An efficient way of finding all fixed points in shallow (low-rank) piece-wise linear recurrent neural networks

Matthijs Pals, University of Tuebingen

September 21, 2023

1 Introduction

Piece-wise linear recurrent neural networks (PLRNNs) have been shown to be able emulate a wide range of dynamical systems, all while being interpretable - all fixed points can be solved for (given a potentially high computational cost). Recently a 'low-rank' variant of the piece-wise model was proposed (shPLRNN; eq. 12 in [1]):

$$\mathbf{z}_t = \mathbf{A}\mathbf{z}_{t-1} + \mathbf{V}^{\mathsf{T}}\mathsf{ReLU}(\mathbf{U}\mathbf{z}_{t-1} - \mathbf{h}_{\mathbf{x}}) - \mathbf{h}_{\mathbf{z}} \tag{1}$$

with $\mathbf{z}_t \in \mathcal{R}^R$ and $\mathbf{V}, \mathbf{U} \in \mathcal{R}^{N\times R}$. To find all fixed points naively, we need to solve 2^N inverses of $R \times R$ matrices. (eq. 39 in [1])

2 Main result

We can obtain all fixed points of a shPLRNN, by solving $\sum_{r=0}^{R} \binom{N}{r}$ inverses of $R \times R$ matrices.

3 Argument

3.1 Obtaining fixed points in PLRNNs

First, following [1]. To find all fixed points, start by redefining ReLU by introducing a matrix:

$$\mathbf{D} = \begin{bmatrix} \mathbf{d}_1 & & & \\ & \mathbf{d}_2 & & \\ & & \ddots & \\ & & & \mathbf{d}_N \end{bmatrix} \tag{2}$$

with $\mathbf{d}_i = \begin{cases} 1, & \text{if } \mathbf{x}_i > \mathbf{b}_i \\ 0, & \text{otherwise} \end{cases}$

Then

$$\mathbf{z}_t = \mathbf{A}\mathbf{z}_{t-1} + \mathbf{V}^{\mathsf{T}}\mathbf{D}\mathbf{U}\mathbf{z}_{t-1} - \mathbf{V}^{\mathsf{T}}\mathbf{D}\mathbf{h}_{\mathbf{x}} - \mathbf{h}_{\mathbf{z}} \tag{3}$$

Next, for all all 2^N configurations $\mathbf{D}^{(j)}$ of $\mathbf{D}$, solve:

$$\mathbf{z}^* = (\mathbf{A} + \mathbf{V}^{\mathsf{T}}\mathbf{D}^{(j)}\mathbf{U} - \mathbf{I})^{-1}(\mathbf{V}^{\mathsf{T}}\mathbf{D}^{(j)}\mathbf{h}_{\mathbf{x}} + \mathbf{h}_{\mathbf{z}}) \tag{4}$$

for $\mathbf{z}^*$, and check whether the obtained $\mathbf{z}^*$ is consistent with the assumed $\mathbf{D}^{(j)}$, if so, you found a fixed point of the shPLRNN.

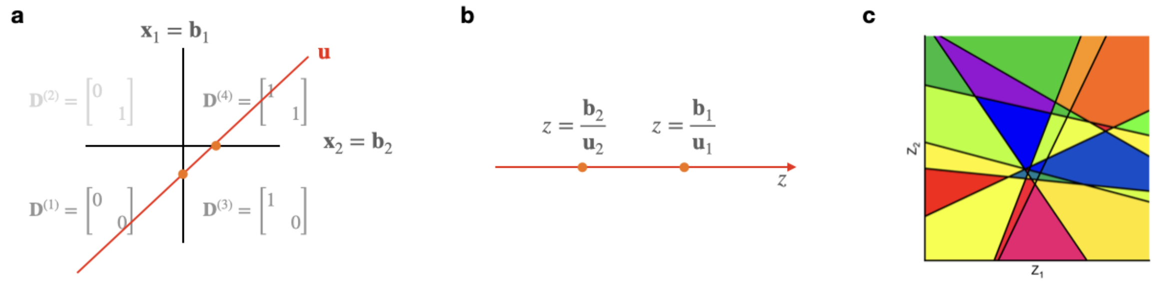

Figure 1: **a**. We can only reach a subset of the subspaces corresponding to each **D**. **b.** Each neuron breaks up the space spanned by **u** with a hyperplane (point in the 1D case) **c.** All subspaces of a randomly initialised network with N=8, R=2.

3.2 Observation 1: a shPLRNN is a low-rank version of a standard PLRNN

Introduce $\mathbf{x}_t \in \mathcal{R}^N$, such that $\mathbf{z}_t$ can be seen as the projection of $\mathbf{x}_t$ on $\mathbf{U}$: $\mathbf{x}_t = \mathbf{U}\mathbf{z}_t$, $\mathbf{z}_t = (\mathbf{U}^\intercal\mathbf{U})^{-1}\mathbf{U}^\intercal\mathbf{x}$. Furthermore, introduce $\mathbf{W} = \mathbf{U}\mathbf{V}^\intercal$ and $\mathbf{A}_\mathbf{x} = \mathbf{U}\mathbf{A}(\mathbf{U}^\intercal\mathbf{U})^{-1}\mathbf{U}^\intercal$

Then we can rewrite equation 1 as:

$$\mathbf{x}_t = \mathbf{A}_\mathbf{x}\mathbf{x}_{t-1} + \mathbf{W}\mathsf{ReLU}(\mathbf{x}_{t-1} - \mathbf{h}_\mathbf{x}) - \mathbf{U}\mathbf{h}_\mathbf{z} \tag{5}$$

a 'standard' PLRNN - however $\mathbf{W}$ and $\mathbf{A}_x$ are low-rank matrices, with span columns of $\mathbf{U}$. As a consequence $\mathbf{x}_t$ is constrained to an R dimensional linear subspace (see also work by e.g. Ostojic' lab).

3.3 Observation 2: not all configurations of D can be reached

To get some intuition, consider $R = 1, N = 2$. The phase space of the model can be broken up in 4 subspaces, each corresponding to one configuration of $\mathbf{D}$ (Figure 1a). The dynamics are however confined to the line spanned by the vector $\mathbf{u}$ (with coordinate z), which can reach at most 3 of the subspaces (red line in Figure 1a).

We can solve for the points where $\mathbf{u}$ crosses a border between subspaces. E.g. for border given by $\mathbf{x}_1 = \mathbf{b}_1$, we have $\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{bmatrix} = z\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix} = \begin{bmatrix}\mathbf{b}_1\\ 0\end{bmatrix}$, thus this border crosses $\mathbf{u}$ at $z = \frac{\mathbf{b}_1}{\mathbf{u}_1}$ (see Figure 1b)

In general, for networks with $R = 1$, each neuron adds 1 point to the line along which the dynamics flow (at $z = \frac{\mathbf{b}_i}{\mathbf{u}_i}$ for the i'th neuron), meaning we can reach at most $N + 1$ subspaces / configurations of $\mathbf{D}$. We only have to solve $N + 1$ 'matrix' inverses to find all inverses!

For $R = 2$ dynamics are confined to a plane, spanned by the two columns of U, and each neuron adds a line to this plane (Figure 1c contains an example with N=8). The amount of configurations of $\mathbf{D}$ we can reach is thus equal to the question, in how many surfaces can we divide a plane with N lines, which is $\frac{1}{2}(N^2 + N) + 1$ — a big improvement over 2^N.

Generalising this to arbitrary R, we have to answer the question, in how many subspaces can we divide an R dimensional space with N hyperplanes, which has the solution $\sum_{r=0}^{R}\binom{N}{r}$

3.4 Finding all reachable configurations of D

We so far showed that we only need to consider $\sum_{r=0}^{R}\binom{N}{r}$ configurations of $\mathbf{D}$, but how do we find these configurations?. First we can solve for all cross sections of hyperplanes (see previous section for r=1). To do this, we have to solve $\binom{N}{r}$ systems of linear equations.

From each cross sections we can reach 2^R possible subregions by moving epsilon away from the cross section. We can find the corresponding $\mathbf{D}$ matrices of each of those subsection as follows. First compute $\mathbf{x} = \mathbf{U}\mathbf{z}$, corresponding to the cross section we are currently considering. Next calculate all the $\mathbf{D}$s corresponding to the found $\mathbf{x}$ — for elements $\mathbf{x}_i$ of $\mathbf{x}$ which are equal to $\mathbf{b}_i$, $\mathbf{D}_i$ can be either 1, or 0. Now we just make a list of all possible $\mathbf{D}$s that we find this way, and finally remove any duplicates.

References

[1] Florian Hess, Zahra Monfared, Manuel Brenner, and Daniel Durstewitz. Generalized teacher forcing for learning chaotic dynamics, 2023.