ECOGRAPHY

Software Note

'cito': an R package for training neural networks using 'torch'

Christian Amesöder[1,2], **Florian Hartig**[1] and **Maximilian Pichler**[1]

[1]Theoretical Ecology, University of Regensburg, Regensburg, Germany
[2]Information Systems, University of Regensburg, Regensburg, Germany

Correspondence: Maximilian Pichler (maximilian.pichler@biologie.uni-regensburg.de)

Ecography
2024: e07143
doi: 10.1111/ecog.07143
Subject Editor:
F. Guillaume Blanchet
Editor-in-Chief: Miguel Araújo
Accepted 20 February 2024

Deep neural networks (DNN) have become a central method in ecology. To build and train DNNs in deep learning (DL) applications, most users rely on one of the major deep learning frameworks, in particular PyTorch or TensorFlow. Using these frameworks, however, requires substantial experience and time. Here, we present 'cito', a user-friendly R package for DL that allows specifying DNNs in the familiar formula syntax used by many R packages. To fit the models, 'cito' takes advantage of the numerically optimized 'torch' library, including the ability to switch between training models on the CPU or the graphics processing unit (GPU) which allows the efficient training of large DNNs. Moreover, 'cito' includes many user-friendly functions for model plotting and analysis, including explainable AI (xAI) metrics for effect sizes and variable importance. All xAI metrics as well as predictions can optionally be bootstrapped to generate confidence intervals, including p-values. To showcase a typical analysis pipeline using 'cito', with its built-in xAI features, we built a species distribution model of the African elephant. We hope that by providing a user-friendly R framework to specify, deploy and interpret DNNs, 'cito' will make this interesting class of models more accessible to ecological data analysis. A stable version of 'cito' can be installed from the comprehensive R archive network (CRAN).

Keywords: classification, machine learning, R language, regression, species distribution model, causal inference, predictive modelling, deep learning

Introduction

Deep neural networks (DNN) are increasingly used in ecology and evolution for regression and classification tasks such as species distribution models, image classification or sound analysis (Christin et al. 2019, Joseph 2020, Strydom et al. 2021, Pichler and Hartig 2023a). State-of-the-art DNNs are almost exclusively implemented and trained in specialized deep learning (DL) frameworks such as PyTorch or Tensorflow (Abadi et al. 2016, Paszke et al. 2019). These frameworks, most of which are implemented in Python, provide flexible and performant functions and classes that allow users to implement and train complex DL architectures, such as large language models (e.g. GPT-3 (Brown et al. 2020); RoBERTA (Liu et al. 2019)) or complex object

NORDIC SOCIETY OIKOS
www.ecography.org

detection models (e.g. Mask R-CNN (He et al. 2017); DeepVit (Zhou et al. 2021)). Their high level of flexibility is appealing to 'power users', but the complexity of these frameworks can be prohibitive or at least repelling for scientists with limited knowledge in the field, and who merely want to use neural networks in standard applications.

In response to this problem, several simplified frontends for the major DL frameworks have been developed. Many of these are also available in R (www.r-project.org), the language used by most ecologists for practical data analysis. Well-known examples are 'Keras' for TensorFlow and luz for 'torch' (Allaire and Chollet 2022, Falbel 2022). Although easier to use, these frontends still follow the syntax of Python rather than the formula-based approach of R used by popular packages such as 'ranger' for training random forests or 'lme4' for fitting mixed-effects models (Bates et al. 2015, Wright and Ziegler 2017). Moreover, DL frontends such as 'Keras' or 'luz' mainly focus on model fitting and include only a very limited set of plots and convenience functions which are common to most R packages. As a result, working with these frontends still requires a considerable amount of training, especially because users have to use additional packages or program code for downstream tasks such as bootstrapping, plots or explainable AI (xAI) metrics by hand.

Besides the mentioned frontends to the large DL frameworks, some specialized R packages for training DNNs exist that more closely adhere to the user interface of other popular R packages, and that use in particular the R formula syntax to specify the model structure. However, these more user-friendly packages often lack crucial functionalities, and most of them do not make use of state-of-the-art DL frameworks for model fitting. For example, R packages such as 'nnet' or 'neuralnet' do not support modern DL techniques, such as advanced regularization techniques (e.g. dropout) to control the bias-variance tradeoff (Venables and Ripley 2002, Fritsch et al. 2019), modern training techniques such as early stopping or learning rate schedulers that help to achieve convergence, or the ability to train large models on GPUs, which can make training inefficient. The 'h2o' package comes with its own Java backend, and while it allows specifying models with the formula syntax, its use in R is cumbersome due to its inability to work with standard R objects (LeDell et al. 2022). The 'brulee' R package (Kuhn and Falbel 2022), which uses 'torch' to train the DNNs specified in standard R syntax, is very similar to the package presented here, but still lacks some features that we consider critical (see section 'Design of the 'cito' package').

Here, we present 'cito', an R package based on the 'torch' DL framework for training fully connected neural networks. 'cito' allows flexible specification of neural networks architectures using the standard R formula syntax for model specification, supports many modern DL techniques (e.g. dropout and elastic net regularization, learning rate schedulers), can take advantage of CPU and GPU hardware for parallelization and, despite its simple user interface, optionally offers a high degree of customization such as user-defined loss functions. Moreover, 'cito' supports many downstream functionalities, such as the possibility to continue the training of existing DNNs with modified training parameters for fine-tuning, or the application of xAI methods to interpret the trained models. As such, 'cito' provides a user-friendly but nevertheless complete analysis pipeline for building neural networks in R.

In the remainder of the paper, we introduce the design principles of 'cito' in more detail, show validation and performance analysis, and showcase the application of 'cito' using it to model the distribution of the African elephant.

Design of the 'cito' package

'torch' backend

'cito' uses 'torch', a variant of PyTorch, as its backend to represent and train the specified neural networks. Until recently, R users who wanted to use PyTorch and Tensorflow had to call their Python bindings through the 'reticulate' package. R packages that relied on this pipeline were thus dependent on appropriate Python installations (e.g. Pichler and Hartig 2021), which often created dependency issues. This issue was solved with the release of 'torch', a native implementation of the 'torch' libraries with an R frontend (Falbel and Luraschi 2022).

Building and training neural networks in 'cito'

With 'torch', R users can essentially use PyTorch functions natively in R, which solves dependency issues, but not the problem that specifying a DNN with 'torch' is complex.

'cito' addresses this problem by providing a simple function, *dnn()*, which combines everything needed to build and train a fully connected neural network in one line of code (see function *vignette('A_Introduction_to_cito')* for more details). The *dnn()* function includes options to modify the network architecture and training process.

After fitting the model, the *dnn()* function returns the trained model as an S3 object that can be used in various ways. One option is the *continue_training()* function, which continues the training for additional epochs (iterations) with the same or modified training hyperparameters or data. Moreover, many standard R functions such as *summary()*, *predict()* or *residuals()* are implemented for the trained models, and additional specialized xAI functions are available for interpreting the fitted networks. More details on these and other functions are available in the R function vignettes provided with the 'cito' package.

Arguments that configure the architecture and influence training are called hyperparameters (Table 1). While they can be crucial to prevent overfitting by controlling the functional complexity of the DNN via regularization, they can also prevent convergence if set to the wrong values. For example, hyperparameters such as the learning rate depend on the scale of the response variable in regression tasks, where inappropriate learning rates can lead to inefficient training. We have tried to find reasonable default values for all hyperparameters in

Table 1: Arguments, including hyperparameters that control the learning process and the architecture of the final network, and their default values in *dnn()*. The defaults for all arguments are set to reasonable values; however, hyperparameters (e.g. learning rate, hidden, activation, bias, epochs, batchsize, lr, lambda, and alpha) typically need to be tuned. Detailed guidance on this can be found in the help file of the *dnn()* function or in the 'cito' R package *vignette('Training_neural_networks')*.

Name	Explanation	Default
Architecture		
hidden	Quantity and size of hidden layers	(50, 50)
activation	Activation function for hidden layers	'selu'
bias	Should hidden nodes have bias	TRUE
Training		
validation	Split data into training and validation set to monitor training	0
epochs	Number of training iterations	100
device	Set to 'cuda' to train on GPU	'cpu'
plot	Visualize loss during training	TRUE
batchsize	Number of samples used for each training step	32
shuffle	Shuffle batches in between epochs	TRUE
lr	Learning rate	0.01
early_stopping	Stops training early based on validation loss	FALSE
bootstrap	Number of bootstrap samples	FALSE
Controlling bias-variance trade-off (regularization)		
lambda	Strength of elastic net regularization	0
alpha	Split of L1 and L2 regularization	0.5
dropout	Dropout probability of a node	0

the *dnn()* function that should be able to handle many datasets, but there is no guarantee that this will always work. In this case, the user must manually adjust the hyperparameters (e.g. the learning rate) as needed. Once convergence issues are solved, hyperparameters are ideally tuned under cross-validation to find the regularization strength that optimizes the bias-variance trade-off (e.g. Pichler and Hartig 2023a).

To help users detect problems with non-convergence or overfitting, 'cito' provides detailed visual feedback on training and validation loss. Unlike other packages (Table 2), we provide a baseline loss (intercept-only model) which can be compared to the training loss to detect poor convergence. The *vignette('B-Training_neural_networks')* explains in detail how to detect and fix such problems.

The lack of uncertainties (standard errors) is an often raised concern for DNNs (Gawlikowski et al. 2021). In 'cito', we provide an option to automatically calculate confidence intervals for all outputs (including xAI metrics and predictions) using bootstrapping. Bootstrapping can be enabled in the *dnn()* function setting, e.g. *dnn(… ,bootstrap = 50)*. Bootstrap standard errors are then automatically propagated through all downstream methods and are also used to generate p-values wherever obvious null hypotheses exist. As bootstrapping can be computationally expensive, we recommend training first without bootstrapping (which is the default) to optimize the training procedure (Fig. 1) and to then enable the bootstrap for the final model after the training pipeline has been finalized.

Performance comparison and validation of 'cito'

After explaining the design of 'cito', we briefly compare its performance and functionality with other packages for implementing neural networks in R. In particular, we consider 'nnet' and 'neuralnet', which each have their own backend

Table 2. Feature comparison of R packages used to build fully connected neural networks.

	'cito'	'brulee'	'h2o'	'neuralnet'	'nnet'
Customizable network architecture	X	X	X	X	
Fit a probability distribution	X		X		
GPU support	X				
Regularization	X	X	X	X	X
Custom loss function	X			X	
Optimization of additional user-defined parameters	X				
Continue training	X				
Class weights for imbalanced data		X			
Learning rate scheduler	X	X	X		
Feature importance (xAI)	X		X		
Partial dependency plots (xAI)	X				
Accumulated local effect plots (xAI)	X				
Uncertainty (confidence intervals and p-values for xAI metrics and predictions)	X				
Baseline loss (to help with the convergence)	X				

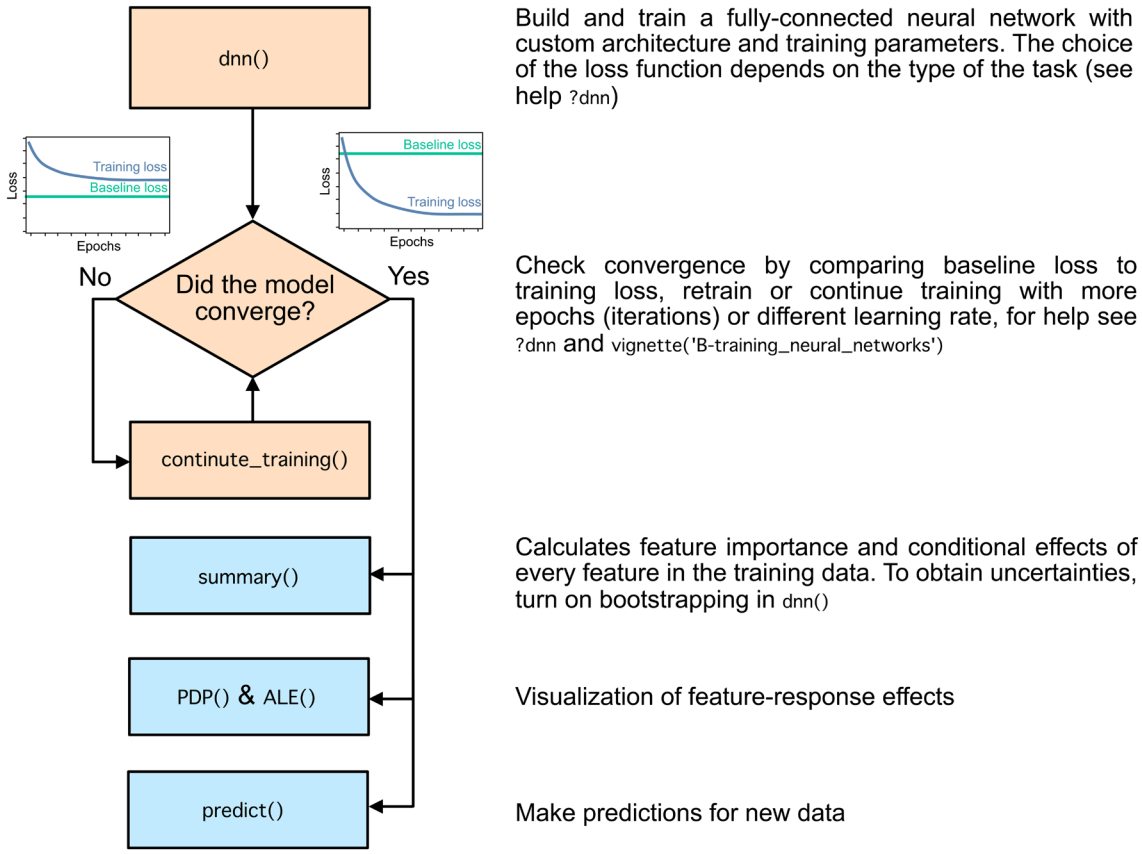

Figure 1. Workflow of building, training and analyzing DNNs with 'cito'. Example workflow and analyses for (multi) species distribution models are available as a vignette (*vignette('C -Example_Species_distribution_modeling')*) or at https://citoverse.github.io/cito/articles.

and are not based on modern DL frameworks (Venables and Ripley 2002, Fritsch et al. 2019), 'h2o', which has a much broader toolkit for training neural networks than the previous two packages (LeDell et al. 2022) and 'brulee' (Kuhn and Falbel 2022) which, similar to 'cito', uses the 'torch' DL framework as a backend.

Our comparison shows that 'cito' implements more functionalities than other packages, in particular GPU support, the possibility to continue training and custom loss functions and, most importantly, tools to interpret the trained DNN models (Table 2).

Looking at the computational performance, measured by the time it takes to train the networks, we find that some of the older packages, in particular 'neuralnet', perform better than the 'torch'-based packages ('brulee' and 'cito') for small networks (Fig. 2). This is probably due to the smaller overhead of these more specialized packages. However, when moving to larger networks (large and especially wide

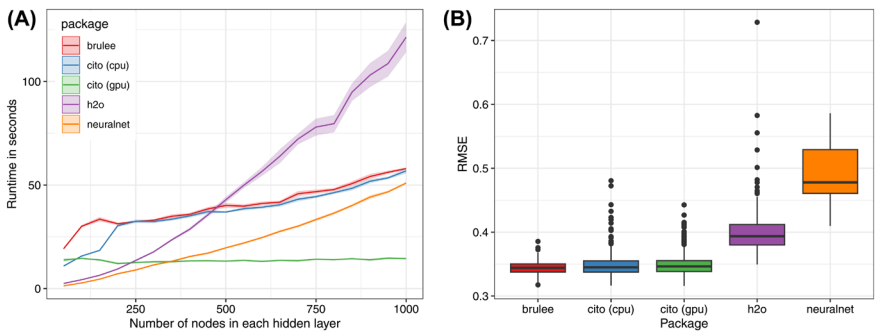

Figure 2. Comparison of runtime and predictive performance of different R deep learning software packages ('brulee', 'h2o', 'neuralnet' and 'cito' (CPU and GPU)). Panel (A) shows the runtime of each package on different network sizes on an Intel Xeon Gold 6246R and a Nvidia RTX A5000. The networks consist of five equally sized layers (50 to 1000 nodes with a step size of 50) and were trained on a simulated dataset of 1000 observations. Panel (B) shows the average root mean square error (RMSE) of the models on a holdout of size 1000 observations (RMSE was averaged over different network sizes). Each network was trained 20 times (the dataset was resampled each time).

networks are often beneficial for achieving low generalization errors (Belkin et al. 2019)), 'cito' can play out one of the main advantage of modern DL frameworks, which is GPU support. On a GPU, training time in 'cito' for our simulated datasets (1000 observations and 20 predictors) is practically independent of the size of the network, confirming the consensus that training large networks requires GPU resources. On a CPU, 'cito' performs on a par with 'brulee', the other 'torch'-based package, but somewhat worse than 'neuralnet'. We interpret these results as showing that, for a simple problem, there is still some overhead in using 'torch' as opposed to a native C implementation. Nevertheless, we would argue that the added flexibility and functionality of 'cito' outweighs this advantage of 'neuralnet'. Moreover, our results suggest that the difference between the 'torch'-based packages ('brulee' and 'cito') and 'neuralnet' lies mainly in the constant overhead needed to set up the models. For large models, their performance is about the same.

Workflow and case study

So far, we have mainly discussed the model training process, which is arguably the core of any machine learning project. Now, we want to comment on the entire workflow when using 'cito' to build and interpret a predictive model. This workflow usually consists of model specification, training, and interpretation and predictions (Fig. 1). To make the discussion of the workflow more accessible to the reader, we illustrate this workflow with the example (following Ryo et al. (2021)) of building a species distribution model (SDM) for the African elephant *Loxodonta africana*.

SDMs are niche models that correlate environment with species occurrence data (Elith and Leathwick 2009). As a case study we used African elephant records used by Ryo et al. (2021) and originally gathered by Angelov (2020), which were compiled from different studies available on the Global Biodiversity Information Facility (Musila et al. 2019, Jlegind 2021, INaturalist Contributors 2022, Navarro 2022). These presence-only data were supplemented by Angelov (2020) with randomly sampled background points (pseudo-absences) to generate a presence-absence signal for the classifier. While it is common in statistical modeling to sample more pseudo-absences than presences (Barbet-Massin et al. 2012), such imbalanced class numbers can be harmful for ML algorithms (Steen et al. 2021). We therefore randomly undersampled pseudo-absences to match the number of observations. As predictors, we used all 19 bioclimatic variables from WorldClim ver. 2 (Fourcade et al. 2018), which were centered and standardized.

Building and training a species-distribution model based on a fully connected neural network with three hidden layers of 50, 50 and 50 nodes and trains it for 50 epochs can be done in a single line of code:

```
nn.fit <- dnn(label~.,
              data=data,
              hidden=c(50, 50, 50),
              loss='binomial',
              epochs=50,
              lr=0.1,
              batchsize=300,
              validation=0.1,
              shuffle=TRUE,
              alpha=0.5,
              lambda=0.005,
              early_stopping=10,
              boostrap=30)
```

During training (without bootstrapping), a plot is displayed in R that monitors the training, validation and baseline loss. This plot can be used to diagnose convergence problems, for example if the training loss does not decrease over time or does not fall below the baseline loss. In this case, it would be advisable to abort and restart the training with different hyperparameters (e.g. lower learning rate), use a learning rate scheduler or perform a systematic hyperparameter tuning. We provide extensive help on this topic in the documentation and in a vignette (*vignette('B-Training_neural_networks')*). Here we show an example where we restart the training with a lower learning rate and a learning rate scheduler that automatically reduces the learning rate if the loss does not decrease in 8 continuous epochs (patience=8) to achieve a better fit:

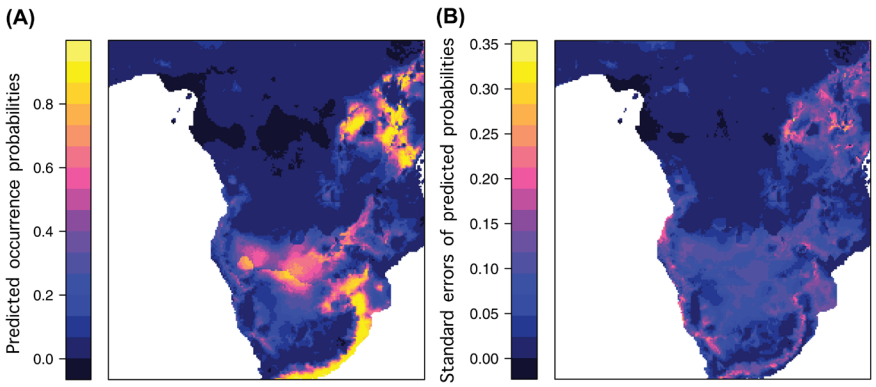

Figure 3. Predictions and standard errors of prediction for the African elephant from a DNN trained by 'cito'. Panel (A) shows the predicted probability of occurrence of the African elephant. Panel (B) shows the standard error for the predicted probabilities (confidence interval).

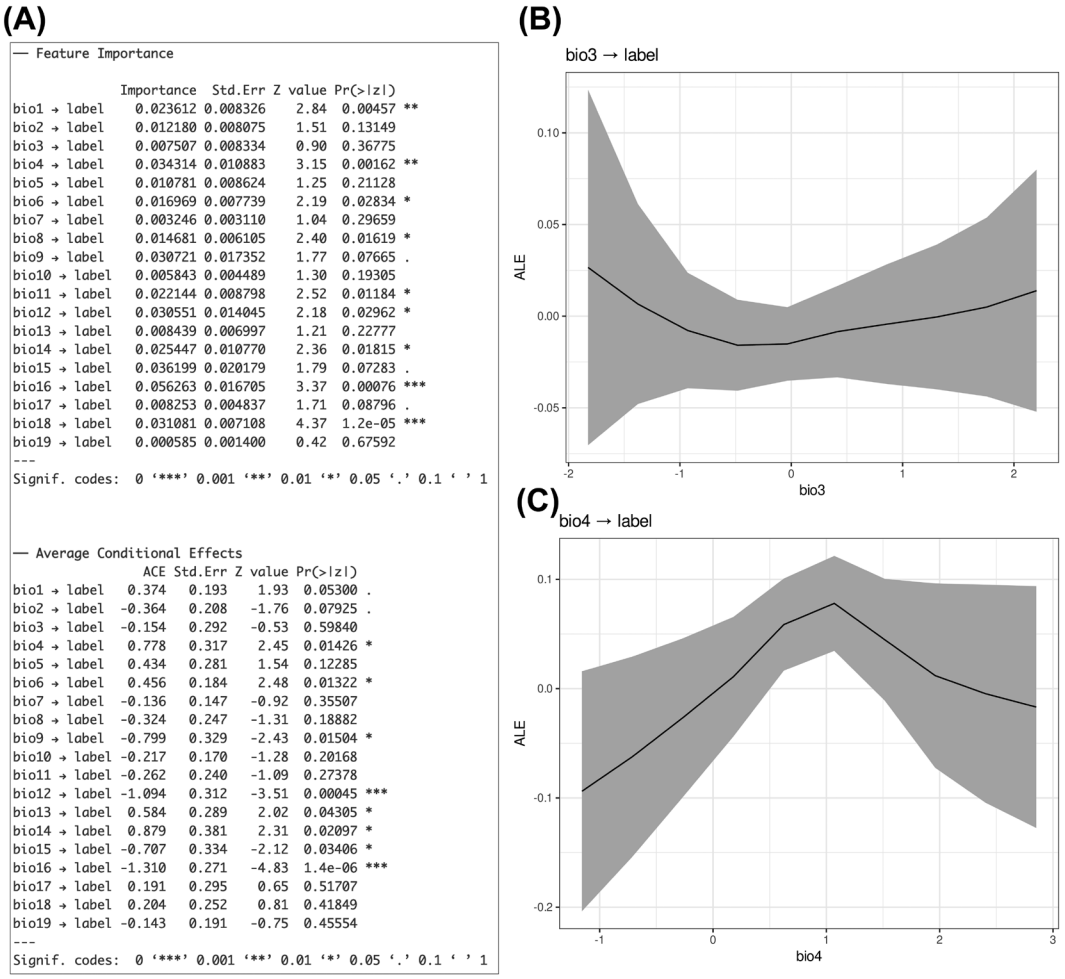

Figure 4. xAI metrics with bootstrapped confidence intervals (+/- 1 standard errors) from the model trained by 'cito'. Panel (A) shows (permutation) feature importances and average conditional effects (approximation of linear effects) from the *summary()* output for the 19 Bioclim variables. Panels (B) and (C) show the accumulated local effect plots (ALE); that is, the change of the predicted occurrence probability, for the Bioclim variables 3 (isothermality) and 4 (temperature seasonality).

```
nn.fit <- continue_training(nn.fit,
                            epochs = 150,
                            changed_params = list(lr = 0.05,
                                                  lr_scheduler = config_lr_scheduler('reduce_on_plateau'),
                                                  patience = 8,
                                                  factor = 0.8))
```

The trained models can be used with a range of in-built functions. The *predict()* function can be used to predict the occurrence probability of the elephant (Fig. 3A). The *summary()* function provides an overview about influential variables by calculating their importances (Fisher et al. 2019) as well as average conditional effects (which are an approximation of linear effects, Scholbeck et al. 2022, Pichler and Hartig 2023b, Fig. 4A). Partial dependency plots (PDP) and averaged local effect plots (ALE) functions can be used to display the effect of specific features on the response, in this case the occurrence probability of the elephant (Fig. 4B–C). If bootstrapping is enabled, 'cito' automatically uses the bootstrap samples to calculate confidence intervals (CI) for the predictions (Fig. 3B, where CI are presented using standard errors for visualization), CIs and p-values for the xAI metrics (Fig. 4A) and CIs for the PDP and ALE plots (Fig. 4B–C).

Conclusion

In conclusion, 'cito' is a powerful and versatile R package for building and training fully connected neural networks with the formula syntax. The package seamlessly integrates into the R

regression ecosystem and removes many hurdles in using neural networks for inexperienced users, but also saves programming time for experienced users who just want to build simple neural networks. The unique combination of features provided by 'cito', such as training on a GPU, using custom loss functions, baseline loss, confidence intervals, modern DL training techniques such as continuing training, learning rate scheduler or early stopping, cannot be found in other packages.

A downside of the simple user interface of 'cito' is that users can only choose between selected architectures. Currently, 'cito' only supports fully connected neural networks, a structure best suitable for tabular data, as in our SDM example. However, the development branch of 'cito' already includes an extension for convolutional neural networks (CNN) that are often used for image and sound recognition in ecology (Christin et al. 2019). While it may be possible to also offer support for some advanced architectures such as recurrent neural networks (RNN), we anticipate that expanding 'cito' to a general set of complex architectures while maintaining our simple user interface will be challenging.

Moreover, despite the simplicity of our interface, it is important to note that the models currently implemented in 'cito' are still complex compared to a linear regression, and users should make sure of understanding the effects of model assumptions and hyperparameter settings. For example, while we provide bootstrap CIs and p-values for all inferential outputs in 'cito', it is not automatically guaranteed that these will respect standard statistical expectations such as controlled type I error or nominal coverage. The reason is that algorithmic structure and hyperparameters in ML algorithms such as DNNs regularize functional complexity and will thus create simplicity biases on predictions and xAI metrics (Shah et al. 2020, Pichler and Hartig 2023b). The performance of the provided metrics should therefore ideally be tested for any specific combination of data and hyperparameters. Future development of 'cito' will aim at supporting users in these tasks by implementing additional functionalities such as internal cross-validation for hyperparameter optimization, gradient-based methods for hyperparameter tuning, and by increasing the statistical reliability of the xAI.

To cite 'cito' or acknowledge its use, cite this Software note as follows, substituting the version of the application that you used for 'version 1.0.2':

Amesöder, C., Hartig, F. and Pichler, M. 2024. 'cito': an R package for training neural networks using 'torch'. – Ecography 2024: e07143 (ver. 1.0.2).

Acknowledgements – We would like to thank Isabelle Halbhuber, Daniel Maar, and Armin Schenk, as well as Guillaume Blanchet and two anonymous reviewers for their valuable comments and suggestions. Open Access funding enabled and organized by Projekt DEAL.

Author contributions

Christian Amesöder: Conceptualization (supporting); Investigation (equal); Methodology (equal); Software (equal); Visualization (equal); Writing – original draft (lead). **Florian Hartig**: Conceptualization (equal); Investigation (equal); Methodology (equal); Supervision (equal); Visualization (equal); Writing – review and editing (supporting). **Maximilian Pichler**: Conceptualization (equal); Investigation (equal); Methodology (equal); Software (equal); Supervision (equal); Visualization (equal); Writing – review and editing (lead).

Transparent peer review

The peer review history for this article is available at https://publons.com/publon/10.1111/ecog.07143.

Data availability statement

The processed datasets for the species distribution model (African elephant) are available from Angelov (2020). We used version 1.0.2 of the 'cito' package. The 'cito' package can be downloaded from CRAN. Documentation and rendered vignettes (under articles) can be found on https://citoverse.github.io/cito/or on the CRAN website of the package at https://cran.r-project.org/web/packages/cito/index.html.

Data are available from the Zenodo Digital Repository: https://zenodo.org/records/10853334 (Amesöder et al. 2024).

References

naturgucker_de 2020. *naturgucker*. – www.gbif.org/dataset/6ac3f774-d9fb-4796-b3e9-92bf6c81c084, accessed 30 May 2020.

Abadi, M. et al. 2016. Tensorflow: large-scale machine learning on heterogeneous distributed systems. – ArXiv Prepr. ArXiv160304467, https://www.tensorflow.org/about/bib.

Allaire, J. J. and Chollet, F. 2022. keras: R Interface to "Keras". – https://cran.r-project.org/web/packages/keras/index.html.

Amesöder, C., Hartig, H. and Pichler, M. 2024 Data from: 'cito': an R package for training neural networks using 'torch'. – Zenodo Digital Repository, https://zenodo.org/records/10853334.

Angelov, B. 2020. boyanangelov/interpretable_sdm: reproducibility fix. – https://zenodo.org/records/4048271.

Barbet-Massin, M., Jiguet, F., Albert, C. H. and Thuiller, W. 2012. Selecting pseudo-absences for species distribution models: how, where and how many? – Methods Ecol. Evol. 3: 327–338.

Bates, D., Mächler, M., Bolker, B. and Walker, S. 2015. Fitting linear mixed-effects models using lme4. – J. Stat. Softw. 67: 1–48.

Belkin, M., Hsu, D., Ma, S. and Mandal, S. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. – Proc. Natl Acad. Sci. USA 116: 15849–15854.

Brown, T. et al. 2020. Language models are few-shot learners. – Adv. Neural Inf. Process. Syst. 33: 1877–1901.

Christin, S., Hervet, É. and Lecomte, N. 2019. Applications for deep learning in ecology. – Methods Ecol. Evol. 10: 1632–1644.

Elith, J. R. and Leathwick, J. R. 2009. Species distribution models: ecological explanation and prediction across space and time. – Annu. Rev. Ecol. Evol. Syst. 40: 677–697.

Falbel, D. 2022. luz: higher level "API" for "torch." – https://CRAN.R-project.org/package=luz.

Falbel, D. and Luraschi, J. 2022. torch: tensors and neural networks with "GPU" acceleration. – https://CRAN.R-project.org/package=torch.

Fisher, A., Rudin, C. and Dominici, F. 2019. All models are wrong, but many are useful: learning a variable's importance by studying an entire class of prediction models simultaneously. – J. Mach. Learn. Res. 20: 1–81.
Fourcade, Y., Besnard, A. G. and Secondi, J. 2018. Paintings predict the distribution of species, or the challenge of selecting environmental predictors and evaluation statistics. – Global Ecol. Biogeogr. 27: 245–256.
Fritsch, S., Guenther, F. and Wright, M. N. 2019. neuralnet: training of neural networks. – https://CRAN.R-project.org/package=torch.
Gawlikowski, J., Tassi, C. R. N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R., Shahzad, M., Yang, W., Bamler, R. and Zhu, X. X. 2021. A survey of uncertainty in deep neural networks. – arXiv preprint arXiv:2107.03342, https://link.springer.com/article/10.1007/s10462-023-10562-9.
He, K., Gkioxari, G., Dollar, P. and Girshick, R. 2017. Mask R-CNN. Presented at the Proceedings of the IEEE International Conference on Computer Vision, pp. 2961–2969, https://arxiv.org/abs/1703.06870.
INaturalist Contributors. 2022. iNaturalist research-grade observations. – https://www.gbif.org/dataset/50c9509d-22c7-4a22-a47d-8c48425ef4a7.
Jlegind 2021. Earth guardians weekly feed. – Questagame.
Joseph, M. B. 2020. Neural hierarchical models of ecological populations. – Ecol. Lett. 23: 734–747.
Kuhn, M. and Falbel, D. 2022. brulee: high-level modeling functions with "torch". – https://CRAN.R-project.org/package=brulee.
LeDell, E., Gill, N., Aiello, S., Fu, A., Candel, A., Click, C., Kraljevic, T., Nykodym, T., Aboyoun, P., Kurka, M. and Malohlava, M. 2022. h2o: R interface for the "H2O" scalable machine learning platform. – https://CRAN.R-project.org/package=h2o.
Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. and Stoyanov, V. 2019. RoBERTa: a robustly optimized BERT pretraining approach. – https://arxiv.org/abs/1907.11692.
Musila, S., Syingi, R., Mutavi, D., Odhiambo, K. and Masinde, S. 2019. Occurrence records of mammal species in Tana River Basin. – National Museums of Kenya.
Navarro, R. 2022. Kenya virtual museum records. – FitzPatrick Institute of African Ornithology.
Paszke, A. et al. 2019. PyTorch: an imperative style, high-performance deep learning library. – Adv. Neural Inf. Process. Syst. 32, https://arxiv.org/abs/1912.01703.
Pichler, M. and Hartig, F. 2021. A new joint species distribution model for faster and more accurate inference of species associations from big community data. – Methods Ecol. Evol. 12: 2159–2173.
Pichler, M. and Hartig, F. 2023a. Machine learning and deep learning—a review for ecologists. – Methods Ecol. Evol. 14: 994–1016.
Pichler, M. and Hartig, F. 2023b. Can predictive models be used for causal inference? – https://arxiv.org/abs/2306.10551.
Ryo, M., Angelov, B., Mammola, S., Kass, J. M., Benito, B. M. and Hartig, F. 2021. Explainable artificial intelligence enhances the ecological interpretability of black-box species distribution models. – Ecography 44: 199–205.
Scholbeck, C. A., Casalicchio, G., Molnar, C., Bischl, B. and Heumann, C. 2022. Marginal effects for non-linear prediction functions. – arXiv preprint arXiv:2201.08837, https://link.springer.com/article/10.1007/s10618-023-00993-x.
Shah, H., Tamuly, K., Raghunathan, A., Jain, P. and Netrapalli, P. 2020. The pitfalls of simplicity bias in neural networks. – Adv. Neural Inf. Process. Syst. 33: 9573–9585.
Steen, V. A., Tingley, M. W., Paton, P. W. C. and Elphick, C. S. 2021. Spatial thinning and class balancing: key choices lead to variation in the performance of species distribution models with citizen science data. – Methods Ecol. Evol. 12: 216–226.
Strydom, T., Catchen, M. D., Banville, F., Caron, D., Dansereau, G., Desjardins-Proulx, P., Forero-Muñoz, N. R., Higino, G., Mercier, B., Gonzalez, A., Gravel, D., Pollock, L. and Poisot, T. 2021. A roadmap towards predicting species interaction networks (across space and time). – Philos. Trans. R. Soc. B. 376: 20210063.
Venables, W. N. and Ripley, B. D. 2002. Modern applied statistics with S, 4th edn. – Springer.
Wright, M. N. and Ziegler, A. 2017. ranger: a fast implementation of random forests for high dimensional data in C++ and R. – J. Stat. Softw. 77: 1–17.
Zhou, D., Kang, B., Jin, X., Yang, L., Lian, X., Jiang, Z., Hou, Q. and Feng, J. 2021. DeepViT: towards deeper vision transformer. – https://arxiv.org/abs/2103.11886.