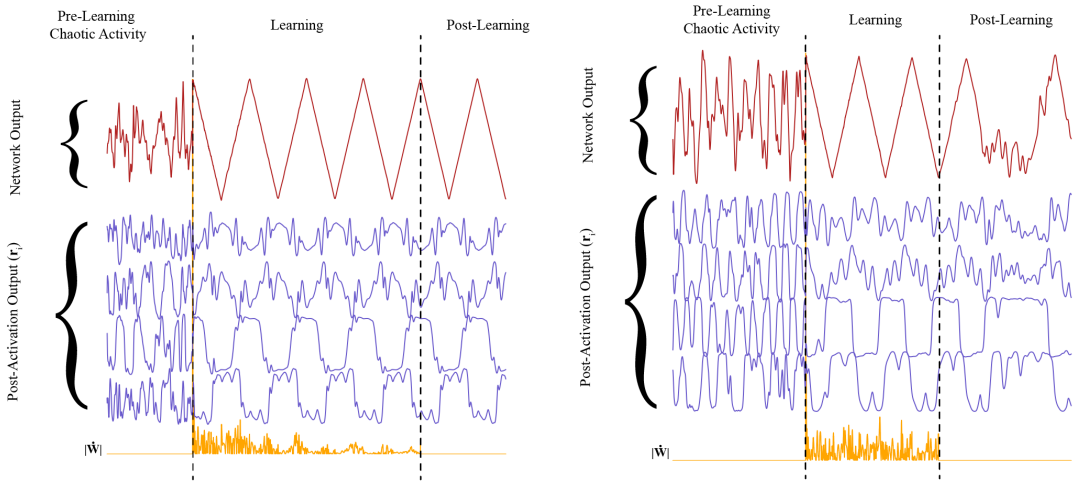
On the left we see what is called the generator network **r** and the feedback network **s**. Like before, the generator network is connected to a readout z which is the output of the network. Instead of a feedback loop, the feedback network is used to regulate the chaos inside the generator. Now two sets of weights are trained: W connecting the network to the readout, and J connecting the generator to the feedback network (in that direction). On the right we see what may be thought of as the most general RNN, a network with only internal connections and an output

readout. In this case all of the weights are modified during learning eliminating the need for any sort of feedback.

Both of these network structures are more biologically realistic than that in figure 1. Having a generator network is analogous to neural populations interacting with each other (regulating in this case). As well, in a real biological setting it is not just a subset of synapses that are modified during learning as every connection can and most likely will be altered. These two networks will have an evolution equation that describes the network activity in time that is similar to equation (7) and can be seen in [SL09].

5 Simple Examples:

In the figure that follows we will see various examples of the FORCE learning process and the functions that a RNN can produce when trained with FORCE. Note that in most of these examples any sort of time scale has been left out. This is because the time aspect is somewhat arbitrary and can be impacted both by the timescale of the target function, and by the timescale of the network (i.e. τ). For reference we set $\tau = 1$ and the target functions are on the scale of 100τ. The network used for these examples consists of $N = 1000$, $g = 1.5$, and $\alpha = 1$.



(a) Success in learning

(b) Failure in learning

Figure 3: Learning process for triangle wave function.

In figure 3 above we see two plots showing the learning process of a network as it attempts to learn a triangle wave function – both plots contain a number of attributes. The red trace indicates the output of the network (z), while the blue traces indicate the post-activation output of individual neurons ($\mathbf{r}_i$). Lastly, the orange trace indicates the magnitude of the time derivative of the weights. We can also see that each plot is divided into sections that indicate pre-learning

spontaneous activity, learning, and post-learning prediction. Initially, the network exhibits chaotic activity, but once learning starts the network begins to produce the target function. This is showing us how FORCE always keeps the error small. In the left plot we see that as learning progresses the weight change gets smaller and smaller. This is what is supposed to happen in FORCE and we see that after learning the network will continue to produce the target function. However, in the right plot we see that training does not proceed for long enough and the weights are never able to stabilize. Thus, the network falls back into its chaotic state after learning is finished.

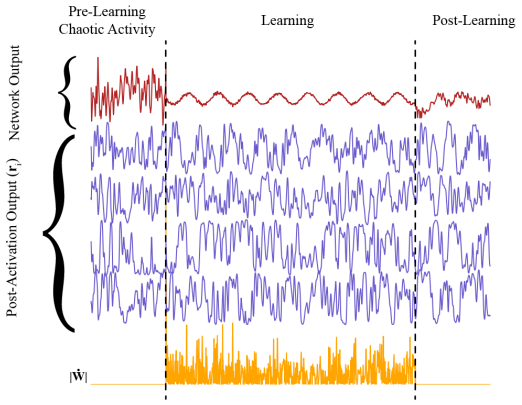


Figure 4: Network fails to learn low amplitude sine wave.

In figure 4 above we see a similar plot as that for figure 3, but with a different target function. We see that the network is trying to learn a low amplitude sine wave, but that it fails to do so. This is caused by the fact that the sine wave has such a low amplitude – $\frac{1}{10}$ versus 1 for the triangle wave. When the error signal gets sent back along the feedback pathway it is too small to effectively suppress the chaos in the network. An easy fix for this is to simply offset the function by some amount, but it gets at a previously discussed aspect of FORCE. In order to effectively train the network the chaos must be suppressed during learning.

In figure 5 we see a number of relatively simple examples of network output for a particular target function. We see that the network is capable of learning all sorts of periodic functions, and we should note that FORCE can also train a network to produce aperiodic functions based of inputs. We do not delve into this however, but more can be seen in [SL09]. In figure 5 we see complex, discontinuous, and noisy periodic functions. In all cases the network is capable of easily learning and producing the target function. In figure 5D we see two sin waves with period 6τ and 500τ. This serves to indicate the dynamic range of a network trained with FORCE. It should be noted that for these functions on the extremes of a networks capabilities learning takes longer.

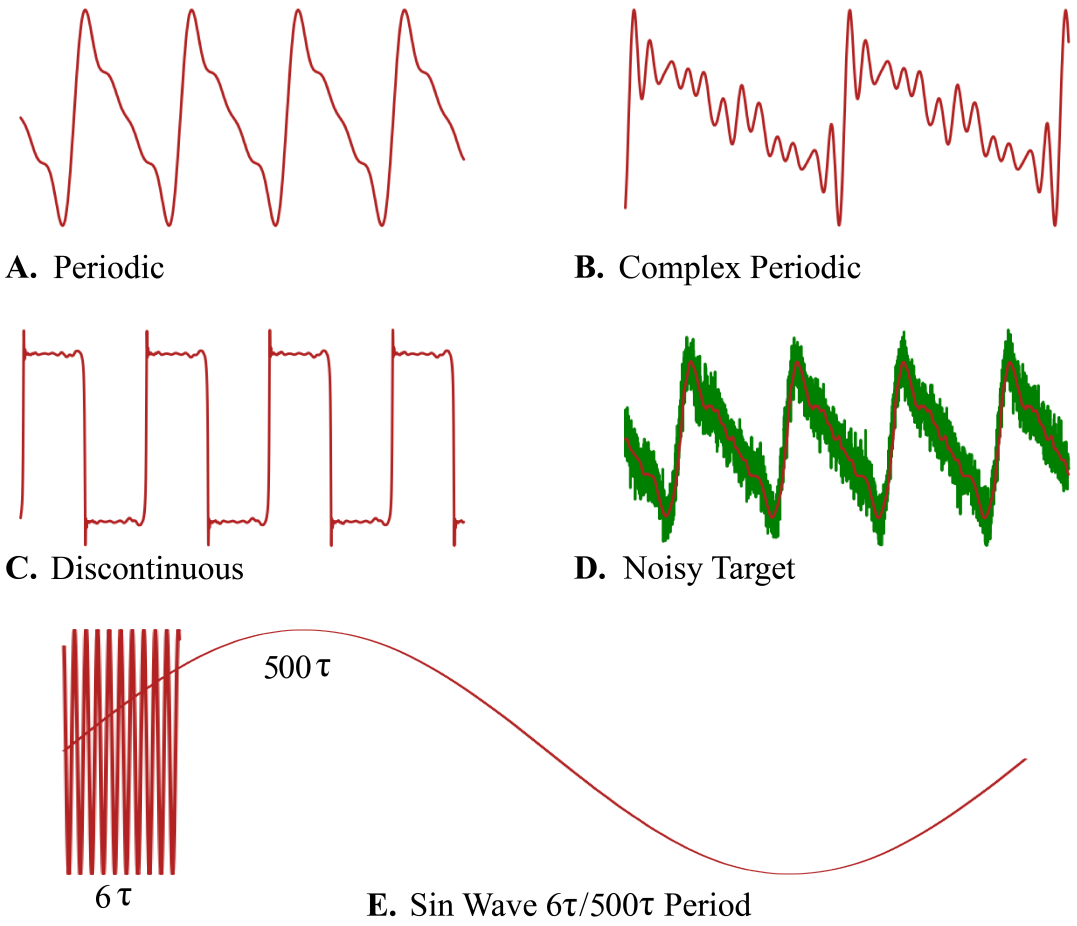


Figure 5: Simple examples of functions a network can produce

6 Principal Component Analysis (PCA):

We will now analyze the network activity using Principle Component Analysis. We will first consider the network trained on the periodic function shown in figure 6 below. The function is a sum of four sinusoids. Training was performed as usual, but during the prediction phase the 8 leading principle components are used to produce the output. The network used for these examples consists of $N = 1000$, $g = 1.5$, and $\alpha = 1$.

After training the network to produce a target function, one can apply PCA to project the network activity onto a few of the principal components. This is simply done by finding the eigenvectors of the leading principle components and projecting the output weights onto a matrix of the associated eigenvectors. Then, one needs to multiply the transformed time series matrix, which is found by multiplying a matrix composed of the vector $\mathbf{r}(t)$ at every time step with the matrix of the leading principle components' eigenvectors, with the projected output weights. This is what has been done in figure 6.

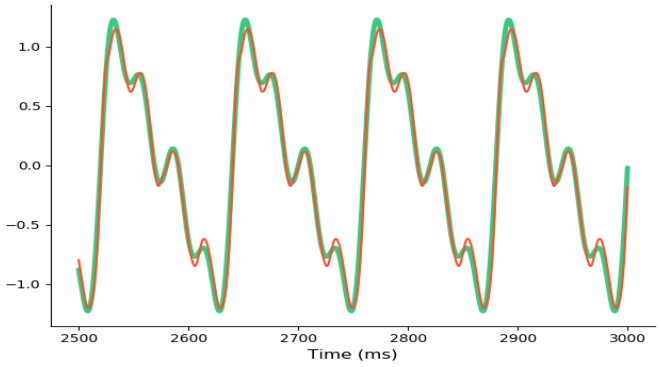


Figure 6: Output after training a network to produce a sum of four sinusoids (green), and the approximation (red) obtained using activity projected onto the 8 leading principal components.

We can see that even with a only a small number of principal components the network is able to accurately produce the target output. This is further validated by the distribution of eigenvalues in figure 7. It indicates most of the network activity can be accounted for by a few leading principal components.

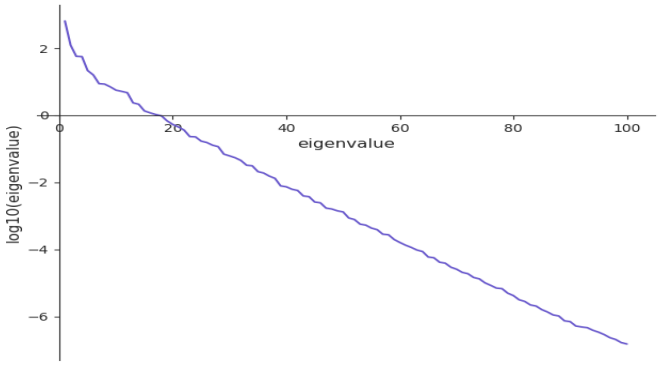


Figure 7: PCA eigenvalues for the network activity that generated the waveform in figure 6. Only the largest 100 of 1000 eigenvalues are shown.

One can also plot the network activity, or the time series matrix, onto the leading principle components. Below is the network activity projected onto the eight leading principle components. As one can see the first principle component's amplitude is the largest and then the rest of the amplitudes begin to decrease. This is what we expect to see because the first principle component contributes the most to explaining the systems variance. This again confirms our intuition earlier that the first few principle components are contributing the most to the networks activity.

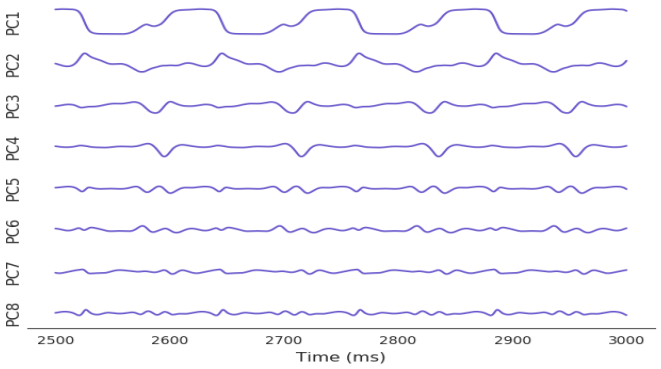


Figure 8: Projections of network activity onto the leading eight PC vectors.

We can extend this idea of principle component analysis to multiple outputs and see if networks that require multiple outputs are more complex and if more of the principle components contribute to the networks activity. The process is very much the same as before. Below are two examples of networks that have been trained to produce 2 outputs and 3 outputs. If we plot the first hundred of the leading principle components' eigenvalues one can see that the graphs are very similar to the network with only one output. This tells us that if we increase the outputs of the network most of the network activity can still be accounted for by a few leading principal components.

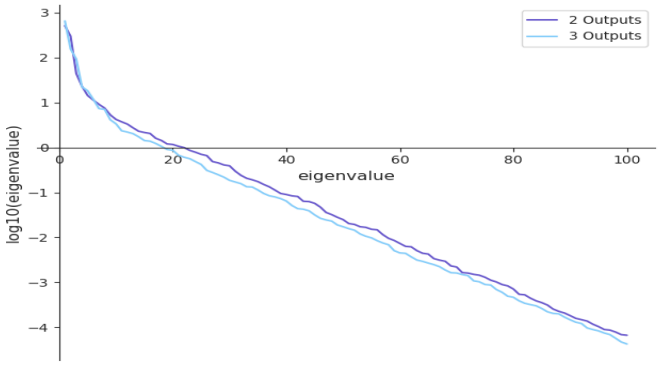
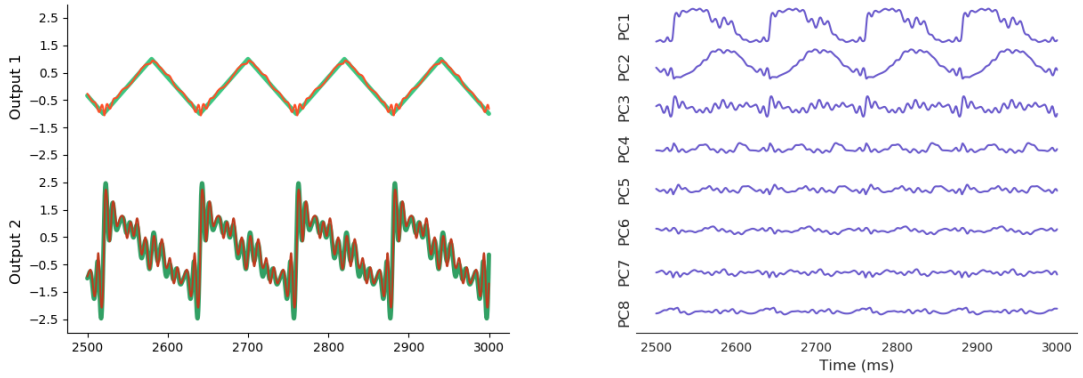


Figure 9: PCA eigenvalues for the network activity that generated the waveform in figures 10 and 11. Only the largest 100 of 1000 eigenvalues are shown.

We can again project the activity of the network onto the eight leading principle components to obtain the target functions. The networks both produce a range of target functions. The 2 output network produces a complex periodic function and a triangle wave function. The 3 output network produces both of those functions plus a discontinuous function as its third output. It appears that most of the target patterns can be obtained from these few projections onto the leading principle components. Out of all the degrees of freedom available (in this case a thousand neuron network) only about eight are actually necessary to produce the target functions.



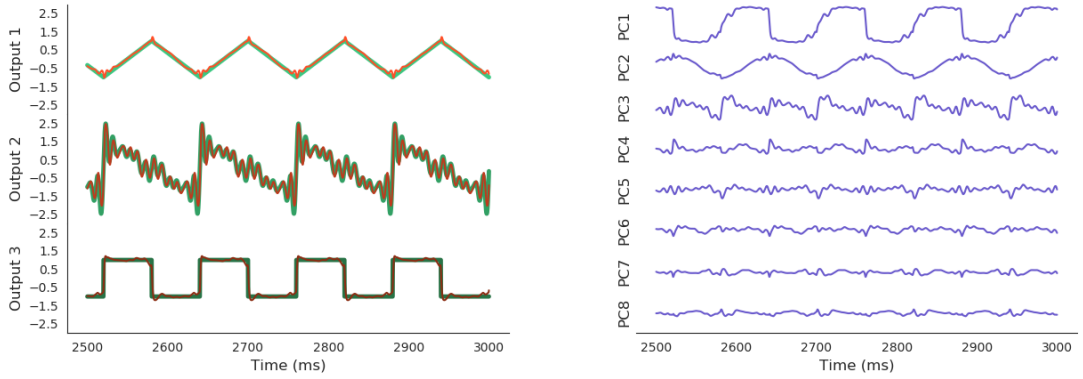
(a) Network output

(b) Projected output

Figure 10:

Left: Network outputs (green), and the approximations (red).

Right: Projections of network activity onto the leading eight PC vectors.



(a) Network output

(b) Projected output

Figure 11:

Left: Network outputs (green), and the approximations (red).

Right: Projections of network activity onto the leading eight PC vectors.

Again, if one looks at the projections of network activity onto each of the leading eight principle components the amplitude of the first component is much larger than the rest, and the amplitudes begin to decrease after the first. This is good because it tells us as the number of outputs increases the complexity of the network activity remains the same.

7 Network Parameters

We have seen in section 4.2 the various parameters that describe the network, and for the examples we have seen up to now they have stayed relatively constant. For example the network size has been fixed at $N = 1000$, the learning rate at $\alpha = 1$, the chaos at $g = 1.5$, and the activation function for the output neurons has been the hyperbolic tangent. However, one may ask why these values are what they are, and what changing these values will do to the network. As well, from a biological perspective we know that any sort of artificial network is only approximating reality, so what do differences in these parameters mean biologically? We will examine some of these parameters and analyze their traits.

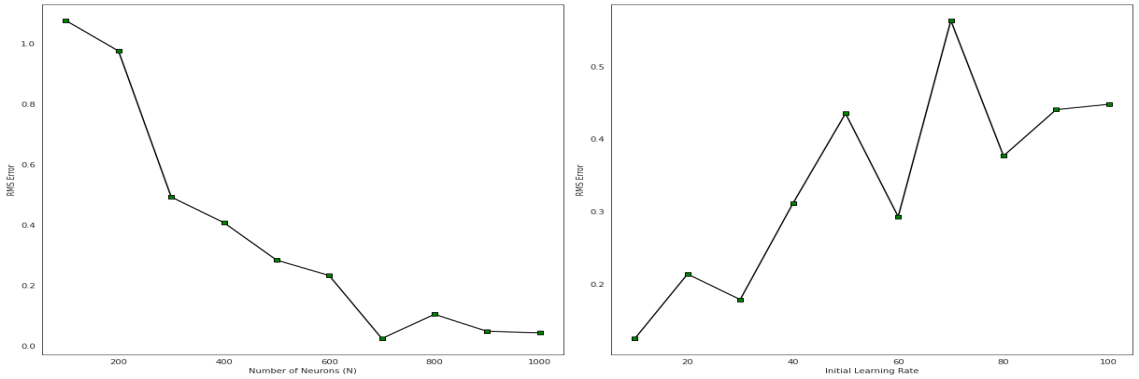
We first consider the hyperbolic tangent that is the activation for the output neurons (i.e $\mathbf{r}_i = tanh(\mathbf{x}_i)$). The hyperbolic is a non-linear activation function that is defined as follows:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{8}$$

This is a commonly used activation function in machine learning and artificial intelligence which maps the input to a range between -1 and 1. It is also important to note that in a biological sense this activation function is a stand in for non-linear neuronal dynamics. The question we ask is how does changing this function affect the network and its ability to learn or predict. There are a plethora of other activation functions to consider, but as it turns out none of them come anywhere close to the the capabilities of *tanh*. There are several classes of functions to consider and the first we will discuss are those that map positive inputs linearly. This includes a linear activation, rectified linear units, exponential linear units, and others. These functions result in the network dynamics exploding with activity shooting off to infinity. This happens because the network is recurrently connected to itself and in the architecture we consider (4.3) there is also a feedback loop sending the error back into the network. During learning if these large magnitude values are fed back into the network the activity just keeps increasing – especially if they are mostly positive e.g ReLU. The next set of functions to consider are those that map the input into some smaller output range. This includes *tanh*, but it also includes the sigmoid function (which maps between 0 and 1) as well as the softmax (probability density) function. It would be fair to assume that given the fact that these functions have more similar dynamics to *tanh* that they would result in similar capabilities in training. However, in practice this does not appear to make any difference. Even shifting the sigmoid to include negative values will still not result in being able to learn a simple function. Essentially what we find is that among the most widely used activation functions the only one that will result in successful training is the hyperbolic tangent function. It allows for both positive and negative outputs and it maps everything to a magnitude less than 1. This allows network dynamics to be kept under control while still having a wide sample of excitatory and inhibitory action.

We next consider three of the most important and obviously impactful network parameters.

These include the network size N, the initial inverse learning rate α, and the chaos of network g. This chaos factor was something that was considered in [SL09], and we are able to recreate their results.



(a) RMS error as a function of network size

(b) RMS error as a function of α

Figure 12: Root Mean Squared (RMS) prediction error as function of network parameters.

In the figure 12 we see the network prediction error (RMS error) as function of the network size and of the parameter alpha. Looking at the network size curve first we see what would be expected. As the network size increases the prediction error steadily drops – at least up to a point. Once we reach around 700 neurons the prediction error does not change much. This should make sense because there is a limit to how much more accurate a network can get once it has learned the function. Throwing more neurons at it wont continue to make it better once the function has been learned. In terms of alpha we should first revisit what this parameter controls. In section 4.2 we discussed how alpha serves as the initial condition for the $\mathbf{P}$ matrix used in recursive least squares. Specifically $\frac{1}{\alpha}$ serves as the initial learning rate for all neurons in the network. However, as learning progresses this learning rate will change depending on the specific neuron. Thus increasing α serves to decrease the initial learning rate – which makes it not immediately obvious why this would increase the prediction error. The answer lies in the way FORCE learning works, which as we have discussed earlier relies on quick reduction in initial error to control chaos. If α is too large then in the update rule seen in equation (6) the weights are updated slower. This can cause the network to be incapable of holding the output close to the target function during learning, resulting in poor prediction.

In figure 13 below we see two plots of functions of the parameter g which indicates the chaos of the internal connections in the network. We note again that $g > 1$ is what leads to chaos and for most of our examples we have taken $g = 1.5$.

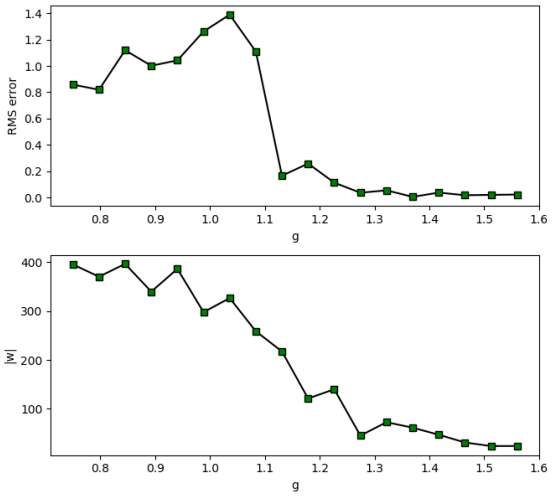


Figure 13: The effect of chaos in learning.

In the first of the two plots we see the root mean squared error as a function of g. Although the initial trace is somewhat uncertain we see a distinct trend as g surpasses 1. The error drops significantly and stays low as the chaos increases. This error is a good indication of the accuracy of the network and we see that more chaos leads to higher accuracy in prediction. This only extends up to a point as too much chaos will prevent FORCE from inducing order during learning. As discussed previously this will prevent successful learning.

In the second of the two plots we see the magnitude of the internal network weights as a function of g. A larger $|W|$ indicates that there are either large positive or large negative values, and because the network in this case is stable in producing its target function we know that a large $|W|$ indicates cancellations between these large values. Thus a smaller $|W|$ indicates that the network is more stable (i.e. less sensitive to perturbations). We see that this weight magnitude declines steadily as chaos increases. This indicates that as the network increases in chaos it becomes more stable.

8 Discussion

FORCE learning is an effective method for training highly chaotic recurrent neural networks like those inspired by biological neural circuits. FORCE works by keeping the error small throughout learning while optimizing for weights that will keep this error small. Importantly FORCE requires that the output feedback into the network is able to suppress chaos during learning. When applied properly we see that FORCE is able to learn a wide range of functions.

From our results in the PCA section we can conclude that the network activity is driven mostly by the leading few principle components. This means that producing the predictions for each target function is not a complex task for a network of a thousand neurons. We saw that when the number of outputs increases the number of principle components that compose most of the

networks activity remains the same. This tells us that increasing the number of outputs does not result in a more complex network and the same amount of network activity in producing one output is similar to that of multiple for the functions that were tested. This makes sense neurobiologically because a population of neurons should be able to handle producing more than one output just as easily as one.

In section 7 we saw that there were a number of components of FORCE that had significant effects on learning and prediction – for which there are two important takeaways. First, the FORCE paradigm can have a significant number of parameter choices that vary depending on the implementation. Some have more immediately obvious impacts on network dynamics while others have more subtle effects. Our investigations into these parameters were relatively limited and were constrained to simple functions. A more thorough investigation into the structure of FORCE could provide some insight for those interested in applying it regularly. As well, this could help in making some connections between parameters and biological analogues. Lastly, the fact that only the hyperbolic tangent function was capable of training a network is quite intriguing. It is perhaps not so surprising that this function works, but more surprising that no other functions do. One thing to note is that the activation functions are artificial replacements for complicated non-linear neural dynamics, However, because FORCE does not require any sort of differentiation (i.e backpropagation) any sort of non-linear function may be applied to the activation even if it would not be usable in a feed forward network.

If given more time, one of the areas we wanted to explore with the FORCE learning regimen was motion capture to replicate human motor outputs. Unfortunately, there were several roadblocks we encountered trying to do so. In theory, and as shown in [SL09], the FORCE regime should be able to predict aperiodic motion, however in this case the network setup would have needed to be much more complex. Had we been able to reproduce human motor output an investigation into the principal components of said network would be revealing. This would help to determine if these motor outputs are relatively low dimensional like we found for multiple simpler outputs.

9 Appendix:

Code Repository: https://github.com/colesturza/APPM4370-Project

10 References

[JH04] H. Jaeger and H. Haas. "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication". In: *Science* 304.5667 (Feb. 2004), pp. 78–80.

[SL09] David Sussillo and Abbott L.F. "Generating Coherent Patterns of Activity from Chaotic Neural Networks". In: *Neuron* 63 (2009), pp. 544–547.

[DeP+18] Brian DePasquale et al. "full-FORCE: A target-based method for training recurrent networks". In: *PloS one* 13.2 (2018).