The Journal of Open Source Software

nnR: A package to perform algebraic operations on neural networks

Shakil Rafi [1,2*] and **Joshua Lee Padgett** [3*]

1 Department of Mathematical Sciences, University of Arkansas, Fayetteville, AR, 72701. **2** Department of Data Science, University of Arkansas, Fayetteville, AR, 72701. **3** Toyota Financial Services, Plano, TX, 75024. ***** These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 20 February 2024
Published: unpublished

Summary

There exists an algebraic way of viewing feedforward multi-layer perceptrons as an ordered tuple of ordered pairs. That is to say a neural network is $\nu = ((W_1, b_1), (W_2, b_2), ...(W_L, b_L))$, for some collection of weight matrices W_i and bias vectors b_i.

These were first introduced in Petersen & Voigtlaender (2018) , extended in (Grohs et al., 2018), and (Jentzen et al., 2023), and along with that several operations such as composition, stacking, scalar left and right multiplication, neural network sums and finally neural networks for squaring and products of real numbers were introduced.

This was extended in (Rafi et al., 2024) to include neural networks for raising to a power, neural network polynomials, neural network exponentials, sines, cosines, and a rudimentary trapezoidal rule implementation.

This is not the standard way neural networks are envisioned, which are as piecewise linear approximants. This way of building out networks that approximate transcendental functions, using algebraic operations of sum, scalar multiplication, stacking, products, and raising to powers, differs markedly from deep learning orthodoxy. It is hoped that this package will find use in the community.

A full explanation of what these algebraic operations are, and what these neural network sines, cosines, and exponents are is explained in (Rafi et al., 2024).

Statement of need

Despite this framework's wide use in e.g. (Grohs et al., 2018), (Grohs et al., 2022), and (Ackermann et al., 2023), in addition to (Grohs et al., 2023) and (Jentzen et al., 2023), this framework has lacked a consistent implementation in a widely available language.

A custom solution is needed as widely available artificial neural network software like PyTorch or TensorFlow are inadequate to the task. Specifically the authors believe that operations such as the Pwr network with its associated tunnel networks cannot adequately be described using the layer structure that PyTorch or indeed TensorFlow require. This is because of the way this network is defined, purely in terms of its associated parameters q and ε. This would lead to an uncountably infinitely many versions of the same network, each needing to be implemented separately.

In addition, this framework only envisions neural networks whose weight and bias parameters are already given. Typical neural network libraries require training via backpropagation, and it is somewhat difficult to see how one may implement a neural network in these common libraries without actually specifying backpropagation.

Finally, all neural network architectures posited to exist in (Rafi et al., 2024), such as Pwr, Xpn, Csn, Sne, and the 1-D interpolation scheme have associated with theme extensive, and sometimes very convoluted bounds for parameter, depth, and accuracy. It is nice to have an implementation in R if only to run simulations, but also to test out some of these bounds that are proposed, and further as pedagogical tools to promote a wider dissemination of this rather new neural network calculus.

As a bonus, R's widely known and easy to use infix notation with %function% allows us to translate directly, theorems in (Rafi et al., 2024), (Grohs et al., 2023), and (Jentzen et al., 2023) to software. For instance $\frac{1}{2} \triangleright (\nu_1 \bullet \nu_2)$ is rendered directly as `0.5 %slm% (nu_1 %comp% nu_2)`.

A concerted effort has been made to make this software as fast as possible, with R vectorization. Future work will be focused on re-implementing this, but using Rcpp. The source code repository for nnR has been archived to Zenodo with https://doi.org/10.5281/zenodo.10672209

Acknowledgements

This work was conducted fully or in part on computational resources at the Arkansas High Performance Computing Center

References

Ackermann, J., Jentzen, A., Kruse, T., Kuckuck, B., & Padgett, J. L. (2023). *Deep neural networks with ReLU, leaky ReLU, and softplus activation provably overcome the curse of dimensionality for kolmogorov partial differential equations with lipschitz nonlinearities in the L^p-sense*. https://arxiv.org/abs/2309.13722

Grohs, P., Hornung, F., Jentzen, A., & Wurstemberger, P. von. (2018). *A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations* (Papers No. 1809.02362). arXiv.org. https://ideas.repec.org/p/arx/papers/1809.02362.html

Grohs, P., Hornung, F., Jentzen, A., & Zimmermann, P. (2023). Space-time error estimates for deep neural network approximations for differential equations. *Advances in Computational Mathematics*, *49*(1), 4. https://doi.org/10.1007/s10444-022-09970-2

Grohs, P., Jentzen, A., & Salimova, D. (2022). Deep neural network approximations for solutions of PDEs based on monte carlo algorithms. *Partial Differential Equations and Applications*, *3*(4). https://doi.org/10.1007/s42985-021-00100-z

Jentzen, A., Kuckuck, B., & Wurstemberger, P. von. (2023). *Mathematical introduction to deep learning: Methods, implementations, and theory*. https://arxiv.org/abs/2310.20360

Petersen, P., & Voigtlaender, F. (2018). Optimal approximation of piecewise smooth functions using deep ReLU neural networks. *Neural Netw*, *108*, 296–330. https://doi.org/10.1016/j.neunet.2018.08.019

Rafi, S., Padgett, J. L., & Nakarmi, U. (2024). Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials. In *arXiv.org*. https://arxiv.org/abs/2402.01058v1