Peaking Into the Black Box of Neural Networks

Recurrent Neural Networks as Dynamical Systems

Andrew Sears, Hanson Mo, Eduard Kovalets, Avinash Joshi

14 December, 2022

AMATH 422/522 Au. 2022

Contents

Abstract

Recurrent neural networks (RNNs) are powerful tools for approximating and predicting time series data, but how they go about doing so is largely shrouded in mystery. By conceiving of RNNs as nonlinear systems of ordinary differential equations, or dynamical systems, and analyzing their hidden states for fixed points, how RNNs implement their computation can be better understood. This procedure from Sussillo and Barak [12] is implemented in the `python` library `pytorch` and used to investigate how a RNN implements the FitzHugh-Nagumo model for neuronal excitation. By extending the procedure further in a novel direction, an approximation of the FitzHugh-Nagumo model's analtyic fixed point v^* can be found without training a RNN on such fixed points; however, the relationship learned between input, I_{ext}, and the approximated fixed point, $\tilde{v}^*$, is linear rather than the cubic relationship of the analytic fixed points, v^*.

1 Introduction

Recurrent neural networks (RNNs) have quickly become one of the most useful tools to reproduce difficult, nonlinear, time and state dependent time-series data in fields such as mathematical finance, neuroscience, artificial intelligence, and machine learning. Even though they are widely used and thorough research has been done to investigate different architectures and training procedures, much of how RNNs implement their computation to reproduce nonlinear dynamics is shrouded in vague mystery under the all-encompassing phrase "Black Box." In order to pry open such black boxes, Dr. David Sussillo and Dr. Omri Barak [12] present a simple procedure in uncovering how RNNs implement their computation through the lens of dynamical systems and techniques to understand nonlinear behavior, principally, finding fixed points in phase space and analyzing trajectories close to such locations using linear stability analysis (LSA). The first half of this paper will seek to motivate and then outline the procedure presented by Sussillo and Barak, then provide an implementation of such a procedure generally in the `python` neural network library `pytorch`.

The problems investigated by Sussillo and Barak, however, are of a "computer science" variety, i.e., examples that focus on problems related to computer related tasks such as averages and memory. While such examples are integral in allowing others to plainly understand one's work, the Sussillo and Barak procedure has not yet been applied to many biological problems [8] or taken to a logical next step — analyzing how RNNs replicate the behavior of nonlinear dynamical systems. For this reason, the second part of the paper will look to apply the Sussillo and Barak procedure to the FitzHugh-Nagumo model for neuronal excitation and analyze the results from applying their procedure to the nonlinear dynamical

system. Before the procedure is introduced, the integral components of it, RNNs, dynamical systems, and linear stability analysis, will be introduced first alongside an introduction to the FitzHugh-Nagumo model. Following this introduction, the procedure will be given in full, it's implementation in `pytorch` discussed, and reproductions of Sussillo and Barak's results shown as validation for the procedure's `pytorch` implementation.

1.1 Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) and, more broadly, neural networks (NNs) have been heralded as universal solvers for any flavor of problem but widely understood as incredibly complex and difficult to rigorously understand. This complexity originates from the combination of numerous linear operations, nonlinear functions, and "learning" capability baked into the "architecture" of a specific neural network. For RNNs, one finds an additional layer of complexity in the so-called "hidden state" variable or "memory" of an individual RNN cell, $\mathbf{x}$. Each RNN cell, $t = 0, 1, 2, \ldots, N$, takes in as input some scalar, vector, or matrix, $\mathbf{u}^t$, and has a series of matrix operations and nonlinear functions applied to it in order to approximate some desired output, $\mathbf{z}^t$. Each successive matrix operation that is applied to the input, $\mathbf{u}^t$, and output, $\mathbf{z}^t$, are called the hidden layers of the cell. The components of the different matrices and tensors that inhabit each hidden layer are trained using a technique called backpropagation and implemented through gradient descent, though both will not be discussed in detail. Briefly, a loss function is employed to quantify how much the network matrices must change based on the predicted output $\tilde{\mathbf{z}}^t$ and the correct output $\mathbf{z}^t$ and calculates how much each component of each hidden layer must change with respect to the error in approximating $\mathbf{z}^t$.

While much of the previous discussion generally applies to most NNs (except for the fact that cells are only found in RNNs), the hidden state variable of cell t, $\mathbf{x}^t$, is unique to RNNs and is where the power of RNNs as time-series data approximators comes from. In the process of predicting $\mathbf{z}^t$ given $\mathbf{u}^t$, $\mathbf{x}^t$ serves as an intermediary variable that expresses the impact of previous computations done by the network on the current cell — the current cell's "memory" of previous computations. In the most vanilla of RNN architectures, $\mathbf{x}^t$ is changed after each cell by some recurrence relation function, $F(\mathbf{x})$, and only changes the following cell's hidden state variable, $\mathbf{x}^{t+1}$. It should be noted that all hidden layer functions and parameters are shared between cells, so the only component inside the RNN that changes between each cell is $\mathbf{x}^t$, the hidden state variable. A simplified graphical representation of the vanilla RNN architecture and structure, in both its unravelled and ravelled state, can be found in Figure **??**. Graphically, the recurrence between cells is given by horizontal arrows from either one side of the cell to the other side, for the ravelled cell diagram, or from one cell to the next moving rightward, for the unravelled diagram. This recurrence, or dependence on previous cells' computations, is found similarly in a class of systems of ordinary differential equations (ODEs) called dynamical systems.

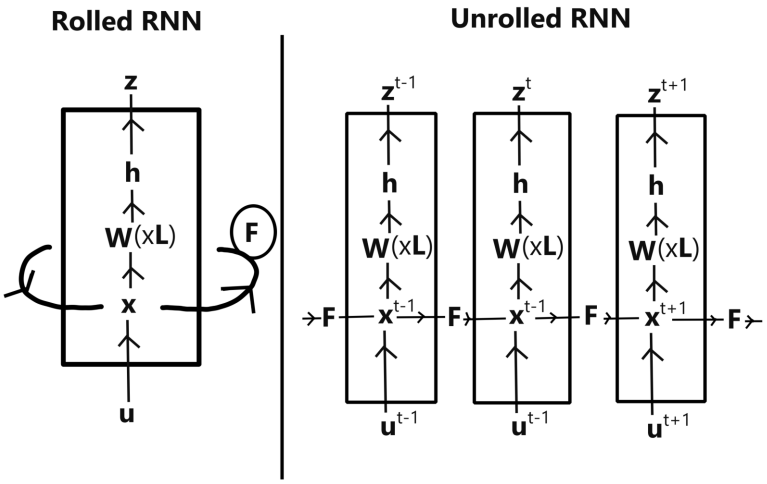


Figure 1: A general RNN architecture with input $\mathbf{u}$ to the RNN followed by a linear transform into the hidden state $\mathbf{x}$. This is then followed by L successive applications of dense linear hidden layers by W_l and passed through a nonlinear function h. Another linear transform is then applied to this and is outputted as $\mathbf{z}$. The recurrence relationship between each cell is defined by the function F. The rolled version of this general RNN architecture is on the right while the unrolled version is on the left.

1.2 Dynamical Systems and Linear Stability Analysis (LSA)

Dynamical systems are nonlinear systems of first-order ordinary differential equations (ODEs) that model the state dependent dynamics of some phenomena [11]. For an m dimensional system with state variable $\mathbf{x}(t) = [x_1(t), x_2(t), \ldots, x_{m-1}(t), x_m(t)]$, the corresponding dynamical system would generalize as

$$\dot{x}_1 = f_1(x_1, \ldots, x_m), \tag{1a}$$
$$\dot{x}_2 = f_2(x_1, \ldots, x_m), \tag{1b}$$
$$\vdots \tag{1c}$$
$$\dot{x}_{m-1} = f_{m-1}(x_1, \ldots, x_m), \tag{1d}$$
$$\dot{x}_m = f_m(x_1, \ldots, x_m). \tag{1e}$$

The set of m, first order nonlinear differential equations has each component's time derivative, $\dot{x}_i$, given by a nonlinear equation dependent on the state of the system $\mathbf{x}$, $f_i(\mathbf{x})$. Put into compact form,

$$\dot{\mathbf{x}} = A(\mathbf{x}). \tag{2}$$

Typically, closed formed solution to dynamical systems are impossible, so qualitative descriptions of system behavior is required to characterize the evolution of such nonlinear dynamical systems. Such qualitative descriptions are found by studying when the dynamical system

does not change, i.e., when $\dot{\mathbf{x}} = 0$. The expression for when each equation defining a component's first derivative is found to not change, $\dot{x}_i = f_i(\mathbf{x}) = 0$, is called the nullcline of component x_i, or the x_i nullcline. Finding the intersection of all m nullclines, $A(\mathbf{x}) = 0$, corresponds to the state in phase space when, if the system is at that state, it will not change for all of time. These locations are called fixed points and are denoted as $\mathbf{x}^*$.

Analysis of behavior around such fixed points $\mathbf{x}^*$ is a key technique in analyzing nonlinear dynamical systems. This technique is called linear stability analysis (LSA). Linear stability analysis (LSA) is defined only short distances away from such fixed points $\mathbf{x}^*$ using a Taylor expansion some small distance away from $\mathbf{x}^*$, $\mathbf{x}(t) = \mathbf{x}^* + \Delta\mathbf{x}(t)$, to quantify if trajectories evolve towards $\mathbf{x}^*$, away from $\mathbf{x}^*$, or orbit around $\mathbf{x}^*$. The corresponding Taylor expansion around $\mathbf{x}^*$ is

$$\frac{d\Delta\mathbf{x}}{dt} = A(\mathbf{x}^*) + \Delta\mathbf{x}(t)J(\mathbf{x}^*) + O(\Delta\mathbf{x}^2(t)), \tag{3}$$

where $J(\mathbf{x})$ is the Jacobian at $\mathbf{x}$ and is defined by

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \dots & \frac{\partial f_1}{\partial x_m}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \dots & \frac{\partial f_2}{\partial x_m}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \frac{\partial f_m}{\partial x_2}(\mathbf{x}) & \dots & \frac{\partial f_m}{\partial x_m}(\mathbf{x}) \end{bmatrix}. \tag{4}$$

By the definition of a fixed point, Equation 3 simplifies to $\frac{d\Delta\mathbf{x}}{dt} = \Delta\mathbf{x}(t)J(\mathbf{x}^*)$; thus, whether trajectories close to $\mathbf{x}^*$ evolve to, away from, or around $\mathbf{x}^*$ is dependent on the values of the m eigenvalues λ_i of the Jacobian matrix $J(\mathbf{x}^*)$ with the associated directions of maximal attraction or repulsion given by the eigenvectors v_i for $i = 1, 2, \dots, m$. $\frac{d\Delta\mathbf{x}}{dt} = \Delta\mathbf{x}(t)J(\mathbf{x}^*)$ can then be decomposed into a general solution

$$\Delta\mathbf{x}(t) = c_1 v_1 e^{\lambda_1 t} + c_2 v_2 e^{\lambda_2 t} + \ldots + c_m v_m e^{\lambda_m t}. \tag{5}$$

The stability of the solution is classified as unstable (trajectories will grow exponentially away from $\mathbf{x}^*$ to $\pm\infty$) if a single $Re(\lambda_i) > 0$ and is stable (trajectories will asymptotically approach $\mathbf{x}^*$) if $Re(\lambda_i) < 0\ \forall\lambda_i$. More complex stability and behavior can take place such as oscillatory behavior or so called "limit cycles," an example of which can be found in the FitzHugh-Nagumo (FHN) model for neuronal excitation, the model that will be studied in this paper.

1.3 The FitzHugh-Nagumo (FHN) Model for Neuronal Excitation

The FitzHugh-Nagumo (FHN) model for neuronal excitation is a two dimensional, nonlinear, system of ordinary differential equations that models the voltage v across a neuron in the presence of an external current, I_{ext}, and a recovery variable, w. The model itself is a simplification of the four dimensional Hodgin-Huxley model for neuronal excitation [5] and was independently conceived of and studied by Richard FitzHugh [2] in 1961 and Jin-ichi Nagumo *et al.* [9] in 1962. In short, to pass signals between cellular structures in the brain, called neurons, one neuron sends an electro-chemical signal through a wire-like structure

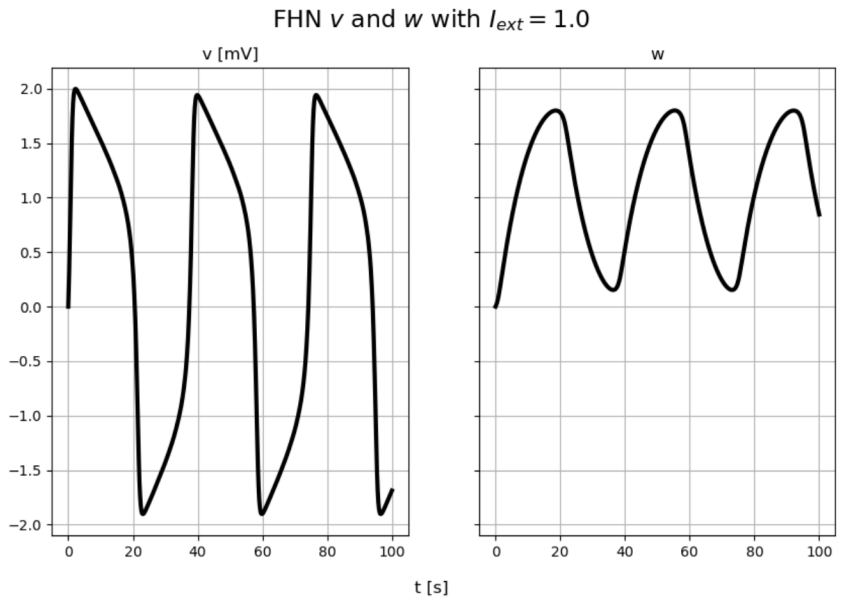


Figure 2: An example trace of the FitzHugh-Nagumo model with $a = 0.7$, $b = 0.8$, $\tau = 12.5$, and $I_{ext} = 1.0$. The left subgraph shows the v variable in time while the right graph shows the w variable in time. Both variables exhibit oscillatory behavior given these parameters.

called the axon of a neuron. When the voltage across the axon is measured as a signal propagates through it, a rapid increase in voltage is measured followed by a rapid decrease in voltage. The FHN model is canonically described by

$$\dot{v} = v - \frac{v^3}{3} - w + I_{ext}, \tag{6a}$$

$$\dot{w} = \frac{1}{\tau}(v + a - bw). \tag{6b}$$

v is the voltage (in millivolts) across a neuron and w is the recovery variable (in millivolts) that defines the rate at which the membrane repolarizes (becomes most negative) after it has spiked (reached its most positive voltage in a cycle). I_{ext} is a constant external current that causes the neuron to spike, while a, b, and τ are constants related to the rate of recovery of the neuronal membrane. The recovery variable, w, represents the combined effect of three different variables from the original Hodgkin-Huxley model and relates to the influx and outflux of three different ion species that precipitate and halt neuronal spiking. For this paper, the constants will be fixed to the following values: $a = 0.7$, $b = 0.8$, and $\tau = 12.5$. An example trajectory of the system using $I_{ext} = 1.0$ can be seen in Figure 2. Because a sufficiently large constant current is applied to the neuron, the neuron repeatedly spikes in an oscillatory manner. The FHN model's unique oscillatory behavior given a sufficiently large external current will be replicated latter using a RNN and then investigated using the following procedure introduced by Sussillo and Barak and outlined in Section 2.

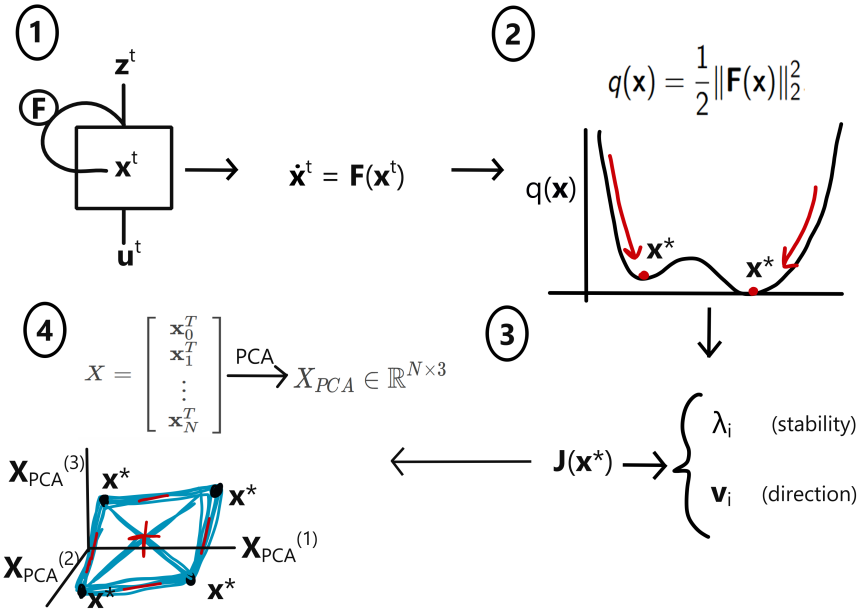


Figure 3: A graphical representation of the Sussillo and Barak procedure in 4 steps starting in the top-left corner going counter-clockwise. 1) A trained RNN's recurrence relation F is formulated as a dynamical system with respect to the hidden state variable $\mathbf{x}$. 2) Optimization is done on the optimization form of F, $q(\mathbf{x})$, to find the fixed points of $\mathbf{x}$. 3) The Jacobian at $\mathbf{x}^*$ is computed to determine the stability of each fixed point. 4) The hidden state trajectories are reconstructed using the first 2 or 3 principle components using PCA to visualize the dynamics of the hidden state at low dimensions.

2 RNNs to Dynamical Systems: The Procedure

Just looking at RNNs and dynamical systems, there seems to be no obvious connection between the two. One is a nonlinear neural network comprised of numerous identical cells that one must train to perform a specific task while the other is a nonlinear system of differential equations that produces complicated and varied behavior. How are these two seemingly distinct products of mathematics related? and how can we leverage the techniques developed for the latter to study the enigmatic former? Both of those questions are answered by first highlighting two defining components that both systems share - nonlinearity and state dependence. The first of the two components, nonlinearity, is obviously shared between both. RNNs are nonlinear by design in order to ensure that they can learn desired temporal behavior, and many (of the interesting varieties of) dynamical systems feature nonlinear terms which precipitates beautiful and complex behavior. The second component, state dependence, is the more nuanced component of both and is critical in formulating RNNs as a dynamical system. To start with dynamical systems, the change in a system, as described generally in Equation 2, is obviously dependent on the current state of the system as $\dot{\mathbf{x}}$ is a function of $\mathbf{x}$. For RNNs, however, the key to their state dependent nature is located in the information transmitted forward from cell-to-cell, its hidden state variable $\mathbf{x}^t$. To apply linear stability analysis to qualitatively understand RNN behavior, the task then becomes formulating RNNs as dynamical systems, the key to Sussillo and Barak's method. The other three steps associated with their method, finding the fixed points of the RNN hidden state, classifying their stability, and visualizing the qualitative dynamics in low dimensions,

naturally follows from the first step. The procedure is displayed graphically in Figure 3.

2.1 From a RNN to a Dynamical System

The ability to formulate RNNs as dynamical systems allows RNNs to be analyzed like dynamical systems; thus, the curtains obstructing how RNNs implement their computation can slowly be drawn back using analytical techniques associated with dynamical systems. To better illustrate this key observation and the first step in the Sussillo and Barak procedure, a proposed RNN (an echostate neural network) from Sussillo and Barak [12] will be used as an example to show how a RNN can be written as a dynamical system. The RNN used by Sussillo and Barak, seen graphically in Figure 4, takes in an I dimensional input $\mathbf{u}$ to cell t, $\mathbf{u}^t$, and is modified by a weight matrix $\mathbf{B}$. This modified input quantity then directly affects $\mathbf{x}^t$. The modified input quantity is then passed through a nonlinear function h, becoming $\mathbf{r}^t$, which then progresses out of the cell by another weight matrix, $\mathbf{W}$, as $\mathbf{z}^t$. The next hidden state variable $\mathbf{x}^{t+1}$ is then adjusted by value of the previous hidden state variable, $\mathbf{x}^t$, as we assume some continuity in calculation, and also by the output, $\mathbf{z}^t$, adjusted by a feedback weight matrix, $\mathbf{W}^{fb}$, and by the output of the nonlinear function, $\mathbf{r}^t$, multiplied by another feedback matrix, $\mathbf{R}$. The recurrence between cells can be seen graphically by the arrows that leave cell t on the right and connect to cell $t+1$ on its left. It follows that, considering the system from cell to cell as a discrete system, the RNN's hidden state is changed by

$$\mathbf{x}^{t+1} = -\mathbf{x}^t + W^{fb}\mathbf{z}^t + R\mathbf{r}^t + B\mathbf{u}^t, \tag{7a}$$

$$\mathbf{r}^t = \mathbf{h}(\mathbf{x}^t) \tag{7b}$$

$$\mathbf{z}^t = W\mathbf{r}, \tag{7c}$$

which naturally extends to continuous time system using limiting arguments and methods like Forward Euler to get

$$\dot{\mathbf{x}}^t = \mathbf{F}(\mathbf{x}^t, \mathbf{u}^t, \mathbf{z}^t; \mathbf{W}, \mathbf{W}^{fb}, \mathbf{B}, \mathbf{R}). \tag{8}$$

In Equation 8, the matrices $\mathbf{W}$, $\mathbf{W}^{fb}$, $\mathbf{B}$, and $\mathbf{R}$ are found after a semicolon indicating they are constants. They become constants after having trained the RNN to replicate the desired dynamics given an input; thus, when discussing RNNs as dynamical systems, only trained RNNs should be analyzed. Using Equation 8, we can now analyze RNNs as dynamical systems around their hidden state fixed points using linear stability analysis.

2.2 Finding $\mathbf{x}^*$ of a RNN and Classifying Stability

To analyze RNNs using linear stability analysis, the fixed points of the system must first be found; however, for most RNNs, translating the recurrence relation into functional form as in Equation 8 is difficult if not impossible and, if possible, is even more difficult to analytically find fixed points as described in Section 1.2. To combat this difficulty and make the Sussillo and Barak procedure more generalizable, an intermediary function that can be

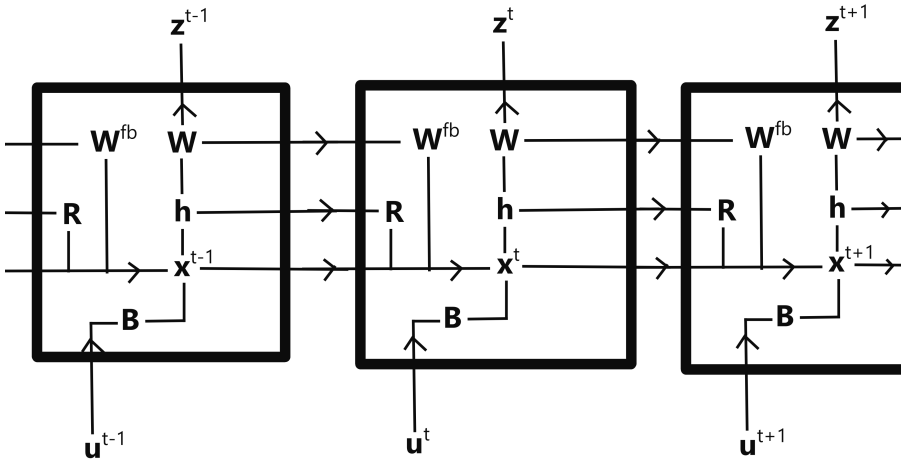


Figure 4: The RNN circuit diagram for the Sussillo and Barak echostate neural network used in their 3-Bit Flip-Flop task and by the authors of this paper to show how RNNs can be formulated as a dynamical system.

optimized is used to numerically approximate fixed points to a high degree of accuracy. This intermediary function, $q(\mathbf{x})$, is defined as

$$q(\mathbf{x}) = \frac{1}{2}\|\mathbf{F}(\mathbf{x})\|_2^2 \tag{9}$$

ensuring that when $q(\mathbf{x}) = 0 \approx \epsilon$, $\mathbf{x} = \mathbf{x}^*$, a fixed point of the RNN hidden state variable, because $\|\cdot\|_2 \geq 0$ and $q(\mathbf{x}) = 0$ only when $\mathbf{F}(\mathbf{x}) = 0$, or when the hidden state is at a fixed point and does not change. ϵ is chosen sufficiently small in order to approximate $\mathbf{x}^*$ else incorrect values of $\mathbf{x}$ would be considered fixed points. The non-linear non-convex optimization algorithm that is used to find when $q(\mathbf{x}) = 0$ and approximate $\mathbf{x}^*$ is gradient descent[1]. In order to ensure that the proposed $\mathbf{x}^*$ is indeed a fixed point, a trained RNN is given the input data and produces a hidden state for each cell and a random selection of those hidden states are then used as initial conditions for the optimization procedure.

After the all fixed points of the RNN hidden state have been found, their stability is then determined by the Jacobian of the system at $\mathbf{x}^*$, outlined in Section 1.2. In practice, a numerical approximation of the Jacobian is used.

2.3 Visualizing Dynamics in Low Dimensions

Since the hidden layer variable is often hundred- or thousand-dimensional, visualizing how each component of $\mathbf{x}$ changes as the RNN predicts an output given an input is impossible and effectively useless. Instead, by distilling the dynamics into a very low dimensional

[1]A brief description of gradient descent can be found here with an excellent video found here by the YouTube channel 3Blue1Brown.

approximation of the high dimensional $\mathbf{x}$, usually either two or three dimensions, the dynamics of the hidden state variable are more easily identified and understood using linear stability analysis (LSA). Using principle component analysis (PCA) (implemented through singular value decomposition [SVD]), the high dimensional dynamics of the hiddens state variable are easily visualized and better understood. Simply, PCA decomposes a matrix $M \in \mathbb{R}^{n \times m}$ into a matrix product of two orthonormal bases and an matrix of singular values which are related to the eigenvalues of the system. Formally, SVD is defined as

$$M = U\Sigma V^T \tag{10}$$

where $U \in \mathbb{R}^{n \times m}$, $\Sigma \in \mathbb{R}^{m \times n}$, and $V^T \in \mathbb{R}^{n \times n}$. U is called the left-singular vectors of M while V is the right-singular vectors of M. The singular values σ_i of Σ are ordered in such a way that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m$ with the associated re-ordering of U and V^T to match. By reconstructing M using only p columns of U and Σ, for $p \leq m$, a lower dimensional approximation of the matrix M, $\tilde{M}$, can be reconstructed. Only the largest p singular values are chosen which corresponds to the first p left-singular vectors of U; thus, $\tilde{M} \in \mathbb{R}^{n \times p}$.

For the RNN, PCA is done on all the state trajectories and fixed points of the RNN by stacking $\mathbf{x}^t$ and $\mathbf{x}^*$ as $X = [x_1^T, x_2^T, \ldots, x_n^T]$ for n cells, and then choosing $p = 2$ or 3. This approximates the dynamics of the hidden state variable of the RNN as a two or three dimensional quantity rather than the large N dimensional quantity it actually is. In practice, however, this entire procedure is implemented using the general neural network framework provided by the `python` library `pytorch`.

3 Model Implementation in `pytorch`

The procedure described in Section 2 and all RNNs are implemented numerically using the `python` neural network library `pytorch` given its robustness and versatility. This `pytorch` implementation is built with help from a `tensorflow` implementation of the same procedure done by Matthew Golub and David Sussillo [4]. Since RNNs are tailored to the results one wishes to reproduce, the following discussion will only be about the Sussillo and Barak procedure, and any necessary comments about model construction or preparation for a specific task will be discussed in their respective sections. Generally, the Sussillo and Barak procedure is divided up into the following three tasks: one, train a RNN to reproduce desired results given a specific input; two, find the hidden state fixed points $\mathbf{x}^*$ using gradient descent; three, visualize the dynamics of the hidden state using PCA in two or three dimensions. Classifying stability was purposefully excluded from implementation given its relative lack of importance for the investigation of the FitzHugh-Nagumo model. A deeper discussion on the exclusion implementing stability via the Jacobian will be included in Section 7.

After the data for the desired task that the RNN will reproduce has been collected, a RNN class is initialized with the appropriate number of hidden layers, the specific nonlinear activation functions, and the size of the hidden state variable. This is done by first

creating the RNN model through `class RNN(nn.Module)` and defining each step in the RNN cell. Because of convergence issues, five linearly connected hidden layers were used using the `nn.Linear` as a part of our RNN class. The nonlinear activation function `ReLU` was employed, but other functions such as `tanh` are also sufficient. The model can then be trained by employing a standard loss function, such as `torch.nn.MSELoss()`[2] and setting the model into train mode by `model.train()`. From here, the inputs are fed one-at-a-time to the model, the loss is calculated and propagated backwards, and the learning parameters from gradient descent are updated (`torch.optimi.Adam` is employed for learning). The model either exits after it has run through a 2000 epochs of training data or if the loss is below 0.001.

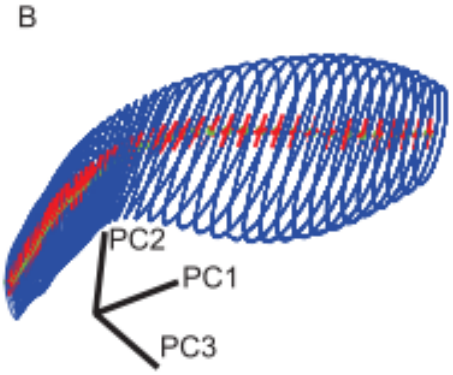
Now that the model has been trained to predict the desired output given a specified input, the fixed points of the hidden state can be found. This is employed through a fixed point class that contains the necessary features defined in Section 2.2. The gradient descent to $\mathbf{x}^*$ is implemented by first taking a hidden state of one time step, $\mathbf{x}^j$, and making it into a `pytorch` variable through `hidden(0) = torch.autograd.Variable(`$\mathbf{x}^j$`)`. From here, the gradient of $\mathbf{x}^j$ is calculated by `hidden(i).grad` and the next step in the iterative procedure defined as `hidden(i+1) = hidden(i) -` η`*hidden(i).grad`. When the value of the `hidden(i)` has not sufficiently changed, the iteration process concludes and the final value of `hidden(i)` is $\mathbf{x}^*$.

Now that all the trajectories of the desired system have been replicated using the RNN and the the fixed points of the RNN have been approximated, visualizing these dynamics in low dimensions using PCA is trivial. The python library `scikit-learn` and their implementation of PCA, `sklearn.decomposition.PCA`, is used. PCA with either two or three components is initialized using `pca = PCA(n_components = p)` and fit to the $\mathbf{X}$ matrix defined in Section 2.3 using `pca.fit(X)`. The trajectories and fixed are reduced to p dimensions using `pca.transform()` and plotted using `matplotlib.pyplot`. While this implementation works generally for most problems that are suspected of having computation fixed points mediating dynamics in the hidden state of an RNN, the implementation of the FitzHugh-Nagumo model will be discussed in further detail in the following section. The implementation will then be validated by recreating the Sine Wave Generator task discussed by Sussillo and Barak.

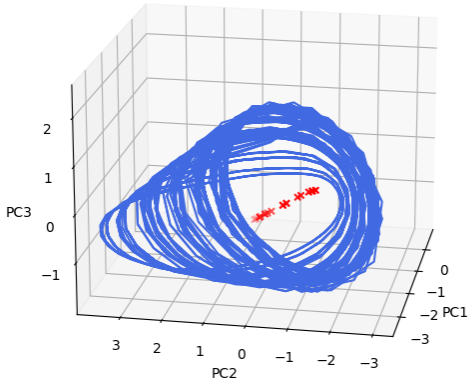
3.1 FHN Implementation

To apply the previous `pytorch` implementation of the Sussillo and Barak procedure to the FitzHugh-Nagumo model, the input and output data must first be collected. For this, data was collected by evolving Equations 6a and 6b from an initial condition of $[v, w] = [0, 0]$ using Forward-Euler with a step size of 0.3 for 100 time units given a constant input

[2]Mean Squared Error (MSE) is a standard measure of how much a predicted set of values deviates from a true set of values and is defined as `MSE`$= \frac{1}{N}\sum_{n=0}^{N-1}(x_i - \tilde{x}_i)^2$ where x_i is i^{th} true value and $\tilde{x}_i$ is the i^{th} predicted value.



(a) Sussillo and Barak



(b) Ours

Figure 5: Sussillo and Barak's found 51 fixed points in their optimization procedure for the Sine Wave Generator task but we were only able to run the optimization procedure 20 times due to computational constraints.

current I_{ext}. The I_{ext} was chosen in order to ensure that the system expressed limit cycle behavior, or oscillatory behavior in phase space, so, given $a = 0.7$, $b = 0.8$, and $\tau = 12.5$, $I_{ext} \in [0.4, 1.4]$. The RNN was trained only to reproduce the voltage, v, but struggled to replicate the dynamics, so the voltages that the RNN predicted where halved. Likewise, the constant input current to the RNN was also halved to ensure the model completed training. When the training was complete, PCA with 2 and 3 components were both used, but only trajectories approximated using PCA with 2 components will be discussed.

4 Sine Wave Generator Example

To verify our implementation of the Sussillo and Barak procedure in `pytorch`, the Sine Wave Generator task was replicated. Simply, the task is, given a constant frequency ω_j in radians, the RNN must output $\sin \omega_j t$. A hidden state dimension of 256 was employed which is similar to the 200 dimensional hidden state used by Sussillo and Barak. Using our `pytorch` implementation, we successfully reproduced a portion of the Sine Wave Generator task and their associated figure. Comparing their results, found in Figure 5a, to our results, found in Figure 5b, it is obvious to see that the procedure's implementation in `pytorch` succeeded. The hidden state, when viewed using 3 PCA components, oscillated in a circular trajectory, traced in blue, in phase space indicating that, to implement the sine wave task, the hidden state oscillated just like a sine wave. These oscillations were localized around unstable fixed points so that, when the task was initialized, the network would naturally evolve to the proper oscillatory trajectory for the hidden state to produce the correct output. Looking at

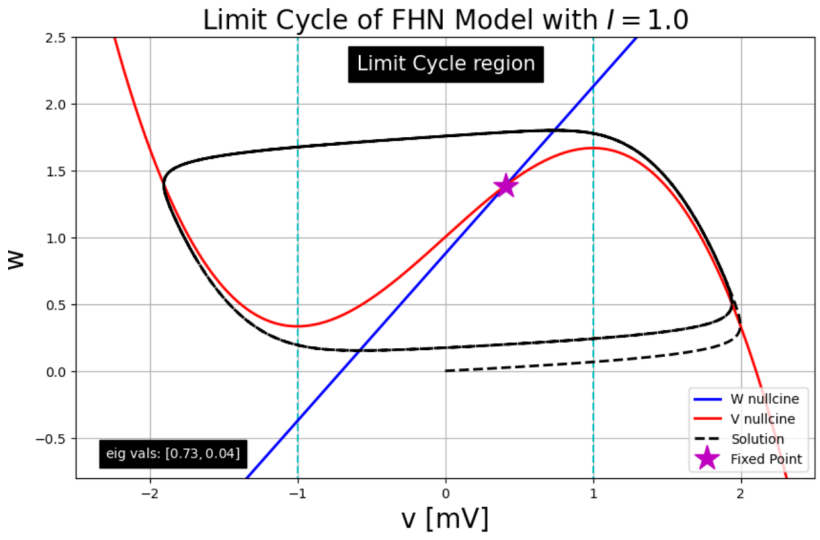


Figure 6: Example trajectory of v and w of FHN model in vw phase space with $v-$nullcline graphed in blue and $w-$nullcline graphed in red. The singular fixed point (starred in magenta) of the system represents the intersection of these fixed points and is unstable by the positive eigenvalues. The horizontal cyan lines represent the relative region where, if a fixed point lies inside that region, a limit cycle forms.

Figure 5b, our network likewise approximates all the unstable fixed points, starred in red, on a 1D manifold that pierces the center of all the oscillatory trajectories. A similar manifold can be found in Figure 5a from Sussillo and Barak.

Due to the long compute times for training thousands of epochs and finding tens of fixed points, the entire range of frequencies outlined by Sussillo and Barak was not used. Likewise, for our methods to converge, 20 random frequencies were chosen from an appropriate range when they were fed as a constant input to the trained RNN and fixed point finder algorithm. Furthermore, stability was not as thoroughly explored by our method, but the capability for stability analysis can be found in our RNN implementation, though it was not employed due to compute and time constraints. Since our procedure has succeeded, the procedure was then applied to the FitzHugh-Nagumo model for neuronal excitation, a model whose fixed points can be analytically calculated.

5 How a RNN Replicates the FitzHugh-Nagumo Model

A major component of Sussillo and Barak's 2013 paper was analyzing computer science problems in order to test their procedure and hypothesis. These problems and results, however, gave no satisfying framework for discussing the relevance of these fixed points to the

problems themselves. For all but one example, the 3-Bit Flip-Flop task, were the fixed points of the RNN's hidden state discussed purely by themselves without reference to the problem or task at hand. The fixed points were simply a product of some computational aspect of the RNN and its reproduction of the associated task; thus, a major branch of the discussion of the method was left barren. This branch was having a RNN reproduce results from a dynamical system which has analytic fixed points, a natural extension of considering RNNs as dynamical system - having RNNs act like dynamical systems. To give this aspect of RNNs as dynamical systems a better treatment, the FitzHugh-Nagumo model was investigated in order to answer whether a connection exists between fixed points of the RNN and fixed points of dynamical systems and if a connection exits, what that entails.

For the reproducing the dynamics of the FitzHugh-Nagumo model using a RNN, two specifics investigations were undertaken. First, the limit cycle behavior of the FHN model will be reproduced by an RNN solely from a constant I_{ext} input and its fixed point analyzed. Secondly, the analytic fixed points of the FHN model will be compared to the hidden state fixed points of the RNN. To complete the second task, a novel extension of Sussillo and Barak's procedure will be introduced in Section 5.2. How the RNN was trained and specific considerations for implementing the Sussillo and Barak procedure for the FHN model can be found in Section 3.1; however, before giving the results of the Sussillo and Barak procedure for the FHN model, the analytic fixed points to the FHN model will be discussed first.

5.1 Analytic Solutions to FitzHugh-Nagumo Model Fixed Points

In order to find the analytic fixed points to the FHN model, dubbed v^* and w^*, the steps outlined in Section 1.2 is employed. The nullclines of Equations 6a and 6b, the $v-$nullcline and $w-$nullcline, respectively, are

$$0 = v^* - \frac{(v^*)^3}{3} - w^* + I_{ext}, \tag{11a}$$

$$0 = v^* + 0.7 - 0.8w^* \tag{11b}$$

for $a = 0.7$ and $b = 0.8$. The intersection(s) of these nullclines corresponds to the fixed point(s) of the system, but, for the parameters specified, there is only one fixed point. The fixed points are found to be

$$v^* = \frac{\sqrt[3]{2}\left[24I_{ext} - 21 + \sqrt{576I_{ext}^2 - 1008I_{ext} + 445}\right]^{2/3} - 2}{2 \times 2^{2/3}\sqrt[3]{24I_{ext} - 21 + \sqrt{576I_{ext}^2 - 1008I_{ext} + 445}}}, \tag{12a}$$

$$w^* = \frac{v^* + 0.7}{0.8}. \tag{12b}$$

v^* corresponds to the only real solution to the cubic $v-$nullcline and is dependent on I_{ext}. The Jacobian of the system is found to be

$$J(v^*) = \begin{bmatrix} 1 - (v^*)^2 & -1 \\ \frac{1}{12.5} & -\frac{0.8}{12.5} \end{bmatrix}. \tag{13}$$

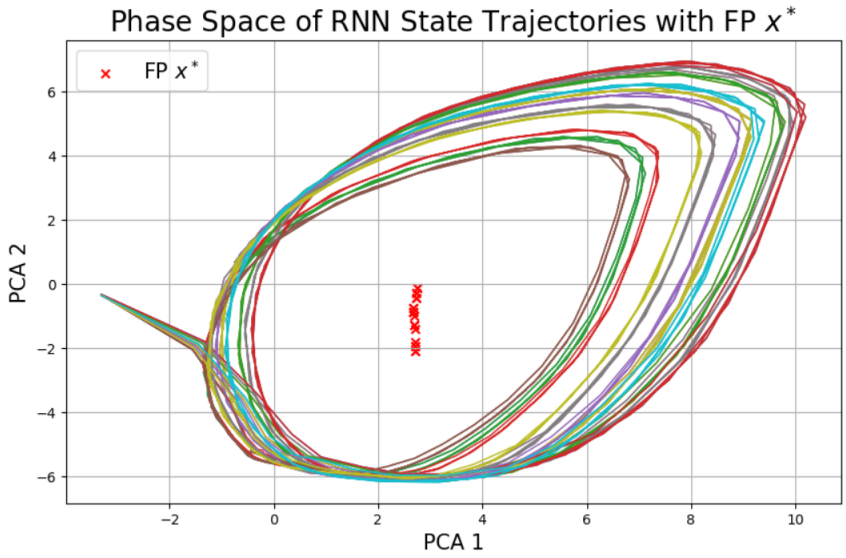


Figure 7: The 14 uniques fixed points of the hidden state are displayed as red crosses and are roughly centered around the corresponding evolutions of the hidden state variable. As I_{ext}, the trajectories become more constricted in PCA space as the hidden variable must oscillate faster to match the dynamics of the FHN.

An example solution of the system in phase space with $I_{ext} = 1.0$ can be found in Figure 6 which also displays the intersection of the two nullclines and the associated limit cycle behavior[3]. Approximately, between $0.4 < I_{ext} < 1.5$, a unstable limit cycle occurs in phase space, so all subsequent values of I_{ext} will be in that range and all fixed points will be unstable. Now that expressions for the analytic fixed points as a function of I_{ext} have been found, the fixed points of the RNN can be shown.

5.2 RNN Hidden State Fixed Points

Following the Sussillo and Barak procedure outlined in Section 3, 14 unique fixed points of the RNN's hidden state were found. These $\mathbf{x}^*$ corresponded to inputs of I^*_{ext} ranging from 0.65 to 1.35, values of which are consistent for the desired limit cycle behavior in the FHN model. The trajectories associated with the 14 different fixed points can be found in Figure 7 reduced to only the two largest contributing PCA components. The multicolored trajectories represent the evolution of the hidden state variable $\mathbf{x}$ as time evolve given a specific I_{ext}. These trajectories constrict in PCA space as I_{ext} is increased because the hidden state variable must output the higher frequency spiking of the FHN model. Like the Sine Wave Generator task, the fixed points of the hidden state variable lie along a 1D manifold in PCA space. These fixed points, however, only vary an appreciable amount with respect to PCA component 2 while PCA component 1 is relatively stationary at around a value of 2.6. These

[3]A more comprehensive discussion of stability and bifurcation of the FHN model can be found in [1].

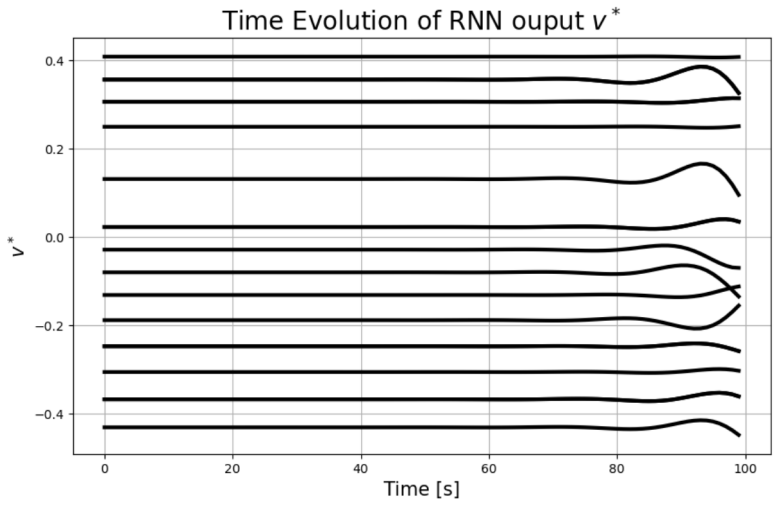


Figure 8: Using the 14 unique fixed points as initilizations to the hidden state of the RNN, corresponding approximations for the FHN fixed point v^* can be obtained. The validity of $\tilde{v}^*$ can be seen graphically as the majority of the time evolution, the value stays constant. Each line represents unique value of $\tilde{v}^*$ from x^*.

fixed points, however, are in PCA space, i.e., the variables aren't representative of the actual fixed points of the FHN model which are in $v - w$ space. Because of this, a method must be developed in order to translate $\mathbf{x}^*$ from PCA/hidden state space to v space.

To find out how to translate the hiddens state fixed points $\mathbf{x}^*$ to $\tilde{v}^*$, the translation of the hidden state's fixed points into v space, we will use the definition of a fixed point and the structure of a RNN. Given a recurrence relationship of the a RNN $F(\mathbf{x})$, the fixed point of the hidden state variable of the RNN is defined as

$$\mathbf{x}^* : \dot{\mathbf{x}} = F(\mathbf{x}) = 0. \tag{14}$$

In other words, if we pass the numerically approximated fixed point $\mathbf{x}^*$ back into the hidden state variable and evolve the system with the associated constant input I^*_{ext}, $\mathbf{x}$ should not change. Using this, fact, the output of the RNN v, should also not change because the only two factors in an RNN that change between cells is the input, which we have held constant by design, and the hidden state variable, which should not change because the system is at a fixed point. Therefore, the output of the system v should also be an approximation of v^* given by the RNN's reproduction of the FHN model. To translate $\mathbf{x}^*$ to $\tilde{v}^*$, the RNN is intialized with a specific $\mathbf{x}^*$ and associated I^*_{ext} and evolved in time. This novel revelation can be seen graphically in Figure 8 which plots the associated $\tilde{v}^*$ to the 14 different $\mathbf{x}^*$ found. Between $0 \leq t \leq 80$, the value of the 14 different $\tilde{v}^*$ does not change appreciable indicating that $\tilde{v}^*$ is an approximation of the FHN v^*. After $t = 80$, however, the value of $\tilde{v}^*$ changes into an oscillatory function instead of constant. This is due to the numerical approximation of $\mathbf{x}^*$ which is not an exact value. Because $\mathbf{x}^*$ is not exact, the error between the true fixed

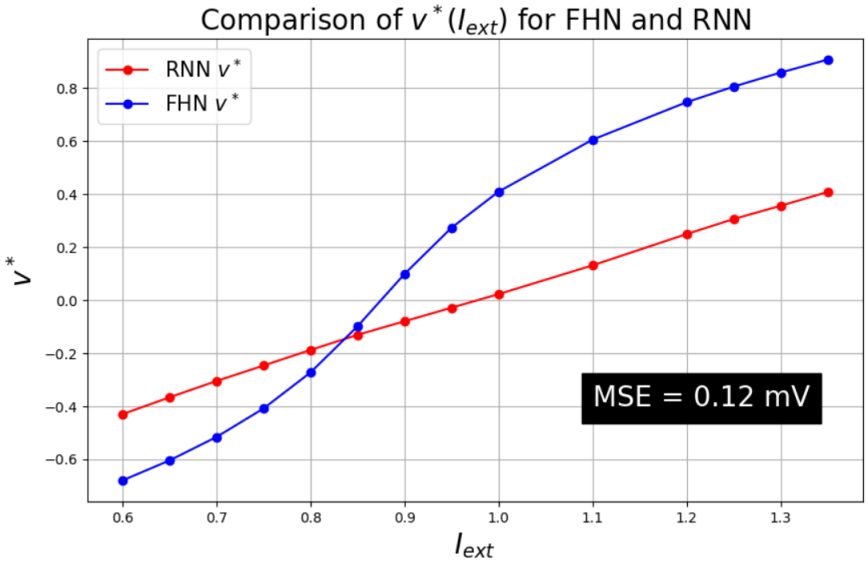


Figure 9: The RNN approximates the FHN fixed point, v^*, as a linear relation with respect to the input, I_{ext}, when actuality, the relationship is cubic. Because of this, the RNN's approximation in red either undershoots or overshoots the actual v^* in blue; thus, the error is about 0.12. The linear relationship, however, can serve as a rough approximation to the actual fixed point.

point and approximated one grows exponentially because the fixed points are unstable. The values of $\tilde{v}^*$ then approaches the limit cycle which defines the behavior of the system. Now that a translation is available for $\mathbf{x}^*$, the fixed points of the RNN can be compared to the fixed points of the FHN model.

5.3 RNN Fixed Points vs. FitzHugh-Nagumo Fixed Points

Using the 14 unique fixed points to the RNN's hidden state and the translation procedure introduced before, the associated 14 $\tilde{v}^*$ were found. From the I^*_{ext} for each of the 14 $\mathbf{x}^*$, the 14 correct fixed points of the FHN model, v^*, found using the explicit solutions to the $v-$ and $w-$ nullclines introduced in Equations 12a and 12b, were computed. The relationship between v^*, the v component of the fixed point tuple for the FHN model, and $\tilde{v}^*$, the translated fixed point of the RNN hidden state, as a function of I_{ext} can be graphically seen in Figure 9. The dotted red line represents $\tilde{v}^*$ and the blue line represents the true v^* from the FHN model.

6 Discussion of Results

From Figure 9, it is plainly apparent that the translated fixed point of the RNN does not exactly match the value of the true v^* nor the shape of the relationship between I_{ext} and v^*; it does, however, match the general trend and approximate value of v^* indicating that the Sussillo and Barak procedure can be naturally extended further to approximate fixed points of dynamical systems without having to do much more work. This assertion is supported by the general trend of the relationship between v^*, in blue, and $\tilde{v}^*$, in red. Both trajectories, while different in shape, trace out the same general relationship between I_{ext} and v^*: as I_{ext} is increased from 0.6 to 1.35, both v^* and $\tilde{v}^*$ increase from about -0.65 and -0.4 to 0.85 and 0.4, respectively. The major difference between the two lines is that v^* is defined as the real root of a cubic equation while $\tilde{v}^*$ is approximated by the RNN as linear. The difference in the order of equation (cubic versus linear) results in an error of about 0.12.

This discrepancy in order might indicate that the RNN utilizes a linear relationship between constant input and translated fixed point v^* purely because of computational ease. For the RNN, a linear relationship might be the least computationally expensive relationship it can use to approximate the behavior of the system because many of its components rely on linear operations. This hypothesis is likely false because of the repeated application of the nonlinear function `ReLU`. Likewise (and more likely), it may be due to something inherent to the problem of predicting an oscillatory variable given a constant input. This linear manifold that defines the relationship between input and predicted output (or translated output) was also noticed by Sussillo and Barak in their Sine Wave Generator example. Their example, has no inherent fixed points to the problem as it does not seek to replicate the dynamics of a dynamical system. This introduces the largest critique of our paper's work. Because only the FHN model has been studied in such a fashion using the Sussillo and Barak method, what little insight has been gleamed from our novel extension onto approximations of fixed points of RNNs and dynamical systems reveals more questions that must be answered.

In literature, work has been done to use the Sussillo and Barak procedure to understand how RNNs implement their computation, such as in Maheswaranathan *et al.* [8], but little to no research has been undertaken to apply the Sussillo and Barak procedure to the study of dynamical systems and the approximation of fixed points. The authors of that paper use a RNN for sentiment analysis of Yelp reviews and show that the linearized dynamics around fixed points can be used to approximate the recurrence relationship between RNN cells. They do this by finding the fixed points of the system and approximate the nonlinear recurrence relationship between cells using the linearized dynamics around such fixed points using the Jacobian at $\mathbf{x}^*$. More importantly, though, they remark that the fixed points lie upon a 1D manifold irregardless of network architecture. This implies that, regardless of task, e.g., sentiment analysis or regression, the structure inherent to RNNs produces low dimensional line attractor/repulsor dynamics. It is known, however, that for other points of interest in phase space (dubbed "slow" points) mediate other computational tasks and are 2D manifolds [12]. Why RNNs produce 1D or 2D manifolds in their hidden state fixed points is not known. Treating RNNs as dynamics systems and even using fixed points of dynamical systems to generate RNNs, however, has had significant treatment in the study

of both dynamical systems and RNNs.

Hopfield's 1982 paper [6], *Neural networks and physical systems with emergent collective computational abilities*, famously uses fixed points in recurrently connected computational neurons to recover corrupted messages which are encoded as fixed points in memory. Likewise, work done by Pollock and Jazayeri [10] utilizes manifold representations of specific tasks to recreate a RNN that reproduces the desired dynamics of said task. Both of these works represents the intersection of dynamical systems and neural networks that uses fixed points to generate NNs that produce desired dynamics. This style of research has also been used by Trischler and D'Eleuterio [13] to directly reproduce nonlinear dynamics in RNNs using vector fields, the realization of the equations that govern nonlinear dynamical systems. Of importance is the ability to reproduce Van der Pol oscillators (a very close relative to the FitzHugh-Nagumo model) but does so using a training procedure which utilizes the vector field of a system and not the trajectories themselves. Again, little to no research has been done on using RNNs, which have been proven to be able to reproduce the dynamics of any dynamical system [3], to approximate fixed points of nonlinear dynamical systems and the behavior of those fixed points.

7 Further Work

The amount of further work that can be done to explore approximating fixed points of nonlinear dynamical systems using trained RNNs and investigating how RNNs implement complex dynamics through the fixed points of their hidden state variable is large. To improve just the results provided in this paper, a finer discretization of training data for v and I_{ext} should be employed to ensure that the RNN exactly captures the dynamics between I_{ext} and the temporal evolution of v. Likewise, the fixed points of the RNN hidden state should be computed for a finer window of I_{ext}. This should also be altered and improved so that a specific I_{ext} can be used for the fixed point approximation scheme outlined in Section 3 rather than a random input current which allowed the scheme to converge. To better understand the process of replicating the FHN model, an investigation into how a model can be trained using unaltered training data rather than the halved training data that was required to prevent problems with gradients (vanishing or exploding gradients). Furthermore, to better understand the behavior of the fixed points, the linear stability analysis (LSA) of the code should be enabled and investigated in comparison to the true LSA classification of the systems fixed points.

More broadly, more nonlinear dynamical systems that have analytic fixed points should be reproduced using RNNs and their fixed points analyzed to see if the extension of the Sussillo and Barak procedure outlined in this applies to other systems. Otherwise, these results might become invalidated. Furthermore, an investigation into what causes specific manifold behavior of the RNN hidden state variable and approximated system fixed points must be undertaken. In another direction, most research into hidden state fixed points or constructing RNNs from fixed points avoid bifurcations or similar qualitative changes in

system behavior (except [7], etc.). Investigating whether RNNs are able to replicate bifurcations, such as in the FHN model, and the associated fixed points of the RNN would reveal more information about what exactly RNNs replicate with respects the behavior of the dynamical systems they reproduce. Nonetheless, both the extension of the Sussillo and Barak procedure outlined in this paper to approximate fixed points of dynamical systems and the fixed points of the RNN hidden state present numerous avenues of further research into how RNNs implement their computation.

8 Conclusion

In this paper, we have successfully adapted a procedure for understanding how RNNs implement their computation through a procedure finding fixed points of a RNNs hidden state proposed by Sussillo and Barak into the `python` library `pytorch`. This procedure is then applied to the FitzHugh-Nagumo model for neuronal excitation to analyze how a RNN implements the limit cycle behavior of the 2 dimensional nonlinear dynamical system. The Sussillo and Barak procedure revealed 14 unique hidden state fixed points that were then translated through into the v variable of the FitzHugh-Nagumo model and compared with the analytic fixed points of the system. Through the translation and comparison, it has been revealed that the RNN can be used to approximate the analytic fixed points of the FitzHugh-Nagumo model, though the relationship between input parameter, I_{ext}, to fixed point is linear while the true relationship is cubic. Further work must therefore be done to better understand if such an extension of conceiving of RNNs as dynamical systems and analyzing their fixed points in such a manner is correct or incorrect.

References

[1] Em, P. V. L. A study of fixed points and hopf bifurcation of fitzhugh-nagumo model. *Can Tho University Journal of Science 54*, 2 (2018), 112–121.

[2] FitzHugh, R. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal 1*, 6 (1961), 445–466.

[3] Funahashi, K.-i., and Nakamura, Y. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks 6*, 6 (1993), 801–806.

[4] Golub, M. D., and Sussillo, D. Fixedpointfinder: A tensorflow toolbox for identifying and characterizing fixed points in recurrent neural networks. *Journal of Open Source Software 3*, 31 (2018), 1003.

[5] Hodgkin, A. L., and Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology 117*, 4 (1952), 500.

[6] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences 79*, 8 (1982), 2554–2558.

[7] Lim, S. H., Theo Giorgini, L., Moon, W., and Wettlaufer, J. S. Predicting critical transitions in multiscale dynamical systems using reservoir computing. *Chaos: An Interdisciplinary Journal of Nonlinear Science 30*, 12 (2020), 123126.

[8] Maheswaranathan, N., Williams, A., Golub, M., Ganguli, S., and Sussillo, D. Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics. *Advances in neural information processing systems 32* (2019).

[9] Nagumo, J., Arimoto, S., and Yoshizawa, S. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE 50*, 10 (1962), 2061–2070.

[10] Pollock, E., and Jazayeri, M. Engineering recurrent neural networks from task-relevant manifolds and dynamics. *PLoS computational biology 16*, 8 (2020), e1008128.

[11] Strogatz, S. H. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[12] Sussillo, D., and Barak, O. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation 25*, 3 (2013), 626–649.

[13] Trischler, A. P., and D'Eleuterio, G. M. Synthesis of recurrent neural networks for dynamical system simulation. *Neural Networks 80* (2016), 67–78.

A Code Appendix for Graphs and Analysis

Note, the code that has to do with the recurrent neural network training, fixed point finding algorithm, and plotting utilities can be found in an attached `zip` folder to this project. Only the `Jupyter` notebook that was used to compare the fixed points for both the RNN and FHN model is attached below.

In [ ]:
```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import pandas as pd

from numpy import linalg
from scipy import integrate
from matplotlib import animation, rc

plt.rcParams["figure.figsize"] = (10,6)
```

In [ ]:
```
a = 0.7
b = 0.8
tau = 12.5

def fhn(t,x,a,b,tau,I): ## FHN RHS
    v = x[0] - (x[0]**3)/3 - x[1] + I
    w = (x[0] + a - b*x[1])/tau
    return [v,w]

def fhn_fixed_point(I): ##calculate fixed point and jacobian
    ## assume a = 0.7, b = 0.8, tau = 12.5
    tau = 12.5
    a = 0.7
    b = 0.8
    #inter = (np.sqrt(576*I**2 - 1008*I + 445) + 24*I - 21)**(1/3)
    #v_star = (inter/(2*(2**(1/3)))) - 1/(inter*(2**(2/3)))
    det = (np.sqrt(576*I**2 - 1008*I + 445) + 24*I - 21)
    v_star = ((2**(1/3))*(det**(2/3)) - 2)/(2*(2**(2/3))*(det)**(1/3)) ##simplified
    w_star = (v_star + a)/b

    jacobian = np.array(([1 - v_star**2, -1], [1/tau, -b/tau]))
    return v_star, w_star, jacobian

T_max = 100
dt = 0.1
tspan = np.arange(0,T_max+dt,dt)
vspan = np.linspace(-3,3,len(tspan))

dI = 0.01
I_list = np.arange(0.15, 1.5+dI, dI)
fhn_sol = np.zeros((len(I_list),len(tspan),2))
W_nullcine = np.zeros((len(I_list),len(tspan)))
V_nullcine = np.zeros((len(I_list),len(tspan)))

fhn_0 = [0,0]

for i in range(len(I_list)):
    sol = sp.integrate.solve_ivp(fhn, [0,T_max], y0 = fhn_0, args = (a,b,tau,I_list[i]),
                                 t_eval = tspan, method = 'RK45')
    fhn_sol[i,:,0] = sol.y[0,:]
    fhn_sol[i,:,1] = sol.y[1,:]
    W_nullcine[i,:] = b*vspan - a
    V_nullcine[i,:] = vspan - ((vspan)**3)/3 + I_list[i]

fp_fhn = np.zeros((3, len(I_list)))

for i, I in enumerate(I_list):
    fp_fhn [:,i] = I, fhn_fixed_point(I)[0], fhn_fixed_point(I)[1]
```

In [ ]:
```
# ### Depricated
# data = '0.75 2.72344875 -0.61066362   0.09077326 0.7  2.74056482      -0.45540864     0.12338276 1.1  2.70886462      -1.5
# clean_data = data.replace('\t',' ').split()

# rnn_fixed_points_old = np.zeros((int(len(clean_data)/4),4))
# for j in range(4):
#     rnn_fixed_points_old[:,j] = [float(x) for i, x in enumerate(clean_data) if i%4 == j]

# idx = np.argsort(rnn_fixed_points_old[:,0])
# rnn_fixed_points_old[:,:] = rnn_fixed_points_old[idx,:]
```

In [ ]:
```
rnn_pca_2 = pd.read_csv('output_1.csv').to_numpy()
rnn_pca_3 = pd.read_csv('output_2.csv').to_numpy()


rnn_pca_2[:,:] = rnn_pca_2[np.argsort(rnn_pca_2[:,0]),:]
rnn_pca_3[:,:] = rnn_pca_3[np.argsort(rnn_pca_3[:,0]),:]

scale_v = 2

rnn_2_wstar = (scale_v*rnn_pca_2[:,3] + a)/b ## uses explicit w-nullcine solution
rnn_3_wstar = (scale_v*rnn_pca_3[:,4] + a)/b

fp_compare = np.zeros((rnn_pca_2.shape[0],6)) ## [Iext, v* RNN, w* RNN, v* FHN, w* FHN, v* diff]
fp_compare[:,0] = rnn_pca_2[:,0]
fp_compare[:,1] = scale_v*rnn_pca_2[:,3]
fp_compare[:,2] = rnn_2_wstar
for i, Iext in enumerate(fp_compare[:,0]):
    fp_compare[i,3], fp_compare[i,4], _ = fhn_fixed_point(Iext)
fp_compare[:,5] = fp_compare[:,1] - fp_compare[:,3]
fp_compare_no_dup = (pd.DataFrame(fp_compare).drop_duplicates()).to_numpy()
```

In [ ]:
```
for i in range(rnn_pca_2.shape[0]):
    plt.plot(2*rnn_pca_2[i,3:], 'k-', lw = 3)
plt.xlabel('Time [s]', fontsize = 15)
plt.ylabel('$v^*$', fontsize = 15)
plt.title('Time Evolution of RNN ouput $v^*$', fontsize = 20)
plt.grid()
plt.show()
```

In [ ]:
```
time_length = 100
iteration = 20

pc_fixed_point = np.loadtxt("PCA2_fixed_points.csv",delimiter=" ", dtype=float)
pc_trajectory = np.loadtxt("PCA2_trajectory.csv",delimiter=" ", dtype=float)

for i in range(iteration):
    plt.plot(pc_trajectory[0, i * time_length:(i + 1) * time_length],
             pc_trajectory[1, i * time_length:(i + 1) * time_length], linewidth=1.1)
plt.scatter(pc_fixed_point[0], pc_fixed_point[1], color='red', marker='x', label = 'FP $x^*$')
plt.xlabel('PCA 1', fontsize = 15)
plt.ylabel('PCA 2', fontsize = 15)
plt.title('Phase Space of RNN State Trajectories with FP $x^*$', fontsize = 20)
plt.grid()
plt.legend(fontsize = 15)
plt.show()
```

In [ ]:
```
pc_fixed_point = np.loadtxt("PCA3_fixed_points.csv",delimiter=" ", dtype=float)
pc_trajectory = np.loadtxt("PCA3_trajectory.csv",delimiter=" ", dtype=float)

fig, ax = plt.subplots(1, subplot_kw={"projection": "3d"}, figsize = (10,10))
for i in range(iteration):
    ax.plot(pc_trajectory[0, i * time_length:(i + 1) * time_length],
            pc_trajectory[1, i * time_length:(i + 1) * time_length],
            pc_trajectory[2, i * time_length:(i + 1) * time_length], linewidth=1.1)
ax.scatter(pc_fixed_point[0], pc_fixed_point[1], pc_fixed_point[2], color='red', marker='x', label = '$x^*$ FP')
ax.set_xlabel('PCA 1', fontsize = 15)
ax.set_ylabel('PCA 2', fontsize = 15)
ax.set_zlabel('PAC 3', fontsize = 15)
ax.view_init(30,45)
plt.title('Phase Space of RNN State Trajectories with FP $x^*$', fontsize = 20)
plt.legend(loc = 'lower center', fontsize = 15)
plt.show()
```

In [ ]:
```
error = np.round(np.sum(fp_compare_no_dup[:,5]**2)/len(fp_compare_no_dup[:,5]), decimals = 2)

plt.plot(fp_compare_no_dup[:,0],fp_compare_no_dup[:,1], 'ro-', label = 'RNN $v^*$')
plt.plot(fp_compare_no_dup[:,0],fp_compare_no_dup[:,3], 'bo-', label = 'FHN $v^*$')
plt.xlabel('$I_{ext}$', fontsize = 20)
plt.ylabel('$v^*$', fontsize = 20)
plt.title('Comparison of $v^*(I_{ext})$ for FHN and RNN', fontsize = 20)
plt.text(1.1,-0.4, f'MSE = {error} mV', c = 'w', backgroundcolor = 'k', fontsize = 20)
plt.grid()
plt.legend(fontsize = 15)
plt.show()
```

In [ ]:
```
fig, ax = plt.subplots(1, 3, subplot_kw={"projection": "3d"}, figsize = (15,15))
for i in range(3):
    ax[i].plot(fp_fhn[1,:],fp_fhn[2,:],fp_fhn[0,:], label ='FHN (Full I range)')
    ax[i].scatter(fp_compare_no_dup[:,3],fp_compare_no_dup[:,4],fp_compare_no_dup[:,0], c = 'k', label = 'FHN')
    ax[i].scatter(fp_compare_no_dup[:,1],fp_compare_no_dup[:,2],fp_compare_no_dup[:,0], c = 'r', label = 'RNN')
    ax[i].set_xlabel('v [mV]', fontsize = 10)
    ax[i].set_xticks(np.arange(-1,1+0.5,0.5))
    ax[i].set_ylabel('w', fontsize = 10)
    if i == 0 or 2:
        ax[i].set_zlabel('$I_{ext}$ [mA]', fontsize = 10)

ax[1].set_zticks([ ])
ax[2].set_yticks([ ])
ax[2].set_ylabel(' ')
ax[1].view_init(90,-90)
ax[2].view_init(0,-90)
ax[1].set_title('Fixed Points $(v*,w*)$ for RNN and FHN', fontsize = 20)
plt.legend(loc = 'upper right', fontsize = 10)
plt.show()
```

In [ ]:
```
C = 85 ## current (I_ext) indexer

v_star, w_star, jacobian = fhn_fixed_point(I_list[C])

eig_l, eig_v = np.linalg.eig(jacobian)

if np.imag(eig_l[0]) != 0 and np.real(eig_l[0]) < 0:
    title = 'Stable Spiral'
else:
    title = 'Limit Cycle'

fig, ax = plt.subplots(1)
ax.vlines(-1, -1, 2.5, color = 'c', linestyles = '--')
ax.vlines(1, -1, 2.5, color = 'c', linestyles = '--')
ax.plot(W_nullcine[C,:], vspan, 'b',lw = 2, label = 'W nullcine')
ax.plot(vspan,V_nullcine[C,:], 'r',lw = 2, label = 'V nullcine')
ax.plot(fhn_sol[C,:,0], fhn_sol[C,:,1], 'k--',lw = 2, label = 'Solution')
ax.plot(v_star,w_star, 'm*', ms = 20, label = 'Fixed Point')
#ax.plot(fp_compare_no_dup[:,1], fp_compare_no_dup[:,2], 'go', lw = 3, label = 'RNN $(v^*,w^*)$')
ax.set_xlabel('v [mV]', fontsize = 20)
ax.set_ylabel('w', fontsize = 20)
ax.set_xlim([-2.5,2.5])
ax.set_ylim([-0.8,2.5])
ax.set_title(f'{title} of FHN Model with $I = {np.round(I_list[C], decimals = 2)}$', fontsize = 20)
ax.text(-2.3, -0.65, f'eig vals: $[{np.round(eig_l[0], decimals = 2)}, {np.round(eig_l[1], decimals = 2)}]$'
         , c = 'w', backgroundcolor = 'k', fontsize = 10)
ax.text(-0.6, 2.25, 'Limit Cycle region',
        c = 'w', backgroundcolor = 'k', fontsize = 15)
ax.legend(loc = 'lower right')
ax.grid()
plt.show()
```

In [ ]:
```
fig, ax = plt.subplots(1,2,sharey=True)
ax[0].plot(tspan, fhn_sol[C,:,0], 'k-', lw = 3)
ax[1].plot(tspan, fhn_sol[C,:,1], 'k-', lw = 3)
fig.supxlabel('t [s]')
ax[0].set_title('v [mV]')
ax[1].set_title('w')
fig.suptitle('FHN $v$ and $w$ with $I_{ext} = 1.0$', fontsize = 17)
ax[0].grid()
ax[1].grid()
plt.show()
```

In [ ]:
```
v_star, w_star, jacobian = fhn_fixed_point(I_list[0])

eig_l, eig_v = np.linalg.eig(jacobian)

if np.imag(eig_l[0]) != 0 and np.real(eig_l[0]) < 0:
    title = 'Stable Spiral'
else:
    title = 'Limit Cycle'


rnn_fp_v_ani, rnn_fp_w_ani  = [], []

fig, ax = plt.subplots(1)
ax.vlines(-1, -1, 2.5, color = 'c', linestyles = '--')
ax.vlines(1, -1, 2.5, color = 'c', linestyles = '--')
ax.plot(W_nullcine[0,:], vspan, 'b',lw = 2, label = 'W nullcine')
vnull, = ax.plot(vspan,V_nullcine[0,:], 'r',lw = 2, label = 'V nullcine')
sol, = ax.plot([], [], 'k--',lw = 2, label = 'Solution')
fhn_fp, = ax.plot([],[], 'm*', ms = 20, label = 'FHN FP')
rnn_fp, = ax.plot([], [], 'g*', ms = 20, label = 'RNN FP')
ax.set_xlabel('v [mV]', fontsize = 20)
ax.set_ylabel('w', fontsize = 20)
ax.set_xlim([-2.5,2.5])
ax.set_ylim([-0.8,2.5])
eig_val_txt = ax.text(-2.3,-0.65,f'eig vals: $[{np.round(eig_l[0], decimals = 2)},{np.round(eig_l[1],decimals = 2)}]$'
        , c = 'w', backgroundcolor = 'k', fontsize = 10)
ax.text(-0.6, 2.25, 'Limit Cycle region',
        c = 'w', backgroundcolor = 'k', fontsize = 15)
ax.legend(loc = 'lower right')
ax.grid()

def update(i):
    v_star, w_star, jacobian = fhn_fixed_point(I_list[i])

    eig_l, eig_v = np.linalg.eig(jacobian)

    if np.imag(eig_l[0]) != 0 and np.real(eig_l[0]) < 0:
        title = 'Stable Spiral'
    else:
        title = 'Limit Cycle'

    vnull.set_data(vspan, V_nullcine[i,:])
    sol.set_data(fhn_sol[i,:,0], fhn_sol[i,:,1])
    fhn_fp.set_data(v_star,w_star)

    if len(rnn_fp_v_ani)!= 14 and np.round(I_list[i], decimals = 2) == np.round(fp_compare_no_dup[len(rnn_fp_v_ani),0],
                                                                                decimals = 2):
        rnn_fp_v_ani.append(fp_compare_no_dup[len(rnn_fp_v_ani),1])
        rnn_fp_w_ani.append(fp_compare_no_dup[len(rnn_fp_w_ani),2])
        rnn_fp.set_data(rnn_fp_v_ani,rnn_fp_w_ani)

    elif len(rnn_fp_v_ani)== 14 and np.round(I_list[i], decimals = 2) == np.round(fp_compare_no_dup[len(rnn_fp_v_ani)-1,0],
                                                                                  decimals = 2):
        rnn_fp_v_ani.append(fp_compare_no_dup[len(rnn_fp_v_ani)-1,1])
        rnn_fp_w_ani.append(fp_compare_no_dup[len(rnn_fp_w_ani)-1,2])
        rnn_fp.set_data(rnn_fp_v_ani,rnn_fp_w_ani)

    eig_val_txt.set_text(f'eig vals: $[{np.round(eig_l[0], decimals = 2)}, {np.round(eig_l[1], decimals = 2)}]$')

    ax.set_title(f'{title} of FHN Model with $I = {np.round(I_list[i], decimals = 2)}$',loc = 'left', fontsize = 20)
    return vnull,sol,fhn_fp,rnn_fp,eig_val_txt

ani = animation.FuncAnimation(fig, update, frames = range(len(I_list)), interval = 1, blit = True)

writergif = animation.PillowWriter(fps = 24)
ani.save('FHN_fp.gif', writer = writergif)
```