NeuralNetTools: Visualization and Analysis Tools for Neural Networks

Marcus W. Beck

Oak Ridge Institute for Science and Education

US Environmental Protection Agency

Abstract

Functions within this package can be used for the interpretation of neural network models created in R, including functions to plot a neural network interpretation diagram, evaluation of variable importance, and a sensitivity analysis of input variables.

Keywords: neural networks, plotnet, sensitivity, variable importance, R.

1. Introduction

Data science is a relatively new paradigm of analysis that focuses on the synthesis of unstructured information from multiple sources to identify patterns or trends 'born from the data' (Kelling *et al.* 2009). A central theme is the focus on data exploration and prediction as compared to hypothesis-testing using domain-specific methods for scientific exploration (Kell and Oliver 2003). Demand for quantitative toolsets to address challenges in data-rich environments has increased drastically with the advancement of techniques for rapid acquisition of data. Fields of research characterized by high-throughput data have a strong foundation in computationally-intensive methods of analysis (e.g., Saeys *et al.* (2007)). Morever, disciplines that have historically been limited by data quantity, such as ecological studies across broad temporal and spatial scales, have also realized the importance of data intensive approaches given the increasing use of novel techniques to acquire information (e.g., Swanson *et al.* (2015)). Regardless of the discipline, quantitative methods that explicitly focus on inductive reasoning can serve a complementary role to conventional, hypothesis-driven approaches to scientific discovery (Kell and Oliver 2003).

Statistical methods that have been used to support data exploration for inductive analysis are numerous (Jain *et al.* 2000). A common theme among these methods is the use of machine-learning algorithms where the primary objective is to identify emergent patterns in the data

with minimal human intervention. Neural networks, in particular, are designed to mimic the neuronal structure of the human brain by 'learning' inherent data structures through adaptive algorithms (Rumelhart *et al.* 1986; Ripley 1996). Although the conceptual model was introduced several decades ago (McCulloch and Pitts 1943), neural networks have had a central role in data intensive science. The most popular form of neural network is the feed-forward multilayer perceptron (MLP) trained using the backpropagation algorithm (Rumelhart *et al.* 1986). This model is typically used to predict the response of one or more variables given one to many explanatory variables. The hallmark feature of the MLP is the characterization of relationships using an arbitrary number of parameters (i.e., the hidden layer) that are chosen through an iterative training process with the backpropation algorithm. Conceptually, the MLP is nothing more than a hyper-parameterized non-linear model that can fit a smooth function to any dataset with almost non-existent residual error (Hornik 1991).

An arbitrarily large number of parameters to fit a neural network provides obvious predictive advantages, but conversely complicates the extraction of critical model information. Information such as variable importance or model sensitivity are necessary aspects of exploratory data analysis that are not easily obtained from a neural network. As such, a common criticism is that neural networks are 'black-boxes' that offer minimal insight into relationships among variables (e.g., Paruelo and Tomasel 1997). Olden and Jackson (2002) provide a rebuttal to this concern by describing methods to extract information from neural networks, most of which were previously available but not commonly used. For example, Olden and Jackson (2002) describe the neural interpretation diagram (NID) for plotting (Özesmi and Özesmi 1999), the Garson algorithm for variable importance (Garson 1991), and the Profile method for sensitivity analysis (Lek *et al.* 1996). These quantitative tools 'illuminate the black box' by disaggregating the network parameters to characterize relationships between variables that are described by the model. In essence, MLP neural networks were developed for prediction but methods described in (Olden and Jackson 2002) leverage these models to describe data signals. Increasing the accesibility of these diagnostic tools will have value for exploratory analysis in data science.

This article describes the **NeuralNetTools** package for R that was developed to improve the breadth and quality of information obtained from the MLP neural network. Functions provided by the package are those previously described in (Olden and Jackson 2002) but have not been available in an open-source programming environment. The reach of the package is all-inclusive such that generic functions were developed using S3 methods for all neural network object classes available in R. The objecives of this article are to 1) provide an overview of the statistical foundation the MLP network, 2) briefly describe similarities and differences between existing neural network packages in R, and 3) describe the theory and application of the primary functions in the **NeuralNetTools** package. The package is currently available on the Comprehensive R Archive Network (CRAN), whereas the development version is maintained as a GitHub repository.

2. Theoretical foundation and existing R packages

An intriguing characteristic of neural networks is the relaxation of assumptions for the distributional characteristics of the response variables. Unlike conventional approaches, neural networks have been popularized by their ability to model response variables with arbitrary distributions and can describe relationships in datasets with noisy or imprecise information.

The typical MLP network is composed of multiple layers that define the transfer of information between input and response layers. Information travels in one direction where a set of values for one to many variables in the input layer propagates through one or more hidden layers to the resulting layer of the response variables. 'Hidden' layers between the input and response layers are key components of a neural network that mediate the transfer of information. Just as the input and response layers are composed of variables or 'nodes', each hidden layer is composed of nodes with weighted connections that define the strength of information flow between layers. 'Bias' layers connected to hidden and response layers may also be used that are analagous to intercept terms in a standard regression model.

Training a neural network model requires identifying the 'optimal' weights that define the connections between the model layers. The optimal weights are those that minimize prediction error for a test dataset that is independent of the training dataset. Training is commonly achieved using the backpropagation algorithm described in detail in (Rumelhart *et al.* 1986). This algorithm identifies the optimal weighting scheme between layers through an iterative process where weights are gradually changed through a forward- and backward-propagation process (Rumelhart *et al.* 1986; Lek and Guégan 2000). The algorithm begins by assigning an arbitrary weighting scheme to the connections in the network, followed by estimating the output in the response variable through the forward-propagation of information through the network, and finally calculating the difference between the predicted and actual value of the response. The weights are then changed through a back-propagation step that begins by changing weights in the output layer and then the remaining hidden layers. The process is repeated until the chosen error function is minimized. Formulaically, a generic MLP neural network can be represented as (Ripley 1996):

$$y_k = f_k \left(\sum_{j=1}^{k} w_{jk} f_j \left(\sum_{i=1}^{j} w_{ij} x_i \right) \right) \tag{1}$$

where the estimated value of the response variable y_k is a summation of the product between multiple input variables x and multiple hidden nodes j, as mediated by the respective weights w and activation functions f_j and f_k for each hidden and output node as the information progresses through the network.

Methods in **NeuralNetTools** were written for several R packages that can be used to create MLP neural networks: **neuralnet** (Fritsch and Guenther 2012), **nnet** (Venables and Ripley 2002), and **RSNNS** (Bergmeir and Benítez 2012). Limited methods were also developed for neural network objects created with the `train` function from the **caret** package (Kuhn 2015). Additional R packages that can create MLP neural networks include **AMORE** that implements the "TAO-robust back-propagation algorithm" for model fitting (Limas *et al.* 2014), **FCNN4R** as an R interface to the **FCNN** C++ library (Klima 2015), **monmlp** for networks with partial monotonicity constraints (Cannon 2015), and **qrnn** for quantile regression neural networks (Cannon 2011). At the time of writing, the CRAN download logs (Csardi 2015) showed that the R packages with methods in **NeuralNetTools** included 93% of all downloads for the available MLP packages, with **nnet** accounting for over 78%. As such, methods have not been included in **NeuralNetTools** for the remaining packages, although further development of **NeuralNetTools** could include additional methods based on popularity. Methods are currently available for `mlp` (**RSNNS**), `nn` (**neuralnet**), `nnet` (**nnet**), and `train` (**caret**, only if the object also inherits from the `nnet` class) objects. Additional `default` or `numeric` methods are available for some of the generic functions.

3. Package structure

The stable release of **NeurelNetTools** can be installed from CRAN and the development version can be installed from GitHub:

```
# install from CRAN
install.packages('NeuralNetTools')
library(NeuralNetTools)

# install from GitHub
install.package('devtools')
library(devtools)
install_github('fawda123/NeuralNetTools')
library(NeuralNetTools)
```

NeuralNetTools includes four main functions that were developed following techniques described in Olden and Jackson (2002) and references therein. These functions include `plotnet` to plot a neural network interpretation diagram, `garson` and `olden` functions to evaluate variable importance, and `lekprofile` for a sensitivity analysis of neural network response to input variables. Most of the functions require the extraction of model weights in a common format for each of the neural network object classes in R. The `neuralweights` function can be used to retrieve model weights for any of the model classes describe above. A two-element `list` is returned with the first element describing the structure of the network and the second element as a named list of weight values for the input model. The function is used internally within the main package functions but may be useful for comparing networks of different object classes.

A sample dataset is also included with **NeuralNetTools**. The `neuraldat` dataset is a simple `data.frame` with 300 rows of observations and columns for two response variables (`Y1` and `Y2`) and three input variables (`X1`, `X2`, and `X3`. The input variables are random observations from a standard normal distribution and the response variables are linear combinations of the input variables with additional random components. The response variables are also standardized from zero to one. A common approach for data preprocessing prior to creating a neural network is to normalize the input variables and to standardize the response variables (Lek and Guégan 2000; Olden and Jackson 2002). Novel datasets can be preprocessed to this common format using the `scale` function from **base** and the `rescale` function from **scales** for the input and response variables, respectively. The examples below use three models created from the `neuraldat` dataset and include `mlp` (**RSNNS**), `nn` (**RSNNS**), and `nnet` (**nnet**) objects (note the syntax differences).

```
# set random seed for initial weights for training
set.seed(123)

# mlp object, RSNNS package
library(RSNNS)
x <- neuraldat[, c('X1', 'X2', 'X3')]
y <- neuraldat[, 'Y1']
mod1 <- mlp(x, y, size = 5)
```

```r
# nn object, neuralnet package
library(neuralnet)
mod2 <- neuralnet(Y1 ~ X1 + X2 + X3, data = neuraldat, hidden = 5)

#nnet object, nnet package
library(nnet)
mod3 <- nnet(Y1 ~ X1 + X2 + X3, data = neuraldat, size = 5)
```

3.1. Visualizing neural networks

The number of existing functions in **R** to view neural networks is minimal. Such tools have practical use for visualizing network architecture and connections between layers that mediate variable importance. To our knowledge, only the **neuralnet** and **FCNN4R** packages provide plotting methods for MLP networks in R. Although useful for viewing the basic structure, the output is minimal and does not include extensive options for customization.

Tbe `plotnet` function in **NeuralNetTools** plots a network as a neural interpretation diagram (Özesmi and Özesmi 1999) and includes several options to customize plot aesthetics. A NID is a modification to the standard conceptual illustration of the MLP network that allows the user to view the thickness and color of the weight connections based on mangnitude and sign, respectively. The default settings plot positive weights between layers as black lines and negative weights as grey lines. Line thickness is in proportion to absolute magnitude of each weight (fig. 1).

```r
par(mar = c(0, 0, 0, 0))
plotnet(mod3, nid = F, circle_col = 'grey', bord_col = 'grey')
plotnet(mod3)
```

A primary and skip layer network can also be plotted for `nnet` models with a skip layer connection. The default is to plot the primary network, whereas the skip layer network can be viewed with `skip = TRUE`. If `nid = TRUE`, the line widths for both the primary and skip layer plots are relative to all weights. Viewing both plots is recommended to see which network has larger relative weights. Plotting a network with only a skip layer (i.e., no hidden layer, `size = 0`) will include bias connections to the output layer, whereas these are not included in the plot of the skip layer if size is greater than zero.

```r
# create a model with a skip layer
modskip <- nnet(Y1 ~ X1 + X2 + X3, data = neuraldat, size = 5, skip = TRUE)

# plot
par(mar = c(0, 0, 0, 0))
plotnet(modskip, skip = TRUE)
plotnet(modskip)
```

The **RSNNS** package provides several algorithms that can be used to prune connections or nodes in a neural network. This approach can remove connection weights between layers or

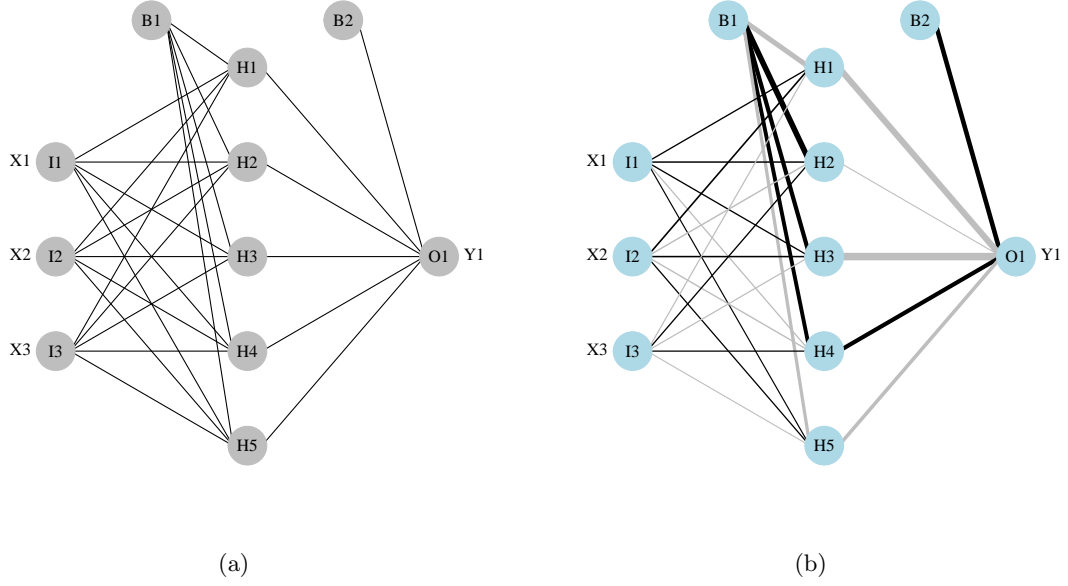(a)   (b)

Figure 1: Examples from the `plotnet` function showing neural networks as a standard graphic (1a) and using the neural interpretation diagram (1b). Labels outside of the nodes represent variable names and labels within the nodes indicate the layer and node (I: input, H: hidden, O: output, B: bias). Default options are changed for the example.
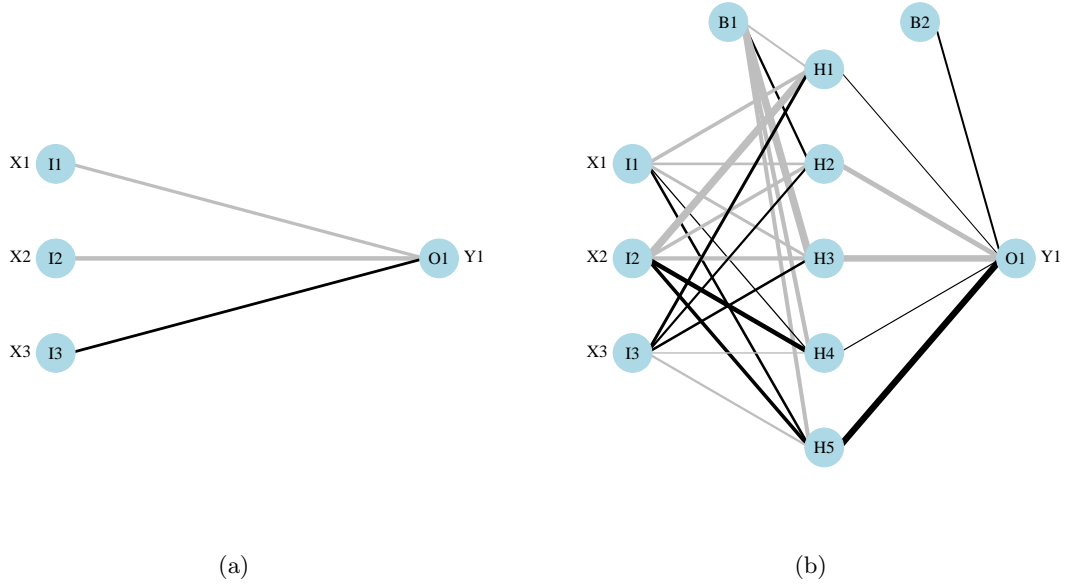

(a)   (b)

Figure 2: Examples from the `plotnet` function showing a neural network with a separate skip layer between the input and output layers. The skip layer (2a) and primary neural network (2b) can be viewed separately with `plotnet`.

input nodes that do not contribute to the predictive performance of the network. Pruning has the benefit of a potential increase in power by increasing the degrees of freedom or the identification of input nodes with low importance. Algorithms in **RSNNS** for weight pruning include magnitude based pruning, Optimal Brain Damage, and Optimal Brain Surgeon, whereas algorithms for node pruning include skeletonization and the non-contributing units method (Zell *et al.* 1998). The `plotnet` function can be used to plot a pruned neural network, with options to omit or display the pruned connections (fig. 3).

```
# pruned model using RSNNS
pruneFuncParams <- list(max_pr_error_increase = 10.0, pr_accepted_error = 1.0,
  no_of_pr_retrain_cycles = 1000, min_error_to_stop = 0.01,
  init_matrix_value = 1e-6, input_pruning = TRUE, hidden_pruning = TRUE)
mod <- mlp(x, y, size = 5, pruneFunc = "OptimalBrainSurgeon",
 pruneFuncParams = pruneFuncParams)

# plot, pruned connections are omitted (default) or included with prune_col
par(mar = c(0, 0, 0, 0))
plotnet(mod, rel_rsc = c(3, 8))
plotnet(mod, prune_col = 'lightblue', rel_rsc = c(3, 8))
```
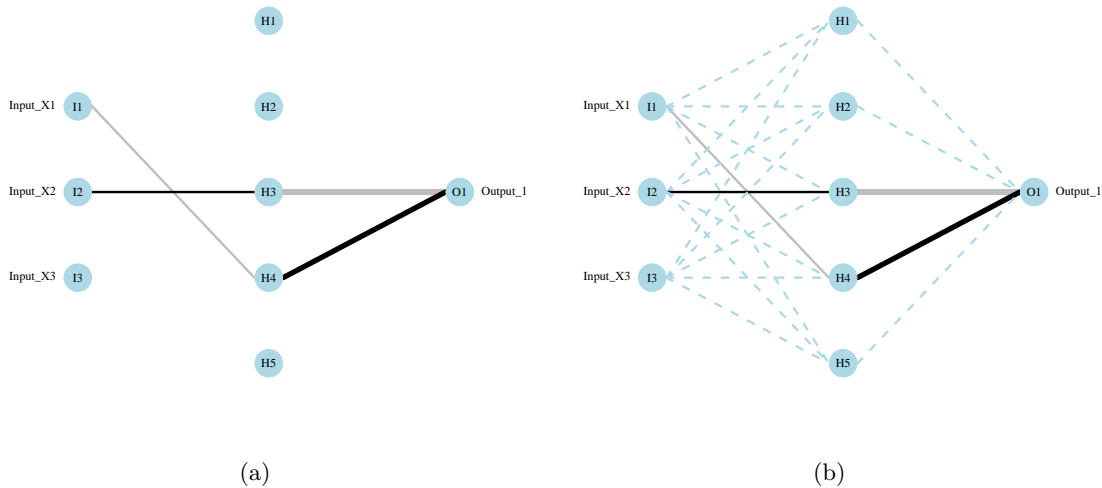

(a) (b)

Figure 3: A pruned neural network using the "Optimal Brain Surgeon" algorithm described in Zell *et al.* (1998). The default plotting behavior of `plotnet` is to omit pruned connections (3a), whereas they can be viewed as dashed lines by including the `prune_col` argument (3b).

3.2. Evaluating variable importance

The primary benefit of visualizing a NID with `plotnet` is the ability to evaluate network architecture and the variation in weighted connections between the layers. Although useful as a general tool, the NID is impractical for evaluating most networks given the amount of

weighted connections. Alternative methods to quantitatively describe a neural network deconstruct the model weights to determine variable importance, whereas similar information can only be qualitatively inferred from `plotnet`. Two algorithms for evaluating variable importance are available in **NeuralNetTools**: Garson's algorithm for relative importance (Garson 1991; Goh 1995) and Olden's conection weights algorithm (Olden *et al.* 2004).

Garson's algorithm was originally described by Garson (1991) and further modified by Goh (1995). The `garson` function is an implementation of the method described in the appendix of Goh (1995). This method identifies the relative importance of each variable as an absolute magnitude by deconstructing all weighted connections between the layers in the network. For each input node, all weights connecting an input through the hidden layer to the response variable are identified to return a list of all weights specific to each input variable. A summed product of the connections for each input node are then scaled relative to all other inputs. A value for each input node indicates relative importance as the absolute magnitude from zero to one. The method is limited in that the direction of the response cannot be determined and only neural networks with one hidden layer and one output node can be evaluated.

The `olden` function is a more flexible approach to evaluate variable importance using the connection weights algorithm described in Olden *et al.* (2004). This method calculates importance as the product of the raw input-hidden and hidden-output connection weights between each input and output node and sums the product across all hidden nodes. An advantage is the relative contributions of each connection weight are maintained in both magnitude and sign, as compared to Garson's algorithm which only considers absolute magnitude. For example, connection weights that change sign (e.g., positive to negative) between the input-hidden to hidden-output layers would have a cancelling effect whereas Garson's algorithm may provide misleading results based on the absolute magnitude. An additional advantage is that Olden's algorithm is capable of evaluating neural networks with multiple hidden layers and response variables. The importance values assigned to each variable are also in units based on the summed product of the connection weights, whereas `garson` returns importance scaled from 0–1.

Both functions have similar implementations and require only a model object as input. The default output is a **ggplot2** bar plot (i.e., `geom_bar`, Wickham 2009) that shows the relative importance of each input variable in the model (fig. 4). The plot aesthetics are based on internal code from the function but can be changed using conventional syntax for **ggplot2**. The importance values can also be returned as a `data.frame` if `bar_plot = FALSE`.

```
# garson and olden functions for each model
garson(mod1)
olden(mod1)
garson(mod2)
olden(mod2)
garson(mod3)
olden(mod3)
```

3.3. Sensitivity analysis

An alternative approach to evaluate variable interactions in a neural network is the Lek profile method, described briefly in Lek *et al.* (1996) and in more detail in Gevrey *et al.* (2003). The
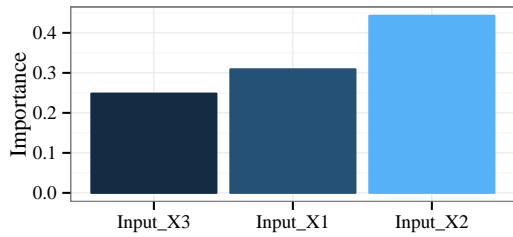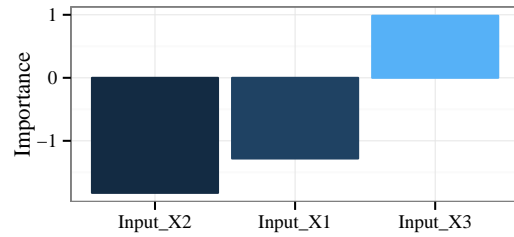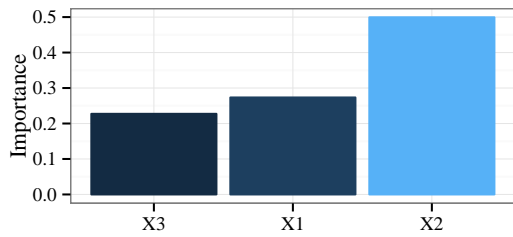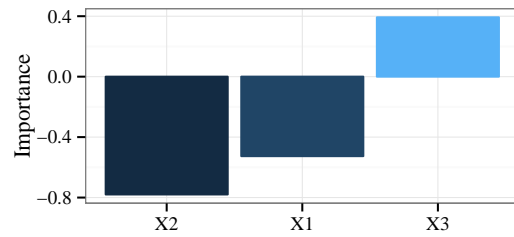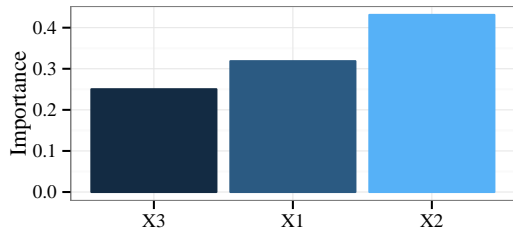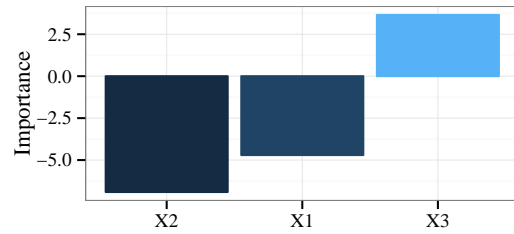
Figure 4: Variable importance for three models using Garson's algorithm for relative importance (`garson`, Garson 1991; Goh 1995) and Olden's conection weights algorithm (`olden`, Olden *et al.* 2004). Garson's algorithm shows importance as absolute values from 0–1 whereas Olden's algorithm preserves sign and magnitude. Importance values for Olden's algorithm are directly from the summed product of model weights and should only be evaluated based on the sign and relative magnitudes between variables.

profile method is fairly generic and can be extended to any statistical model in **R** with a `predict` method. However, it is one of few methods used to evaluate sensitivity in neural networks. The profile method differs fundamentally from the variable importance algorithms by evaluating the behavior of response variables across different values of the input variables.

The `lekprofile` function evaluates the effect of explanatory variables by returning a plot of the predicted response across the range of values for each variable. A critical aspect of the method is the remaining explanatory variables that are held constant when separately evaluating each input variable. The `lekprofile` function provides two options for determining constant values of unevaluated explanatory variables. The first option following the original profile method evaluates effects of each variable while holding the remaining expalanatory variables at different quantiles (e.g., minimum, 20th percentile, maximum, figs. 5a and 6a). This is implemented by creating a matrix of values for explanatory variables where the number of rows is the number of observations in the original dataset and the number of columns is the number of explanatory variables. All explanatory variables are held at their median (or other constant value), while the variable of interest is sequenced from its minimum to maximum value across the range of observations. This matrix is then used to predict values of the response variable from a fitted model object. This is repeated for each explanatory variable to obtain all response curves. Constant values are defined by passing one or more numeric values in the range 0–1 to the `group_vals` that define the quantiles, with the default as minimum, 20th, 40th, 60th, 80th, and maximum percentiles.

An alternative implementation of `lekprofile` is to group the unevaluated explanatory variables using natural groupings defined by statistical properties of the data. Covariance among predictors may present unlikely scenarios if all unevaluated variables at held at the same level (e.g., high values for one variable may be unlikely to co-occur with high values for a second variable). To address this issue, a second option is available to hold unevaluated variables at mean values defined by natural clusters in the data (figs. 5b and 6b). Clusters are identified using k-means clustering (`kmeans` from base **stats**, Hartigan and Wong 1979) of the input variables if the argument passed to `group_vals` is an integer value greater than one. The centers (means) of the clusters are then used as constant values for the unevaluated variables. Beck *et al.* (2014) provide an example of the clustering method for `lekprofile` by evaluating response of a lake health index to different explanatory variables. Lake clusters were identified given covariance among explanatory variables, such that holding explanatory variables at related values created more ecologically relevant response curves. Similar plots can be created with `lekprofile` as response curves (fig. 5) or to visualize the groupings for unevaluated explanatory variables (fig. 6). For the latter case, `split_show = TRUE` will return a stacked bar plot for each group with heights in each bar proportional to the constant values.

```
# lek profile, quantile evaluation
lekprofile(mod3)

# lek profile, quantile groups
lekprofile(mod3, group_show = TRUE)

# lek profile, cluster evaluation
lekprofile(mod3, group_vals = 6)
```

(a) Response with quantiles
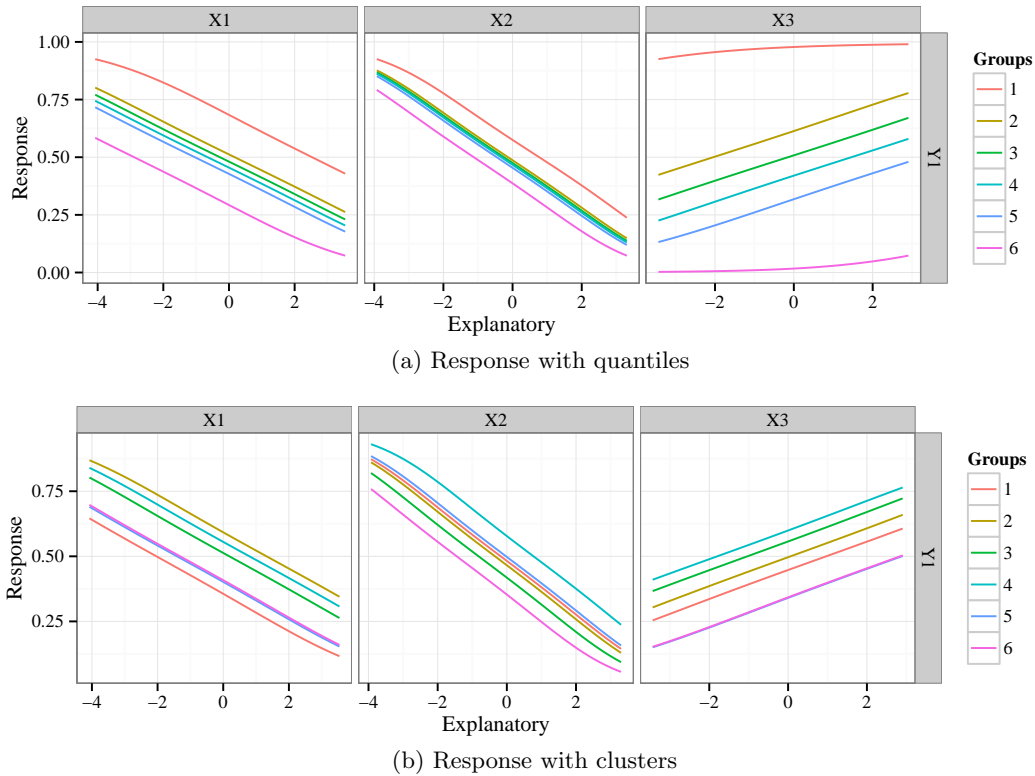

(b) Response with clusters

Figure 5: Sensitivity analysis of a neural network using the Lek profile method to evaluate the effects of explanatory variables. Figure 5a groups unevaluated explanatory variables at quantiles and fig. 5b groups by cluster means from k-means clustering of all explanatory variables. Values at which explanatory variables are held constant in each plot are shown in figs. 6a and 6b, respectively.

```
# lek profile, cluster groups
lekprofile(mod3, group_vals = 6, group_show = TRUE)
```

4. Applied example

Although **NeuralnetTools** provides several methods to extract information from a neural network, it does not provide explicit guidance for developing the initial model. A more challenging aspect is understanding the effects of network architecture, use of training and validation datasets, and implications for the bias-variance tradeoff with model over- or under-fitting during model development. A detailed discussion of these issues is beyond the scope of this paper, although an example application is presented below to introduce a 'best practices' approach to neural network development. The models presented above, including the `neuraldat` dataset, are contrived examples to illustrate use of the **NeuralNetTools** package and they do not demonstrate a comprehensive application of neural networks in practice. Several texts provide explicit guidelines on developing neural network models (e.g., Ripley 1996; Lek and Guégan 2000; Maier and Dandy 2000). In general, the following should be considered during model development:
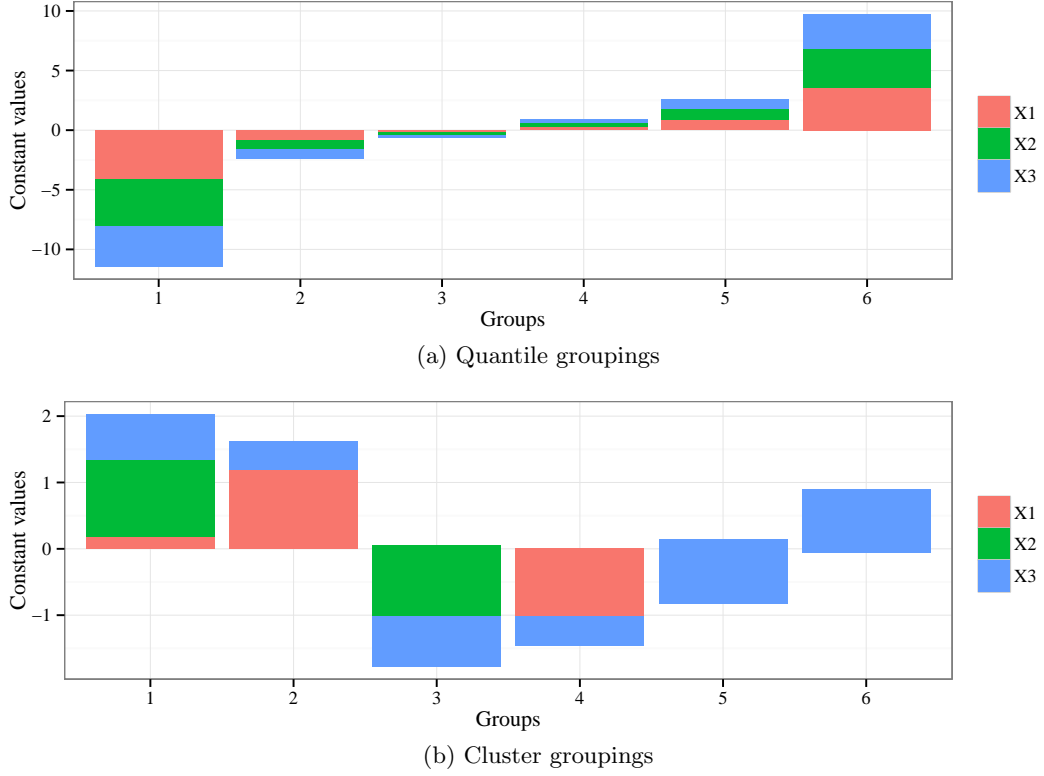
(a) Quantile groupings

(b) Cluster groupings

Figure 6: Bar plots for values in each group of unevaluated explanatory variables in figs. 5a and 5b. Figure 6a shows default quantile groupings set at the minimum, 20th, 40th, 60th, 80th, and maximum percentiles. For example, variables are held at zero for group 1 (i.e., stacked bars with no height) for the minimum value, whereas group 6 holds variables at their maximum of one. Figure 6b shows the cluster centers for each variable in each group identified from *k*-means clustering.

- Initial data pre-processing to normalize inputs and standardize response,

- Network architecture including number of hidden layers, number of nodes in each hidden layer, inclusion of bias or skip layers, pruning weights or inputs,

- Split of dataset into a training and test dataset, e.g., 2:1, 3:1, leave-one-out, etc.,

- Initial starting weights for the back-propagation algorithm, and

- Criteria for stopping model training iterations, e.g., error convergance tolerance, maximum number of iterations, minimum error on test dataset, etc.

```
library(nycflights13)
library(dplyr)

# data preprocessing
# select UA carrior for month of december and relevant variables
# normalize and standardize variables
tomod <- filter(flights, month == 12 & carrier == 'UA') %>%
  select(arr_delay, dep_delay, dep_time, arr_time, air_time, distance) %>%
  mutate_each(funs(scale), -arr_delay) %>%
  mutate_each(funs(as.numeric), -arr_delay) %>%
  mutate(arr_delay = scales::rescale(arr_delay, to = c(0, 1))) %>%
  data.frame

# create two models
# random starting weights are different for each
ex1 <- nnet(arr_delay ~ ., size = 5, linout = TRUE, data = tomod)
ex2 <- nnet(arr_delay ~ ., size = 5, linout = TRUE, data = tomod)

# view olden results
olden(ex1)
olden(ex2)
```

5. Conclusions

The **NeuralNetTools** package provides a simple approach that improves the quality of information obtained from a feed-forward MLP neural network. Functions in the package can be used to visualize a neural network using a neural interpretation diagram (`plotnet`), evaluate variable importance (`garson`, `olden`), and conduct a sensitivity analysis (`lekprofile`). Methods are available for the most used CRAN packages that can create neural networks (**caret**, **neuralnet**, **nnet**, **RSNNS**), whereas additional methods will be made available based on future popularity of the remaining packages (**AMORE**, **FCNN4R**, **monmlp**, **qrnn**). A primary objective of the package is to dispel the myth that supervised neural networks for prediction are 'black boxes' that provide no information about underlying relationships between the variables (Paruelo and Tomasel 1997; Olden and Jackson 2002). Although neural networks are considered complex relative to more conventional approaches, the theoretical foundation

has many parallels with traditional modelling techniques that can allow insight into causation. Moreover, the model fitting process requires the minimization of an objective function such that conventional techniques to evaluate model sensitivity or performance (e.g., cross-validation) can be used with neural networks. Functions in **NeuralNetTools** can facilitate the selection of the optimal network architecture or can be used for post-hoc assessment.

A more compelling issue is when and how to apply neural networks given alternative methods for data-intensive exploration. The popularity of the MLP neural network is partially the cause of misperceptions and generalizations of their benefits as modelling tools (Burke and Ignizio 1997). Indeed, the 'neural' component of neural networks is advertised as a mathematical representation of the network of synaptic impulses in the human brain, although the former is of course an over-simplification of the latter. Several examples have shown that the MLP network may provide comparable predictive performance as similar statistical methods (Feng and Wang 2002; Razi and Athappilly 2005; Beck *et al.* 2014). As such, the neural network should be considered a tool in the larger toolbox of data-intensive methods that should be used after examination of the tradeoffs between techniques, with particular emphasis on the needs of a given dataset or research question. For example, an appropriate method should be chosen based on power given the sample size, expected linear or non-linear interactions between variables, distributional forms of the response, and other relevant concerns that are routinely considered during exploratory data analysis. The true value of the **NeuralNetTools** package is a set of analysis tools that can inform a more balanced assessment given the alternative and complementary methods for data-intensive exploration.

6. Acknowledgments

References

Beck MW, Wilson BN, Vondracek B, Hatch LK (2014). "Application of neural networks to quantify the utility of indices of biotic integrity for biological monitoring." *Ecological Indicators*, **45**, 195–208.

Bergmeir C, Benítez JM (2012). "Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS." *Journal of Statistical Software*, **46**(7), 1–26. URL http://www.jstatsoft.org/v46/i07/.

Burke L, Ignizio JP (1997). "A practical overview of neural networks." *Journal of Intelligent Manufacturing*, **8**(3), 157–165.

Cannon AJ (2011). "Quantile regression neural networks: implementation in R and application to precipitation downscaling." *Computers & Geosciences*, **37**, 1277–1284. doi:10.1016/j.cageo.2010.07.005.

Cannon AJ (2015). *monmlp: Monotone Multi-Layer Perceptron Neural Network*. R package version 1.1.3, URL http://CRAN.R-project.org/package=monmlp.

Csardi G (2015). *cranlogs: Download Logs from the 'RStudio' 'CRAN' Mirror.* R package version 2.1.0, URL https://github.com/metacran/cranlogs.

Feng CX, Wang X (2002). "Digitizing uncertainty modeling for reverse engineering applications: regression versus neural networks." *Journal of Intelligent Manufacturing*, **13**(3), 189–199.

Fritsch S, Guenther F (2012). *neuralnet: Training of neural networks.* R package version 1.32, URL http://CRAN.R-project.org/package=neuralnet.

Garson GD (1991). "Interpreting neural network connection weights." *Artificial Intelligence Expert*, **6**(4), 46–51.

Gevrey M, Dimopoulos I, Lek S (2003). "Review and comparison of methods to study the contribution of variables in artificial neural network models." *Ecological Modelling*, **160**(3), 249–264.

Goh ATC (1995). "Back-propagation neural networks for modeling complex systems." *Artificial Intelligence in Engineering*, **9**(3), 143–151.

Hartigan JA, Wong MA (1979). "A K-means clustering algorithm." *Applied Statistics*, **28**(1), 100–108.

Hornik K (1991). "Approximation capabilities of multilayer feedforward networks." *Neural Networks*, **4**(2), 251–257.

Jain AK, Duin RPW, Mao JC (2000). "Statistical pattern recognition: A review." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(1), 4–37.

Kell DB, Oliver SG (2003). "Here is the evidence, now what is the hypothesis? The complementary roles of inductive and hypothesis-driven science in the post-genomic era." *BioEssays*, **26**(1), 99–105.

Kelling S, Hochachka WM, Fink D, Riedewald M, Caruana R, Ballard G, Hooker G (2009). "Data-intensive science: A new paradigm for biodiversity studies." *BioScience*, **59**(7), 613–620.

Klima G (2015). *FCNN4R: Fast Compressed Neural Networks for R.* R package version 0.4.1.

Kuhn M (2015). *caret: Classification and Regression Training.* R package version 6.0-52, URL http://CRAN.R-project.org/package=caret.

Lek S, Delacoste M, Baran P, Dimopoulos I, Lauga J, Aulagnier S (1996). "Application of neural networks to modelling nonlinear relationships in ecology." *Ecological Modelling*, **90**(1), 39–52.

Lek S, Guégan JF (2000). *Artificial Neuronal Networks: Application to Ecology and Evolution.* Springer-Verlag, Berlin, Germany.

Limas MC, Meré JBO, Marcos AG, de Pisón Ascacibar FJM, Espinoza AVP, Elías FA, Ramos JMP (2014). *AMORE: A MORE flexible neural network package.* R package version 0.2-15, URL http://CRAN.R-project.org/package=AMORE.

Maier HR, Dandy GC (2000). "Neural networks for the prediction and forecasting of water resources variables: A review of modelling issues and applications." *Environmental Modelling and Software*, **15**(1), 101–124.

McCulloch WS, Pitts W (1943). "A logical calculus of the ideas imminent in nervous activity." *Bulletin of Mathematical Biophysics*, **5**, 115–133.

Olden JD, Jackson DA (2002). "Illuminating the "black box": A randomization approach for understanding variable contributions in artifical neural networks." *Ecological Modelling*, **154**(1-2), 135–150.

Olden JD, Joy MK, Death RG (2004). "An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data." *Ecological Modelling*, **178**(3-4), 389–397.

Özesmi SL, Özesmi U (1999). "An artificial neural network approach to spatial habitat modelling with interspecific interaction." *Ecological Modelling*, **116**(1), 15–31.

Paruelo JM, Tomasel F (1997). "Prediction of functional characteristics of ecosystems: A comparison of artificial neural networks and regression models." *Ecological Modelling*, **98**(2-3), 173–186.

Razi MA, Athappilly K (2005). "A comparative predictive analysis of neural networks (NNs), nonlinear regression and classification and regression tree (CART) models." *Expert Systems and Applications*, **29**(1), 65–74.

Ripley BD (1996). *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, United Kingdom.

Rumelhart DE, Hinton GE, Williams RJ (1986). "Learning representations by back-propagating errors." *Nature*, **323**(6088), 533–536.

Saeys Y, Inza I, naga PL (2007). "A review of feature selection techniques in bioinformatics." *Bioinformatics*, **23**(19), 2507–2517.

Swanson A, Kosmala M, Lintott C, Simpson R, Smith A, Packer C (2015). "Snapshot Serengeti: High-frequency annotated camera trap images of 40 mammalian species in African savanna." *Scientific Data*, **2**, 150026.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S.* Fourth edition. Springer, New York, New York. URL http://www.stats.ox.ac.uk/pub/MASS4.

Wickham H (2009). *ggplot2: Elegant Graphics for Data Analysis.* Springer. New York. ISBN 978-0-387-98140-6. URL http://had.co.nz/ggplot2/book.

Zell A, Mamier G, Vogt M, Mache N, Hübner R, Döring S, Herrmann KU, Soyez T, Schmalzl M, Sommer T, Hatzigeorgiou A, Posselt D, Schreiner T, Kett B, Clemente G, Wieland J, Gatter J (1998). *SNNS: Stuttgart Neural Network Simulator, User Manual, Version 4.2.* University of Stuttgart and WSI, University of Tübingen, URL http://www.ra.cs.uni-tuebingen.de/SNNS/.

Affiliation:

Marcus W. Beck
Oak Ridge Institude for Science and Education
US Environmental Protection Agency
National Health and Environmental Effects Research Laboratory
Gulf Ecology Division, 1 Sabine Island Drive
Gulf Breeze, Florida, 32561, USA
E-mail: beck.marcus@epa.gov