Final Project

Daniel Dager

Introduction

In this report, we aim to replicate and explain key aspects of the paper "Generating Coherent Patterns of Activity from Chaotic Neural Networks." The paper introduces the First-Order Reduced and Controlled Error (FORCE) algorithm, a method for training recurrent neural networks (RNNs) to generate specific patterns of neural activity from chaotic spontaneous activity. We will explicitly define the FORCE algorithm, chaotic spontaneous activity, and the Recursive Least Squares (RLS) method as they are crucial to understanding the underlying principles of this work. Additionally, we will reproduce and analyze key figures from the paper, demonstrating the FORCE algorithm's effectiveness in transforming chaotic activity into coherent patterns.

The fundamental issue with training recurrent neural networks is that modification of the output weights can have unintended downstream consequences, even if at the time of modification, the modification does move the network closer to the target output. This is due to the inherent complexities and non-linearities of recurrent networks.

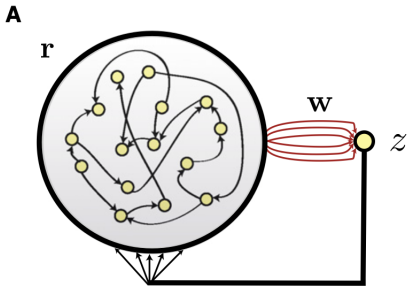


Figure 1A: A typical recurrent generator network, where $\boldsymbol{r}$ is a vector of firing rates, $\boldsymbol{w}$ is a vector of modifiable weights, and z is the output of the system.

Jaeger and Haas, in their paper "Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication" (2004), solve this problem by "clamping" the feedback signal. During training, rather than feeding the network its own output z, they fed it the target output f.

A key advantage to FORCE training is that this clamping is no longer necessary. "First-Order Reduced and Controlled Error" refers to an algorithm that reduces error magnitudes during training so that the feedback signal can be perserved without causing rampant instability.

Crucially, the algorithm described by the authors employs recursive least squares (RLS) to update the weights of the network. To describe RLS modification, we start by defining an $N \times N$ matrix $\boldsymbol{P}$, a running estimate of the inverse of the correlation matrix of the network rates r plus a regularization term:

$$\boldsymbol{P} = (\sum_t \boldsymbol{r}(t)\boldsymbol{r}^T(t) + \alpha \boldsymbol{I})^{-1} \tag{1}$$

The matrix $\boldsymbol{P}$ is initalized as $\boldsymbol{P}(0) = \frac{\boldsymbol{I}}{\alpha}$, where α is some constant. It updated at each time step t where the weights are updated, according to the following update rule:

$$\mathbf{P}(t) = \mathbf{P}(t - \Delta t) - \frac{\mathbf{P}(t - \Delta t)\mathbf{r}(t)\mathbf{r}^T(t)\mathbf{P}(t - \Delta t)}{1 + \mathbf{r}^T(t)\mathbf{P}(t - \Delta t)\mathbf{r}(t)}. \tag{2}$$

We also define the error prior to the weight update at time t, $e_{-}(t)$, as:

$$e_{-}(t) = \mathbf{w}^T(t - \Delta t)\mathbf{r}(t) - f(t) \tag{3}$$

Finally, the weights are updated according to the following rule:

$$\mathbf{w}(t) = \mathbf{w}(t - \Delta t) - e_{-}(t)\mathbf{P}(t)\mathbf{r}(t) \tag{4}$$

We can see from this update rule that the error is always constrained by the matrix $\boldsymbol{P}$. This is what is meant by "First-Order Reduced and Controlled Error." First-Order refers to the fact that initial output errors (errors directly following a weight modification) are the ones being reduced and controlled, rather than the higher-order effects of weight modifications which may manifest themselves later.

Before proceeding, we might do well to describe another concept which motivates the FORCE algorithm: chaotic spontaneous activity. In the context of neural networks, chaotic spontaneous activity refers to the virtually unpredictable patterns of neural activity that can emerge from a network of neurons. This is activity which is highly sensative to initial conditions and pertubations.

The authors postulate that behavior can be attributed to the reorganization of spontaneous neural activity in organism into coherent patterns. The FORCE algorithm is meant as one possible explantion for how this reorganization may come about.

Reproduction of Figures

In this next section, we will reproduce and analyze key figures from the paper. We will begin by reproducing figures 2A-C, which demonstrate the chaotic spontaneous activity of an untrained network of 1000 neurons, the network's output and the magnitude of the changes in the network's weights during training, and the network's output after training, respectively.

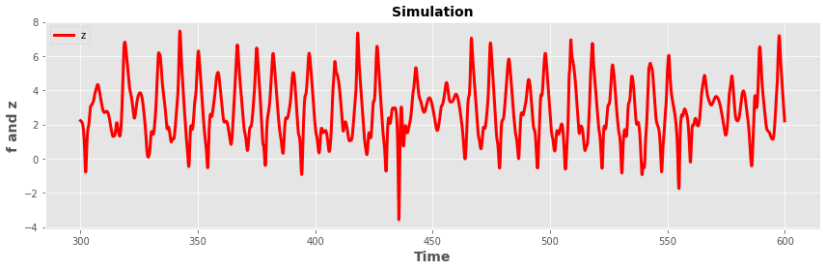


Figure 2A: Chaotic spontaneous activity of an untrained network of 1000 neurons. There is no magnitude in the change of weights because the weights are not yet being modified.

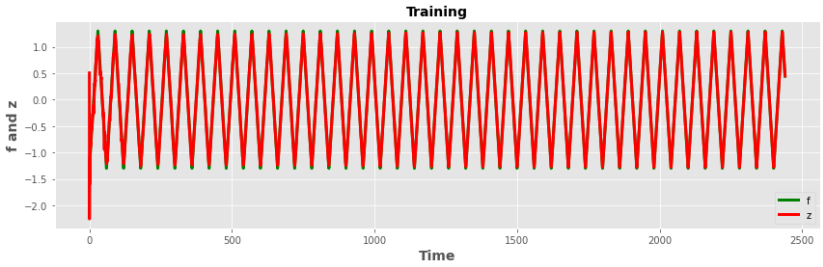


Figure 2B: The network's output during training. As described by the authors, the network's output immediately matches the target function.

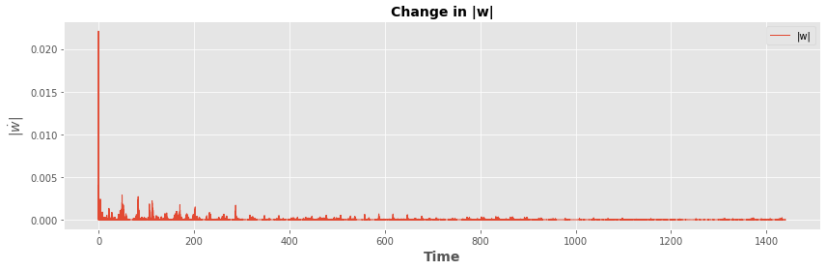


Figure 2C: The magnitudes of the changes in the network's weights during training. They start off relatively high, but decrease to almost zero by the end of the training phase.

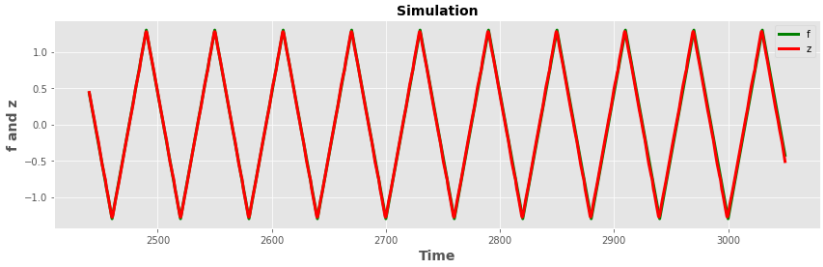


Figure 2C: Post-training output corresponds to the target function. We know there is no magnitude in the change of weights because the weights are no longer being modified.

Now we will reproduce figures 2D, 2E, 2F, 2G, and 2I, all of which show the network's ability to generate different target functions after training:

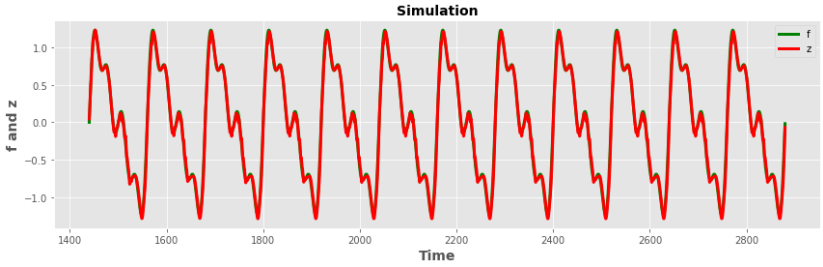


Figure 2D: Here the network reproduces a target function which is the sum of four sinusoidal waves.

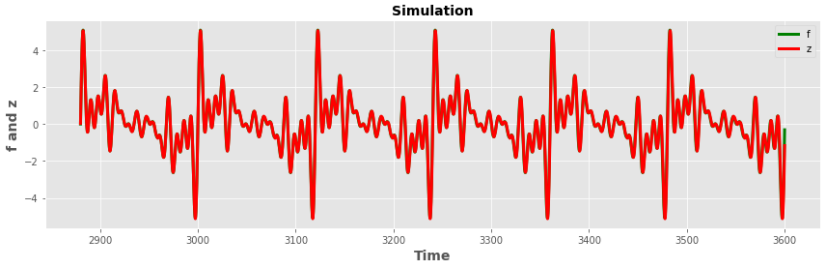


Figure 2E: The network reproduces a target function which is the sum of sixteen sinusoidal waves.

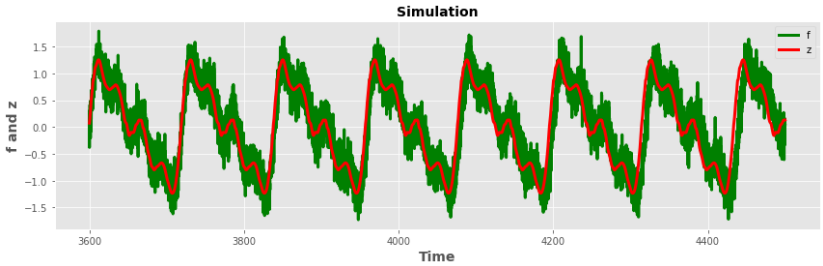


Figure 2F: Now the network recovers a target function which is the sum of four sinusoidal waves which was perturbed by a Gaussian noise term.

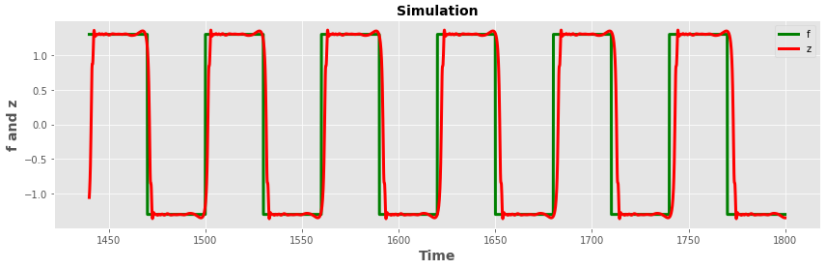


Figure 2G: The network reproduces a discontinous target function, the square wave.

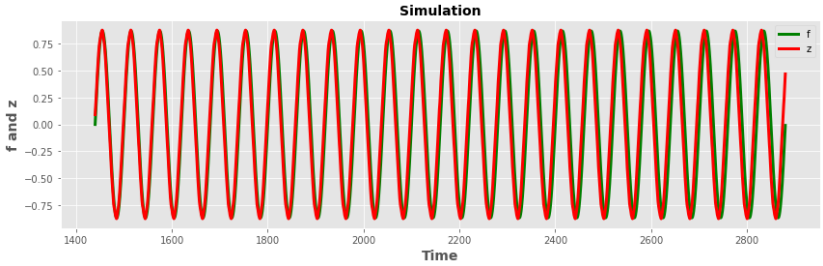


Figure 2I: The network reproduces a target function which is a high frequency sine wave.

Now we move on to PCA analysis of the network's neural activity. We begin with figure 3A, which shows the projection of neural activity onto the first eight principal components, after the network has been trained to reproduce a target function which is the sum of four sinusoidal waves:

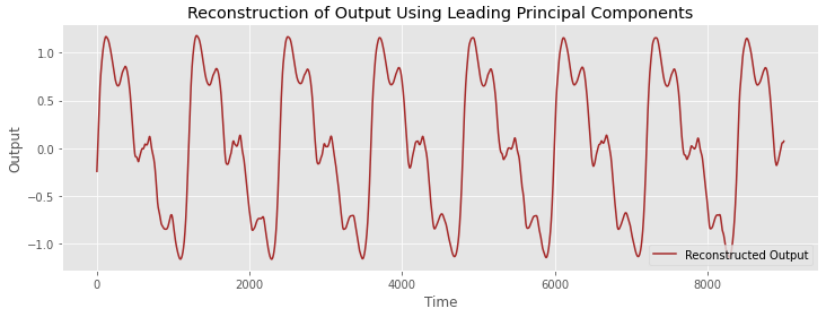
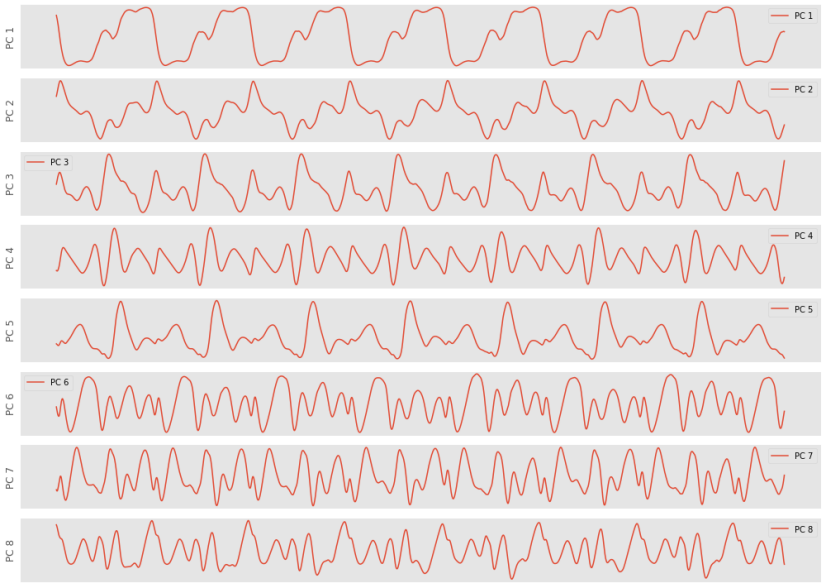


Figure 3A: The reconstructed output clearly matches that of figure 2D. This demonstrates that the firing rates of the trained network encode the target function.

We also reproduce figure 3B, which shows the projection of the network's neural activity onto the first eight principal components during a post-training simulation:



Next, we reproduce figure 3C, which gives a sense of the power of the principal components of the network's neural activity during a post-training simulation. We take the log of the eigenvalues of the PCA decomposition of the network's neural activity and see that only the first twenty-three components have significant power. This differs slightly from the authors' results, but the general trend is the same:

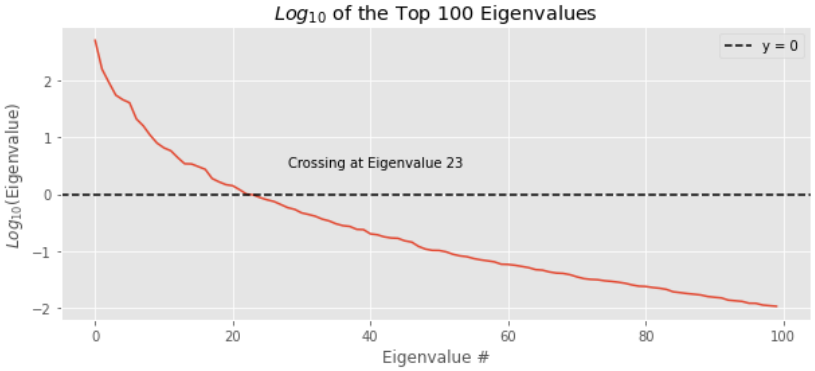


Figure 3C: The log of the eigenvalues of the PCA decomposition of the network's neural activity.

Finally, we reproduce figure 3E, where we plot the projections of the weight vectors onto the first two principal components over the course of five training sessions, and observe that the network converges to the same solution from different initial conditions:

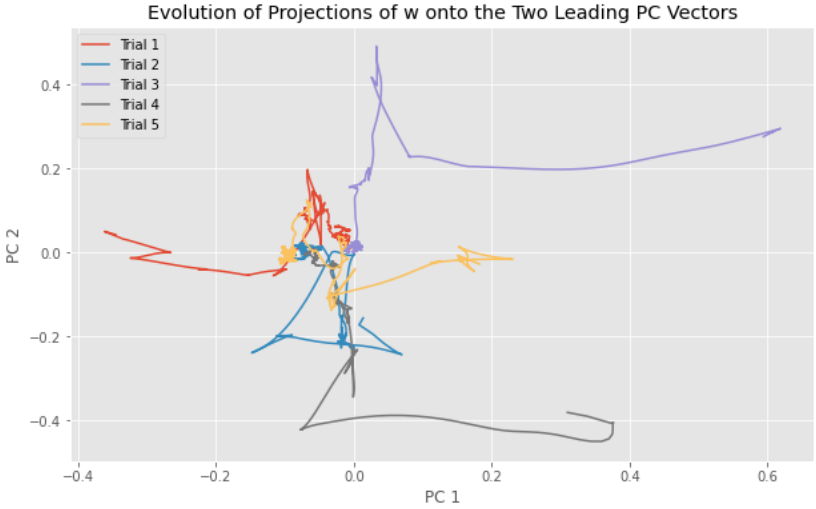


Figure 3E: The projections of the weight vectors onto the first two principal components over the course of five training sessions.

Conclusion

In conclusion, this report demonstrated the application of the FORCE learning algorithm to train recurrent neural networks to reproduce desired patterns of neural activity. The implementation involved generating various target functions, such as sine waves and a discontinous functions like the triangle and square waves. Through the use of principal component analysis (PCA), we examined the projections of neural activities and the evolution of output weights, highlighting the network's capability to converge from different initial conditions. The findings confirm that the FORCE algorithm effectively stabilizes chaotic neural activity, enabling the RNN to generate coherent patterns that closely match the target functions.

This represents a significant advancement in understanding how chaotic systems can be reorganize themselves so as to have a controlled output. The potential biological and cognitive implications of this algorithm are profound. Perhaps the brain utilizes a similar mechanism to stabilize and control chaotic neural activity in order to produce consistent behaviors.