Explaining RNN computations through hidden state dynamics

Abstract

Recurrent neural networks (RNNs) work in a brain-inspired way by repeatedly updating an internal state using feedback and an input signal in non-linear way. This enables them, with the right synaptic strengths, to perform dynamical tasks like sequence prediction and text-understanding. However, because the internal state is high-dimensional and the updating is nonlinear and convoluted, the mechanisms underlying their computations are very hard to decipher. One way to tackle this problem, proposed by Sussillo and Barak in [1], is to study the linearized dynamics of the system around points of very slow movement. This can reveal crucial aspects of how RNNs implement their computations. First, we briefly describe their methodology and reproduce an example application from their paper. Second, we extend a little bit their analysis and play around with their example to gather some intuition. Third, we use dynamics to examine how RNNs perform the general task of implementing Finite State Automaton. We end by discussing our results and offering general observations. See `https://imgur.com/a/doNAGkM` for animations complementing the figures.

1 RNNs and dynamical systems

Throughout this work we will use a variation of traditional RNNs : Gated Recurrent Units (GRU). See Definition 1 for more details. Given a fixed input $\mathbf{x}_0$, a GRU gives rise to a discrete time autonomous dynamical system $\mathbf{h} \mapsto g(\mathbf{h})$ where :

$$\begin{aligned} g(\mathbf{h}) =& (1 - \sigma(W_z\mathbf{x}_0 + U_z\mathbf{h} + b_z)) \odot \mathbf{h} + \\ & \sigma(W_z\mathbf{x}_0 + U_z\mathbf{h} + b_z) \tanh(W_h\mathbf{x}_0 + U_h(\sigma(W_r\mathbf{x}_0 + U_r\mathbf{h} + b_r) \odot \mathbf{h}) + b_h) \end{aligned} \tag{1}$$

We only write the explicit expression to get a sense of the difficulty one might have studying this dynamical system from the ground up. Because we cannot find the fixed points of the map g analytically, we have to resort to numerical methods : we minimize the "speed" of the dynamics given by the function

$$q(\mathbf{x}) = \|g(\mathbf{x}) - \mathbf{x}\|^2$$

It is clear that $q(\mathbf{x}) = 0$ iff $\mathbf{x}$ is a fixed point. However because we are using numerical methods and that this function is not convex, we are most likely to find points where $q(\mathbf{x})$ is small but not 0. To see why this is not a big problem, suppose $g(\mathbf{x}^*) = \mathbf{x}^* + \mathbf{v}$. Taylor-expanding g we get

$$g(\mathbf{x}^* + \mathbf{y}) = g(\mathbf{x}^*) + Dg(\mathbf{x}^*)\mathbf{y} + \frac{1}{2}\mathbf{y}^\top D^2 g(\mathbf{x}^*)\mathbf{y} + \ldots = \mathbf{x}^* + \mathbf{v} + Dg(\mathbf{x}^*)\mathbf{y} + \frac{1}{2}\mathbf{y}^\top D^2 g(\mathbf{x}^*)\mathbf{y} + \ldots$$

Therefore, ignoring higher order terms, the dynamics are well approximated by the affine transformation $\mathbf{y} \mapsto Dg(\mathbf{x}^*)\mathbf{y} + \mathbf{v}$. However, if $\|\mathbf{v}\|$ is dominated by $\|Dg(\mathbf{x}^*)\mathbf{y}\|$ (which happens when $q(\mathbf{x}^*)$ is very small) then the linear map $Dg(\mathbf{x}^*)$ can still be of use to approximate the dynamics. We refer to such points as *slow points* and treat them the same way as fixed points for this work.

As initial conditions for our unconstrained minimization problem, we take the values of the GRUs hidden state during a computation of interest (we'll call it a *hidden trajectory*) and we perturb them with small Gaussian noise sample the space more uniformly.

2 Sussillo's Flip-Flop task

In this section, we reproduce one of the RNN dynamics analysis found in [1]. Then we explore the effects of slightly modifying the task and make general observations. See Figure 1 for a description of the task.

2.1 Classical analysis

We trained a GRU to perform the task and then ran the minimization procedure for every possible input to get 27 groups of fixed points. Then, for every fixed points, we computed the Jacobian matrix and its eigenvector decomposition. The key to visualizing the results is to perform a principal components analysis (PCA) on the hidden states sampled during a run of the RNN and observe that most of variance can be explained by only the first 3 PCs (complicated way of saying that the hidden state spends most of his time on a 3D subspace). Therefore, we can draw an approximate phase portrait by projecting everything on the first 3 PCs. See Figure 2 for our results.

We observe that under input $(0,0,0)$, the GRU is able to hold his state through a cube-like fixed point structure in his dynamics where each vertex seems to be associated to a state. Then, the pulses induce dynamics that make the hidden state move to the octant encoding the desired state. This is well illustrated by the positioning of the attractors corresponding to the pulse, as shown in the figures. We note that in the original paper the only consider the fixed points with respect to $(0,0,0)$ input.

Lastly, for all saddle nodes, most directions are stable (at most 3 aren't) and the unstable ones are aligned with first PCA directions, meaning that they funnel all the phase space into the computational 3-dimension subspace.

2.2 2-bit Flip-Flop

A GRU solving the 2-bit Flip-Flop task exhibits dynamics that are analogous to the 3-bit case, except now in 2 dimensions (see Figure 3). We will use this version of the task from now on because it is way easier to visualize.

2.3 Delay between input and response

We now ask the GRUs to wait d time steps after the pulse to react to it. To study the effect of that change, we trained GRUs for increasing value of d, from 0 to 8 (Figure 4)

First, we observe that the general structure of the low-dimensional dynamics is unchanged. However, as d increases, the modulus of the eigenvalues corresponding to linearized systems approach 1 and become complex. (Figure 5). This means that the dynamics gets globally slower and more rotational. In fact, this is the way the GRUs implement the delayed reaction : hidden states take d steps to arrive at destination after a pulse.

2.4 Impact of the encoding

Here we study the impact of the way we encode the problem to the GRU on the dynamics produced. To that effect, we train networks to do two slightly modified version of the task : one where pulses are 1-2 instead of ± 1 and one where inputs and outputs are encoded as one-hot vectors. Their dynamics are shown in Figures 6 and 7. The general structure of the state space is conserved, which means that the dynamics are linked to the task in an abstract sense and not just to a particular formulation. However, there are some notable differences. We observe that the relationships between fixed points associated with different inputs reflects the relationship between those inputs in the encoding space. For the 1-2 pulses, groups of fixed points corresponding to inputs $(0, 0)$, $(1, 0)$ and $(2, 0)$ are on the same line one after the other, like the inputs themselves. For the one-hot encoding, no relationship is enforced and the fixed points arrange mostly disordered way.

Therefore, the nice geometrical interpretability we saw earlier in the positioning of the pulses fixed points was an artifact of the encoding.

We also note that for the 1-2 pulses, the "saddle nodes" mediating the fixed points have no unstable directions. We don't know how to interpret that, other than by saying that there can be variations within the same general structure.

3 Generalization of the Flip-Flop task with FSAs

A Finite State Automaton (FSA) is an abstract model of computation defining a machine that possesses a finite number of states and rules to move between them in response to seeing an input (see Definition 2). We consider it because it can be seen as generalizing the Flip-Flop task.

We designed a framework to train GRUs to implement arbitrary FSAs in an attempt to understand how RNNs solve this simple class of problems through dynamics.

Figures 8, 9 and 10 show three different FSAs along with the dynamics of a GRU that implements them. They are also accompanied by a short description. In the next section, we use these examples along with what we saw earlier to draw general insights into how GRUs implement FSAs.

3.1 How GRUs implment Finite State Automaton

First, although the main principal components give us a good idea of the dynamics, it is clear that the GRUs use more dimensions to make their computation. To illustrate this, observe that for most examples the position of the hidden state on first 2 PCs doesn't completely determine the output state (especially obvious in the 1-2 flip-flop task). Also, in the 3 FSA examples, trajectories starting from reconstructed PCA points (pale gray lines in the background) don't behave exactly like the hidden state during an actual run. It is therefore important to keep in mind that low-dimensional dynamics don't capture the full computational capabilities of the RNN.

Nevertheless, we can use them to extract general ideas about how GRUs implement FSAs :

- Output states correspond to regions of the state space (not necessarily on the first PCs).
- $$\begin{tikzcd} q_0 \arrow[loop above, "0"] & q_1 \arrow[loop above, "0"] \end{tikzcd}$$ can be achieved by forming stable nodes in the regions corresponding to each state. Saddle nodes are present to funnel all the state regions to the appropriate fixed point.
- $$\begin{tikzcd} q_0 \arrow[r, "1"] & q_1 \end{tikzcd}$$ can be achieved by the "1" dynamics moving hidden state from one state region to the other. This can be chained many times to recognize input patterns like in Figure 9 and 10.
- $$\begin{tikzcd} q_0 \arrow[r, "1"] & q_1 \arrow[r, "1"] & q_2 \arrow[ll, bend right, "1"'] \end{tikzcd}$$, or any cycle of constant input can be achieved by the formation of a stable periodic orbit that attracts nearby points and make them cycle through different state regions (Figure 8).

It seems that GRUs can implement arbitrary FSA by combining these mechanisms in various ways, like in the four examples we gave.

4 Conclusion

For the kind of tasks we considered, we saw that the dynamics of the GRU's hidden state are low-dimensional and can be understood by analyzing at the fixed points and associated linear dynamics. This gives us the ability to explain the behavior of the RNN with respect to the hidden state, saying things like "the RNN holds his state under constant $(0, 0)$ input because the hidden state dynamics has fixed points in regions of phase space that the output layer associates to the states". However, this is still a high-level description of the RNN's behavior. It is a bit like if we were trying to reverse engineer a computer and that instead of just being able to look at the screen we can now make sense of the electrical signals sent to it. Most of the processing has already been done at this point. While this a good first step, it does not *explain* how the computer works in any meaningful way. Similarly, we think that to get a satisfying explanation for the GRU's behavior (if it is even possible) we are gonna have to look deeper (ie. the equations of Definition 1).

Therefore, rather than "opening the black box" (refering to the title of [1]), we think that this type of analysis offers us a new pair of glasses through which we can look at it. In particular, it enables us to see regularities in the way different computations are implemented. Overall, we think it is a useful tool to gather intuition, generate hypotheses and guide further explorations. We are excited to see future development this idea (see Related works).

References

[1] David Sussillo and Omri Barak. Opening the black box : Low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Computation*, 25:626–649, 2013.

[2] Niru Maheswaranathan, Alex H. Williams, Matthew D. Golub, Surya Ganguli, and David Sussillo. Universality and individuality in neural dynamics across large populations of recurrent networks, 2019.

[3] Kyle Aitken, Vinay V. Ramasesh, Ankush Garg, Yuan Cao, David Sussillo, and Niru Maheswaranathan. The geometry of integration in text classification rnns, 2020.

[4] Niru Maheswaranathan and David Sussillo. How recurrent networks implement contextual processing in sentiment analysis, 2020.

[5] Antônio H. Ribeiro, Koen Tiels, Luis A. Aguirre, and Thomas B. Schön. Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness, 2020.

[6] Ryan C.Williamson, Brent Doiron, Matthew A.Smith, and Byron M.Yu. Bridging large-scale neuronal recordings and large-scale network models using dimensionality reduction. *Current Opinion in Neurobiology*, 55:40–47, Apr 2019.

[7] David Sussillo. Computation thru dynamics. `https://github.com/google-research/computation-thru-dynamics/`, 2020.

[8] Matt Golub. Fixedpointfinder. `https://github.com/mattgolub/fixed-point-finder/`, 2020.

5 Supplementary material

5.1 Definitions

Definition 1 (Gated Recurrent Unit)**.** A *Gated Recurrent Unit* is a type of RNN defined by the following equations :

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \qquad r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

$$\hat{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \qquad h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

where $r_t, z_t, \hat{h}_t, h_t \in \mathbb{R}^n$ are respectively the update gate, forget gate, memory state and hidden state. The $W_i \in \mathbb{R}^{m \times n}$, $U_i \in \mathbb{R}^{n \times n}$ and $b_i \in \mathbb{R}^n$ are learned parameters where n is the state of the hidden state and m the size of the input. Finaly, $\sigma = \frac{1}{1+e^{-x}}$ is the sigmoid function.

Definition 2 (Finite State Automaton)**.** A *Finite State Automaton* (FSA) is defined by the 4-tuple

$$(Q, \Sigma, \delta, q_0)$$

where Q is a finite set called the *states*, Σ is a finite set called the *alphabet* (the possible inputs), $\delta : Q \times \Sigma \to Q$ is the *transition function* and q_0 the *starting state.*

5.2 Methods

All the computations were performed using the Wolfram Language with Mathematica. We drew inspiration from [7] and [8]. The Mathematica notebook is

5.3 Related works

It has been showed in [2] that certain aspects of the low-dimensional dynamics are invariant of the type of RNN architecture used. Recently, RNN performing more natural tasks like text classification and sentiment analysis have been analyzed with dynamics ([3],[4]). The bifurcations creating the fixed points during training have been studied by [5]. Finally, works like [6] bridge artificial and biological neural networks using dynamics.

5.4 Figures

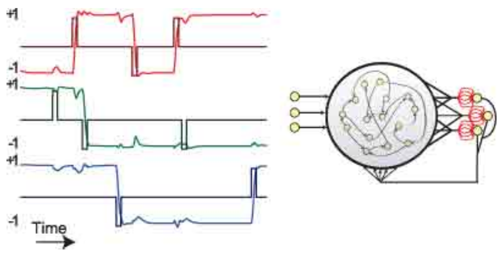

Figure 1: Example inputs and outputs for the 3-bit flip-flop task.(Left) Input are the black lines, outputs are the colored lines. Inputs are streams of 0s with randomly timed ±1 pulses. The network must always output the value of the last pulse. It must deal with the 3 bits independently. (Right) Depiction of the RNN.

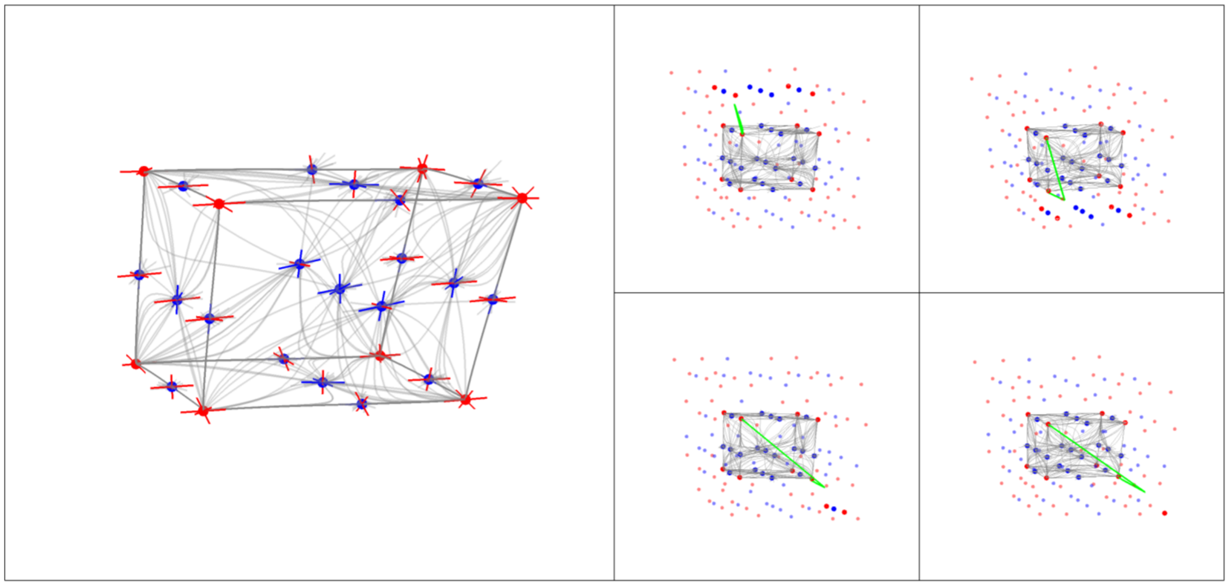

Figure 2: Low-dimensional phase space representation of the 3-bit Flip-Flop task. Red points correspond to stable fixed points. Blue point have are fixed points with at least 1 unstable eigenvalue (saddle nodes). The pale gray lines show trajectories starting near each saddle node under constant $(0, 0, 0)$ input. (Left) Fixed points with respect to $(0, 0, 0)$. Colored lines emanating from the points correspond to the direction of the 3 largest eigenvalues.(Right)Fixed points for ALL inputs combined. The green lines are the trajectories starting at the point corresponding to state $(0, 0, 1)$ (top left corner of the cube) followed by a pulse and then few $(0, 0, 0)$ inputs. Pulses shown are $(0, 0, 1), (0, 0, -1), (1, 0, -1)$ and $(1, 1, -1)$, from left to right, top to bottom. Darker points (apart from the 3x3 inner cube) are the fixed points corresponding to the pulse

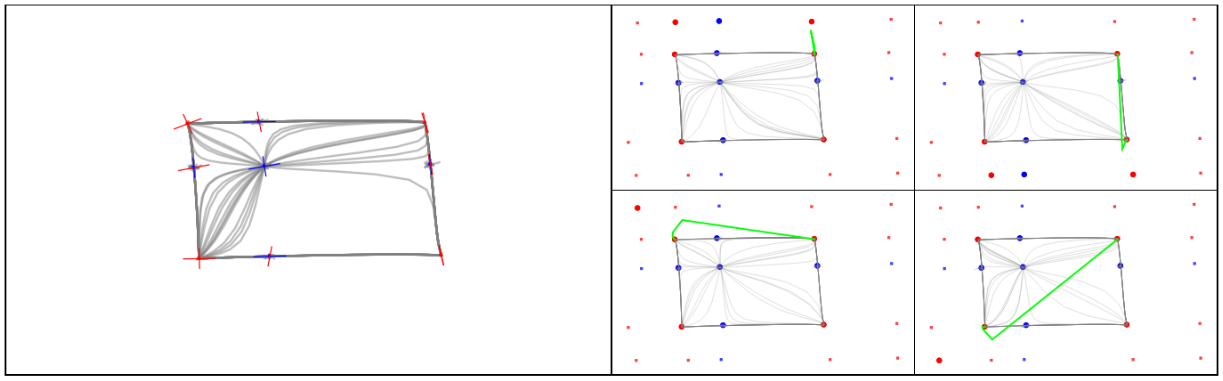

Figure 3: Low-dimensional phase space representation of 2-bit Flip-Flop task. Analogous to 3-bit counterpart. Trajectories on the right show the state $(1, 1)$ (top right corner of square) responding to pulses from $(0, 1), (0, -1), (-1, 1)$ and $(-1, -1)$ (from left to right, top to bottom).

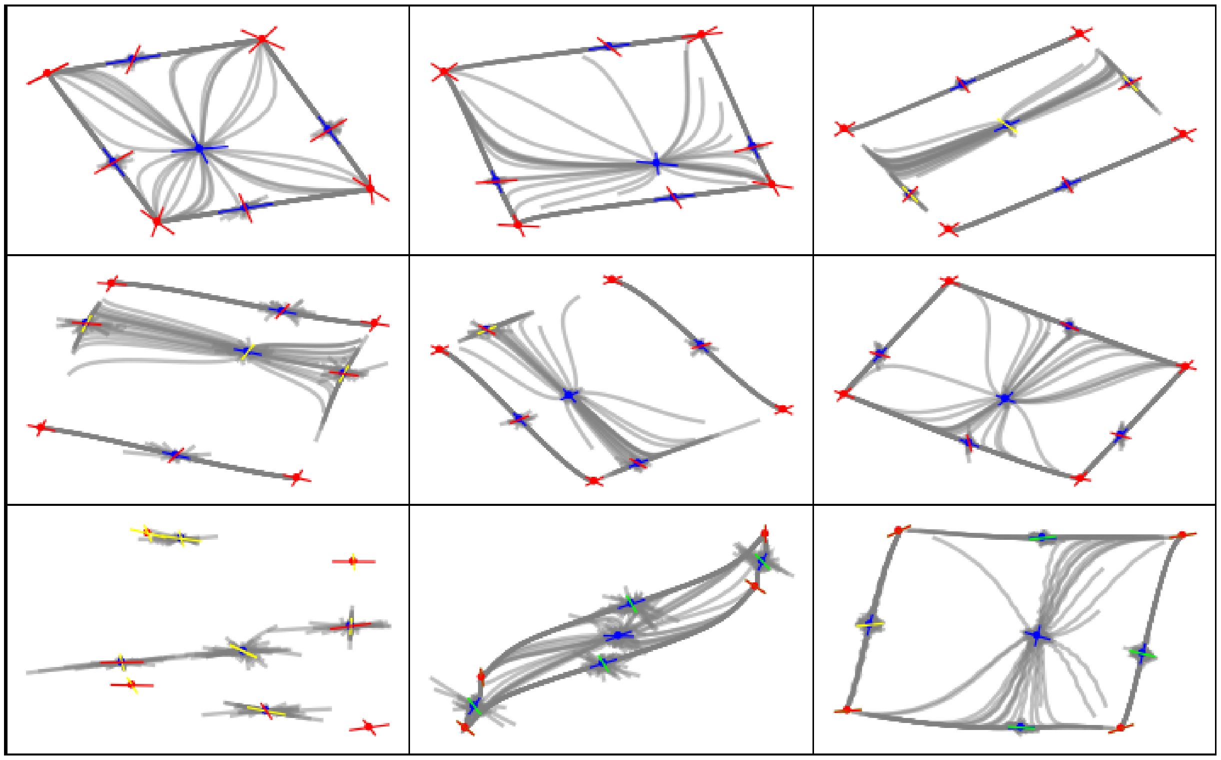

Figure 4: Each graph corresponds to the structure of the fixed points of the $(0, 0)$ dynamics for a specific delay. Delay goes from 0 to 8 as we move from left to right, top to bottom. Yellow lines correspond unit eigenvalue and green lines correspond to -1 eigenvalue (2-cycle).

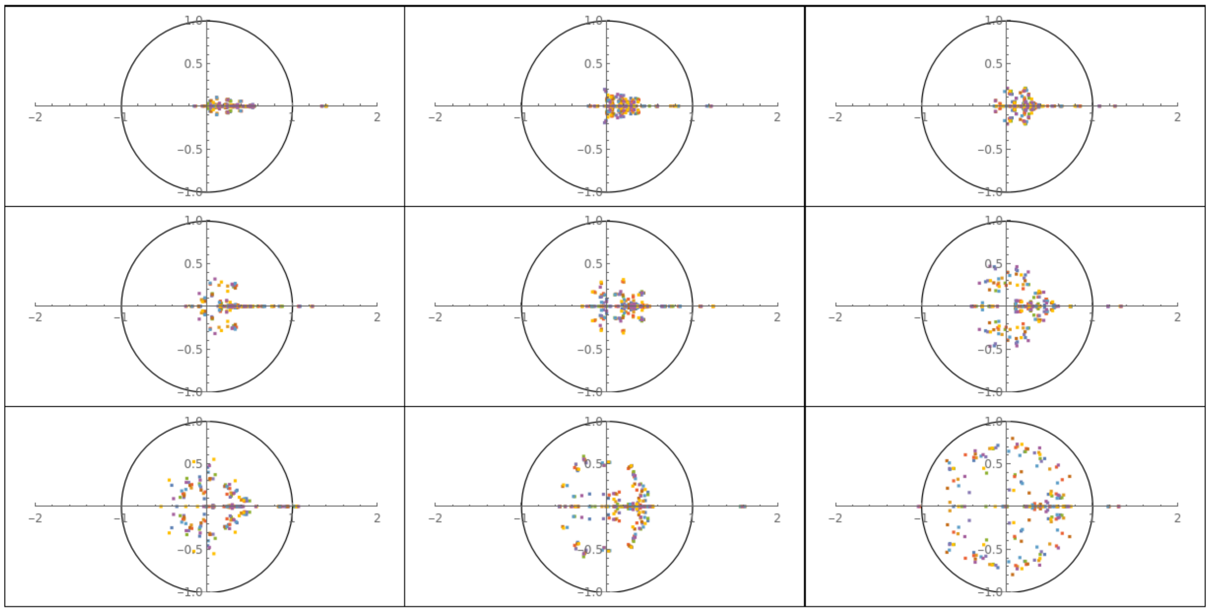

Figure 5: Each graph corresponds the combined eigenspectrum of all fixed points for the $(0, 0)$ dynamics and specific delay. Delay goes from 0 to 8 as we move from left to right, top to bottom.

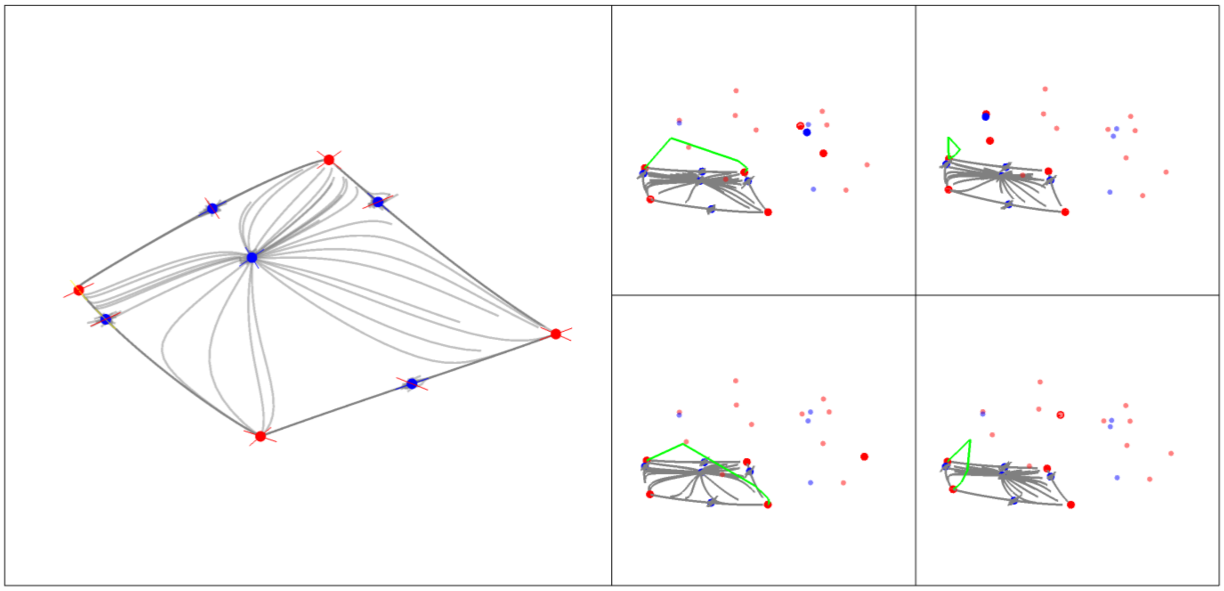

Figure 6: Low-dimensional phase space representation of 2-bit Flip-Flop task with inputs are outputs encoded as one-hot vectors. Plots on the right are the projections <u>on the first 3 PCs</u>. Trajectories on the right show the state $(1, 1)$ (top left corner of square) responding to pulses from $(0, -1)$, $(0, 1)$, $(-1, -1)$ and $(-1, 1)$ (from left to right, top to bottom).

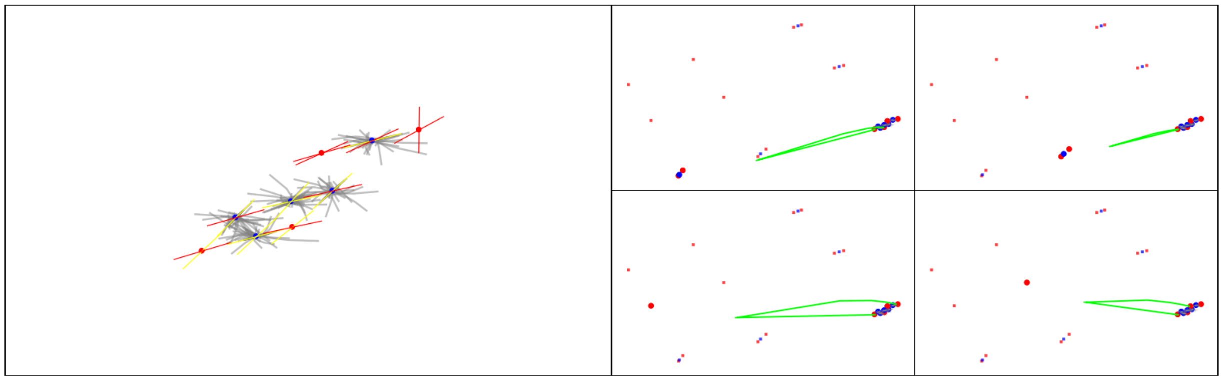

Figure 7: Low-dimensional phase space representation of 2-bit Flip-Flop task with 1-2 pulses. Trajectories on the right show the state $(1, 1)$ (bottom left corner of square) responding to pulses from $(0, 2)$, $(0, 1)$, $(2, 1)$ and $(1, 1)$ (from left to right, top to bottom).

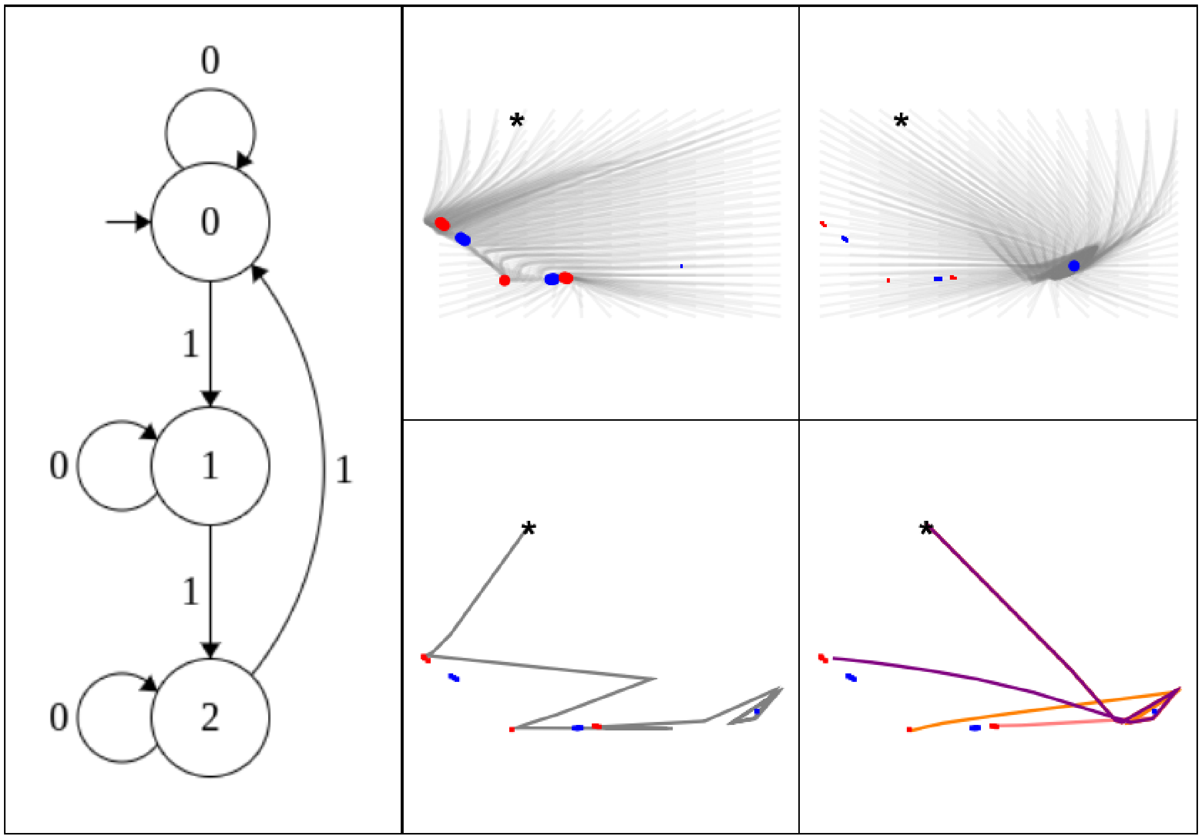

Figure 8: Low-dimensional dynamics for the modulo 3 counter GRU (FSM on the left). The black star corresponds to the initial state of the GRU (always the same).(Top) Phase portrait for dynamics under constant 0 input (left) and constant 1 input (right). Pale gray lines are trajectories starting from hidden states obtained by reconstructing uniformly spaced points in the projected PCA plane. "Active" fixed points are larger. Under constant 0 input, hidden states converge to one of the 3 stable nodes (red) on the left depending on their starting location in phase space. The fixed point on corresponding to input 1 (on the right) is surrounded by a stable limit cycle of period 3. (Bottom left) Trajectory corresponding to input $(0,0,0,1,0,0,0,1,0,0,0,1,1,1,1,1,1,1,1)$. (Bottom right) Three trajectories corresponding to a variable number of input 1 (6, 7 and 8) followed by 0s. The whole system is fine tuned so that the "1" dynamics will always leave the hidden state in a place that will converge to the adequate stable node when inputs become 0. In particular, each of the stable limit cycle points correspond to one state.

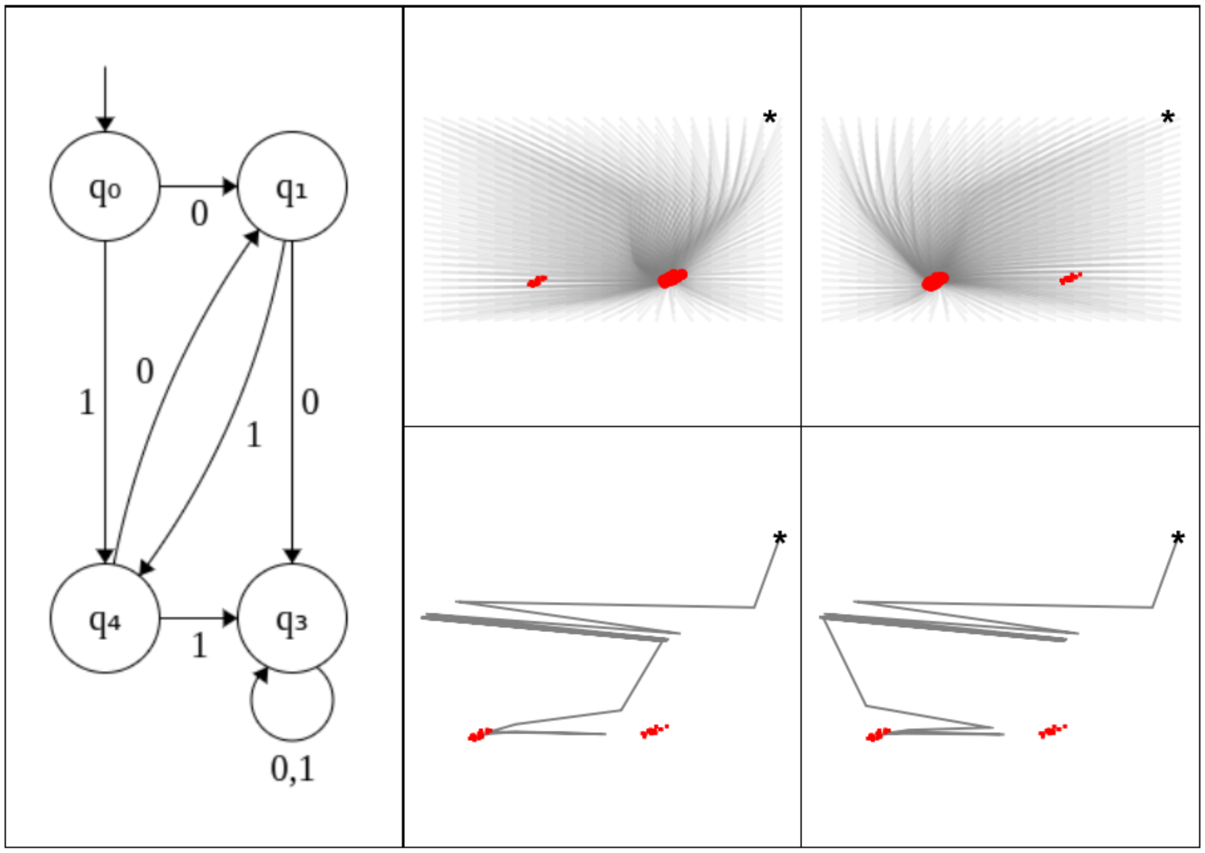

Figure 9: Low-dimensional dynamics for the GRU associated with the FSM on the left. The task consists of going into an "trapping" state ($q3$) once any symbol is repeated. The black star corresponds to the initial state of the GRU (always the same). (Top) Phase portrait for dynamics under constant 0 input (left) and constant 1 input (right). Pale gray lines are trajectories starting from hidden states obtained by reconstructing uniformly spaced points in the projected PCA plane. "Active" fixed points are larger. The dynamics under input 0 consists of pushing phase space to the right side and then to down to the bunch of stable nodes. Same thing but on the opposite side when input is 1. (Bottom) Trajectories resulting from input $(0,1,0,1,0,1,0,1,0,0,1,0,1,0)$ on left and $(0,1,0,1,0,1,0,1,1,0,1,0,1,0)$ on right. Therefore, while the input is alternating between 0 and 1, the hidden state oscillates "above" the fixed points in the phase portrait. When a symbol repeats itself, the hidden state undergoes the downward pull of one of the fixed point and gets trapped at the bottom forever.

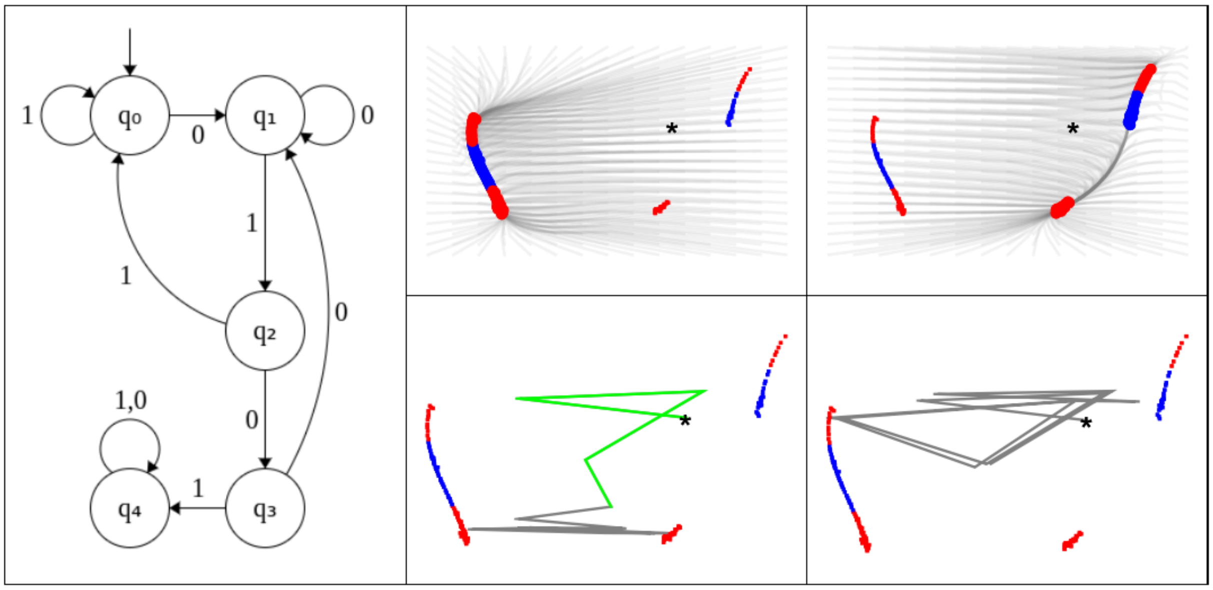

Figure 10: Low-dimensional dynamics for the GRU associated with the FSM on the left. The task consists of going into an ”trapping” state (q_4) once the the substring 0101 is seen. The black star corresponds to the initial state of the GRU (always the same). (Top) Phase portrait for dynamics under constant 0 input (left) and constant 1 input (right). Pale gray lines are trajectories starting from hidden states obtained by reconstructing uniformly spaced points in the projected PCA plane. ”Active” fixed points are larger. (Bottom left) Trajectoy corresponding to input $(0, 1, 0, 1, 0, 1, 0, 1, 1, 0)$, containing the desired substring. The part of the trajectory corresponding to reading 0101 is in green. (Bottom right) Trajectory corresponding to input $(0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0)$, not containing 0101. The system is fine tunes so that the only way to get to the bottom of the phase portrait is reading 0101. And once it is there it cannot leave.