Neural computation

https://doi.org/10.1038/s42256-023-00674-w

A 'programming' framework for recurrent neural networks

Manuel Beiran, Camille A. Spencer-Salmon & Kanaka Rajan

A 'programming'-like approach provides a one-step algorithm to find network parameters for recurrent neural networks that can model complex dynamical systems.

Recurrent neural networks (RNNs) are foundational tools for performing tasks that require time-dependent computation. They make use of recurrent connections between individual network units to process sequential inputs and generate a wide range of dynamics. Moreover, they are ideal for tasks that require extensive distributed parallel processing such as determining whether a given maze has a solution[1]. However, the major obstacle to using RNNs for such complex computations is finding the right set of network parameters for each task. The most common approaches optimize recurrent networks via supervised training algorithms in which the network is simulated many times, processing a large number of different input–output pairs, and slowly correcting errors. In this issue of *Nature Machine Intelligence*, Kim and Bassett focus on an alternative approach, adopting tools from neuroscience and software development[2]. Their work presents an innovative method to design RNNs that are capable of performing complex computations based on the direct engineering of network parameters. Unlike standard training techniques, this one-step algorithm determines appropriate network parameters analytically, effectively 'programming' network computations in a manner analogous to how programming languages work on computer hardware.

The fact that RNNs (in particular, reservoir computers) can universally approximate any dynamical system has been shown theoretically decades ago[3,4]. However, this result lacked one important ingredient for practical use: how to find the right network parameters for universally approximating dynamics. The current standard training approach of defining a loss function and adjusting network parameters along the gradient direction, called backpropagation, has been used with remarkable success in fields such as language processing and systems neuroscience. This success is due among other factors to the availability of backpropagation algorithms (ADAM[5]), accessible software libraries for implementation (for example, pytorch[6]), progress in computing hardware, and careful selection of network architectures and hyperparameters. Perhaps overshadowed by the empirical success of this approach, alternative techniques for designing and optimizing neural networks, such as the neural engineering framework by Eliasmith et al.[7], have received comparatively less attention. Neural engineering can find suitable network parameters via a one-step method that avoids simulating the network multiple times. Inspired by this approach, Kim and Bassett propose a specific 'programming' algorithm that allows recurrent neural networks to solve tasks. This algorithm complements popular gradient-based training paradigms. While programming

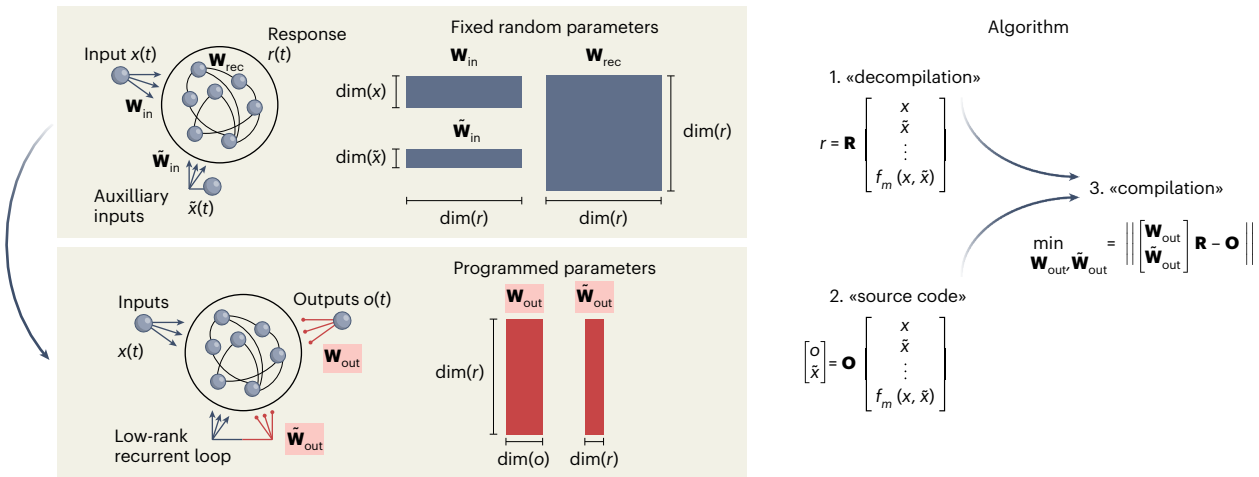

Fig. 1 | Programming of a recurrent neural network. Without needing to simulate multiple training epochs, Kim and Bassett propose a method for constructing RNNs to perform computational tasks. Given an RNN (top left) with fixed recurrent connectivity $\mathbf{W}_{rec}$ receiving (possibly time-varying) inputs x and auxiliary inputs $\tilde{x}$ through input weights $\mathbf{W}_{in}$ and $\tilde{\mathbf{W}}_{in}$ respectively, the network produces an initial response $r(t)$. The 'programming' algorithm then finds the output weights for the readout $\mathbf{W}_{out}$ and the recurrent loop $\tilde{\mathbf{W}}_{out}$ to obtain the desired time-varying outputs $o(t)$(bottom left). The algorithm (right) leverages the relationship between the response of the hidden units in the network and the inputs and nonlinear combinations f_m of the inputs (1). This relationship is described through the matrix $\mathbf{R}$. In parallel, the desired outputs can be as well mapped to the inputs, through the matrix $\mathbf{O}$ (2). Finally, the mismatch between the desired and initial output are minimized, such that the programmed RNN matches both constraints (3).

RNNs requires a fair amount of user knowledge and its performance has not yet been shown to match training algorithms, it provides a path to operating networks in a fast, reliable and predictable manner.

Kim and Bassett's central strategy for programming RNNs lies in randomly fixing some of the network parameters and subsequently inferring the remaining learnable parameters based on the target outputs and the fixed random parameters (Fig. 1, left). This programming is implemented through a least-squares regression problem, and its solution often effectively relies on low-rank network connectivity[8]. The authors show that this method can accurately implement different dynamics by initializing the network at a fixed point, and approximating to first order the RNN's response as a function of the input and transformations of the input (Fig. 1, right). Such simple principles are then applied to build RNNs for a deft set of applications ranging from implementation of virtual machines, to logic gates and ping-pong video games, without the need for subsequent trial-and-error corrections of the network parameters.

In recent times, research in neural networks is expanding its focus beyond performance in computation towards additional emergent properties. One such emergent property is the study of inductive biases and generalization. For many real-world applications, it is important that RNNs not only solve predefined input–output mappings, but that they do so with appropriate priors, such that these networks are capable of generalizing to novel inputs while remaining robust to noise. For instance, a network should not fail when presented with adversarial inputs, will ideally remain stable to small perturbations, and should generalize smoothly to unseen examples. However, understanding how such emergent biases are shaped by the learning algorithm is currently a challenging problem in RNNs trained via backpropagation. Alternative methods for designing RNNs with simpler algorithms, such as the 'programming' of RNNs, may also simplify the study of such inductive biases in the future.

Another exciting research direction in neural networks is the emergence of compositionality: the property by which networks solving different computations can be flexibly combined such that the resulting network can make use of the abilities of the individual networks comprising it[9,10]. A promising result from Kim and Bassett is that they show how a network programmed to act as a logic gate can be combined with others to implement more complex strategies. Future work will determine whether and how such emergent properties like generalization and modularity can be guaranteed with some specific programming code for RNNs, or whether other classes of neural architectures or training algorithms are required.

Overall, the creative approach by Kim and Bassett invokes a vast range of possibilities for engineering recurrent neural networks beyond gradient-based training. While the net advantages of such alternative algorithms will need to be considered in the context of their specific application (in terms of available hardware, end-goal, desired robustness and so on), only by considering a variety of complementary options will we be able to unlock the full potential of RNNs.

Manuel Beiran [1,2] ✉, **Camille A. Spencer-Salmon**[2] & **Kanaka Rajan** [2] ✉

[1]Mortimer B. Zuckerman Mind Brain Behavior Institute, Columbia University, New York, NY, USA. [2]Nash Family Department of Neuroscience, Icahn School of Medicine, Mount Sinai, New York, NY, USA.

✉e-mail: mb4878@columbia.edu; kanaka.rajan@mssm.edu

Published online: 12 June 2023

References

1. Larsen, B. W. & Druckmann, S. *PLoS Comput. Biol.* **18**, e1010227 (2022).
2. Kim, J. Z. & Bassett, D. S. *Nat. Mach. Intell.* https://doi.org/10.1038/s42256-023-00668-8 (2023).
3. Doya, K. *IEEE Transactions on Neural Networks* **1**, 1–6 (1993).
4. Maass, W., Joshi, P. & Sontag, E. D. *PLoS Comput. Biol.* **3**, e165 (2007).
5. Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. In *Proc. 3rd International Conference on Learning Representations (ICLR)* (ICLR, 2015)
6. Paszke, A. et al. *NIPS 2017 Autodiff Workshop* (2017).
7. Eliasmith, C. & Anderson, C. H. Neural engineering: Computation, representation, and dynamics in neurobiological systems. (MIT Press, 2003).
8. Beiran, M., Dubreuil, A., Valente, A., Mastrogiuseppe, F. & Ostojic, S. *Neural Comput.* **33**, 1572–1615 (2021).
9. Yang, G. R., Joglekar, M. R., Song, H. F., Newsome, W. T. & Wang, X.-J. *Nat. Neurosci.* **22**, 297–306 (2019).
10. Márton, C. D., Gagnon, L., Lajoie, G. & Rajan, K. Preprint at *arXiv* https://doi.org/10.48550/arXiv.2105.14108 (2021).

Competing interests

The authors declare no competing interests.