CHARTING AND NAVIGATING THE SPACE OF SOLUTIONS FOR RECURRENT NEURAL NETWORKS SUPPLEMENTARY MATERIAL

Anonymous Author(s)
Affiliation
Address
email

1 2D RNN

1.1 Training methods

The four parameters, $W \in \mathbb{R}^{2\times 2}$, were iid sampled from a uniform distribution $\mathcal{U}(-1.5, 1.5)$. We implemented the continuous time dynamics of the RNN in PyTorch[1] using the package torchdiffeq[2, 3].

Network parameters were optimized using gradient descent with no momentum and a learning rate of 0.03. During each step of learning, the network dynamics were simulated for a single trajectory (Equation 1 in main text), and the loss $\mathcal{L}$ (Eq 2 in main text)was used to compute the gradient. We then normalised the gradient by its Frobenius norm and scaled it by $\sqrt{\mathcal{L}}$ before updating the parameters. This is a heuristic choice to motivate convergence of learning even when gradients are small for some pathological initializations. Networks were trained for 2000 epochs, and only RNNs with $\mathcal{L} < 10^{-5}$ were accepted as solutions.

1.2 Different nonlinearity

To examine the effect of training hyperparameters on the space of solutions, we used $\phi :=$ ReLU instead of $\phi := \tanh$ that was used in the main text. We find that this choice indeed leads to different solution types. Specifically, ReLU RNNs did not converge to limit-cycle solutions. Some converged to non-zero fixed points, accompanied by a saddle point at the origin, as in the main text. In addition, two other solution types arised in this setting. A stable origin with large transient amplification, as in the yellow curve of Figure 1), and a diverging trajectory, shown by the dark curve in the same figure. The effect of the different non-linearity is also seen in the distribution of trace and determinants of the solutions (Figure 2), where limit cycles are absent for ReLu, and stable solutions (bottom-right quadrant) are absent for $\tanh$.

2 Timing task

2.1 Training process

2.1.1 Network architecture

We studied three different RNN architectures and their exact equations are all summarized below. The trained parameters are the weights W and biases b. The function $\sigma(z) = (1 + exp(-z))^{-1}$ is the sigmoid function, $h_t \in \mathbb{R}^N$ and $u_t \in \{0, 1\}^2$ are the state and the input at time t.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

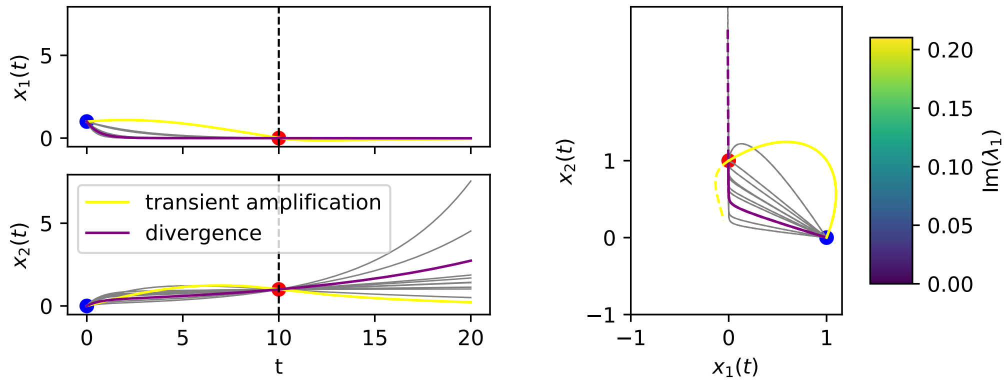

Figure 1: Trajectories of several 2D RNN solutions for $\phi :=$ ReLU.

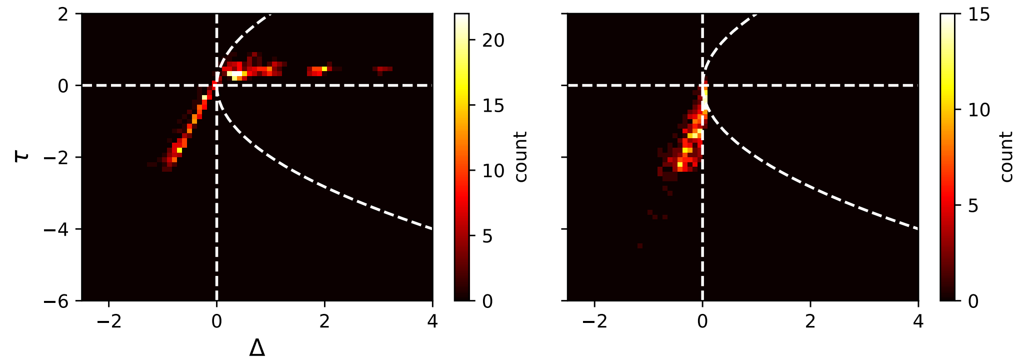

Figure 2: A 2D histogram of trace τ and determinant Δ of 2D RNNs solutions for the task specified in the main text. $\phi := \tanh$ (left) and $\phi :=$ ReLU (right).

Vanilla [4]

$$h_t = \tanh\left(W_{ih}u_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}\right) \tag{1}$$

GRU [5]

$$\begin{aligned} r_t &= \sigma\left(W_{ir}u_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}\right) && (2)\\ z_t &= \sigma\left(W_{iz}u_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}\right) && (3)\\ n_t &= \tanh\left(W_{in}u_t + b_{in} + r_t * \left(W_{hn}h_{t-1} + b_{hn}\right)\right) && (4)\\ h_t &= (1 - z_t) * n_t + z_t * h_{t-1} && (5) \end{aligned}$$

LSTM [6]

$$\begin{aligned} i_t &= \sigma\left(W_{ii}u_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right) && (6)\\ f_t &= \sigma\left(W_{if}u_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right) && (7)\\ g_t &= \tanh\left(W_{ig}u_t + b_{ig} + r_t * \left(W_{hg}h_{t-1} + b_{hg}\right)\right) && (8)\\ o_t &= \sigma\left(W_{io}u_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right) && (9)\\ c_t &= f_t * c_{t-1} + i_t * g_t && (10)\\ h_t &= o_t * \tanh(c_t) && (11) \end{aligned}$$

The units had $N = 20, \ldots, 50$ hidden neurons and the output of the network at every time-step is an affine readout of the internal state. h_0 was always initialized to zero.

2.1.2 Task and trial structure

Each trial was comprised of seven consecutive epochs, as demonstrated in Figure 3. The *Ready* pulse was given after $20-30$ steps. Both inputs and the required output were binary sequences with ones during each pulse (10 steps long) and zero elsewhere. When working with intervals from the range $[t_s^{min}, t_s^{max}]$, the length of all trials was set to $2 * t_s^{max} + 100$. This allowed the network time to relax back to rest for at least 70 steps after emitting a *Go* pulse. The training set always included 512

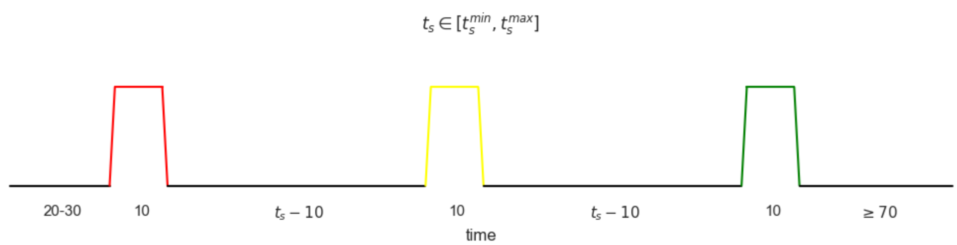

Figure 3: Ready-Set-Go timeline

random trials so, on average, every interval was included more than 5 times.

2.1.3 Training protocol

All networks were trained using *Adam* [7] for 10000 epochs with a batch size of 64 and a decaying learning rate starting from $1e-3$ up until $1e-4$. Unless stated otherwise, the training set was comprised of 512 trials and their order was shuffled at the beginning of each epoch. We estimated the network's performance with mean squared error (MSE), and training was halted when the minimal threshold of 10^{-5} was achieved over the training set.

2.2 Feature extraction

The feature extraction process in this work can be divided into two. First, we describe how we extracted numerical features from the neural activity during training. Later, we describe how we extracted topological features by analyzing the network dynamics outside the training set. Often in our analysis we quantified the neural velocity in phase space. For that purpose, we used the scalar function

$$q(h_t) = \|h_{t+1} - h_t\|_2, \quad t \in \mathbb{N} \tag{12}$$

that was introduced in [8]. This function estimates the distance that the network crosses in a single time-step, given its current location.

2.3 Neural features

As described in the main text, we extracted various features from the neural activity during the training set. These were related to the major dynamical objects and task epochs. Below, we explain and define the features according to the relevant epoch.

Ready-Set features As seen in the main text (FIGURE WITH CIRCULAR TRAJECTORY), the shape of the Ready-Set trajectory can indicate whether the network will eventually converge to a limit cycle. We thus considered the minimal and maximal curvature, the speed at its end, and the ratio between its initial and final speed. All these features were measured on a logarithmic scale.

Set-Go features The Set-Go manifold was defined by collecting the network states corresponding to trials of all delays ($t_s \in [30, 120]$), and using the time points from 10 after the Set pulse until 10 before the Go pulse. Because this is a two-dimensional manifold (time by trials), we calculated the aspect ratio as follows. The nominator was the cumulative length of the trajectory corresponding to the initial states across all trials. The denominator was the length of the full trajectory of the longest trial ($t_s = 120$). Similarly, we extracted the aspect ratio with respect to the final states of the Set-Go manifold. We also measured how this ratio changes as a function of time in the following manner.. Later, we calculated how the length of the Set-Go trajectory changes as a function of the time interval that is being encoded, by fitting a linear regressor to the mapping $t_s \rightarrow ||\text{Set-Go}(t_s)||_2^2$ and extracting its slope as a feature. To account for whether time-coding is concentrated in single neurons or distributed across the population, we measured the following quantity. For each neuron, we considered all points on the Set-Go manifold. We used linear regression to map the activity of the neuron to the time remaining until the Go pulse. The minimal error across all neurons was used as a feature.

Ready-Set & Set-Go features Here, we focused on the relationship between the trajectory Ready-Set(t_s^{max}) and Set-Go(t_s^{max}). We extracted as features the Pearson correlation and the angle between them, the ratio between their speeds, and the width of their separating hyper-plane obtained from Linear SVM.

Figure 4 shows the density-histogram of each feature for each architecture.

2.4 Topological

As we discussed in the main text, defining what is a solution is not trivial. In the context of dynamical systems, obtaining a qualitative description of the phase space is often enough. However, this description requires full knowledge of the dynamical objects, which is often inaccessible. Particularly, the transient nature of the Ready-Set-Go timing task renders the dynamical landscape less structured and more difficult to analyze with classical dynamical systems tools. Therefore, to understand the mechanism of the network we will suffice in identifying the key areas of the dynamics and the transitions between them. In the RSG task, following the Ready cue the network entire activity falls into one of the following categories: Ready-Set, Set-Go, Go, and its final configuration. These state-space regions are not mutually exclusive, thus each different realization of them may indicate a different algorithm. We derived a set of binary features (Figure 5) to partition the networks into solution sets with distinct topological structures. To measure whether the Ready-Set and the Set-Go epochs are the same objects (Fig 5 A) we fitted a linear SVM classifier to separate the neural states that constitute these two epochs. If the classification was not successful, we considered them the same. To measure whether the network transitions to the Go epoch after the Ready (Fig 5 E) we let it evolve from there for 200 steps and measured its output. We initially let the network evolve after the Ready pulse for 5000 steps and saved the final state and the neural velocity at that state. We applied a threshold of 10^{-6} to determine whether it converges to a fixed point (Fig 5 D). To measure whether a transition to Go occurs from any state (5 B) or whether its activity is periodic (Fig 5 C) we let it evolve spontaneously for 1000 steps from that saved state and measured its output and periodicity respectively. Using these features, we divided the space of solutions into six distinct clusters.

2.5 Different views of same object

To see whether the neural data from the training set contains information about the topology of the networks, we evaluated the ability of the neural features to predict the topological classification we described earlier. This was done by a Cross-Validation procedure that included 50 repetitions of fitting a Decision Tree classifier to a randomly selected 70% of the data, and then evaluating the *kappa-cohen* score and the confusion matrix of the classification on the remaining validation set. For each architecture separately and combined, the mean and the standard errors of these two measurements across all repetitions are shown in 6.

References

[1] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[2] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *Advances in Neural Information Processing Systems*, 2018.

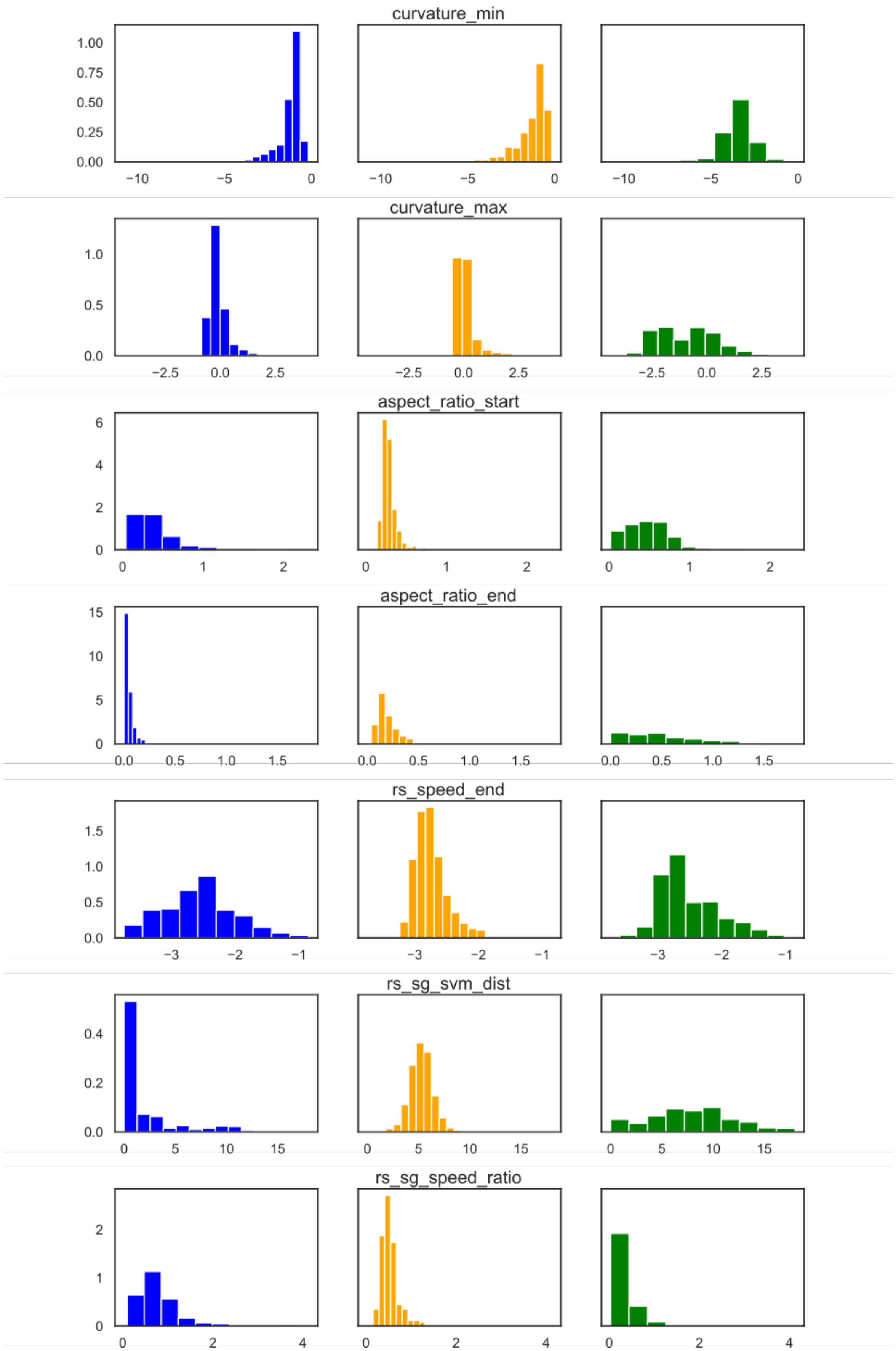

Figure 4: The histogram of every feature, shown for each architecture.

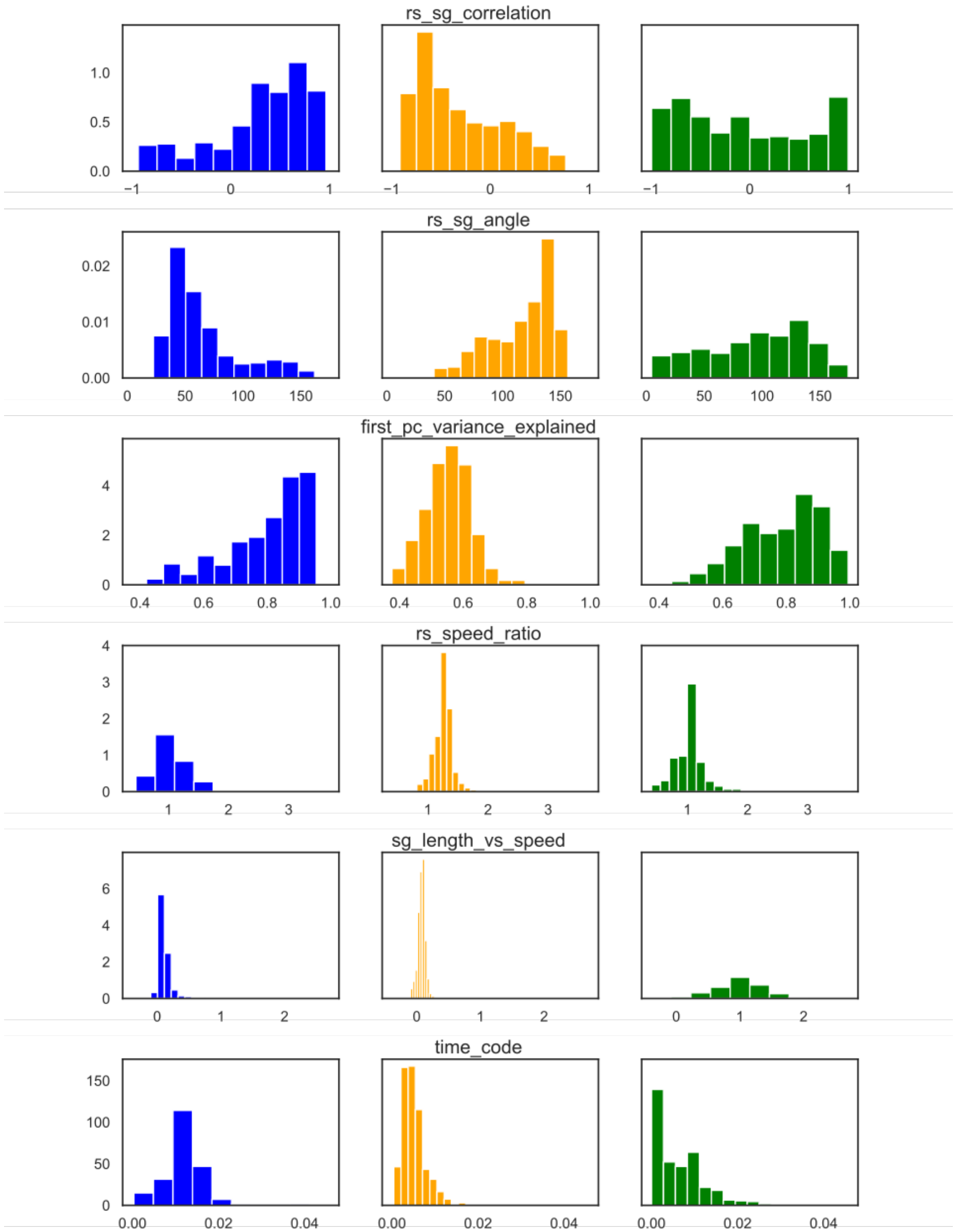

Figure 4: The histogram of every feature, shown for each architecture. (cont.)

[3] R. T. Q. Chen, B. Amos, and M. Nickel, "Learning neural event functions for ordinary differential equations," *International Conference on Learning Representations*, 2021.

[4] J. L. Elman, "Finding Structure in Time," vol. 14, no. 2, pp. 179–211. [Online]. Available: http://doi.wiley.com/10.1207/s15516709cog1402_1

[5] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pp. 1724–1734.

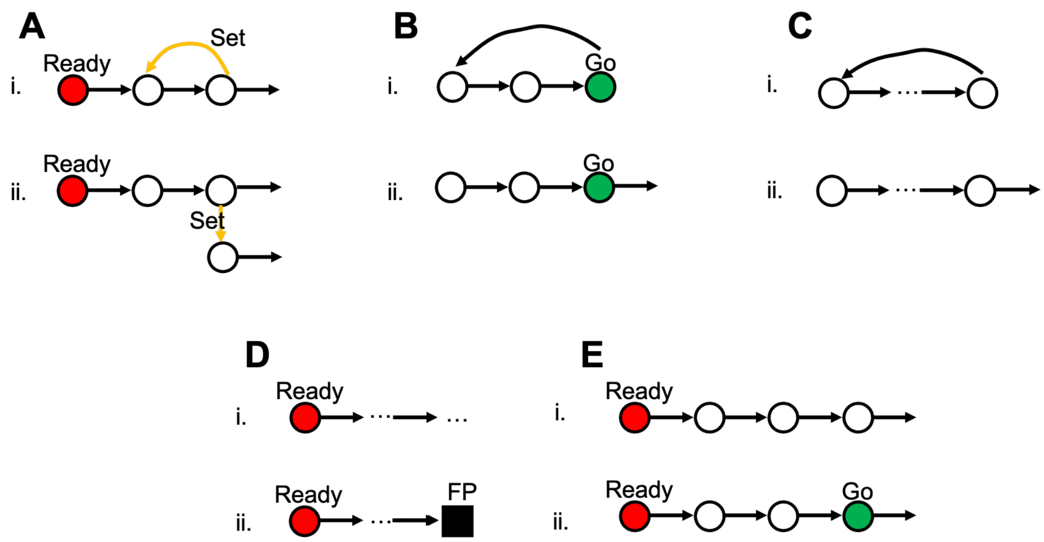

Figure 5: The five topological properties targeted by the binary features. Each feature separates alternatives i and ii. **A** Whether the Ready-Set epoch is identical to the Set-Go (i) or not (ii). **B** Whether the final state transitions to the Go epoch autonomously (i) or not (ii). **C** Whether the network enters a limit-cycle (i) or not (ii). **D** Whether the network reaches a fixed points autonomously from the Ready-Set epoch (i) or not (ii). **E** Whether the Ready-Set epoch transitions to the Go epoch autonomously (i) or not (ii).

[Online]. Available: http://aclweb.org/anthology/D14-1179

[6] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," vol. 9, no. 8, pp. 1735–1780. [Online]. Available: https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735

[7] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. [Online]. Available: http://arxiv.org/abs/1412.6980

[8] D. Sussillo and O. Barak, "Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks," vol. 25, no. 3, pp. 626–649. [Online]. Available: https://www.mitpressjournals.org/doi/abs/10.1162/NECO_a_00409

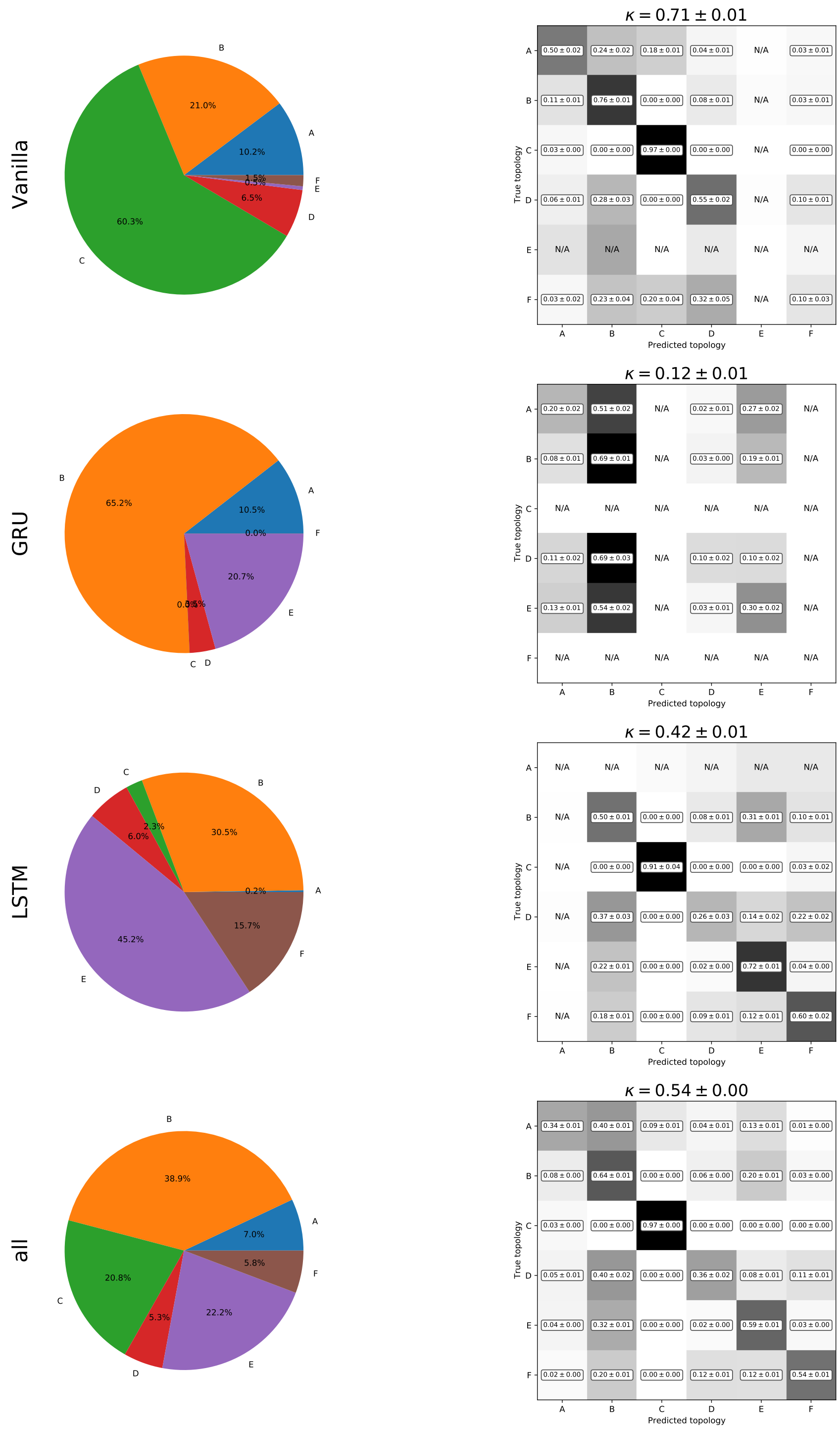

Figure 6: Left: The distribution of the different topologies for each architecture, and for all networks combined. Right: Confusion matrices (mean and standard errors) obtained from 50 repetitions of a Decision-Tree classifier. Cells corresponding to underrepresented topologies are shown as N/A. The value of Cohen's κ is shown for each matrix.