Visualizing Linear RNNs Through Unrolling

Josue Casco-Rodriguez[*], **Tyler Burley**[*], **CJ Barberan**[†], **Ahmed Imtiaz Humayun**[*], **Randall Balestriero**[‡], **Richard G. Baraniuk**[*]
[*]Rice University, [†]Microsoft, [‡]Brown University

Abstract

Neural networks are revolutionizing artificial intelligence (AI), but suffer from poor explainability; for example, recurrent neural networks (RNNs) hold massive potential for sequential or real-time information processing, but their recurrences exacerbate explainability issues and make understanding or predicting RNN behavior difficult. One way to explain neural networks is SplineCam, which illustrates a 2D projection of a neural network's analytical form—however, it does not natively support RNNs. We circumvent this limitation by using linearly-recurrent RNNs, which can be unrolled into feedforward networks. We apply the resulting method, dubbed SplineCam-Linear-RNN, to linearly-recurrent RNNs trained on biosignal data and sequential MNIST. Our procedure enables: (1) unprecedented visualization of the decision boundary and complexity of an RNN, and (2) visualization of the frequency sensitivity of RNNs around individual data points.

1 Introduction

Recurrent neural networks (RNNs) imbue artificial intelligence (AI) with the ability to process data sequentially or in real-time, and are foundational to the progress of fields like time-series processing, language modeling, and robotics [1]. For example, RNNs can process biosignals in real-time to swiftly assist technologies such as epilepsy detection or prosthetic control [2].

However, RNNs suffer from an exacerbated form of a malady inherent to deep learning: poor explainability and interpretability. Deep neural networks are already very nonlinear nonlinear and thus highly difficult to understand, interpret, or describe in the same way that humans can describe mathematical functions or explain the reasoning behind a decision. RNNs compound the complexity of deep networks, and thus the challenges of explainability and interpretation, by recurrently processing data in a sequential fashion. RNNs are crucial to the algorithms behind safety-critical real-time applications like medicine [2, 3], automation [4], and forecasting [5], and yet rigorous methods to explain—i.e., accurately interpret [6]—the inner workings of RNNs or how they may fail are sorely lacking.

To elucidate the inner workings of RNNs, we extend a new theoretically sound neural network illustration method, SplineCam [7], to a family of RNNs with linear recurrences ("linear RNNs"). First, we train linear RNNs [8] on an electromyography (EMG) dataset, the PhysioNet 2017 dataset [9][1], and the sequential MNIST [10–12] dataset. Then, we unroll the RNN so that it is compatible with SplineCam, which allows us to:

1. Illustrate the complexity, decision boundary, and adversarial examples of the RNN in 2D.
2. Probe the frequency-sensitivity of linear RNNs around individual data samples.

[1]to our knowledge, this may be the first instance of linear RNNs for EMG and PhysioNet classification.

Preprint. Under review.

2 Background

2.1 Related Work

Neural Networks as Continuous Piece-Wise Affine Functions. Deep neural networks are often composed of linear transformations followed by rectified-linear (ReLU) nonlinearities. Networks that only use continuous piecewise-linear (CPWL) nonlinearities, such as (leaky) ReLUs, are themselves elaborate CPWL functions. Spline theory [13] analyzes CPWL networks to produce several recent insights into key facets of neural network behavior like normalization and adversarial robustness [14, 15]. One notable recent insight is SplineCam [7], which precisely visualizes the decision boundary, adversarial examples, and complexity of a CPWL neural network by computing the boundaries of linear regions of a neural network in a 2D space around input samples.

RNNs, Splines, and Explainability. Pre-existing work in explaining RNNs has largely avoided incorporating spline theory, resulting in a lack of precise methods to visualize and probe the function represented by an RNN. One popular method of explaining RNNs is through attention mechanisms [16, 17] that use a variety of heuristics (e.g., RNN weights [18] or gradients [17]) or additional neural networks ([19–21]) to estimate the amount of attention, or importance, the RNN places on each timestep of an input sequence. Another method of explaining RNNs is through generating adversarial examples [16]: existing generation methods include gradient descent [17] and causal regression [22]. As for spline theory, Wang et al. [23] proposed some insights into the functional nature of RNNs and hidden state initialization, but since then RNN spline theory has received little attention. Pre-existing works have not developed a training-free method of visualizing the precise analytical form of RNNs.

2.2 Neural Networks

Feedforward. Feedforward neural networks are nonlinear operators composed from a cascade of linear and nonlinear functions. We focus on CPWL networks, in which, at each layer ℓ, linear preactivations, with weights $\boldsymbol{W}_\ell$ and bias $\boldsymbol{b}_\ell$, are fed through a (leaky) ReLU σ_ℓ:

$$f_\ell(\boldsymbol{X}) = \sigma_\ell(f_{\ell-1}(\boldsymbol{X})\boldsymbol{W}_\ell + b_\ell), \quad f_0(\boldsymbol{X}) = \boldsymbol{X} \tag{1}$$

Recurrent. Unlike feedforward networks, RNNs use both feedforward layers (Equation 1) and recurrent layers that process inputs sequentially. A simple recurrent layer with parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ has a hidden state h_k which is updated at each timestep by a nonlinear function $\sigma_h(\cdot)$[2] of its previous value h_{k-1} and the newest input signal u_k. The value of h_k is then transmitted to the next layers in an RNN via the processed hidden state y_k:

$$h_k = \sigma_h(\boldsymbol{A}h_{k-1} + \boldsymbol{B}u_k), \quad y_k = \boldsymbol{C}h_k + \boldsymbol{D}u_k \tag{2}$$

Linear Recurrent. Performant RNNs traditionally used nonlinear recurrences $\sigma_h(\cdot)$ (Equation 2), even if it incurred gradient instability and computational inefficiency [8, 12]. However, recent work has shown that "linear RNNs"—RNNs with linear recurrences ($\sigma_h(x) = x$), careful initialization, and specific parameterization—can match and out-perform traditional RNNs with nonlinear recurrences via temporal compression of information, gradient stabilization, and parallelization [8, 12, 25].

3 Methods

Models, Data, and Unrolling. We train linear RNNs on two biosignal classification datasets—a PhysioNet electrocardiograph (ECG) dataset, and an EMG dataset—and the sequential MNIST dataset. Our RNNs use the parameterization recently perscribed by Orvieto et al. [8], and the classification decision for each sample is taken solely from the logits of the final timestep. The convolutional representation of linear RNNs allows us to unroll each recurrent layer into a simple feedforward layer, which is compatible with SplineCam. Our work is focused on linear RNNs because RNNs with nonlinear recurrences are far more computationally expensive to unroll.

[2]The RNN is CPWL if and only if $\sigma_h(\cdot)$ is CPWL. Gated RNNs (e.g., GRUs, LSTMs) [24] are not CPWL.

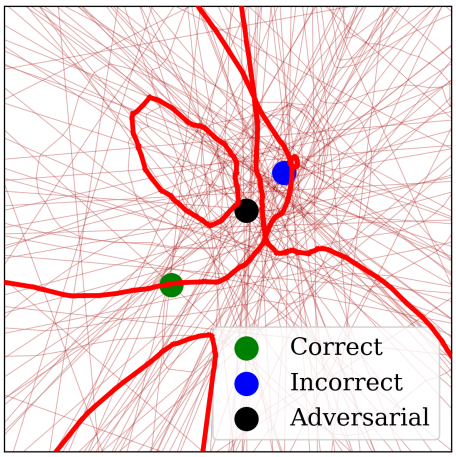

Figure 1: **SplineCam-Linear-RNN can illustrate the decision boundary and complexity of a linear RNN around two multiple-channel samples.** Plotted is a random 2D projection, from SplineCam-Linear-RNN, of a linear RNN trained on EMG biosignal data. The 2D projection is anchored around two sequences from the same class: one correctly, and one incorrectly classified by the RNN. Through SplineCam-Linear-RNN, we can: (1) observe the complexity (partition density, thin red lines) around data samples, and (2) sample adversarial examples near the decision boundary (thick red lines). The two anchor sequences and the adversarial sequence in between them are in Figure 3.

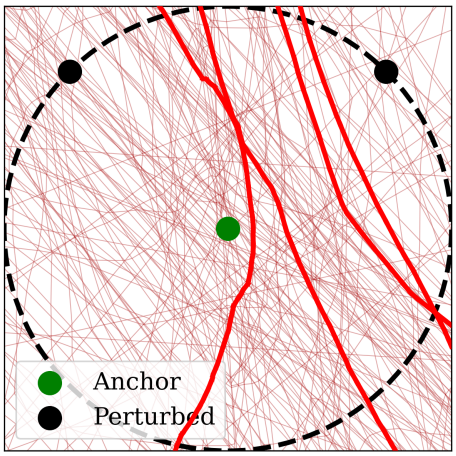

Figure 2: **SplineCam-Linear-RNN can illustrate the frequency-sensitivity of a linear RNN around a sequential MNIST data sample.** Here, instead of random 2D projections (Figure 1), we use sinusoidal projections of a fixed frequency, anchored around a single sequential MNIST sample (green). Coordinates on the dashed circle are sinusoidal perturbations of the same frequency and amplitude, but different phases. We see that the linear RNN is sensitive to the phase of the sinusoidal perturbation applied to the anchor image: one perturbation (black) is on the same side of the decision boundary as the anchor, while the other is not. The anchor and its sinusoidal perturbations are in Figure 4.

Frequency-Sensitivity. SplineCam typically depicts a neural network on a random 2D projection of its input domain, centered around one or two data samples [7]. We propose to also plot the behavior of a neural network around a 2D sinusoidal projection of the input domain centered around a single data sample. Specifically, the horizontal and vertical dimensions of the projection correspond to cosine and sine perturbations, of the same frequency, that are applied to the given data sample. By plotting the RNN's behavior with respect to sinusoidal perturbations of the same frequency, we can see how sinusoidal perturbations of a given frequency—but any phase, akin to a *unit circle* of a given frequency—affect the complexity and decision of the RNN around a single data sample; in other words, for a given frequency and data sample, we can see the *frequency sensitivity* of an RNN.

4 Results

4.1 Illustrating the Decision Boundary and Complexity of Linear RNNs

Figures 1 and 3 show how SplineCam-Linear-RNN visualizes the complexity and decision boundary of linear RNNs and generates adversarial samples. Essentially, SplineCam-Linear-RNN calculates a 2D projection of the exact analytical form of a linear RNN around one or two data samples, and then uses this analytical form to generate adversarial samples and exactly illustrate the complexity and behavior of the linear RNN near data samples. In any SplineCam plot, each thin line is a 2D projection of the border between piecewise-linear regions of a neuron's behavior—for (leaky) ReLU networks, this border is where the preactivation of a given neuron equals 0.

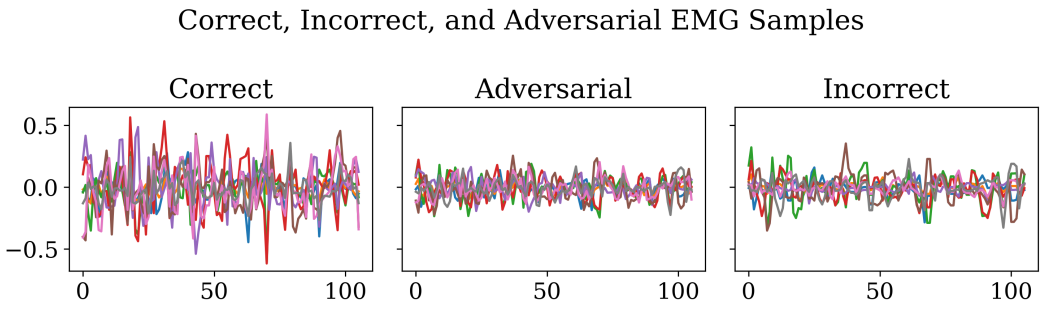

Figure 3: **SplineCam-Linear-RNN can sample adversarial samples between two multi-channel time series.** Here are the EMG samples from Figure 1. Each color corresponds to a different channel, and the x-axis simply corresponds to the temporal dimension.

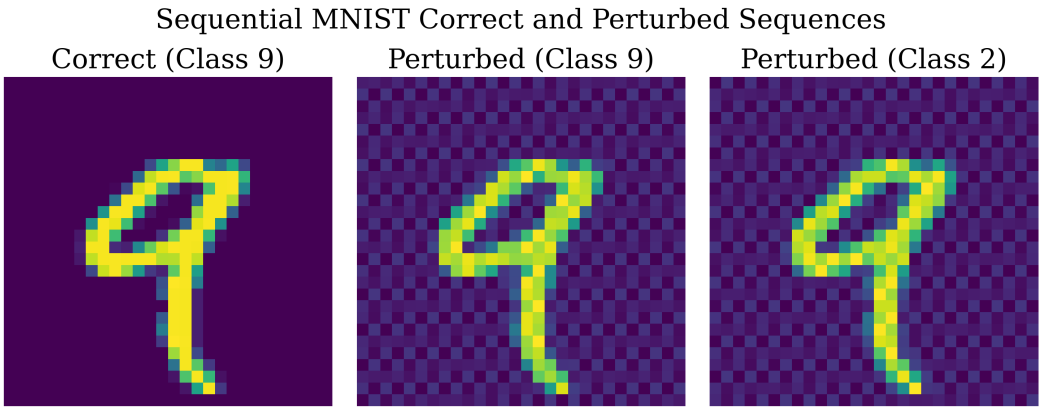

Figure 4: **SplineCam-Linear-RNN can find adversarial samples through frequency-sensitivity.** Here are one correctly classified sequence and two perturbed sequences with the same perturbation frequency, but different phases. The difference between the two perturbation phases is minimal, but changes the classification decision. These perturbations are from SplineCam-Linear-RNN (Figure 2). These images were originally sequential MNIST sequences, but have been reshaped for visualization.

4.2 Probing the Frequency-Sensitivity of Linear RNNs

Aside from illustrating the spline partitions and decision boundary of a neural network along a random 2D projection of the input plane, SplineCam-Linear-RNN can also illustrate the behavior of a neural network along a sinusoidal 2D projection of a given frequency and variable phase, akin to a *unit circle* of fixed frequency around a given data sample. The resulting visualizations are shown in Figures 2 and 4: they show how the behavior of linear RNNs can vary with respect to the frequency chosen.

5 Conclusions and Future Work

We have presented SplineCam-Linear-RNN, a holistic algorithm that covers how to unroll, visualize, and understand RNNs with linear recurrences. Unlike any other existing algorithms, SplineCam-Linear-RNN can: (1) rigorously visualize the complexity and decision boundary of an RNN, and (2) visualize the frequency-sensitivity of RNNs around given data samples and frequencies. Future works include: (a) understanding whether the nature of LRU unrolling induces any specific properties in decision boundary arrangements compared to vanilla convolutional or feedforward networks; (b) applying SplineCam-Linear-RNN to regression tasks; (c) using other 2D projection bases relevant to sequential data, such as wavelets; (d) using selective 2D projections that can focus on certain timesteps or regions of interest; and (e) visualizing how the 2D projection of a data sample moves as the data sample evolves over time, perhaps by projecting sliding windows of a data sample. Additionally, extending SplineCam-Linear-RNN to RNNs with nonlinear recurrences would be interesting.

6 Acknowledgements

This work was supported by NSF grants CCF-1911094 and IIS-1730574; ONR grants N00014-23-1-2714, N00014-24-1-2225, and MURI N00014-20-1-2787; AFOSR grant FA9550-22-1-0060; DOE grant DE-SC0020345; DOI grant 140D0423C0076; and a Vannevar Bush Faculty Fellowship, ONR grant N00014-18-1-2047.

References

[1] Ibomoiye Domor Mienye, Theo G. Swart, and George Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, 15(9), 2024. doi: 10.3390/info15090517.

[2] Yassin Khalifa, Danilo Mandic, and Ervin Sejdić. A review of hidden markov models and recurrent neural networks for event detection and localization in biomedical signals. *Information Fusion*, 69, 2021. doi: https://doi.org/10.1016/j.inffus.2020.11.008.

[3] Shitong Mao and Ervin Sejdić. A review of recurrent neural network-based methods in computational physiology. *IEEE Transactions on Neural Networks and Learning Systems*, 34 (10), 2022.

[4] Ramya S Nair and P Supriya. Robotic path planning using recurrent neural networks. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2020.

[5] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1), 2021.

[6] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, 2018.

[7] Ahmed Imtiaz Humayun, Randall Balestriero, Guha Balakrishnan, and Richard G. Baraniuk. SplineCam: Exact visualization and characterization of deep network geometry and decision boundaries. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[8] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning (ICML)*, 2023.

[9] Gari D Clifford, Chengyu Liu, Benjamin Moody, H Lehman Li-wei, Ikaro Silva, Qiao Li, AE Johnson, and Roger G Mark. AF classification from a short single lead ECG recording: The PhysioNet/computing in cardiology challenge 2017. In *2017 Computing in Cardiology (CinC)*, 2017.

[10] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[11] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2016.

[12] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022.

[13] Randall Balestriero and Richard G Baraniuk. Mad max: Affine spline insights into deep learning. *Proceedings of the IEEE*, 109(5), 2020.

[14] Randall Balestriero and Richard G Baraniuk. Batch normalization explained. *arXiv preprint arXiv:2209.14778*, 2022.

[15] Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Deep networks always grok and here is why. *arXiv preprint arXiv:2402.15555*, 2024.

[16] Thomas Rojat, Raphaël Puget, David Filliat, Javier Del Ser, Rodolphe Gelin, and Natalia Díaz-Rodríguez. Explainable artificial intelligence (XAI) on timeseries data: A survey. *arXiv preprint arXiv:2104.00950*, 2021.

[17] Ziqi Zhao, Yucheng Shi, Shushan Wu, Fan Yang, Wenzhan Song, and Ninghao Liu. Interpretation of time-series deep models: A survey. *arXiv preprint arXiv:2305.14582*, 2023.

[18] Wendong Ge, Jin-Won Huh, Yu Rang Park, Jae-Ho Lee, Young-Hak Kim, and Alexander Turchin. An interpretable icu mortality prediction model based on logistic regression and recurrent neural networks with lstm units. In *AMIA Annual Symposium Proceedings*, volume 2018, 2018.

[19] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. Lstm fully convolutional networks for time series classification. *IEEE Access*, 6, 2017.

[20] Cedric Schockaert, Reinhard Leperlier, and Assaad Moawad. Attention mechanism for multivariate time series recurrent model interpretability applied to the ironmaking industry. *arXiv preprint arXiv:2007.12617*, 2020.

[21] Cj Barberan, Sina Alemmohammad, Naiming Liu, Randall Balestriero, and Richard Baraniuk. NeuroView-RNN: It's about time. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, 2022. doi: 10.1145/3531146.3533224.

[22] Aditya Chattopadhyay, Piyushi Manupriya, Anirban Sarkar, and Vineeth N Balasubramanian. Neural network attributions: A causal perspective. In *International Conference on Machine Learning (ICML)*, 2019.

[23] Zichao Wang, Randall Balestriero, and Richard Baraniuk. A max-affine spline perspective of recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[24] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7), 2019.

[25] Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *International Conference on Learning Representations (ICLR)*, 2023.