A review of R neural network packages (with NNbenchmark): accuracy and ease of use

by Salsabila Mahdi, Akshaj Verma, Christophe Dutang, Patrice Kiener, John C. Nash

Abstract

In the last three decades, neural networks (NN) have evolved from an academic topic to a common scientific computing tool. CRAN currently hosts approximately 80 packages in May 2020 involving neural network modeling, some offering more than one algorithm. However, to our knowledge, there is no comprehensive study which checks the accuracy, the reliability and the ease-of-use of those NN packages.

In this paper, we attempted to test this rather large number of packages against a common set of datasets with different levels of complexity, and to benchmark and rank them with certain metrics.

Restricting our evaluation to regression algorithms applied on the one-hidden layer perceptron and ignoring those for classification or other specialized purposes, there were approximately 60 package::algorithm pairs left to test. The criteria used in our benchmark were: (i) the accuracy, i.e. the ability to find the global minima on 13 datasets, measured by the Root Mean Square Error (RMSE) in a limited number of iterations; (ii) the speed of the training algorithm; (iii) the availability of helpful utilities; (iv) and the quality of the documentation.

We have attempted to give a score for each evaluation criterion and to rank each package::algorithm pair in a global table. Overall, 15 pairs are considered accurate and reliable and can be recommended for daily usage. Most others should be avoided as they are either less accurate, too slow, too difficult to handle, or have poor or no documentation.

To carry out this work, we developed various codes and templates, as well as the NNbenchmark package used for testing. This material is available at `https://akshajverma.com/NNbenchmarkWeb/index.html` and `https://github.com/pkR-pkR/NNbenchmark`, and can be used to verify our work and, we hope, improve both packages and their evaluation. Finally, we provide some hints and features to guide the development of an idealized neural network package for R.

Introduction

The R Project for Statistical Computing (`www.r-project.org`), as any opensource platform, relies on its contributors to keep it up to date. Neural networks (NN), inspired on the brain's own connections system, are a class of models in the growing field of machine learning for which R has a number of tools. During the last 30 years, neural networks have evolved from an academic topic to a common tool in scientific computing. Previously, neural networks were considered more theory than practice, partly because the algorithms used were computationally demanding.

As a convenience in the general conversation, the same term is used in a generic manner for different model structures and applications: multilayer perceptron for regression, multilayer perceptron for classification, multilayer perceptron for specialized applications, recurrent neural network for autoregressive time series, convolutional neural networks for dimension reduction and pattern recognition, deep neural networks for image or voice recognition. Most of the above types of neural networks can be found in R packages hosted on CRAN but without any warranty about the accuracy or the speed of computation. This is an issue as many poor algorithms are available in the literature and hence poor packages implemented on CRAN.

A neural network algorithm requires complicated calculations to improve the model control parameters. As with other optimization problems, the gradient of the chosen cost function that indicates the lack of suitability of the model is sought. This lets us improve the model by changing the parameters in the negative gradient direction. Parameters for the model are generally obtained using part of the available data (a training set) and tested on the remaining data. Modern software allows much of this work, including approximation of the gradient, to be carried out without a large effort by the the user.

The training process can generally be made more efficient if we can also approximate second derivatives of the cost function, allowing us to use its curvature via the Hessian matrix. There are a large number of approaches, of which quasi-Newton algorithms are perhaps the most common and useful. Within this group, methods based on the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm for updating the (inverse) Hessian approximation provide several well-known examples. In conducting this study, we believed that these second-order algorithms would perform better than first-order methods for fit-in-memory datasets.

Regardless of our belief, we wished to be able to conduct a thorough examination of these training algorithms in R. There are many packages, but barely any information to allow comparison. Our work, reported here, aims to provide a framework for benchmarking neural network packages. We restrict our examination to packages for R, and in this report focus on those that provide neural networks of the perceptron type, that is, one input layer, one normalized layer, one hidden layer with a nonlinear activation function that is usually the hyperbolic tangent tanh(), and one output output layer. The criteria used in our benchmark were: (i) the accuracy, i.e. the ability to find the global minima on 13 datasets in a limited number of iterations; (ii) the speed of the training algorithm; (iii) the availability of helpful utilities; (iv) and the quality of the documentation. We restricted our evaluation to regression algorithms applied on the one-hidden layer perceptron and ignored those for classification or other specialized purposes.

Neural Networks: the perceptron

Here, we give a short description of the one hidden layer perceptron. As the "layer" term suggests it, some terms come from the representation of graphs whereas some other terms come from the traditional literature on nonlinear models.

Using the graph description, a one-hidden layer neural network is made of 3 parts: (i) the layer of the input(s), (ii) the hidden layer which consists of independant neurons, each of them performing two operations: a linear combination of the inputs plus an offset, then a nonlinear function applied on this linear combination. (iii) the layer of the output(s) which is a linear combination of the output of the nonlinar functions in the hidden layer.

The nonlinear function used in the hidden layer must have the following four properties: continuous, differentiable, monotonic, bounded. The logistic function, the hyperbolic tangent function and the arctangent functions are the usual candidates.

The above description has a simple mathematical equivalence. Let us give two examples.

The model $y = a1 + a2 * tanh(a3 + a4 * x) + a5 * tanh(a6 + a7 * x) + a8 * tanh(a9 + a10 * x)$ describes a neural network with one input, three hidden neurons, one output model where x is the input, tanh() is the activation function, y is the output and $a1, .., a10$ are the parameters.

The model $y = a1 + a2 * atan(a3 + a4 * x1 + a5 * x2 + a6 * x3 + a7 * x4 + a8 * x5) + a9 * atan(a10 + a11 * x1 + a12 * x2 + a13 * x3 + a14 * x4 + a15 * x5) + a16 * atan(a17 + a18 * x1 + a19 * x2 + a20 * x3 + a21 * x4 + a22 * x5)$ describes a neural network with five inputs, three hidden neurons, one output model where x is the input, atan() is the activation function, y is the output and $a1, .., a22$ are the parameters.

In order to get large gradients at the first steps of the training algorithm, it is recommended to use normalized inputs, normalized outputs, odd functions like the hyperbolic tangent function or the arctangent function and small random values to initialize the parameters, for instance extracted from the N(0, 0.1) distribution. Such good practices help find good local minima and possibly the global minimum.

The dataset used for the training is assumed to have a number of rows much larger than the number of parameters. While «much larger» is subject to discussion, values of 3 to 5 are generally accepted. (In experimental design, some iterative strategies start with a dataset having a number of experiments/lines equal to 1.8 times the number of parameters and then increase the number of experiments to finetune the model.)

It is rather clear from the mathematical formula above that neural networks of perceptron type are nonlinear models and require for their parameter estimation some training algorithms that can handle (highly) nonlinear models. Indeed, the intrinsic and parametric curvatures of such models are usually very high and, with so many parameters, the Jacobian matrix might exhibit some collinearities between its columns and become nearly singular. As a result, appropriate algorithms for such dataset::model pairs are rather limited and well-known. They are the second-order algorithms like BFGS and Levenberg-Marquardt (and Horsehoe?).

JN??: What is the Horseshoe? Probably I know it by a different name. Also Levenberg-Marquardt is a stabilization of the Jacobian that could be applied to several algorithms. Should we say 'Levenberg-Marquardt stabilized Gauss-Newton', which is what nlsr uses.

Unfortunately, due to some simple literature on the gradient and the hype around "deep neural networks" that manipulate ultra-large models with hundreds or thousands parameters and sometimes more parameters than examples in the datasets, many papers and many R packages emphasize the use of first-order gradient algorithms. In the case of the perceptron, this is an error and the goal of this paper is to demonstrate it.

JN??: replace previous paragraph with ??

Unfortunately, there are widely-discussed articles concerning the gradient and "deep neural networks" that manipulate ultra-large models with hundreds or thousands parameters and sometimes more parameters than examples in the datasets. These, along with some R packages, emphasize the use of first-order gradient algorithms. In the case of the perceptron, we contend this is an error, and provide evidence to that effect in this paper.

Methodology

When training neural networks, we attempt to tune a set of hyperparameters so that the RMSE is minimized. When our method for such adjustment can no longer reduce the RMSE, we say that the given algorithm has converged. In practice, some implementations of algorithms require that we stop the optimization process in exceptional situations (e.g., a divide by zero), or a pre-set limit on the number of steps or elapsed time is reached. Thus the implementation may terminate even when the algorithm has not converged. At termination, we desire that the RMSE be "small". Note that other measures could be used to judge when to accept that the optimization is converged, for instance the Mean Absolute Error (MAE). However, the MAE is not used either in a convergence test nor in our overall ranking as there does not appear to be consensus on its use. See, e.g., (Willmott and Matsuura, 2005; Chai and Draxler, 2014).

In our tests, a termination limit for second-order algorithms is 200 iterations. On the other hand, first-order algorithms were set to several values, depending on how well and how fast they converged: `maxit1storderA=1000` iterations, `maxit1storderB=10000` iterations, and `maxit1storderC=100000` iterations. The full list of the maximum iteration number per package:algorithm is given in Appendix C. We were unable to completely harmonize the hyperparameters as an appropriate learning rate differed between package, despite the algorithm being similarly named.

We measure performance primarily by relative computing time between methods on a particular computing platform. We could also count measures of iterations, function evaluations or similar quantities that indicate the computing effort. We note that differences in machine architecture and in the attached libraries (e.g., BLAS choices for R) will modify our measures. We are putting our tools on a Github repository so that further evaluation can be made by ourselves and others as hardware and software evolves.

The resulting files recording performance that are in our repository were mostly generated by one of us (SM) on a Windows system build 10.0.18362.752 with an i7-8750H CPU, a Intel(R) UHD Graphics 630 and NVIDIA GeForce GTX 1060 chip, and 16 GB of RAM. The authors have multiple computers with different operating software and configurations, and our experiences on those systems appear to be consistent with the results from the test system described above.

Our research process was divided into 3 phases.

Phase 1 - Preparation of benchmark datasets

Datasets => NEED TO BE FINISHED??

All the datasets we use cannot generally be modeled using a non-iterative calculation such as Ordinary Least Squares. Varying levels of difficulty in modeling the different data sets are intended to allow us to further classify different algorithms and the packages that implement them. Sonja Surjanovic and Derek Bingham of Simon Fraser University created a useful website from which three of the multivariate datasets were drawn. We note the link, name and difficulty level of the three datasets:
- `http://www.sfu.ca/~ssurjano/fried.html` (Friedman - average)
- `http://www.sfu.ca/~ssurjano/detpep10curv.html` (Dette - medium)
- `http://www.sfu.ca/~ssurjano/ishigami.html` (Ishigami - high)
The other multivariate dataset, Ref153, was taken from ...

Three of the univariate datasets we used were taken from a website of the US National Institute for Standards and Technology (NIST): `https://www.itl.nist.gov/div898/strd/nls/nls_main.shtml`. (Gauss1 - low; Gauss2 - low; Gauss3 - average)

Univariate datasets Dmod1, Dmod2 are from ...

Dreyfus1 is a pure neural network which has no error. This can make it difficult for algorithms that assume an error exists. Dreyfus2 is Dreyfus1 with errors. NeuroOne from ...

Finally, we also consider a Simon Wood test dataset, used in (Wood, 2011) for benchmarking generalized additive models. Precisely, we consider a generation of Gaussian random variates Y_i, $i = 1, \ldots, n$ with the mean μ_i defined as

$$\mu_i = 1 + f_0(x_{i,0}) + f_1(x_{i,1}) + f_2(x_{i,2}) + f_3(x_{i,3}) + f_4(x_{i,4}) + f_0(x_{i,5})$$

and standard deviation $\sigma = 1/4$ where f_j are Simon Wood's smooth functions defined in Appendix B, $x_{i,j}$ are uniform variates and $n = 20,000$.

Packages

Using **RWsearch** (Kiener, 2020) we sought to automate the process of searching for neural network packages. All packages that have "neural network" as a keyword in the package title or in the package description were included. In May 2020, around 80 packages falls into this category. Packages **nlsr**, **minpack.lm**, **caret** were added because the former 2 are important implementations of second-order algorithms while the latter is the first cited meta package in the CRAN's task view for machine learning, `https://CRAN.R-project.org/view=MachineLearning`, as well as the dependency for some of the other packages tested. Restricting to regression analysis left us with 49 package::algorithm pairs in 2019 and 60 package::algorithm pairs in 2020.

Phase 2 - Review of packages and development of a benchmarking template

From documentation and example code, we learned that not all packages selected by the automated search fit the scope of our research. Some have no function to generate neural networks. Others were not regression neural networks of the perceptron type or were only intended for very specific purposes. Basically, each package was inspected 3 times.

1. The discard/not discard phase: depending on the package, this could be decided as easily as looking at the DESCRIPTION file or having to go through the process of making the code and seeing the results.

2. Benchmarking with template that was developed in 2019 and encapsulated in the functions of 2020, keeping notes of whether or not the package was easy to use.

Templates for Testing Accuracy and Speed

As we inspected the packages, we developed a template for benchmarking. The structure of this template (for each package) is as follows:

1. Set up the test environment - loading of packages, setting working directory and options;
2. Summary of tested datasets;
3. Loop over datasets:
   a. setting parameters for a specific dataset,
   b. selecting benchmark options,
   c. training a neural network with a tuned functions for each package,
   d. calculation of convergence metrics (RMSE, MAE, WAE),
   e. plot each training over one initial graph, then plot the best result,
   f. add results to the appropriate existing record (*.csv file) and
   g. clear the environment for next loop.
4. Clearing up the environment for the next package. It is optional to print warnings.

To simplify this process, we developed tools in the NNbenchmark package, of which the first version was created as part of GSoC 2019. In GSoC 2020, 3 functions encapsulating the template, that had been made generic with an extensive use of the incredible `do.call` function, were added:

1. In `trainPredict_1mth1data` a neural network is trained on one dataset and then used for predictions, with several utilities. Then, the performance of the neural network is exported, plotted and/or summarized.
2. `trainPredict_1data` serves as a wrapper function for trainPredict_1mth1data for multiple methods.
3. `trainPredict_1pkg` serves as a wrapper function for trainPredict_1mth1data for multiple datasets.

A function for the summary of accuracy and speed, `NNsummary`, was also added. The package repository is https://github.com/pkR-pkR/NNbenchmark, with package templates in https://github.com/pkR-pkR/NNbenchmarkTemplates.

3. summarizing or re-reviewing the tested packages utility functions & documentation

Ease of Use Scoring

We define an ease-of-use measure based on what we considered a user would need when using a neural network package for nonlinear regression, namely, utility functions and sufficient documentation.

1. Utilities (1 star)

   a. a predict function exists
   b. scaling capabilities exist

2. Sufficient documentation (2 stars)

   a. the existence of useful example/vignette = (1 star)
      - clear, with regression = 2 points
      - unclear, examples use iris or are for classification only = 1 point
      - no examples = 0 points

   b. input/output is clearly documented, e.g., what values are expected and returned by a function = (1 star)
      - clear input and output = 2 points
      - only one is clear = 1 point
      - both are not documented = 0 points

The ease-of-use measure ranges from 0 to 3 stars.

Phase 3 - Collection of and analysis of results

Results collection

Looping over the datasets using each package template, we collected results in the relevant package directories in the templates repository.

Analysis

To rank how well a package converged and its speed, we developed the following method:

1. The results datasets are loaded into the R environment as one large list. The dataset names, package:algorithm names and all 10 run numbers, durations, and RMSE are extracted from that list
2. For the duration score (DUR), the duration is averaged by dataset. 3 criteria for the RMSE score by dataset are calculated:

   a. The minimum value of RMSE for each package:algorithm as a measure of their best performance
   b. The median value of RMSE for each package:algorithm as a measure of their average performance, without the influence of outliers
   c. The spread of the RMSE values for each package which is measured by the difference between the median and the minimum RMSE (d51)

3. Then, the ranks are calculated for every dataset and the results are merged into one wide dataframe.

   a. The duration rank only depends on the duration.
   b. For minimum RMSE values, ties are decided by duration mean, then the RMSE median
   c. For median RMSE values, ties are decided by the RMSE minimum, then the duration mean
   d. The d51 rank only depends on itself

4. A global score for all datasets is found by a sum of the ranks (of duration, minimum RMSE, median RMSE, d51 RMSE) of each package:algorithm for each dataset
5. The final table is the result of ranking by the global minimum RMSE scores for each package:algorithm

Results

Table 1 gives the RMSE ranks, time ranks, and scores for ease of use for each package::algorithm pair. A more complete list of metric scores and hyperparameters is given in Table 2 in Appendix C.

Tables

Discussion and Recommendations

The following is a list of packages we included in this study, with brief descriptions.

1. **AMORE** (Limas et al., 2020),
2. **ANN2** (Lammers, 2020),
3. **appnn** (Família et al., 2015),
4. **autoencoder** (Dubossarsky and Tyshetskiy, 2015),
5. **automl** (Boulangé, 2020),
6. **BNN** (Jia, 2018),
7. **brnn** (Rodriguez and Gianola, 2020),
8. **Buddle** (Kim, 2020),
9. **CaDENCE** (Cannon, 2017a),
10. **cld2** (Ooms, 2018),
11. **cld3** (Ooms, 2020),
12. **condmixt** (Carreau, 2020),
13. **DamiaNN** (Siniakowicz, 2016),
14. **deep** (Mayer, 2019),
15. **deepdive** (Balakrishnan, 2020),
16. **deepnet** (Rong, 2014),
17. **deepNN** (Taylor, 2020),
18. **DNMF** (Jia and Zhang, 2015),
19. **elmNNRcpp** (Mouselimis and Gosso, 2020),
20. **ELMR** (Petrozziello, 2015),
21. **EnsembleBase** (Mahani and Sharabiani, 2016),
22. **evclass** (Denoeux, 2017),
23. **gamlss.add** (Stasinopoulos et al., 2020),
24. **gcForest** (Jing, 2018),
25. **GMDH** (Dag and Yozgatligil, 2016),
26. **GMDH2** (Dag et al., 2019),
27. **GMDHreg** (Tilve, 2020),
28. **gnn** (Hofert and Prasad, 2020),
29. **grnn** (Chasset, 2013a),
30. **h2o** (LeDell et al., 2020),
31. **hybridEnsemble** (Ballings et al., 2015),
32. **isingLenzMC** (Suzen, 2016),
33. **keras** (Allaire and Chollet, 2020),
34. **kerasR** (Arnold, 2017),
35. **leabRa** (Titz, 2017),
36. **learNN** (Quast, 2015),
37. **LilRhino** (Barton, 2019),
38. **minpack.lm** (Elzhov et al., 2016),
39. **MachineShop** (Smith, 2020),
40. **monmlp** (Cannon, 2017b),
41. **neural** (Nagy, 2014),
42. **neuralnet** (Fritsch et al., 2019),
43. **NeuralNetTools** (Beck, 2018),
44. **NeuralSens** (Portela González et al., 2020),
45. **NlinTS** (Hmamouche, 2020),

Table 1: Result from Tested Packages

	Individual score			Global score	
Package	Util	Doc	Algorithm	Time	RMSE
AMORE	1	3.0	ADAPTgd	10	34
	1	3.0	ADAPTgdwm	17	25
	1	3.0	BATCHgd	39	40
	1	3.0	BATCHgdwm	40	39
ANN2	2	3.0	adam	16	33
	2	3.0	rmsprop	14	28
	2	3.0	sgd	12	41
automl	1	3.0	trainwgrad_adam	50	18
	1	3.0	trainwgrad_RMSprop	47	26
	1	3.0	trainwpso	57	42
brnn	2	4.0	Gauss-Newton	8	13
CaDENCE	2	3.0	optim(BFGS)	46	10
	2	3.0	pso_psoptim	54	54
	2	3.0	Rprop	56	51
caret	2	3.0	avNNet_nnet_optim(BFGS)	9	22
deepdive	2	3.0	adam	32	45
	2	3.0	gradientDescent	52	58
	2	3.0	momentum	53	56
	2	3.0	rmsProp	34	53
deepnet	1	3.0	BP	23	18
elmNNRcpp	2	3.0	ELM	1	59
ELMR	2	3.0	ELM	2	60
EnsembleBase	1	1.0	nnet_optim(BFGS)	5	12
h2o	2	2.0	first-order	51	11
keras	2	0.0	adadelta	60	47
	2	0.0	adagrad	58	36
	2	0.0	adam	42	35
	2	0.0	adamax	48	23
	2	0.0	nadam	44	36
	2	0.0	rmsprop	37	52
	2	0.0	sgd	48	43
MachineShop	1	3.0	nnet_optim(BFGS)	6	4
minpack.lm	1	3.5	Levenberg-Marquardt	13	24
monmlp	2	3.5	optimx(BFGS)	26	9
	2	3.5	optimx(Nelder-Mead)	32	46
neuralnet	1	3.0	backprop	37	48
	1	3.0	rprop-	21	21
	1	3.0	rprop+	18	20
	1	3.0	sag	41	38
	1	3.0	slr	31	30
nlsr	1	4.0	NashLM	18	1
nnet	1	3.0	optim (BFGS)	3	3
qrnn	2	3.0	nlm()	28	15
radiant.model	2	2.0	nnet_optim(BFGS)	11	7
rminer	2	3.5	nnet_optim(BFGS)	14	2
RSNNS	2	3.0	BackpropBatch	43	49
	2	3.0	BackpropChunk	26	29
	2	3.0	BackpropMomentum	25	30
	2	3.0	BackpropWeightDecay	29	32
	2	3.0	Quickprop	45	57
	2	3.0	Rprop	24	16
	2	3.0	SCG	30	17
	2	3.0	Std_Backpropagation	22	27
snnR	2	2.0	SemiSmoothNewton	7	49
traineR	1	2.5	nnet_optim(BFGS)	4	6
validann	1	4.0	optim(BFGS)	35	4
	1	4.0	optim(CG)	59	8
	1	4.0	optim(L-BFGS-B)	36	14
	1	4.0	optim(Nelder-Mead)	55	44
	1	4.0	optim(SANN)	20	55

46. **nlsr** (Nash and Murdoch, 2019),
47. **nnet** (Ripley, 2020),
48. **nnetpredint** (Ding, 2015),
49. **nnfor** (Kourentzes, 2019),
50. **nntrf** (Aler and Valls, 2020),
51. **nnli2bRcpp** (Nikolaidis, 2020),
52. **onnx** (Tang and ONNX Authors, 2018),
53. **OptimClassifier** (Perez-Martin et al., 2020),
54. **OSTSC** (Dixon et al., 2017),
55. **pnn** (Chasset, 2013b),
56. **polyreg** (Matloff et al., 2020),
57. **predictoR** (with contributions from Diego Jimenez A. and D., 2020),
58. **qrnn** (Cannon, 2019),
59. **QuantumOps** (Resch, 2020),
60. **quarrint** (Barthelemy et al., 2016),
61. **radiant.model** (Nijs, 2020),
62. **rasclass** (Wiesmann and Quinn, 2016),
63. **rcane** (Suresh et al., 2018),
64. **regressoR** (Rodriguez R., 2019a),
65. **rminer** (Cortez, 2020),
66. **rnn** (Quast and Fichou, 2020),
67. **RSNNS** (Bergmeir, 2019),
68. **ruta** (Charte et al., 2019),
69. **simpleNeural** (Dernoncourt, 2020),
70. **snnR** (Wang et al., 2017),
71. **softmaxreg** (Ding, 2016),
72. **Sojourn.Data** (Hibbing and Lyden, 2019),
73. **spnn** (Ebrahimi, 2020),
74. **TeachNet** (Steinbuss, 2018),
75. **tensorflow** (Allaire and Tang, 2020),
76. **tfestimators** (Allaire et al., 2018),
77. **trackdem** (Bruijning et al., 2020),
78. **TrafficBDE** (Chatzopoulou et al., 2018),
79. **tsensembler** (Cerqueira et al., 2017),
80. **validann** (Humphrey, 2017),
81. **zFactor** (Reyes, 2019).

2nd order algorithms

Out of all the algorithms, the following second algorithms generally performed better in terms of convergence despite being set to a much lower number of iterations, to be precise a fifth or even less, than the first-order algorithms.

An important finding is that 11 out of 15 of these package::algorithms use the algorithms included in `optim` from **stats**. 2 of them, **CaDENCE**'s BFGS (Cannon, 2017a) and **validann**'s BFGS and L-BFGS-B (Humphrey, 2017), do so with no intermediate package. However, it is not clearly stated in CaDENCE's documentation that optim's BFGS algorithm is used and not one of the other algorithms. Furthermore, the mention of Nelder-Mead in the documentation might lead users to believe that optim's Nelder-Mead is used instead. Speed and variation between results are also not as good as other package's that use optim. This could be because CaDENCE is intended for probabilistic nonlinear models with a full title of "Conditional Density Estimation Network Construction and Evaluation". On the other hand, validann is clearly a package that allows a user to use all optim's algorithms. validann::L-BFGS-B ranks lower than validann::BFGS in just about everything for most runs, despite the former being more sophisticated. This is probably due to our efforts to harmonize parameters under-utilizing the possibilities of the L-BFGS-B algorithm. Both CaDENCE and validann's BFGS are outperformed by nnet, especially in terms of speed.

nnet (Ripley, 2020) differs from the two packages because it uses the C code from optim (converted earlier from Fortran) instead of calling optim from R. It also only implements the BFGS algorithm. This could be what allows it to be faster. nnet is only beaten by the Extreme Learning Machine (ELM) algorithms in terms of speed. However, there is a larger variation between results (see the RMSEd51.score in Appendix C) in comparison to validann::BFGS. Most likely, the different default values are the cause of this. For instance, nnet uses a range of initial random weights of 0.7 while validann uses a value of 0.5. In spite of these results, the real reason most authors or users are likely to choose nnet is because it ships with base R and is even mentioned as the very first package in CRAN's task view for machine learning.

Our research found that 6 of the 11 packages that use optim do so through nnet. Moreover, 8 packages for neural networks, though not tested, use nnet. The total number of nnet dependencies found through a search through the offline database of CRAN with RWsearch came up with 136 packages, although some might be using nnet for the multinomial log-linear models, not neural networks. As for the ones we tested, there were several similarities and differences. The packages that use nnet for neuralnetworks are often meta packages with a host of other machine learning algorithms. **caret** (Kuhn, 2020), also mentioned in the taskview, boasts 238 methods with around 13 different neural network packages with somewhat deceivingly simple name of "Classification and Regression Training". It has many pre-processing utilities available, as well as other tools.

EnsembleBase (Mahani and Sharabiani, 2016) maybe useful for those who wish to make ensembles and test a grid of parameters although the documentation. **MachineShop** (Smith, 2020) has 51 algorithms, with some additional information about the response variable types in the second vignette, functions for preprocessing and tuning, performance assesment, and presentation of results. **radiant.model** (Nijs, 2020) has an unchangeable maxit of 10000 in the original package. Perhaps the author thought this was reasonable as source of the algorithm, nnet, is quite fast. We changed this to harmonize the parameters. **rminer** (Cortez, 2020) is the only package dependant on nnet that ranks above nnet at number 2 for minimum RMSE, and even number 1 in some runs. It also ranks number 1 on the other accuracy measures (median RMSE, minimum MAE, minimum WAE) and is only behind deepdive and minpack.lm in terms of results that are consistent and do not vary (RMSEd51). The difference is probably from the change of maximum allowable weights in rminer to 10000 from 1000 in nnet, which is also probably the reason it its fits are slower. **traineR** (Rodriguez R., 2019b) claims to unify the different methods of creating models between several learning algorithms.

Something worth noting is that nnet and validann do not have external normalization, and it is especially recommended for validann . However, some of the packages dependent on nnet do have this utility and it is included in the scoring for ease of use. With NNbenchmark, this is done through setting scale = TRUE in the function `prepare.ZZ`. Note that scaling might lead to complicating the constraints which is not always worth it. Regardless, users might want to have the utility and most likely want a clear explanation of the method chosen to center the variables. Scaling is one of the things that **optimx** (Nash and Varadhan, 2020) incorporates in an attempt to make a more useful version of optim that only allows for changing the sign of the function which might not even be considered as scaling (Nash, 2014).

Recommended: 1st order algorithms

validann optim CG -slow RSNNS SCG h2o back-propagation RSNNS Rprop ANN2 adam CaDENCE Rprop -SLOW deepnet BP AMORE ADAPTgdwm AMORE ADAPTgd ANN2 sgd automl trainwgrad ANN2 rmsprop RSNNS BackpropChunk RSNNS BackWeightDecay RSNNS Std_Backpropagation RSNNS BackpropMomentum automl trainwpso validann optim NelderMead snnR Semi Smooth Newton RSNNS BackpropBatch validann optim SANN monmlp optimx Nelder Mead

Not recommended: 1st order algorithms <- DISCUSS CUTOFF

By package ELMR, elmNNRcpp - fast ELM algorithms. Unfortunately, can't finetune, does not converge well. neuralnet: a large ammount of iterations, slow, erratic failures tensorflow: NOT EASY TO USE, subsequently keras, tfestimators, ruta ... user needs to understand the language However, advanced users might be able to highly specify a neural network to their needs (customization?)

By algorithm: neuralnet rprop+ neuralnet rprop- neuralnet slr - once ranked well with 100000 iterations AMORE BATCHgd CaDENCE pso psoptim - need to reconfigure? elmNNRcpp - fast, no iterations RSNNS Quickprop (?) AMORE BATCHgdwm tensorflow MomentumOptimizer tensorflow AdamOptimizer ELMR - fast, no iterations tensorflow GradientDescentOptimizer keras rmsprop keras adagrad keras sgd keras adadelta tensorflow AdagradOptimizer keras adam tensorflow FtrlOptimizer neuralnetwork sag tensorflow AdadeltaOptimizer neuralnet backprop - note, might not actually reflect standings, somehow from template to template the learning rate disappeared. Will fix this in future runs

Untested => TO DO - LIST

Conclusion and perspective

Positives

- We are happy to note the existence of neural network packages in R with algorithms that converge well.
- **nnet**, which uses ?? need font choice?? optim's BFGS method, is already often chosen to represent neural networks for packages that are either a collection of independent machine learning algorithms, ensembles, or even applications in a field such as ... ?? need to complete sentence??. JN??: Why is this positive?
- R users have access to a wide variety of neural network methods, including from libraries of other programming languages and using many different types of algorithms. ?? have we defined hyperparameters?? hyperparameters, and uses

Negatives

- We are disappointed that many of the packages we reviewed had poor documentation.
- It would be helpful if there were more packages with (different) second order algorithms. A number of the
- We often found it difficult to discover what default starting values were used for model parameters, or

Future work

As the field of neural networks continue to grow, there will always be more algorithms to validate. For current packages available for R, we believe our research should be extended to encompass more types of neural networks and their data formats (classifier neural networks, recurrent neural networks, and so on). Different rating schemes and different settings for package functions should also be tried out, though such work suffers from the curse of too many dimensions of effort.

We also aspire to the possibility of an idealized neural network package, in particular for R. From the work in this paper, we believe that such a package should offer

- ease of use in setting up computations and attaching data
- the incorporation of second-order optimization methods in training the model, with appropriate computation of derivative information and appropriate internal steps, e.g., Levenberg-Marquardt stabilization
- good documentation and controls, especially of initial values for model parameters

Acknowledgements

This work was possible due to the support of the Google Summer of Code initiative for R during years 2019 and 2020. Students Salsabila Mahdi (2019 and 2020) and Akshaj Verma (2019) are grateful to Google for the financial support.

Bibliography

R. Aler and J. Valls. *nntrf: Supervised Data Transformation by Means of Neural Network Hidden Layers*, 2020. URL https://CRAN.R-project.org/package=nntrf. R package version 0.1.3. [p9]

J. Allaire and F. Chollet. *keras: R Interface to 'Keras'*, 2020. URL https://CRAN.R-project.org/package=keras. R package version 2.3.0.0. [p7]

J. Allaire and Y. Tang. *tensorflow: R Interface to 'TensorFlow'*, 2020. URL https://CRAN.R-project.org/package=tensorflow. R package version 2.2.0. [p9]

J. Allaire, Y. Tang, K. Ushey, and K. Kuo. *tfestimators: Interface to 'TensorFlow' Estimators*, 2018. URL https://CRAN.R-project.org/package=tfestimators. R package version 1.9.1. [p9]

T. Arnold. *kerasR: R Interface to the Keras Deep Learning Library*, 2017. URL https://CRAN.R-project.org/package=kerasR. R package version 0.6.1. [p7]

R. Balakrishnan. *deepdive: Deep Learning for General Purpose*, 2020. URL https://CRAN.R-project.org/package=deepdive. R package version 1.0.1. [p7]

M. Ballings, D. Vercamer, and D. Van den Poel. *hybridEnsemble: Build, Deploy and Evaluate Hybrid Ensembles*, 2015. URL https://CRAN.R-project.org/package=hybridEnsemble. R package version 1.0.0. [p7]

J. Barthelemy, T. Carletti, L. Collier, V. Hallet, M. Moriame, and A. Sartenaer. *quarrint: Interaction Prediction Between Groundwater and Quarry Extension Using Discrete Choice Models and Artificial Neural Networks*, 2016. URL https://CRAN.R-project.org/package=quarrint. R package version 1.0.0. [p9]

T. Barton. *LilRhino: For Implementation of Feed Reduction, Learning Examples, NLP and Code Management*, 2019. URL https://CRAN.R-project.org/package=LilRhino. R package version 1.2.0. [p7]

M. W. Beck. *NeuralNetTools: Visualization and Analysis Tools for Neural Networks*, 2018. URL https://CRAN.R-project.org/package=NeuralNetTools. R package version 1.5.2. [p7]

C. Bergmeir. *RSNNS: Neural Networks using the Stuttgart Neural Network Simulator (SNNS)*, 2019. URL https://CRAN.R-project.org/package=RSNNS. R package version 0.4-12. [p9]

A. Boulangé. *automl: Deep Learning with Metaheuristic*, 2020. URL https://CRAN.R-project.org/package=automl. R package version 1.3.2. [p7]

M. Bruijning, M. D. Visser, C. A. Hallmann, and E. Jongejans. *trackdem: Particle Tracking and Demography*, 2020. URL https://CRAN.R-project.org/package=trackdem. R package version 0.5.2. [p9]

A. J. Cannon. *CaDENCE: Conditional Density Estimation Network Construction and Evaluation*, 2017a. URL https://CRAN.R-project.org/package=CaDENCE. R package version 1.2.5. [p7, 9]

A. J. Cannon. *monmlp: Multi-Layer Perceptron Neural Network with Optional Monotonicity Constraints*, 2017b. URL https://CRAN.R-project.org/package=monmlp. R package version 1.1.5. [p7]

A. J. Cannon. *qrnn: Quantile Regression Neural Network*, 2019. URL https://CRAN.R-project.org/package=qrnn. R package version 2.0.5. [p9]

J. Carreau. *condmixt: Conditional Density Estimation with Neural Network Conditional Mixtures*, 2020. URL https://CRAN.R-project.org/package=condmixt. R package version 1.1. [p7]

V. Cerqueira, L. Torgo, and C. Soares. Arbitrated ensemble for solar radiation forecasting. *International Work-Conference on Artificial Neural Networks*, pages 720–732, 2017. The R package tsensembler has been archived as of July 2020 because it has not been maintained. [p9]

T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)?– arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3): 1247–1250, 2014. [p3]

D. Charte, F. Charte, and F. Herrera. *ruta: Implementation of Unsupervised Neural Architectures*, 2019. URL https://CRAN.R-project.org/package=ruta. R package version 1.1.0. [p9]

P.-O. Chasset. *grnn: General regression neural network*, 2013a. URL https://CRAN.R-project.org/package=grnn. R package version 0.1.0. [p7]

P.-O. Chasset. *pnn: Probabilistic neural networks*, 2013b. URL https://CRAN.R-project.org/package=pnn. R package version 1.0.1. [p9]

A. Chatzopoulou, K. Koupidis, and C. Bratsas. *TrafficBDE: Traffic Status Prediction in Urban Places using Neural Network Models*, 2018. URL https://CRAN.R-project.org/package=TrafficBDE. R package version 0.1.0. [p9]

P. Cortez. *rminer: Data Mining Classification and Regression Methods*, 2020. URL https://CRAN.R-project.org/package=rminer. R package version 1.4.5. [p9, 10]

O. Dag and C. Yozgatligil. *GMDH: Short Term Forecasting via GMDH-Type Neural Network Algorithms*, 2016. URL https://CRAN.R-project.org/package=GMDH. R package version 1.6. [p7]

O. Dag, E. Karabulut, and R. Alpar. *GMDH2: Binary Classification via GMDH-Type Neural Network Algorithms*, 2019. URL https://CRAN.R-project.org/package=GMDH2. R package version 1.5. [p7]

T. Denoeux. *evclass: Evidential Distance-Based Classification*, 2017. URL https://CRAN.R-project.org/package=evclass. R package version 1.1.1. [p7]

D. Dernoncourt. *simpleNeural: An Easy to Use Multilayer Perceptron*, 2020. URL https://CRAN.R-project.org/package=simpleNeural. R package version 0.1.3. [p9]

X. Ding. *nnetpredint: Prediction Intervals of Multi-Layer Neural Networks*, 2015. URL https://CRAN.R-project.org/package=nnetpredint. R package version 1.2. [p9]

X. Ding. *softmaxreg: Training Multi-Layer Neural Network for Softmax Regression and Classification*, 2016. URL https://CRAN.R-project.org/package=softmaxreg. R package version 1.2. [p9]

M. Dixon, D. Klabjan, and L. Wei. *OSTSC: Over Sampling for Time Series Classification*, 2017. URL https://CRAN.R-project.org/package=OSTSC. R package version 0.0.1. [p9]

E. Dubossarsky and Y. Tyshetskiy. *autoencoder: Sparse Autoencoder for Automatic Learning of Representative Features from Unlabeled Data*, 2015. URL https://CRAN.R-project.org/package=autoencoder. R package version 1.1. [p7]

R. Ebrahimi. *spnn: Scale Invariant Probabilistic Neural Networks*, 2020. URL https://CRAN.R-project.org/package=spnn. R package version 1.2.1. [p9]

T. V. Elzhov, K. M. Mullen, A.-N. Spiess, and B. Bolker. *minpack.lm: R Interface to the Levenberg-Marquardt Nonlinear Least-Squares Algorithm Found in MINPACK, Plus Support for Bounds*, 2016. URL https://CRAN.R-project.org/package=minpack.lm. R package version 1.2-1. [p7]

C. Família, S. R. Dennison, A. Quintas, and D. A. Phoenix. *appnn: Amyloid Propensity Prediction Neural Network*, 2015. URL https://CRAN.R-project.org/package=appnn. R package version 1.0-0. [p7]

S. Fritsch, F. Guenther, and M. N. Wright. *neuralnet: Training of Neural Networks*, 2019. URL https://CRAN.R-project.org/package=neuralnet. R package version 1.44.2. [p7]

P. R. Hibbing and K. Lyden. *Sojourn.Data: Supporting Objects for Sojourn Accelerometer Methods*, 2019. URL https://CRAN.R-project.org/package=Sojourn.Data. R package version 0.1.0. [p9]

Y. Hmamouche. *NlinTS: Models for Non Linear Causality Detection in Time Series*, 2020. URL https://CRAN.R-project.org/package=NlinTS. R package version 1.4.2. [p7]

M. Hofert and A. Prasad. *gnn: Generative Neural Networks*, 2020. URL https://CRAN.R-project.org/package=gnn. R package version 0.0-2. [p7]

G. B. Humphrey. *validann: Validation Tools for Artificial Neural Networks*, 2017. URL https://CRAN.R-project.org/package=validann. R package version 1.2.1. [p9]

B. Jia. *BNN: Bayesian Neural Network for High-Dimensional Nonlinear Variable Selection*, 2018. URL https://CRAN.R-project.org/package=BNN. R package version 1.0.2. [p7]

Z. Jia and X. Zhang. *DNMF: Discriminant Non-Negative Matrix Factorization*, 2015. URL https://CRAN.R-project.org/package=DNMF. R package version 1.3. [p7]

X. Jing. *gcForest: Deep Forest Model*, 2018. URL https://CRAN.R-project.org/package=gcForest. R package version 0.2.7. [p7]

P. Kiener. *RWsearch: Lazy Search in R Packages, Task Views, CRAN, the Web. All-in-One Download*, 2020. URL https://CRAN.R-project.org/package=RWsearch. R package version 4.8.0. [p4]

J. Kim. *Buddle: A Deep Learning for Statistical Classification and Regression Analysis with Random Effects*, 2020. URL https://CRAN.R-project.org/package=Buddle. R package version 2.0.1. [p7]

N. Kourentzes. *nnfor: Time Series Forecasting with Neural Networks*, 2019. URL https://CRAN.R-project.org/package=nnfor. R package version 0.9.6. [p9]

M. Kuhn. *caret: Classification and Regression Training*, 2020. URL https://CRAN.R-project.org/package=caret. R package version 6.0-86. [p10]

B. Lammers. *ANN2: Artificial Neural Networks for Anomaly Detection*, 2020. URL https://CRAN.R-project.org/package=ANN2. R package version 2.3.3. [p7]

E. LeDell, N. Gill, S. Aiello, A. Fu, A. Candel, C. Click, T. Kraljevic, T. Nykodym, P. Aboyoun, M. Kurka, and M. Malohlava. *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*, 2020. URL https://CRAN.R-project.org/package=h2o. R package version 3.30.0.1. [p7]

M. C. Limas, J. B. O. Mere, A. G. Marcos, F. J. M. de Pison Ascacibar, A. V. P. Espinoza, F. A. Elias, and J. M. P. Ramos. *AMORE: Artificial Neural Network Training and Simulating*, 2020. URL https://CRAN.R-project.org/package=AMORE. R package version 0.2-16. [p7]

A. S. Mahani and M. T. Sharabiani. *EnsembleBase: Extensible Package for Parallel, Batch Training of Base Learners for Ensemble Modeling*, 2016. URL https://CRAN.R-project.org/package=EnsembleBase. R package version 1.0.2. [p7, 10]

N. Matloff, X. Cheng, P. Mohanty, B. Khomtchouk, M. Kotila, R. Yancey, R. Tucker, A. Zhao, and T. Jiang. *polyreg: Polynomial Regression*, 2020. URL https://CRAN.R-project.org/package=polyreg. R package version 0.6.7. [p9]

B. L. Mayer. *deep: A Neural Networks Framework*, 2019. URL https://CRAN.R-project.org/package=deep. R package version 0.1.0. [p7]

L. Mouselimis and A. Gosso. *elmNNRcpp: The Extreme Learning Machine Algorithm*, 2020. URL https://CRAN.R-project.org/package=elmNNRcpp. R package version 1.0.2. [p7]

A. Nagy. *neural: Neural Networks*, 2014. URL https://CRAN.R-project.org/package=neural. R package version 1.4.2.2. [p7]

J. C. Nash. *Nonlinear Parameter Optimization Using R Tools*. John Wiley & Sons: Chichester, May 2014. ISBN 978-1-118-56928-3. Companion website (see http://www.wiley.com//legacy/wileychi/nash/). [p10]

J. C. Nash and D. Murdoch. *nlsr: Functions for Nonlinear Least Squares Solutions*, 2019. URL https://CRAN.R-project.org/package=nlsr. R package version 2019.9.7. [p9]

J. C. Nash and R. Varadhan. *optimx: Expanded Replacement and Extension of the 'optim' Function*, 2020. URL https://CRAN.R-project.org/package=optimx. R package version 2020-4.2. [p10]

V. Nijs. *radiant.model: Model Menu for Radiant: Business Analytics using R and Shiny*, 2020. URL https://CRAN.R-project.org/package=radiant.model. R package version 1.3.10. [p9, 10]

V. Nikolaidis. *nnlib2Rcpp: A Collection of Neural Networks*, 2020. URL https://CRAN.R-project.org/package=nnlib2Rcpp. R package version 0.1.4. [p9]

J. Ooms. *cld2: Google's Compact Language Detector 2*, 2018. URL https://CRAN.R-project.org/package=cld2. R package version 1.2. [p7]

J. Ooms. *cld3: Google's Compact Language Detector 3*, 2020. URL https://CRAN.R-project.org/package=cld3. R package version 1.3. [p7]

A. Perez-Martin, A. Perez-Torregrosa, M. Vaca-Lamata, and A. J. Verdu-Jover. *OptimClassifier: Create the Best Train for Classification Models*, 2020. URL https://CRAN.R-project.org/package=OptimClassifier. R package version 0.1.5. [p9]

A. Petrozziello. *ELMR: Extreme Machine Learning (ELM)*, 2015. URL https://CRAN.R-project.org/package=ELMR. R package version 1.0. [p7]

J. Portela González, A. Muñoz San Roque, and J. Pizarroso Gonzalo. *NeuralSens: Sensitivity Analysis of Neural Networks*, 2020. URL https://CRAN.R-project.org/package=NeuralSens. R package version 0.2.1. [p7]

B. Quast. *learNN: Examples of Neural Networks*, 2015. URL https://CRAN.R-project.org/package=learNN. R package version 0.2.0. [p7]

B. Quast and D. Fichou. *rnn: Recurrent Neural Network*, 2020. URL https://CRAN.R-project.org/package=rnn. R package version 1.4.0. [p9]

S. Resch. *QuantumOps: Performs Common Linear Algebra Operations Used in Quantum Computing and Implements Quantum Algorithms*, 2020. URL https://CRAN.R-project.org/package=QuantumOps. R package version 3.0.1. [p9]

A. R. Reyes. *zFactor: Calculate the Compressibility Factor 'z' for Hydrocarbon Gases*, 2019. URL https://CRAN.R-project.org/package=zFactor. R package version 0.1.9. [p9]

B. Ripley. *nnet: Feed-Forward Neural Networks and Multinomial Log-Linear Models*, 2020. URL https://CRAN.R-project.org/package=nnet. R package version 7.3-14. [p9, 10]

P. P. Rodriguez and D. Gianola. *brnn: Bayesian Regularization for Feed-Forward Neural Networks*, 2020. URL https://CRAN.R-project.org/package=brnn. R package version 0.8. [p7]

O. Rodriguez R. *regressoR: Regression Data Analysis System*, 2019a. URL https://CRAN.R-project.org/package=regressoR. R package version 1.1.9. [p9]

O. Rodriguez R. *traineR: Predictive Models Homologator*, 2019b. URL https://CRAN.R-project.org/package=traineR. R package version 1.0.0. [p10]

X. Rong. *deepnet: deep learning toolkit in R*, 2014. URL https://CRAN.R-project.org/package=deepnet. R package version 0.2. [p7]

D. Siniakowicz. *DamiaNN: Neural Network Numerai*, 2016. URL https://CRAN.R-project.org/package=DamiaNN. R package version 1.0.0. [p7]

B. J. Smith. *MachineShop: Machine Learning Models and Tools*, 2020. URL https://CRAN.R-project.org/package=MachineShop. R package version 2.5.0. [p7, 10]

M. Stasinopoulos, B. Rigby, V. Voudouris, and D. Kiose. *gamlss.add: Extra Additive Terms for Generalized Additive Models for Location Scale and Shape*, 2020. URL https://CRAN.R-project.org/package=gamlss.add. R package version 5.1-6. [p7]

G. Steinbuss. *TeachNet: Fits Neural Networks to Learn About Backpropagation*, 2018. URL https://CRAN.R-project.org/package=TeachNet. R package version 0.7.1. [p9]

A. Suresh, S. Acharekar, H. Chao, and S. Y. Biradar. *rcane: Different Numeric Optimizations to Estimate Parameter Coefficients*, 2018. URL https://CRAN.R-project.org/package=rcane. R package version 1.0. [p9]

M. Suzen. *isingLenzMC: Monte Carlo for Classical Ising Model*, 2016. URL https://CRAN.R-project.org/package=isingLenzMC. R package version 0.2.5. [p7]

Y. Tang and ONNX Authors. *onnx: R Interface to 'ONNX'*, 2018. URL https://CRAN.R-project.org/package=onnx. R package version 0.0.2. [p9]

B. Taylor. *deepNN: Deep Learning*, 2020. URL https://CRAN.R-project.org/package=deepNN. R package version 1.0. [p7]

M. V. Tilve. *GMDHreg: Regression using GMDH Algorithms*, 2020. URL https://CRAN.R-project.org/package=GMDHreg. R package version 0.2.1. [p7]

J. Titz. *leabRa: The Artificial Neural Networks Algorithm Leabra*, 2017. URL https://CRAN.R-project.org/package=leabRa. R package version 0.1.0. [p7]

Y. Wang, P. Lin, Z. Chen, Z. Bao, and G. J. M. Rosa. *snnR: Sparse Neural Networks for Genomic Selection in Animal Breeding*, 2017. URL https://CRAN.R-project.org/package=snnR. R package version 1.0. [p9]

D. Wiesmann and D. Quinn. *rasclass: Supervised Raster Image Classification*, 2016. URL https://CRAN.R-project.org/package=rasclass. R package version 0.2.2. [p9]

C. J. Willmott and K. Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1): 79–82, 2005. [p3]

O. R. R. with contributions from Diego Jimenez A. and A. N. D. *predictoR: Predictive Data Analysis System*, 2020. URL https://CRAN.R-project.org/package=predictoR. R package version 1.1.3. [p9]

S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(1):3–36, 2011. [p4]

Appendix

Appendix A

Consider a set of observations y_i and its corresponding predictions $\hat{y}_i$ for $i = 1, \ldots, n$. The three metrics used were:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|,\ RMSE = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},\ WAE = \frac{1}{n}\max_{i=1,\ldots,n}|y_i - \hat{y}_i|.$$

These values represent the absolute, the squared and the maximum norm of residual vectors.

Appendix B

We define three smooth functions for Simon Wood's test dataset

$$f_0 = 5 * \sin(2\pi x),\ f_1 = exp(3 * x) - 7 f_2 = 0.5x^{11} * (10(1-x))^6 - 10(10 * x)^3 * (1-x)^{10},$$

$$f_3 = 15\exp(-5|x-1/2|) - 6,\ f_4 = 2 - 1_{(x<=1/3)}(6*x)^3 - 1_{(x>=2/3)}(6-6*x)^3 - 1_{(2/3>x>1/3)}(8+2\sin(9*(x-1/$$

Appendix C

Appendix D

Appendix ??

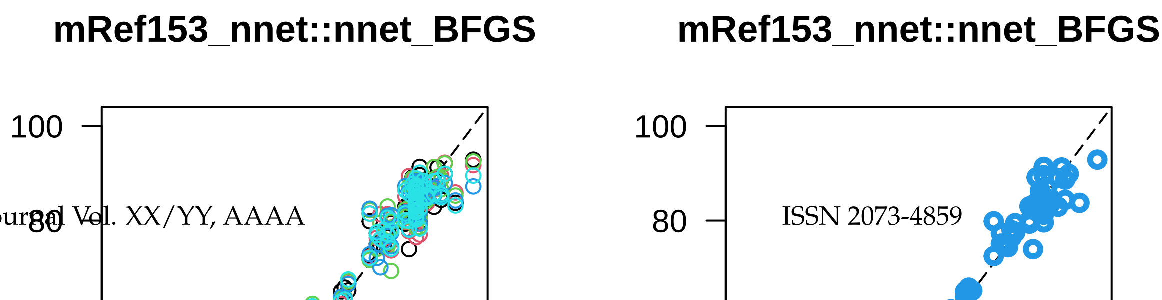
```
library(NNbenchmark)
nrep <- 5
odir <- tempdir()

library(nnet)
nnet.method <- "BFGS"
hyperParams.nnet <- function(...) {
    return (list(iter=200, trace=FALSE))
}
NNtrain.nnet <- function(x, y, dataxy, formula, neur, method, hyperParams, ...) {

    hyper_params <- do.call(hyperParams, list(...))

    NNreg <- nnet::nnet(x, y, size = neur, linout = TRUE, maxit = hyper_params$iter, trace=hyper_params$trace
    return(NNreg)
}
NNpredict.nnet  <- function(object, x, ...) { predict(object, newdata=x) }
NNclose.nnet    <- function() {  if("package:nnet" %in% search())
                                detach("package:nnet", unload=TRUE) }
nnet.prepareZZ  <- list(xdmv = "d", ydmv = "v", zdm = "d", scale = TRUE)

res <- trainPredict_1pkg(4:5, pkgname = "nnet", pkgfun = "nnet", nnet.method,
  prepareZZ.arg = nnet.prepareZZ, nrep = nrep, doplot = TRUE,
  csvfile = FALSE, rdafile = FALSE, odir = odir, echo = FALSE)
```

Table 2: All convergence scores per package:algorithm

	Input parameter			Score			
Num	Input format	Maxit	Learn. rate	RMSE median	RMSE d51	MAE	WAE
1	x & y	1000	0.01	24	8	26	19
2	x & y	1000	0.01	22	28	17	26
3	x & y	10000	0.1	37	24	42	31
4	x & y	10000	0.1	33	14	36	27
5	x & y	1000	0.01	27	26	28	23
6	x & y	1000	0.01	26	33	27	22
7	x & y	1000	0.01	36	20	35	28
8	x & y	1000	0.01	20	35	16	19
9	x & y	1000	0.01	31	50	29	38
10	x & y	1000	-	40	49	40	37
11	x & y	200	-	11	9	12	11
12	x & y	200	-	28	48	23	39
13	x & y	1000	-	56	56	54	56
14	x & y	1000	0.01	54	60	52	58
15	x & y	200	-	13	30	14	12
16	x & y	10000	0.4	41	1	37	44
17	x & y	10000	0.8	57	2	57	53
18	x & y	1000	0.8	52	3	53	51
19	x & y	1000	0.8	45	4	47	50
20	x & y	1000	0.8	18	38	24	17
21	x & y	-	-	59	55	59	59
22	fmla & data	-	-	60	54	60	60
23	x & y	200	-	15	33	15	15
24	"y" & data	10000	0.01	7	7	8	8
25	x & y	10000	0.1	48	27	51	41
26	x & y	10000	0.1	42	51	41	33
27	x & y	10000	0.1	28	44	30	25
28	x & y	10000	0.1	16	19	20	16
29	x & y	10000	0.1	38	58	39	40
30	x & y	10000	0.1	54	57	55	54
31	x & y	10000	0.1	44	47	43	43
32	fmla & data	200	-	9	20	10	7
33	full fmla & data	200	-	18	5	19	14
34	x & y	200	-	10	17	9	10
35	x & y	10000	-	46	46	43	47
36	fmla & data	100000	0.001	50	11	48	45
37	fmla & data	100000	-	21	41	22	18
38	fmla & data	100000	-	23	40	21	24
39	fmla & data	100000	-	50	59	46	52
40	fmla & data	100000	-	38	37	38	46
41	full fmla & data	200	-	3	16	3	6
42	x & y	200	-	2	17	2	3
43	x & y	200	-	14	22	7	35
44	"y" & data	200	-	8	32	11	9
45	fmla & data	200	-	1	6	1	1
46	x & y	10000	0.1	47	25	50	49
47	x & y	1000	-	34	41	32	34
48	x & y	1000	-	35	38	34	30
49	x & y	1000	-	30	43	33	32
50	x & y	10000	-	58	36	58	57
51	x & y	1000	-	24	51	25	29
52	x & y	1000	-	16	22	18	19
53	x & y	1000	0.1	32	30	31	36
54	x & y	200	-	48	13	49	47
55	fmla & data	200	-	5	15	6	2
56	x & y	200	-	4	11	4	5
57	x & y	1000	-	6	10	5	4
58	x & y	200	-	12	29	13	12
59	x & y	10000	-	43	45	45	41
60	x & y	1000	-	53	53	56	55

Table 3: Review of Ommitted Packages

No	Name (package)	Category	Comment
1	**appnn**	AP	This package provides a feed forward neural network to predict the amyloidogenicity propensity of polypeptide sequences
2	**autoencoder**	AP	This package provides a sparse autoencoder, an unsupervised algorithm that learns useful features from the data its given
3	**BNN**	RE*	This package uses a feed forward neural network to perform regression as provided in the examples, however, it is unclear whether it fits the form of perceptron that is the scope of our research. Moreover, it states that it is intended for variable selection. Although how exactly the package would be used to do so isn't accessible in the package, especially considering the source code is based on .c code that users of R might not understand. It's performance is slow, which may have to do with the 100.000 iterations it needs, although quite accurate for simple datasets.
4	**Buddle**	RE**	(errors)
5	**cld2**	00	
6	**cld3**	AP	
7	**condmixt**	AP	
8	**deep**	CL	
9	**DALEX2**	00	removed keyword, included in 2019
10	**DamiaNN**	RE**	(errors) exported functions, still doesn't work
11	**DChaos**	??	removed keyword for some reason, need to check out!
12	**deepNN**	RE**	(errors) I/O weird, ragged vector array
13	**DNMF**	AP	
14	**evclass**	CL	
15	**gamlss.add**	RE	there is some code but dist not appropriate
16	**gcForest**	00	
17	**GMDH**	TS	
18	**GMDH2**	CL	
19	**GMDHreg**	RE*	
20	**grnn**	RE**	
21	**hybridEnsemble**	??	
22	**isingLenzMC**	AP	
23	**leabRa**	??	
24	**learNN**	??	
25	**LilRhino**	AP	
26	**neural**	CL	
27	**NeuralNetTools**	UT	tools for neural networks
28	**NeuralSens**	UT	tools for neural networks
29	**NlinTS**	TS	Time Series
30	**nnetpredint**	UT	confidence intervals for NN
31	**nnfor**	TS	Times Series, uses neuralnet
32	**nntrf**	UT	
33	**onnx**		provides an open source format
34	**OptimClassifier**		choose classifier parameters, nnet
35	**OSTSC**		solving oversampling classification
36	**passt**		
36	**pnn**		Probabilistic
37	**polyreg**		polyregression as alternative to NN
38	**predictoR**		shiny interface, neuralnet
39	**ProcData**		
40	**QuantumOps**		classifies MNIST, Schuld (2018), removed keyword, in 2019
41	**quarrint**		specified classifier for quarry data
42	**rasclass**		classifier for raster images, nnet?
43	**rcane**		
44	**regressoR**		a manual rich version of predictoR
45	**rnn**		Recurrent
46	**RTextTools**		
47	**ruta**		
48	**simpleNeural**		
49	**softmaxreg**		
50	**Sojourn.Data**		sojourn Accelerometer methods, nnet?
51	**spnn**		classifier, probabilistic
52	**studyStrap**		
53	**TeachNet**		classifier, selfbuilt, slow
54	**tensorflow**		
55	**tfestimators**		
56	**trackdem**		classifier for particle tracking
57	**TrafficBDE**	RE*	
58	**tsfgrnn**		
59	**yap**		
60	**yager**	RE*	
61	**zFactor**	AP	'compressibility' of hydrocarbon gas



Salsabila Mahdi
Universitas Syiah Kuala
JL. Syech Abdurrauf No.3, Aceh 23111, Indonesia
bila.mahdi@mhs.unsyiah.ac.id

Akshaj Verma
Manipal Institute of Technology
Manipal, Karnataka, 576104, India
akshajverma7@gmail.com

Christophe Dutang
University Paris-Dauphine, University PSL, CNRS, CEREMADE
Place du Maréchal de Lattre de Tassigny, 75016 Paris, France
dutang@ceremade.dauphine.fr

Patrice Kiener
InModelia
5 rue Malebranche, 75005 Paris, France
patrice.kiener@inmodelia.com

John C. Nash
Telfer School of Management, University of Ottawa
55 Laurier Avenue East, Ottawa, Ontario K1N 6N5 Canada
nashjc@uottawa.ca