NeuralSens: Sensitivity Analysis of Neural Networks

Jaime Pizarroso ⓘ
Universidad Pontificia
Comillas

José Portela ⓘ
Universidad Pontificia
Comillas

Antonio Muñoz ⓘ
Universidad Pontificia
Comillas

Abstract

This article presents the **NeuralSens** package that can be used to perform sensitivity analysis of neural networks using the partial derivatives method. The main function of the package calculates the partial derivatives of the output with regard to the input variables of a multi-layer perceptron model, which can be used to evaluate variable importance based on sensitivity measures and characterize relationships between input and output variables. Methods to calculate partial derivatives are provided for objects trained using common neural network packages in R, and a 'numeric' method is provided for objects from packages which are not included. The package also includes functions to plot the information obtained from the sensitivity analysis.

The article contains an overview of techniques for obtaining information from neural network models, a theoretical foundation of how partial derivatives are calculated, a description of the package functions, and applied examples to compare **NeuralSens** functions with analogous functions from other available R packages.

Keywords: neural networks, sensitivity, analysis, variable importance, R, **NeuralSens**.

1. Introduction

As the volume of available information increases in various fields, the number of situations where data-intensive analysis can be applied also grows simultaneously (Philip Chen and Zhang 2014; Valduriez *et al.* 2018).This analysis can be used to extract useful information and supports decision-making (Sun, Sun, and Strang 2018).

Machine-learning algorithms are commonly used in data-intensive analysis (Hastie, Tibshirani, and Friedman 2001; Butler, Davies, Cartwright, Isayev, and Walsh 2018; Vu *et al.* 2018), as they are able to detect patterns and relations in the data without being explicitly programmed. Artificial neural networks (ANN) are one of the most popular machine-learning algorithms due to their versatility. ANNs were designed to mimic the biological neural struc-

tures of animal brains (McCulloch and Pitts 1943) by "learning" to perform tasks by considering examples and modifying their structure through iterative algorithms (Rojas 1996). The form of ANN that is discussed in this paper is the feed-forward multilayer perceptron (MLP, Rumelhart, Hinton, and Williams 1986). MLPs are one of the most popular form of ANNs and have been used in a wide variety of applications (Mosavi, Salimi, Faizollahzadeh Ardabili, Rabczuk, Shamshirband, and Varkonyi-Koczy 2019; Smalley 2017; Hornik, Stinchcombe, and White 1989). This model consists of interconnected units, called nodes or perceptrons, that are arranged in layers. The first layer consists of inputs (or independent variables), the final layer is the output layer, and the layers in between are known as hidden layers (Özesmi and Özesmi 1999). Assuming that there is a relationship between the outputs and the inputs, the goal of the MLP is to approximate a non-linear function to represent the relationship between the output and the input variables of a given dataset with minimal residual error (Hornik 1991; Cybenko 1989).

Neural networks provide predictive advantages when compared to other models, such as the ability to implicitly detect complex non-linear relationships between dependent and independent variables. However, the complexity of neural networks makes it difficult to obtain information on how the model uses the input variables to predict the output. Finding methods for extracting information on how the input variables affect the response variable has been a recurrent topic of research in neural networks (Olden, Joy, and Death 2004; Zhang, Beck, Winkler, Huang, Sibanda, and Goyal 2018). Some examples are:

1. Neural interpretation diagram (NID) as described by Özesmi and Özesmi (1999) for plotting the ANN structure. A NID is a modified version of the standard representation of neural networks which changes the color and thickness of the connections between neurons based on the sign and magnitude of its weight.

2. Garson's method for variable importance (Garson 1991). It consists of summing the product of the absolute value of the weights connecting the input variable to the response variable through the hidden layer. Afterwards, the result is scaled relative to all other input variables. The relative importance of each input variable is given as a value from zero to one.

3. Olden's method for variable importance (Olden *et al.* 2004). This method is similar to Garson's, but it uses the real value instead of the absolute value of the connection weights and it does not scale the result.

4. Input perturbation (Scardi and Harding 1999; Gevrey, Dimopoulos, and Lek 2003). It consists of adding an amount of white noise to each input variable while maintaining the other inputs at a constant value. The resulting change in a chosen error metric for each input perturbation represents the relative importance of each input variable.

5. Profile method for sensitivity analysis (Lek, Delacoste, Baran, Dimopoulos, Lauga, and Aulagnier 1996). Similar to the input perturbation algorithm, it changes the value of one input variable while maintaining the other variables at a constant value. These constant values are different quantiles of each variable, therefore a plot of model predictions across the range of values of the input is obtained. A modification of this algorithm is proposed in Beck (2018). To avoid unlikely combinations of input values, a clustering technique is applied to the training dataset and the center values of the clusters are used instead of the quantile values.

6. Partial derivatives method for sensitivity analysis (Dimopoulos, Bourret, and Lek 1995; Dimopoulos, Chronopoulos, Chronopoulou-Sereli, and Lek 1999; Muñoz and Czernichow 1998; White and Racine 2001). It performs a sensitivity analysis by computing the partial derivatives of the ANN outputs with regard to the input neurons evaluated on the samples of the training dataset (or an analogous dataset).

7. Partial dependence plot (PDP, Friedman 2001; Goldstein, Kapelner, Bleich, and Pitkin 2015). PDPs help visualize the relationship between a subset of the input variables and the response while accounting for the average effect of the other inputs. For each input, the partial dependence of the response with regard to the selected input is calculated following two steps. Firstly, individual conditional expectation (ICE) curves are obtained, one for each sample of the training dataset. The ICE curve for sample k is built by obtaining the model response using the input values at sample k, except for the input variable of interest, whose value is replaced by other values it has taken in the training dataset. Finally, the PDP curve for the selected variable is calculated as the mean of the ICE curves obtained.

8. Local interpretable model-agnostic explanations (Ribeiro, Singh, and Guestrin 2016). The complex neural network model is explained by approximating it locally with an interpretable model, such as a linear regression or a decision tree model.

9. Forward stepwise addition (Gevrey *et al.* 2003). It consists of rebuilding the neural network by sequentially adding an input neuron and its corresponding weights. The change in each step in a chosen error metric represents the relative importance of the corresponding input.

10. Backward stepwise elimination (Gevrey *et al.* 2003). It consists of rebuilding the neural network by sequentially removing an input neuron and its corresponding weights. The change in each step in a chosen error metric represents the relative importance of the corresponding input.

These methods help with neural network diagnosis by retrieving useful information from the model. However, these methods have some disadvantages: NID can be difficult to interpret given the amount of connections in most networks, Garson's and Olden's algorithms only account for the weights of the input variable connections in the hidden layer, and Lek's profile method may present analyses of input scenarios not represented by the input training data or require other methods like clustering (using the center of the clusters instead of the range quantiles of the input variables) with its inherent disadvantages (Xu and Tian 2015). Partial dependence plots have a similar disadvantage as they might provide misleading information if the value of the output variable depends not only on the variable of interest but also on compound effects of input variables. Local linearization is useful for interpreting the input variable importance in specific regions of the dataset, but it does not give a quantitative importance measure for the entire dataset. Forward stepwise addition and backward stepwise elimination perform a more exhaustive analysis, but are computationally expensive and may produce different results based on the order in which the inputs are added/removed and the initial training conditions of each model.

The partial derivatives method overcomes these disadvantages by analytically calculating the derivative of each output variable with regard to each input variable evaluated on each data

sample of a given dataset. The contribution of each input is calculated in both magnitude and sign taking into account not only the connection weights and the activation functions, but also the values of each input. By using all the samples of the dataset, the effect of the input variables in the response is calculated for the real values of the data, avoiding information loss due to clustering. Analytically calculating the derivatives results in more robust diagnostic information, because it depends solely on how well the neural network predicts the output. As long as the neural network predicts the output variable with enough precision, the derivatives will be the same regardless of the training conditions and the structure of the network (Beck 2018).

As stated before, the main objective of the proposed methods is to extract information from a given neural network model. For example, unnecessary inputs may lead to a higher complexity of the neural structure and prevent finding the optimal model, thus, affecting the performance of the neural network. Several researchers defend the ability of the partial derivatives method to determine whether an explanatory variable is irrelevant for predicting the response of the neural network (White and Racine 2001; Zurada, Malinowski, and Cloete 1994; Engelbrecht, Cloete, and Zurada 1995). Pruning the neural network of these irrelevant inputs improves the capability of the neural network to model the relationship between response and explanatory variables and, consequently, the quality of information that can be extracted from the model.

Using the partial derivatives method has some disadvantages that should be noted. The operations required to calculate partial derivatives are time-consuming when compared to other methods such as Garson's and Olden's. The computing time grows as the size of the neural network or the size of the database used to calculate the partial derivatives increases. Additionally, the input variables should be normalized when using this method, as otherwise the value of the partial derivatives may depend on the scale of each variable and produce misleading results. However, its advantages with regard to other methods make sensitivity analysis a very useful technique for interpreting and improving neural network models.

This article describes the **NeuralSens** package (Portela, Muñoz, and Pizarroso 2022) for R (R Core Team 2021) which can be used to perform sensitivity analysis of MLP neural networks using partial derivatives. The main function of the package includes methods for MLP objects from the most popular neural network packages available in R. To the authors' knowledge, there is no other R package that calculates the partial derivatives of a neural network. The **NeuralSens** package is available at the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=NeuralSens`, and the development version is maintained as a GitHub repository at `https://github.com/JaiPizGon/NeuralSens`. It should be mentioned that other algorithms to analyze neural networks are already implemented in R: NID, Garson's, Olden's and Lek's profile algorithms are implemented in **NeuralNetTools** (Beck 2018), the partial dependence plots method is implemented in **pdp** (Greenwell 2017) and local linearization is implemented in **lime** (Pedersen and Benesty 2021).

The rest of this article is structured as follows. Section 2 describes the theory of the functions in the **NeuralSens** package, along with references to general introductions to neural networks. Section 3 presents the architecture details of the package. Section 4 shows applied examples for using the **NeuralSens** package, comparing the results with packages currently available in R. Finally, Section 5 concludes the article.

2. Theoretical foundation

The **NeuralSens** package has been designed to calculate the partial derivatives of the output with regard to the inputs of a MLP model in R. The remainder of this section explains the theory of multilayer perceptron models, how to calculate the partial derivatives of the output of this type of model with regard to its inputs and some sensitivity measures proposed by the authors.

2.1. Multilayer perceptron

A fully-connected feed-forward MLP has one-way connections from the units of one layer to all neurons of the subsequent layer. Each time the output of one unit travels along one connection to another unit, it is multiplied by the weight of the connection. At each unit the inputs are summed and a constant, or bias, is added. Once all the input terms of each unit are summed, an activation function is applied to the result.

Figure 1 shows the scheme of a neuron in a MLP model and represent graphically the operations in Equation 1.

For each neuron, the output y_k^l of the k-th neuron in the l-th layer can be calculated by:

$$y_k^l = \phi_k^l \left(z_k^l \right) = \phi_k^l \left(\sum_{j=1}^{n^{l-1}} w_{kj}^l \cdot y_j^{l-1} + w_{k0}^l \cdot b^l \right) \tag{1}$$

where z_k^l refers to the weighted sum of the neuron inputs, n^{l-1} refers to the number of neurons in the $(l-1)$-th layer, w_{kj}^l refers to the weight of the connection between the j-th neuron in the $(l-1)$-th layer and the k-th neuron in the l-th layer, ϕ_k^l refers to the activation function of the k-th neuron in l-th layer, b^l refers to the bias in the l-th layer and $\cdot$ refers to the scalar product operation. For the input layer thus holds $l = 1$, $y_j^{1-1} = x_j$, $w_{kj}^1 = 1$ and $b^1 = 0$.
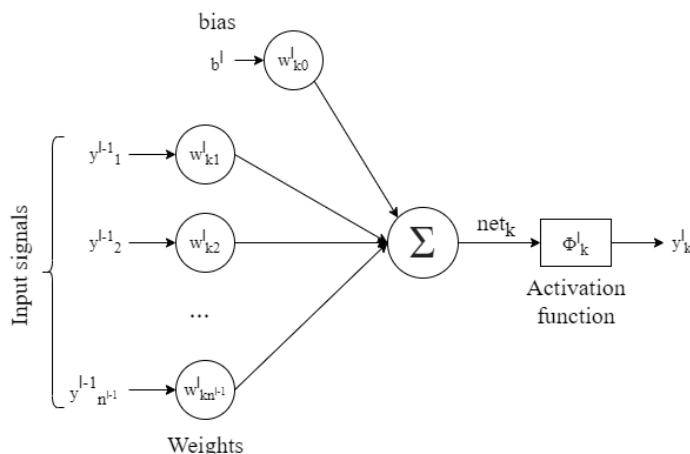

Figure 1: Scheme of the k-th neuron in the l-th layer of a MLP model. ϕ_k^l represent the activation function of the neuron, b^l represent the bias of the l-th layer, y_j^{l-1} represent the output of the j-th neuron in the previous layer and w_{jk}^l represent the weight of the connection between the neuron and the j-th neuron of the previous layer.
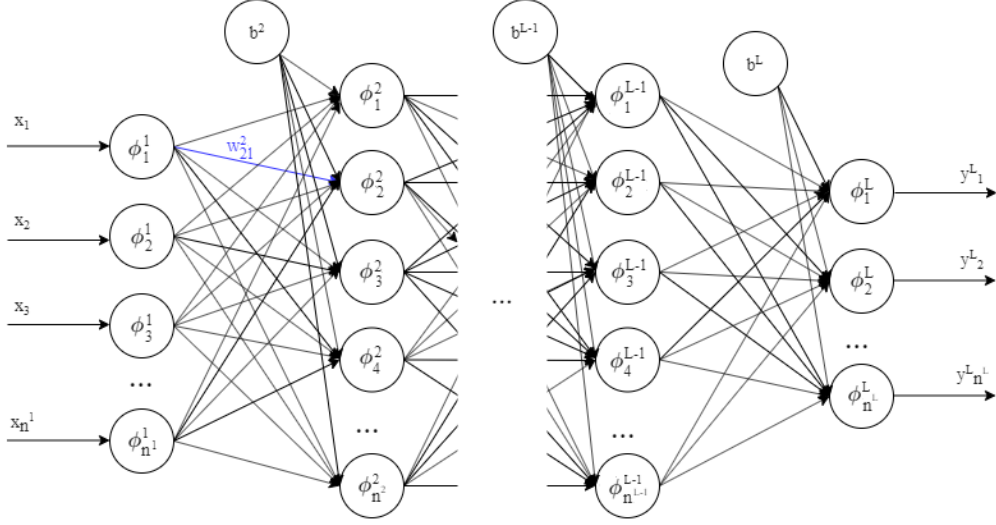
Figure 2: General multilayer perceptron structure with L layers. ϕ_j^i represent the activation function of the j-th neuron in the i-th layer, b^i represent the bias of the i-th layer, x_k represent the input variables and y_k represent the output variables.

Figure 2 can be treated as a general MLP model. A MLP can have L layers, and each layer l $(1 \leqslant l \leqslant L)$ has n^l $(n^l \geqslant 1)$ neurons. n^1 stands for the input layer and n^L for the output layer. For each layer l the input dimension is equal to the output dimension of layer $(l-1)$. For a neuron i $(1 \leqslant i \leqslant n^l)$ in layer l, its input vector, weight vector and output are $\mathbf{yb}^{l-1} = \left(b^l, y_1^{l-1}, \cdots, y_{n^{l-1}}^{l-1} \right)$, $\mathbf{w}_i^l = \left(w_{i0}^l, w_{i1}^l, \cdots, w_{in^{l-1}}^l \right)^\top$ and $y_i^l = \phi_i^l \left(z_i^l \right) = \phi_i^l \left(\mathbf{yb}^{l-1} \cdot \mathbf{w}_i^l \right)$ respectively, where $\phi_i^l : \mathbb{R} \to \mathbb{R}$ refers to the neuron activation function and $\cdot$ refers to the matrix multiplication operator. For each layer l, its input vector is $\mathbf{yb}^{l-1}$, its weight matrix is $\mathbf{W}^l = \left[\mathbf{w}_1^l \cdots \mathbf{w}_{n^l}^l \right]$ and its output vector is $\mathbf{y}^l = \left(y_i^l, \cdots, y_{n^l}^l \right) = \Phi^l \left(\mathbf{z}^l \right) = \Phi^l \left(\mathbf{yb}^{l-1} \cdot \mathbf{W}^l \right)$, where $\Phi^l : \mathbb{R}^{n^l} \to \mathbb{R}^{n^l}$ is a vector-valued function defined as $\Phi^l(\mathbf{z}) = (\phi_1^l(z_1), \cdots, \phi_{n^l}^l(z_{n^l}))$.

Weights in the neural structure determine how the information flows from the input layer to the output layer. Identifying the optimal weights that minimize the prediction error of a dataset is called training the neural network. There are different algorithms to identify these weights, being the most used the backpropagation algorithm described in Rumelhart *et al.* (1986). Explaining these training algorithms are out of the scope of this paper.

2.2. Partial derivatives

The sensitivity analysis performed by the **NeuralSens** package is based on the partial derivatives method. This method consists in calculating the derivative of the output with regard to the inputs of the neural network. These partial derivatives are called sensitivity, and are defined as:

$$s_{ik}\big|_{\mathbf{x}_n} = \frac{\partial y_k}{\partial x_i} (\mathbf{x}_n)$$

where $\mathbf{x}_n$ refers to the n sample of the dataset used to perform the sensitivity analysis and $s_{ik}\big|_{\mathbf{x}_n}$ refers to the sensitivity of the output of the k-th neuron in the output layer with regard to the input of the i-th neuron in the input layer evaluated in $\mathbf{x}_n$. We calculate these

sensitivities applying the chain rule to the partial derivatives of the inner layers (derivatives of Equation 1 for each neuron in the hidden layers). The partial derivatives of the inner layers are defined following the next equations:

- Derivative of z_k^l (input of the k-th neuron in the l-th layer) with regard to y_i^{l-1} (output of the i-th neuron in the $(l-1)$-th layer). This partial derivative corresponds to the weight of the connection between the k-th neuron in the l-th layer and the i-th neuron in the $(l-1)$-th layer:

$$\frac{\partial z_k^l}{\partial y_i^{l-1}} = w_{ki}^l \qquad (2)$$

- Derivative of y_k^l (output of the the k-th neuron in the l-th layer) with regard to z_i^l (input of the i-th neuron in the l-th layer):

$$\left.\frac{\partial y_k^l}{\partial z_i^l}\right|_{z_i^l} = \frac{\partial \phi_k^l}{\partial z_i^l}\left(z_i^l\right) \qquad (3)$$

where $\frac{\partial \phi_k^l}{\partial z_i^l}$ refers to the partial derivative of the activation function of the k-th neuron in the l-th layer with regard to the input of the k-th neuron in the l-th layer evaluated for the input z_i^l of the i-th neuron in the l-th layer.

Equations 2 and 3 have been implemented in the package in matrix form to reduce computational time following the next equations:

$$\frac{\partial \mathbf{z}_{[1\times n^l]}^l}{\partial \mathbf{y}_{[1\times n^{l-1}]}^{l-1}} = \begin{bmatrix} \frac{\partial z_1^l}{\partial y_1^{l-1}} & \frac{\partial z_2^l}{\partial y_1^{l-1}} & \cdots & \frac{\partial z_{n^l}^l}{\partial y_1^{l-1}} \\ \frac{\partial z_1^l}{\partial y_2^{l-1}} & \frac{\partial z_2^l}{\partial y_2^{l-1}} & \cdots & \frac{\partial z_{n^l}^l}{\partial y_2^{l-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_1^l}{\partial y_{n^{l-1}}^{l-1}} & \frac{\partial z_2^l}{\partial y_{n^{l-1}}^{l-1}} & \cdots & \frac{\partial z_{n^l}^l}{\partial y_{n^{l-1}}^{l-1}} \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{21}^l & \cdots & w_{n^l 1}^l \\ w_{12}^l & w_{22}^l & \cdots & w_{n^l 2}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{1n^{l-1}}^l & w_{2n^{l-1}}^l & \cdots & w_{n^l n^{l-1}}^l \end{bmatrix} = \mathbf{W}_{[n^{l-1}\times n^l]}^{*l}$$

$$\mathbf{J}_{l_{[n^l \times n^l]}}^l = \frac{\partial \mathbf{y}_{[1\times n^l]}^l}{\partial \mathbf{z}_{[1\times n^l]}^l} = \begin{bmatrix} \frac{\partial y_1^l}{\partial z_1^l} & \frac{\partial y_2^l}{\partial z_1^l} & \cdots & \frac{\partial y_{n^l}^l}{\partial z_1^l} \\ \frac{\partial y_1^l}{\partial z_2^l} & \frac{\partial y_2^l}{\partial z_2^l} & \cdots & \frac{\partial y_{n^l}^l}{\partial z_2^l} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1^l}{\partial z_{n^l}^l} & \frac{\partial y_2^l}{\partial z_{n^l}^l} & \cdots & \frac{\partial y_{n^l}^l}{\partial z_{n^l}^l} \end{bmatrix} = \begin{bmatrix} \frac{\partial \phi_1^l}{\partial z_1^l}\left(z_1^l\right) & \frac{\partial \phi_2^l}{\partial z_1^l}\left(z_1^l\right) & \cdots & \frac{\partial \phi_{n^l}^l}{\partial z_1^l}\left(z_1^l\right) \\ \frac{\partial \phi_1^l}{\partial z_2^l}\left(z_2^l\right) & \frac{\partial \phi_2^l}{\partial z_2^l}\left(z_2^l\right) & \cdots & \frac{\partial \phi_{n^l}^l}{\partial z_2^l}\left(z_2^l\right) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi_1^l}{\partial z_{n^l}^l}\left(z_{n^l}^l\right) & \frac{\partial \phi_2^l}{\partial z_{n^l}^l}\left(z_{n^l}^l\right) & \cdots & \frac{\partial \phi_{n^l}^l}{\partial z_{n^l}^l}\left(z_{n^l}^l\right) \end{bmatrix}$$

where $\mathbf{W}^{*l}$ is the reduced weight matrix of the l-th layer and $\mathbf{J}_l^l$ is the Jacobian matrix of the outputs in the l-th layer with respect to the inputs in the l-th layer.

Following the chain rule, the Jacobian matrix of the outputs in the l-th layer with regard to the inputs in the $(l-j)$-th layer can be calculated by:

$$\mathbf{J}_{l-j_{[n^l \times n^j]}}^l = \prod_{h=j}^{l-1}(\mathbf{J}_{h_{[n^h \times n^h]}}^h \cdot \mathbf{W}_{[n^h \times n^{h+1}]}^{*(h+1)}) \cdot \mathbf{J}_{l_{[n^l \times n^l]}}^l \qquad (4)$$

where $1 \leqslant k \leqslant (l-1)$ and $2 \leqslant l \leqslant L$. Using this equation with $l = L$ and $j = 1$, the partial derivatives of the outputs with regard to the inputs of the MLP are obtained.

2.3. Sensitivity measures

Once the sensitivity has been obtained for each variable and observation, different measures can be calculated to analyze the results. The authors propose the following sensitivity measures to summarize the information obtained by evaluating the sensitivity of the outputs for all the input samples X_n of the provided dataset:

- *Mean sensitivity* of the output of the k-th neuron in the output layer with regard to the i-th input variable:

$$S_{ik}^{avg} = \frac{\sum_{j=1}^{N} s_{ik}\big|_{\mathbf{x}_j}}{N} \tag{5}$$

  where N is the number of samples in the dataset.

- *Sensitivity standard deviation* of the output of the k-th neuron in the output layer with regard to the i-th input variable:

$$S_{ik}^{sd} = \sigma\left(s_{ik}\big|_{\mathbf{x}_j}\right); j \in 1, \dots, N \tag{6}$$

  where N is the number of samples in the dataset and σ refers to the standard deviation function.

- *Mean squared sensitivity* of the output of the k-th neuron in the output layer with regard to the i-th input variable (Yeh and Cheng 2010; Zurada *et al.* 1994):

$$S_{ik}^{sq} = \sqrt{\frac{\sum_{j=1}^{N} \left(s_{ik}\big|_{\mathbf{x}_j}\right)^2}{N}} \tag{7}$$

  where N is the number of samples in the dataset.

In case there are more than one output neuron, such as in a multi-class classification problem, these measures can be generalized to obtain sensitivity measures of the whole model as follows:

- *Mean sensitivity* with regard to the i-th input variable:

$$S_{i}^{avg} = \frac{\sum_{k=1}^{n^L} S_{ik}^{avg}}{n^L} \tag{8}$$

- *Sensitivity standard deviation* with regard to the i-th input variable:

$$S_{i}^{sd} = \sqrt{\frac{\sum_{k=1}^{n^L} \left(\left(S_{ik}^{sd}\right)^2 + \left(S_{ik}^{avg} - S_{i}^{avg}\right)^2\right)}{n^L}} \tag{9}$$

- *Mean squared sensitivity* with regard to the i-th input variable (Yeh and Cheng 2010):

$$S_{i}^{sq} = \left(\frac{\sum_{k=1}^{n^L} \sqrt{S_{ik}^{sq}}}{n^L}\right)^2 \tag{10}$$

Methods in **NeuralSens** to calculate the sensitivities of a neural network and the proposed sensitivities measures were written for several R packages that can be used to create MLP neural networks: class 'nn' from **neuralnet** package (Fritsch, Guenther, and Wright 2019), class 'nnet' from **nnet** package (Venables and Ripley 2002), class 'mlp' from **RSNNS** (Bergmeir and Benítez 2012), classes 'H2ORegressionModel' and 'H2OMultinomialModel' from **h2o** package (LeDell *et al.* 2022), 'list' from **neural** package (Nagy 2014) and classnnetar from **forecast** package (Hyndman and Khandakar 2008). The same methods are applied to neural network objects created with the `train()` function from the **caret** package (Kuhn 2008) only if these 'train' objects inherit from the available packages the "class" attribute. Methods have not been included in **NeuralSens** for other packages that can create MLP neural networks, although further developments of **NeuralSens** could include additional methods. An additional method for class 'numeric' is available to use with the basic information of the model (weights, structure and activation functions of the neurons). Examples on how to use this 'numeric' method can be found in Appendix A.

3. Package structure

The functionalities of the package **NeuralSens** is based on the new R class 'SensMLP' defined inside the package itself. **NeuralSens** includes four main functions based on this class to perform the sensitivity analysis of a MLP model described in the previous section:

- `SensAnalysisMLP()`: S3 method to perform the sensitivity analysis using partial derivatives of the outputs with regard to the inputs of the MLP model. This function returns a 'SensMLP' object with the results of the sensitivity analysis.

- `SensitivityPlots()`: Graphically represent the sensitivity measures of a 'SensMLP' object.

- `SensFeaturePlot()`: Graphically represent the relation between the sensitivities of a 'SensMLP' object and the value of the input variables.

- `SensTimePlot()`: Graphically represent the evolution among time of the sensitivities of a 'SensMLP' object.

Each of these functions are detailed in the rest of this section. The output of the last three functions are plots created with **ggplot2** package functions (Wickham 2016).

3.1. The R class 'SensMLP'

The **NeuralSens** package defines an S3 object called 'SensMLP' as a list with the following components:

- `sens`: 'list' of 'data.frames', one per neuron in the output layer, with the S_{ik}^{avg}, S_{ik}^{sd} and S_{ik}^{sq} sensitivity measures described in Section 2.3 (Equations 5, 6 and 7). Each row of the `data.frame` contains the sensitivity measures with regard to a specific input.

- `raw_sens`: 'list' of 'matrixes', one per neuron in the output layer, with the sensitivities calculated following Equation 4 with $l = L$ and $j = 1$. Each column of each 'matrix'

contains the sensitivities of the output with regard to a specific input and each row contains the sensitivities with regard to all the inputs corresponding to the same row of the `trData` component.

- `mlp_struct`: 'numeric' 'vector' indicating the number of neurons in each layer of the MLP model.

- `trData`: Typically a 'data.frame' which contains the dataset used to calculate the sensitivities.

- `coefnames`: 'character' 'vector' with the names of the input variables of the MLP model.

- `output_name`: 'character' 'vector' with the names of the output variables of the MLP model.

Functions described in Sections 3.3 (`SensitivityPlots()`), 3.4 (`SensTimePlot()`) and 3.5 (`SensFeaturePlot()`) can be accessed through the plot method of the 'SensMLP' object. `print()` and `summary()` methods are also available for obtaining information on the sensitivities and sensitivity measures of the 'SensMLP' object. Examples of these methods are presented in the remaining sections.

3.2. MLP sensitivity analysis

The `SensAnalysisMLP()` function calculates the partial derivatives of a MLP model. This function consists of an S3 method (Chambers and Hastie 1992) to extract the basic information of the model (weights, structure and activation functions) based on the model `class` attribute and to pass this information to the default method. This default method calculates the sensitivities of the model as described in Section 2, and creates a `SensMLP` object with the result of the sensitivity analysis. `SensAnalysisMLP()` function performs all operations using matrix calculus to reduce the computational time.

In the current version of **NeuralSens** (version 1.0.0), the accepted activation functions are shown in Table 1. To calculate the sensitivities, the function assumes that all the neurons in a defined layer has the same activation function.

In order to show how `SensAnalysisMLP()` is used, we use a simulated dataset to train an MLP model of class 'nn' (**RSNNS**). The dataset consists of a 'data.frame' with 1500 rows of observations and four columns for three input variables (`X1`, `X2`, `X3`) and one output variable (`Y`). The input variables are random observations of a normal distribution with zero mean and standard deviation equal to 1. The output Y is created following Equation 11 based on X_1 and X_2:

$$Y = (X_1)^2 - 0.5 \cdot X_2 + 0.1 \cdot \varepsilon \tag{11}$$

where ε is random noise generated using a normal distribution with zero mean and standard deviation equal to 1. X_3 is given to the model for training and a proper fitted model would find no relation between X_3 and Y.

`?NeuralSens::simdata` can be executed to obtain more information about the data. The library is loaded by executing the following code:

```
R> library("NeuralSens")
```

Name	Function	Derivative
sigmoid	$f(z) = \frac{1}{1+e^{-z}}$	$\frac{\partial f}{\partial z}(z) = \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right)$
tanh	$f(z) = \tanh(z)$	$\frac{\partial f}{\partial z}(z) = 1 - \tanh(z)$
linear	$f(z) = z$	$f'(z) = 1$
ReLU	$f(z) = \begin{cases} 0 & \text{when } z \leqslant 0 \\ z & \text{when } z > 0 \end{cases}$	$\frac{\partial f}{\partial z}(z) = \begin{cases} 0 & \text{when } z \leqslant 0 \\ 1 & \text{when } z > 0 \end{cases}$
arctan	$f(z) = \arctan(z)$	$\frac{\partial f}{\partial z}(z) = \frac{1}{1+e^z}$
softplus	$f(z) = \ln(1 + e^z)$	$\frac{\partial f}{\partial z}(z) = \frac{1}{1+e^{-z}}$
softmax	$f_i(\mathbf{z}) = \frac{e^{z_i}}{\sum\limits_{k}(e^{z_k})}$	$\frac{\partial f_i}{\partial z_j}(\mathbf{z}) = \begin{cases} f_i(\mathbf{z}) \cdot (1 - f_j(\mathbf{z})) & \text{when } i = j \\ -f_j(\mathbf{z}) \cdot f_i(\mathbf{z}) & \text{when } i \neq j \end{cases}$

Table 1: Accepted activation functions and their derivatives in `SensAnalysisMLP()`, where z refers to the input value of the neuron as described in Equation 1 and $\mathbf{z}$ refers to the vector of input values of the neuron layer.

To test the functionality of the `SensAnalysisMLP()` function, `mlp()` function from **RSNNS** package trains a neural network model using the `simdata` dataset.

```
R> library("RSNNS")
R> set.seed(150)
R> mod1 <- mlp(simdata[, c("X1", "X2", "X3")], simdata[, "Y"], maxit = 1000,
+    size = 10, linOut = TRUE)
```

`SensAnalysisMLP()` is used to perform a sensitivity analysis to `mod1` using the same dataset as in training:

```
R> sens <- SensAnalysisMLP(mod1, trData = simdata, output_name = "Y",
+    plot = FALSE)
```

`sens` is a 'SensMLP' object and methods of that class can be used to explore the sensitivity analysis:

```
R> class(sens)

[1] "SensMLP"

R> summary(sens)

Sensitivity analysis of 3-10-1 MLP network.

Sensitivity measures of each output:
$Y
```

```
            mean        std meanSensSQ
X1 -0.005406908 1.94524276 1.94476390
X2 -0.485564931 0.06734504 0.49021056
X3 -0.003200699 0.02971083 0.02987535
```

`summary()` method prints the sensitivity measures of the output with regard to the inputs of the model. These measures are calculated using the sensitivities displayed when using the `print()` method described below. The `mean` column (S_{ik}^{avg}) shows the mean effect of the input variable on the output. The `std` column (S_{ik}^{sd}) shows the variance of the input variable's effect on the output along the input space. These columns provide information on the relation between inputs and output variables:

- If both `mean` (S_{ik}^{avg}) and `std` (S_{ik}^{sd}) are near zero, it indicates that the output is not related to the input, because for all the training data the sensitivity of the output with regard to that input is approximately zero.

- If `mean` (S_{ik}^{avg}) is different from zero and `std` (S_{ik}^{sd}) is near zero, it indicates that the output has a linear relationship with the input, because for all the training data the sensitivity of the output with regard to that input is approximately constant.

- If `std` (S_{ik}^{sd}) is different from zero, regardless of the value of `mean` (S_{ik}^{avg}), it indicates that the output has a non-linear relationship with the input, because the relation between the output and the input vary depending on the value of the input.

Setting an upper bound for `std` to be considered close to zero so that the relationship between output and input can be considered as linear is a non-trivial endeavor. The authors are working on a statistic to test whether the functional relationship between an input and an output variable can be considered linear and, if successful, it will be included in a future version of the package.

In the example, the `mean` and `std` values show:

- X_1 has mean ≈ 0 and standard deviation ≈ 2. This means it has a non-constant, i.e., non-linear effect on the response variable.

- X_2 has mean ≈ 0.5 and standard deviation ≈ 0. This means it has a constant, i.e., linear effect on the response variable.

- X_3 has mean ≈ 0 and standard deviation ≈ 0. This means it has no effect on the response variable.

An input variable may be considered significant if their sensitivities $s_{ik}\big|_{\mathbf{x}_j}$ are significantly different from zero, whether they are positive or negative. In other words, a variable is considered to be significant when changes in the input variable produce significant changes in the output variable of the model. White and Racine (2001) conclude that the statistic $\left(S_{ik}^{sq}\right)^2 = \frac{\sum_{j=1}^{N}\left(s_{ik}\big|_{\mathbf{x}_j}\right)^2}{N}$ is a valid indicator to identify if a variable is irrelevant. Moreover, S_{ik}^{sq} is a measure of the changes in the output due to local changes in the input. Thus, S_{ik}^{sq} can be defined as a measure of the importance of the input variables from a perturbation analysis

point of view, in the sense that small changes in that input will produce larger changes in the output.

'SensMLP' class has also a `print()` method to show the sensitivities of the output with regard to the inputs evaluated in each of the rows of the `trData` component of the `sens` object. A second argument `n` may be used to specify how many rows to display (by default `n = 5`).

```
R> print(sens, n = 2)


Sensitivity analysis of 3-10-1 MLP network.

  2000 samples

Sensitivities of each output (only 2 first samples):
$Y
             X1          X2           X3
[1,]   2.08642384 -0.4462707 -0.044063158
[2,]  -0.34976334 -0.3547381  0.014188363
```

3.3. Visualizing neural network sensitivity measures

Sensitivity measures of the output variables are useful for quantitative analysis. However, it can be difficult to compare sensitivity metrics when a large number of input variables are used. In order to visualize information on the calculated sensitivities, the authors propose the following plots:

1. Label plot representing the relationship between S_{ik}^{avg} (x-axis) and S_k^{std} (y-axis).

2. Bar plot that shows S_k^{sq} for each input variable.

3. Density plot that shows the distribution of output sensitivities with regard to each input (Muñoz and Czernichow 1998):

   - The narrow distribution of sensitivity values for `X2` (corresponding to a constant sensitivity) indicates a linear relationship between this input and the output of the neural net.

   - The wide distribution of sensitivity values for `X1` (corresponding to a variable sensitivity) indicates a non-linear relationship between this input and the output.

   When the height of at least one of the distributions is greater than 10 times the height of the smallest distribution, then an extra plot is created using the `facet_zoom()` function of the **ggforce** package (Pedersen 2021). These plots provides a better representation of the sensitivity distributions.

These plots can be obtained using the `SensitivityPlots()` function and a 'SensMLP' object calculated using `SensAnalysisMLP()`. To obtain the plots of Figure 3:

```
R> SensitivityPlots(sens)
```
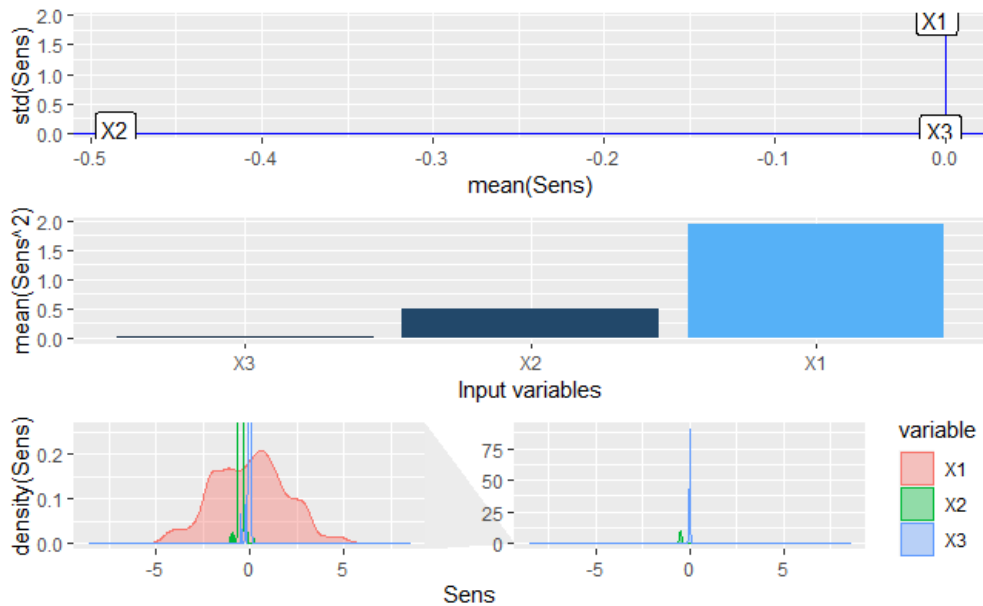
Figure 3: Example from the `SensitivityPlots()` function showing plots specified in Section 3.3. First plot shows the relation between the mean and standard deviation of the sensitivities, the second plot shows the square of the sensitivities and the third and fourth plots show the distribution of the sensitivities.

Or they can be generated using the `plot()` method of the 'SensMLP' object:

`R> plot(sens)`

In this case, the first plot of Figure 3 shows that `Y` has a negative linear relationship with `X2` ($std \approx 0$ and $mean < 0$), no relationship with `X3` ($std \approx 0$ and $mean \approx 0$) and a non-linear relationship with `X1` (std different from 0). The second plot shows that `X3` barely affects the response variable, being `X1` and `X2` the inputs with most effect on the output.

3.4. Visualizing neural network sensitivity over time

A common application of neural networks is time series forecasting. Analyzing how sensitivities evolve over time can provide a better understanding of the effect of explanatory variables on the output variables.

`SensTimePlot()` returns a sequence plot of the raw sensitivities calculated by the function `SensAnalysisMLP()`. The x-axis is related to a `numeric` or `Posixct`/`Posixlt` variable containing the time information of each sample. The y-axis is related to the sensitivities of the output with regard to each input.

In order to show how this function can be used, the `DAILY_DEMAND_TR` dataset is used to create a model of class 'train' from **caret** package (Kuhn 2008). This dataset is similar to the `elecdaily` dataset from **fpp2** R package (Hyndman 2020). However, `DAILY_DEMAND_TR` contains almost five years of daily data (`elecdaily` only one year), which makes it more suitable for training a neural network. It is composed of the following variables:

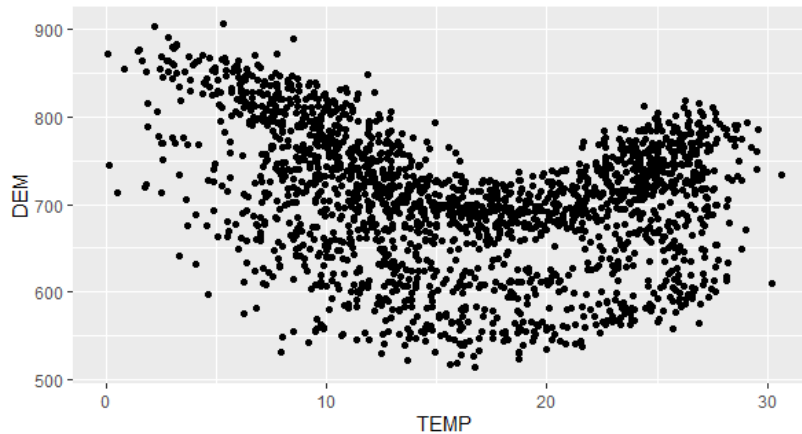- `DATE`: Date of the sample, one per day from 2007-07-02 to 2012-11-30.

Figure 4: Relation between the output variable, `DEM`, and the input `TEMP` of `DAILY_DEMAND_TR` database.

- `TEMP`: Mean daily temperature in °C in Madrid, Spain.

- `WD`: Working day, continuous parameter which represents the effect on the daily consumption of electricity as a percentage of the expected electricity demand of that day with regard to the demand of the reference day of the same week Moral-Carcedo and Vicéns-Otero (2005). In this case, Wednesday is the reference day ($\mathrm{WD}_{Wed} \approx 1$).

- `DEM`: Total daily electricity demand in GWh for Madrid, Spain.

The following code creates the plot in Figure 4:

```
R> library("ggplot2")
R> ggplot(DAILY_DEMAND_TR) + geom_point(aes(x = TEMP, y = DEM))
```

Figure 4 shows the relationship between the electricity demand and the temperature. A non-linear effect can be observed, where the demand increases for low temperatures (due to heating systems) and for high temperatures (due to air conditioners).

The following code scales the data, create a `train` neural network model and apply the `SensTimePlot()` function to two years of the data:

```
R> DAILY_DEMAND_TR[, 4] <- DAILY_DEMAND_TR[, 4]/10
R> DAILY_DEMAND_TR[, 2] <- DAILY_DEMAND_TR[, 2]/100
R> library("caret")
R> set.seed(150)
R> mod2 <- train(form = DEM~TEMP + WD, data = DAILY_DEMAND_TR,
+    method = "nnet", linout = TRUE, maxit = 250, metric = "RMSE",
+    tuneGrid = data.frame(size = 5, decay = 0.1),
+    preProcess = c("center", "scale"), trControl = trainControl())
R> SensTimePlot(mod2, DAILY_DEMAND_TR[1:(365*2), ], output_name = "DEM",
+    date.var = DAILY_DEMAND_TR[1:(365*2), 1], facet = TRUE)
```
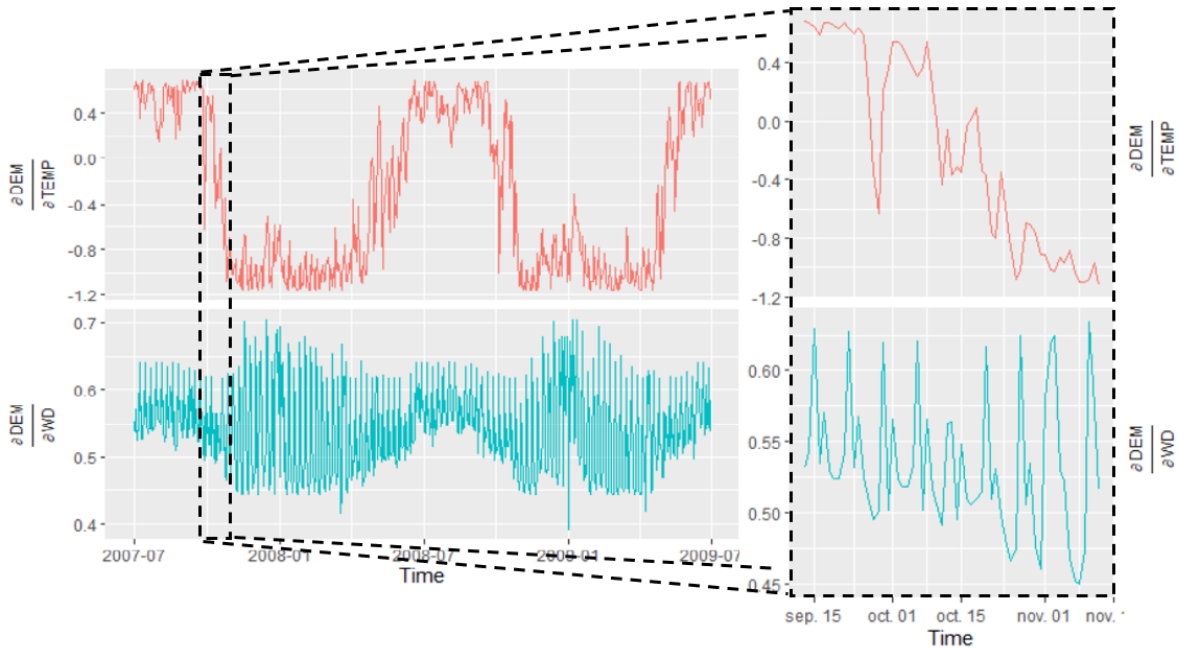
Figure 5: Example from the `SensTimePlot()` function, showing how the sensitivities for each of the inputs evolve over time.

Figure 5 shows that the temperature variable has a seasonal effect on the response variable. In summer, the temperature is higher and cooling systems demand more electricity, therefore the demand is directly proportional to the temperature. In winter, the temperature is lower and heating systems demand more electricity, hence the demand is inversely proportional to the temperature. The sensitivity of the output with regard to `WD` has also a seasonal effect, with higher variance in winter than in summer and greater sensitivity in weekends. Figure 5 can also be generated using the `plot()` method of a 'SensMLP' object:

```
R> sens2 <- SensAnalysisMLP(mod2, trData = DAILY_DEMAND_TR[1:(365*2),],
+    output_name = "DEM", plot = FALSE)
R> plot(sens2, plotType = "time", facet = TRUE,
+    date.var = DAILY_DEMAND_TR[1:(365*2), 1])
```

3.5. Visualizing the sensitivity relation as a function of the input values

Sometimes it is useful to know how the value of the input variables affects the sensitivity of the response variables. The `SensFeaturePlot()` function produces a violin plot to show the probability density of the output sensitivities with regard to each input. It also plots a jitter strip chart for each input, where the width of the jitter is controlled by the density distribution of the data (Pedersen 2021). The color of the points is proportional to the value of each input variable, which display whether the relation of the output with the input is relatively constant within a range of input values.

The following code produce the plot of Figure 6:

```
R> SensFeaturePlot(mod2, fdata = DAILY_DEMAND_TR[1:(365*2),])
```
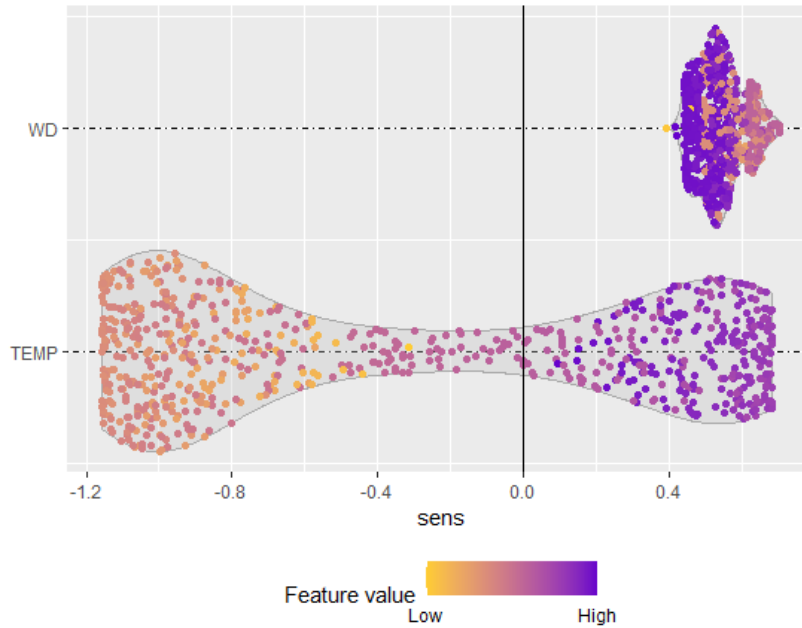
Figure 6: Example from the `SensFeaturePlot()` function, showing the relationship of the sensitivities with the value of the inputs.

It can also be generated using the `plot()` method of a 'SensMLP' object:

```
R> plot(sens2, plotType = "features")
```

In accordance with the information extracted from Figure 5, Figure 6 shows that the sensitivity of the output with regard to the temperature is negative when the temperature is low and positive when the temperature is high. It also shows that the sensitivity of the output with regard to `WD` is higher in the weekends (lower values of `WD`).

3.6. Extending package functionalities to other MLP models

The current version of **NeuralSens** package (version 1.0.0), includes methods of `SensAnalysisMLP()` function for 'nn' (**neuralnet**), 'nnet' (**nnet**), 'H2ORegressionModel' and 'H2OMultinomialModel' (**h2o**), 'mlp' (**RSNNS**), 'list' (**neural**), 'nnetar' (**forecast**) and 'train' (**caret**) (only if the object inherits the class attribute from another of the available packages). Additionally, a 'numeric' method is available to perform sensitivity analysis of a new neural network model using only the weights of the model, its neural structure, and the activation function of the layers and their derivatives. The first information that must be extracted from the model are the weights of the connections between layers. These weights must be passed to the first argument of the `SensAnalysisMLP()` function as a 'numeric' 'vector', concatenating the weights of the layers in order from the first hidden layer ($l = 2$) to the output layer ($l = L$). The bias weight should be added to the vector before the weights of the same layer, following the equation below:

$$wts = [b^2, w_{11}^2, w_{21}^2, \ldots, w_{n^2 n^1}^2, b^3, w_{11}^3, \ldots, b^L, w_{11}^L, \ldots, w_{n^L n^{L-1}}^L] \tag{12}$$

If the model has no bias, the bias weights must be set to 0 ($b^l = 0$).

The second information is the neural structure of the model. The structure of the model must be passed to the `mlpstr` argument as a 'numeric' 'vector' equal in length to the number of layers in the network. Each number specifies the number of neurons in each layer, starting with the input layer and ending with the output layer:

The last information that must be provided are the activation functions of each layer and their derivatives. If the activation function of a layer is one of those provided by the package (shown in Table 1), the function can be specified using its name. If the activation function is not one of those provided in the package, it should be passed as a function. The same applies to the derivative of the activation function. The activation function $\Phi^l(\mathbf{z}^l)$ of a layer l and its derivative $\frac{\partial(\Phi^l(\mathbf{z}^l))}{\partial(\mathbf{z}^l)}$ must meet the following conditions:

- $\Phi^l(\mathbf{z}^l)$ must return a 'vector' with the same length as $\mathbf{z}^l$. The activation function of each neuron may be different, as long as this condition is met:

$$\Phi^l(\mathbf{z}^l) = \begin{bmatrix} \phi_1^l(z_1^l) \\ \phi_2^l(z_2^l) \\ \vdots \\ \phi_{n^l}^l(z_{n^l}^l) \end{bmatrix}$$

- $\frac{\partial(\Phi^l(\mathbf{z}^l))}{\partial(\mathbf{z}^l)}$ must return a square 'matrix' with the derivative of $\Phi^l(\mathbf{z}^l)$ with regard to each component of $\mathbf{z}^l$:

$$\frac{\partial(\Phi^l(\mathbf{z}^l))}{\partial(\mathbf{z}^l)} = \begin{bmatrix} \frac{\partial \phi_1^l}{\partial z_1^l}\left(z_1^l\right) & \frac{\partial \phi_2^l}{\partial z_1^l}\left(z_1^l\right) & \cdots & \frac{\partial \phi_{n^l}^l}{\partial z_1^l}\left(z_1^l\right) \\ \frac{\partial \phi_1^l}{\partial z_2^l}\left(z_2^l\right) & \frac{\partial \phi_2^l}{\partial z_2^l}\left(z_2^l\right) & \cdots & \frac{\partial \phi_{n^l}^l}{\partial z_2^l}\left(z_2^l\right) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi_1^l}{\partial z_{n^l}^l}\left(z_{n^l}^l\right) & \frac{\partial \phi_2^l}{\partial z_{n^l}^l}\left(z_{n^l}^l\right) & \cdots & \frac{\partial \phi_{n^l}^l}{\partial z_{n^l}^l}\left(z_{n^l}^l\right) \end{bmatrix}$$

Examples of how to use the `SensAnalysisMLP()` function with new packages can be found in Appendix A.

3.7. Effect of network structure and training conditions

An important advantage of sensitivity analysis based on partial derivatives is the robustness of the analysis results regardless of the model's neural structure. Other methods such as Olden rely heavily on the neural structure and the initial starting weights. A similar analysis to the one performed on `olden()` in Beck (2018) has been performed on the `SensAnalysisMLP()` function. To observe the effect of the neural structure on the sensitivity metrics, these metrics have been calculated for models with 1, 10 and 20 neurons in the hidden layer. For each neuron level, 50 models with different random initial weights are trained. If the neural structure and the initial starting weights have no effect on the sensitivity metrics, these metrics should be the same for all the models. `simdata` dataset is used to train the models.

Figure 7 shows the mean value of the sensitivity metrics from the 50 models for each neural structure. It also shows the minimum and maximum value of the metric to display the effect
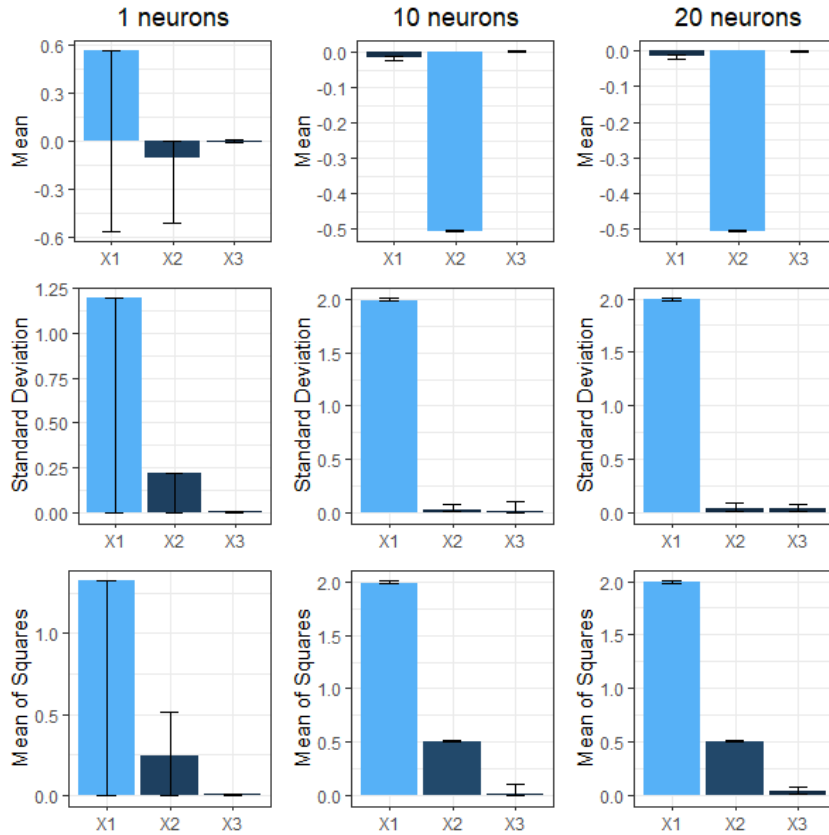
Figure 7: Robustness analysis of sensitivity metrics for three neural network models with different number of neurons in the hidden layer. Fifty different models for each neuron level are trained with random initial weights. `geom_bar` shows the mean value of the sensitivity metrics and `geom_errorbar` shows the minimum and maximum value of the sensitivity metrics.

of the neural structure and the initial weights values. An important conclusion that can be derived from Figure 7 is that with enough neurons in the hidden layer, i.e., if the model can predict the output with enough precision; variance of sensitivity metrics is negligible compared to the value of the metric.

4. Further examples and comparison with other methods

This section contains several examples in which the functions of the **NeuralSens** package are compared with similar functions from other R packages. Section 4.1 trains an MLP for classification to compare `SensAnalysisMLP()` with `olden()`, `garson()` (Beck 2018) and `plot_explanations()` (Pedersen and Benesty 2021). Section 4.2 trains an MLP for regression to compare `SensAnalysisMLP()` and `SensFeaturePlot()` with `lekprofile()` (Beck 2018) and `partial()` (Greenwell 2017).

Topics such as data pre-processing or network architecture should be considered before model development. Discussions about these ideas have been already held (Cannas, Fanni, See, and Sias 2006; Amasyali and El-Gohary 2018; Maier and Dandy 2000; Lek *et al.* 1996) and are beyond the scope of this paper.

(a) Sepal-related variables.
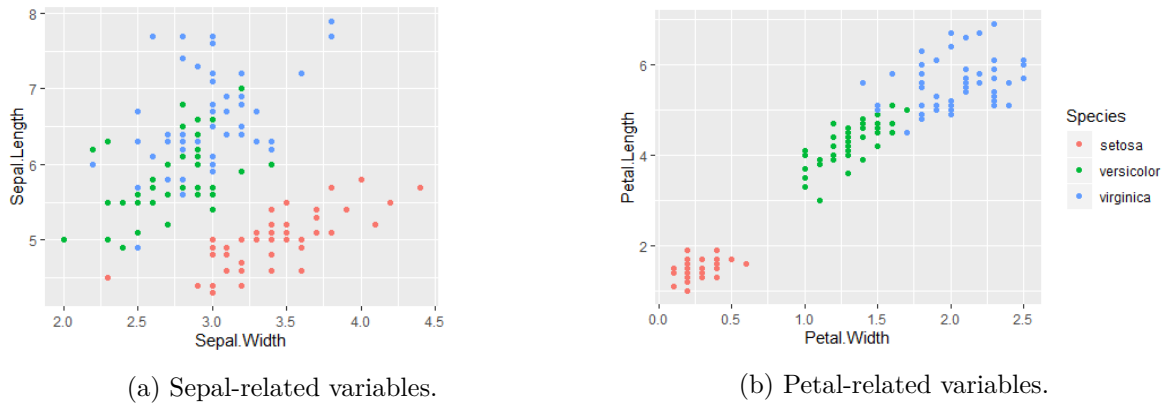

(b) Petal-related variables.

Figure 8: (a) `geom_point` plot representing the variables `Sepal.Length` and `Sepal.Width`, (b) `geom_point` plot representing the variables `Petal.Length` and `Petal.Width`.

4.1. Multilayer perceptron for classification

In this example a multilayer perceptron is trained using the well-known `iris` dataset included in R. Figure 8 shows two scatterplots comparing the petal-related and sepal-related variables of the flowers in the dataset. It can be seen that `setosa` species have a smaller petal size than the other two species and a shorter sepal. It also shows that `virginica` and `versicolor` species have a similar sepal size, but the latter has a slightly smaller petal size.

The `train()` function from the **caret** package creates a new neural network model to predict the species of each flowers based on petal and sepal dimensions.

```
R> set.seed(150)
R> mod3 <- caret::train(Species~., data = iris, preProcess = c("center",
+    "scale"), method = "nnet", linout = TRUE, trControl = trainControl(),
+    tuneGrid = data.frame(size = 5, decay = 0.1), metric = "Accuracy")
```

`SensAnalysisMLP()` function calculates the sensitivities of the model, providing information of the relationships between each output class and each input variable.

```
R> sens4 <- SensAnalysisMLP(mod3)
R> summary(sens4)


Sensitivity analysis of 4-5-3 MLP network.


Sensitivity measures of each output:
$setosa
                  mean         std meanSensSQ
Sepal.Length -0.01346027 0.02698245 0.03007287
Sepal.Width   0.03116669 0.05055589 0.05924712
Petal.Length -0.07133925 0.10172196 0.12396638
Petal.Width  -0.06846395 0.09654030 0.11808983


$versicolor
```

```
                   mean        std meanSensSQ
Sepal.Length  0.035833076 0.02252307 0.04228375
Sepal.Width  -0.002101449 0.03925762 0.03918294
Petal.Length -0.099221391 0.17568963 0.20126091
Petal.Width  -0.093949654 0.16300239 0.18766775


$virginica
                   mean        std meanSensSQ
Sepal.Length -0.01803139 0.02742334 0.03274380
Sepal.Width  -0.04199856 0.05663125 0.07035338
Petal.Length  0.20543515 0.28413548 0.34985476
Petal.Width   0.19731097 0.27236735 0.33559057
```

The sensitivity metrics for each of the output provides information on how the neural network uses the data to predict the output:

- The `setosa` class has a greater probability when `Petal.Length`, `Petal.Width` and `Sepal.Length` variables decrease, or the `Sepal.Width` variable increases.

- The `versicolor` class has a greater probability when `Petal.Length` and `Petal.Width` variables decrease, or the `Sepal.Length` variable increases.

- The `virginica` class has a greater probability when the `Petal.Length` and `Petal.Width` variables increase, and `Sepal.Length` and `Sepal.Width` variables decrease.

This information corresponds to what is observed in Figure 8, where `setosa` class is characterized by a low value of `Petal.Length`, `Petal.Width` and `Sepal.Length` variables, and `versicolor` and `virginica` classes are differentiated by the value of the `Petal.Length` and `Petal.Width` variables.

`garson()` and `olden()` method from the **NeuralNetTools** package (Beck 2018) provide information on input importance. As they provide information related to the first output neuron, the comparison with `SensAnalysisMLP()` is done using the sensitivity measures for the first output class.

```
R> SensitivityPlots(sens, der = FALSE, output = "setosa")
R> garson(mod3)
R> olden(mod3)
```

Figure 9a shows the sensitivity metrics calculated by `SensAnalysisMLP()`, Figure 9b shows `garson()`'s importance metrics for the input variables and Figure 9c shows `olden()`'s importance metrics for the input variables. The mean value of the sensitivities in the top chart of Figure 9a is similar to `olden()`'s metrics observed in 9c, and the S_{ik}^{sq} values in the barplot of Figure 9a are similar to `garson()`'s values observed in Figure 9b. It must be noted that values from `SensAnalysisMLP()` are more robust against changes in the neural structure and initial weights as stated in Section 3.7 and Beck (2018).

The **lime** (Pedersen and Benesty 2021) package can also be used to obtain information on the neural network model. In this case, `lime()` and `explain()` functions train a decision

(a) Model 3, `SensitivityPlots()`.

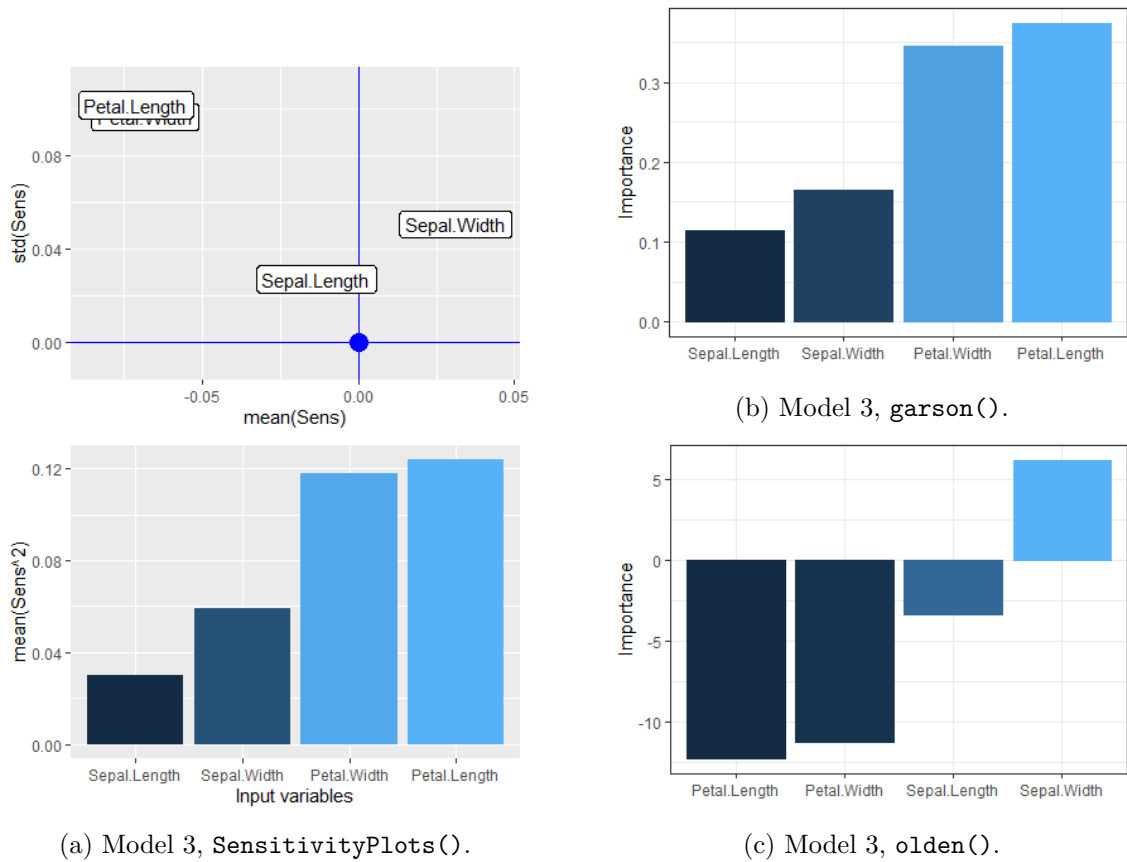(b) Model 3, `garson()`.

(c) Model 3, `olden()`.

Figure 9: (a) Global sensitivity measures plots of `mod3` for the `iris` dataset using `SensitivityPlots()`, (b) variable importance using `garson()` from **NeuralNetTools**, (c) variable importance using `olden()` from **NeuralNetTools**.

tree model using the entire dataset to interpret how the neural network predicts the class of three different samples (one for each iris species) using all the features in the training dataset. `plot_explanations()` function shows graphically the information given by the decision tree.

```
R> library("lime")
R> plot_explanations(explain(iris[c(1, 51, 101),], lime(iris, mod3),
+    n_labels = 3, n_features = 4))
```

Figure 10 confirms the relationships between the inputs and the output variable. However, this method does not provide a quantitative measure for the importance of each input. Due to the lack of quantitative measures for input importance this method can not be directly compared to the other methods exposed in this section (Figure 9).

Sometimes it may be more interesting to obtain global importance measures instead of measures for each output variable. **NeuralSens** allows us to obtain global measures using the `CombineSens()` function. It computes the sensitivity measures of the whole model following Equations 8, 9 and 10. These global measures are an indicator of how much, on average, the output probabilities change when an input variable changes.
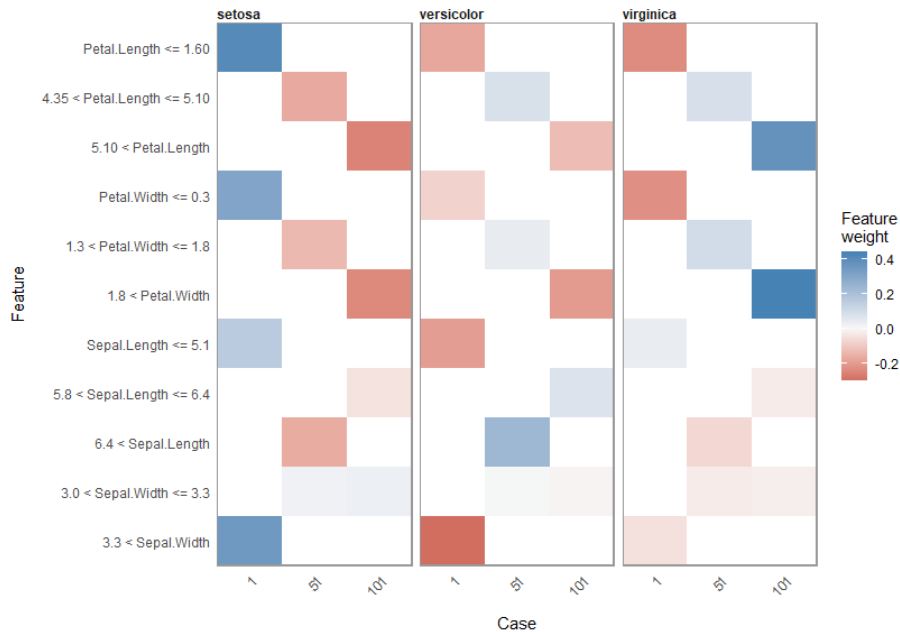
Figure 10: Faceted heatmap-style plots generated by applying the `plot_explanations()` function to `mod3` for three selected samples of `iris` dataset. Each plot represents the contribution (positive or negative) of each feature to the probability of a specific class of `iris[, "Species"]` for the specified data samples.

```
R> summary(CombineSens(sens))


Sensitivity analysis of 4-5-3 MLP network.


Sensitivity measures of each output:
$Combined
                    mean          std meanSensSQ
Sepal.Length  0.001447138 0.03545616 0.03484419
Sepal.Width  -0.004311108 0.05770054 0.05547536
Petal.Length  0.011624836 0.24404886 0.21535630
Petal.Width   0.011632454 0.23246026 0.20434926
```

4.2. Multilayer perceptron for regression

The `Boston` dataset from the **MASS** (Ripley 2022) package is used to train an **nnet** (Venables and Ripley 2002) model. This dataset contains information collected by the U.S. Census Service on housing in the suburbs of Boston (run `?MASS::Boston` to obtain more information about the dataset).

The objective of the model is to predict the nitric oxides concentration (parts per 10 million), stored in the `nox` variable. The input variables of the model are `zn` (proportion of residential land zoned for lots over 25,000 sq.ft.), `rad` (index of accessibility to radial highways) and

lstat (lower status of the population (percent)). `scale()` function standardizes the input variables. `SensFeaturePlot()`, `SensAnalysisMLP()`, `lekprofile()` (**NeuralNetTools**, Beck 2018) and `pdp()` (**pdp**, Greenwell 2017) functions analyze the relationships of the output with regard to the inputs.

```
R> data("Boston", package = "MASS")
R> Boston <- as.data.frame(scale(Boston[, c("zn", "rad", "lstat", "nox")]))
R> set.seed(150)
R> mod4 <- nnet::nnet(nox ~ ., data = Boston, size = 15, decay = 0.1,
+    maxit = 150)
R> library("NeuralNetTools")
R> lekprofile(mod4, group_vals = 6)
R> lekprofile(mod4, group_vals = 6, group_show = TRUE)
R> library("pdp")
R> pdps <- list()
R> for (i in 1:3) {
+    pdps[[i]] <- autoplot(partial(mod4, pred.var = names(Boston)[i],
+       train = Boston, ice = TRUE), train = Boston,
+       center = TRUE, alpha = 0.2, rug = TRUE) +
+    theme_bw() + ylab("nox")
+  }
R> gridExtra::grid.arrange(grobs = pdps, nrow = 1)
R> SensFeaturePlot(mod4, fdata = Boston, output_name = "nox")
R> SensAnalysisMLP(mod4, trData = Boston, output_name = "nox")
```

Figures 11a and 11b display the results of Lek's profile method from the **NeuralNetTools** package. To prevent the analysis of non-representative scenarios in the input dataset, a k-means clustering with 6 clusters has been applied to the dataset. Each subplot in Figure 11a shows the evolution of the output variable when varying the input variable of interest across a range of values corresponding to the center of the k-means clusters. The other inputs remain constant in their values in the center of each k-means cluster. Figure 11b shows the value of the input variables at each of the cluster centers.

Figure 11c shows the PDP and ICE plots of `mod4` output with regard to each input variable. An ICE curve is calculated by maintaining the input variable of interest x_i at a value x_{ik}, where x_{ik} is the value of x_i in the k row of the dataset, and varying all other inputs across their values in the dataset. For a given dataset with N samples, there would be N ICE curves for each input variable. The PDP curve can be calculated as the mean of these ICE curves. PDP shows the marginal effect a given input variable has on the response variable of the neural network model, averaging the effects of the rest of the input variables.

Calculating all ICE curves shows how the output variable change in the entire input space of the dataset. This comes at a large computational cost, since the number of curves that must be calculated are directly proportional to the number of samples and number of input variables. The computational time can be reduced by calculating only the PDP or a reduced number of ICE curves, but in that case some important scenarios might be ignored.
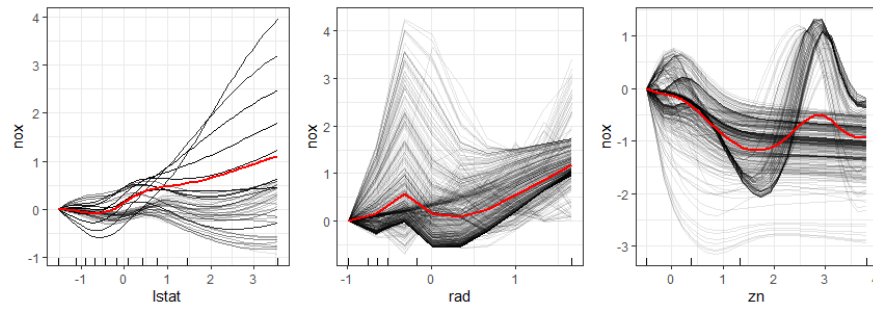
`SensFeaturePlot()` function performs an analysis similar to Lek's and **pdp** by plotting the sensitivity of the output with regard to the input colored proportionally to the value of the input variable. On the one hand, `lekprofile()` function indicates that lstat and rad have
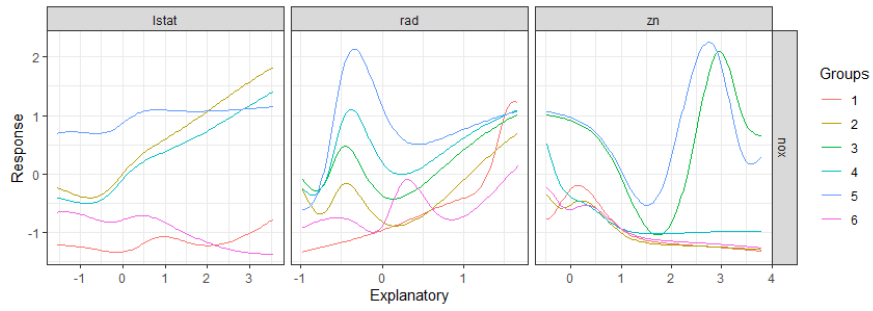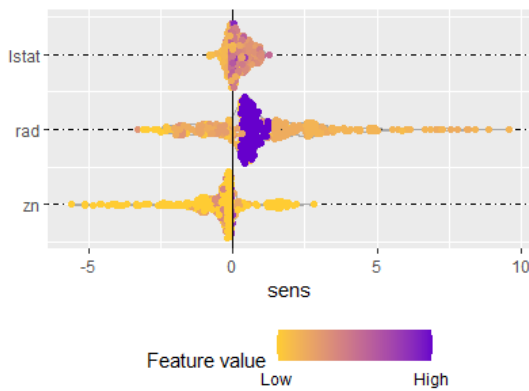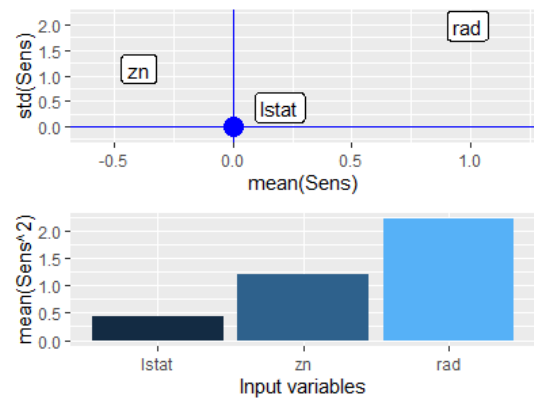
(a) Model 4, `lekprofile()`.
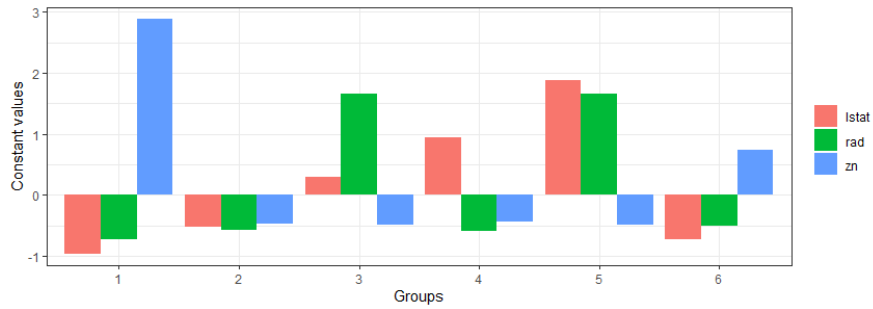
(b) Cluster grouping of `lekprofile()`.

(c) Model 4, `pdp` and `ICE` plots.

(d) Model 4, `SensFeaturePlot()`.

(e) Model 4, `SensAnalysisMLP()` plots.

Figure 11: (a) Sensitivity analysis of a neural network using `lekprofile()` from **Neural-NetTools**. (b) Values at which explanatory variables are held constant for each cluster in `lekprofile()`. (c) Partial dependence plots (red) and individual conditional expectation plots (black) of `mod4`. (d) `SensFeaturePlot()` applied to `mod4`. (e) `SensAnalysisMLP()` applied to `mod4`.

a direct relationship with the output and `zn` has an inverse relationship with the output. Figure 11a also suggests that all the input variables have a non-linear relationship with the output. On the other hand, Figure 11d shows that the sensitivity of the output has an approximately linear relationship with `zn`, and a non-linear relationship with the other two inputs. In this case, the `lekprofile()` function gives more information about the model, but it can be difficult to understand how the value of the input variables in each group affects the output variable.

`SensAnalysisMLP()` can be used to obtain more information on variable importance and relationships between variables. In this case, the `rad` variable affects the output the most, information which can be difficult to extract from the other functions.
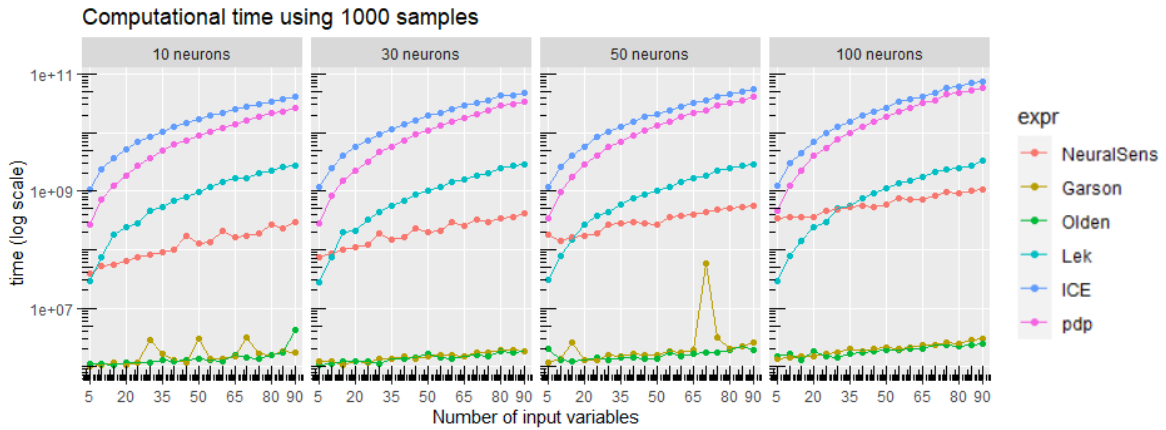
4.3. Computational cost

As the size of neural network models increase exponentially to solve more complex tasks, performing a sensitivity analysis of a model could become an intensive computational task. Sensitivity analysis using partial derivatives requires matrix calculus where the size of the matrices is directly proportional to the size of the hidden layers. As the number of neurons in hidden layers increase, the time to perform these calculations grows rapidly.

A comparison of how much time is required by a sensitivity analysis using the different methods included in Section 4 has been performed. This comparison has been carried out using the `YearPredictionMSD` dataset (Dua and Karra Taniskidou 2017). This dataset consists of 90 input variables and 515345 samples. The authors propose to measure the computational time when varying the number of input variables, the number of samples and the size of the hidden layer of a single hidden layer MLP model.

This analysis has been performed on a computer with the following specs: processor Intel(R) Core(TM) i7-8700 @3.20 GHz, 32 GB of RAM memory, R version 3.6.3 (2020-02-29), platform x86_64-w64-mingw32/x64 (64-bit) and running under Windows 10 x64 (build 18362).

Figure 12 shows the computational time of each function varying the number of training samples, the number of input variables and the number of neurons in the hidden layer. Some conclusions can be reached from this figure:

- The fastest functions are the `olden()` and `garson()` functions from the **NeuralNetTools** package, because they only perform a sum of the weight matrices of the model. As it does not depend of the number of samples, the computational time is directly proportional to the size of the neural network layers (input and hidden layers).

- `lekprofile()` from **NeuralNetTools** and `SensAnalysisMLP()` from **NeuralSens** need of the similar computational times.h `SensAnalysisMLP()` is affected by the number of neurons in the hidden layer and the number of samples, as the size of the matrices increases with the number of neurons and the number of matrices multiplication increase with the number of samples. `lekprofile()` is more affected by the number of input variables, because the number of curves to be created increase with the number of input variables. However, due to the fact that it uses only a fixed number of scenarios it does not depend on the number of samples. Since the number of neurons in the hidden layer barely affects the computational time to predict the output variable in each scenario, this parameter does not affect the computational time of lekprofile.

(a) Computational time using 1000 training samples.
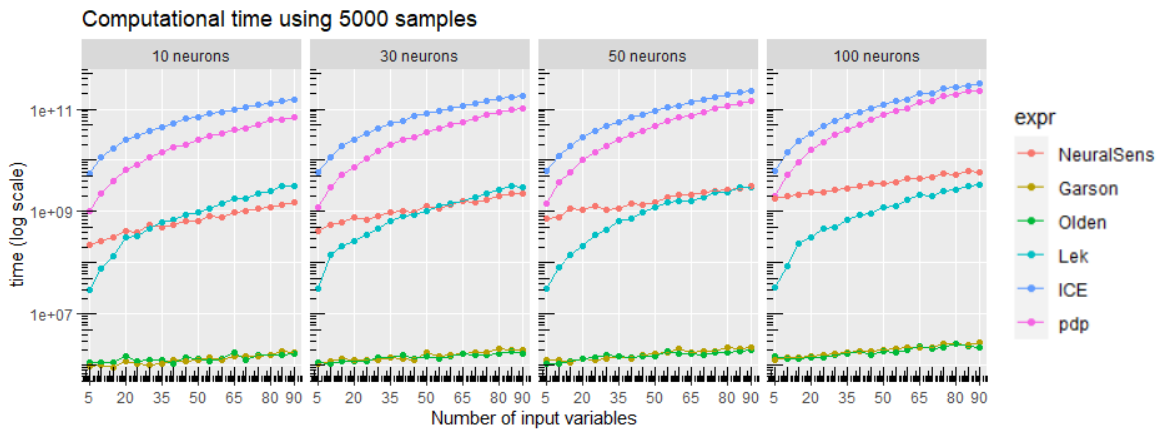
(b) Computational time using 5000 training samples.
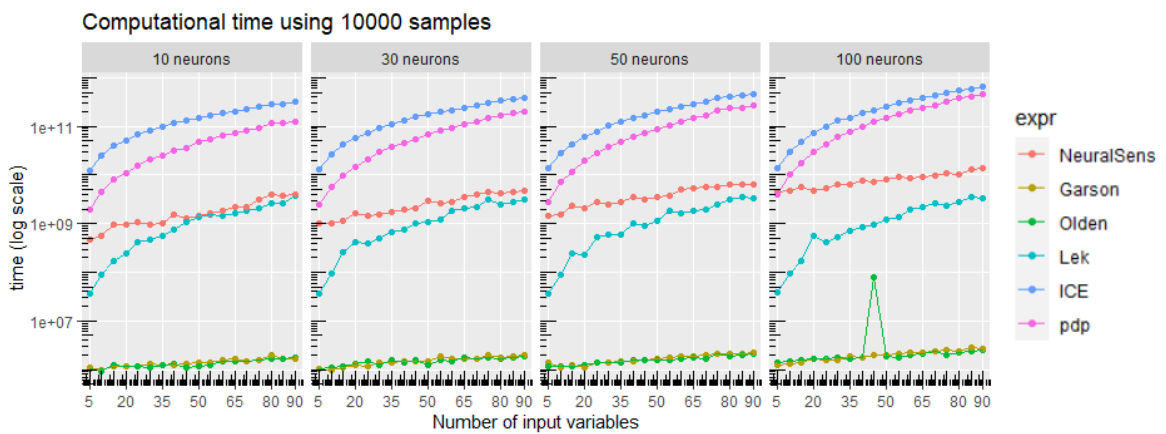
(c) Computational time using 10000 training samples.

Figure 12: Computational time of the different sensitivity analysis methods with different number of training samples (1000, 5000 and 10000 training samples), number of input variables (from 5 to 90 input variables) and number of neurons in the hidden layer (10, 30, 50 and 100 neurons).

- The slowest function is the `partial()` function from the **pdp** package when calculating all ICE curves. Calculating all ICE curves instead of only the PDP curves adds a noticeable amount of computational time. However, if the form of the ICE curves is not constant throughout the samples of the dataset (as in Figure 11c), showing only the PDP curve gives misleading information as the form of the PDP curve does not resemble all the ICE curves. The computational time for `partial()` is directly proportional to the number of samples and the number of input variables, because the number of curves to be calculated increase exponentially as these parameters increase.

In addition to these conclusions, it must be mentioned that this analysis has been performed using a model with only one output variable. If there were several output variables, to obtain analogous information as `SensAnalysisMLP()` the other functions must be called once for each output. Because of this, the computational time of all the functions except `SensAnalysisMLP()` must be multiplied by the number of output variables of the model in order to obtain an approximate idea of the computational time they require.

5. Conclusions

The **NeuralSens** package provides functions to extract information from a fitted feed-forward MLP neural network in R. These functions can be used to obtain the partial derivatives of the neural network response variables with regard to the input variables and to generate plots to obtain different information on the network using these partial derivatives. Methods are available for the following CRAN packages: '`nn`' (**neuralnet**), '`nnet`' (**nnet**), '`mlp`' (**RSNNS**), '`H2ORegressionModel`' and '`H2OMultinomialModel`' (**h2o**), '`list`' (**neural**), '`nnetar`' (**forecast**) and '`train`' (**caret**) (only if the object inherits the class attribute from another package). An additional method for class '`numeric`' is available to use with the basic information of the model (weights, structure and activation functions of the neurons).

The main objective of the package is to help the user to understand how the neural network uses the inputs to predict the output. This information may be useful for simplifying the neural network structure by eliminating the inputs which have no effect on the output. It could also provide a deeper understanding on the problem and the relationship among variables. **NeuralSens** is another tool among several other methods for exploratory data analysis and model evaluation, and it can be used with other packages (Beck 2018; Greenwell 2017) to obtain more information on the neural network model. Nevertheless, it must be noted that sensitivity analysis using partial derivatives provides information about variable relationships such as PDP or ICE plots significantly faster. Moreover, it also provides variable importance measures like Garson's or Olden's methods and these measures are independent of the neural structure and training conditions of the model as long as it predicts the output with enough precision.

Improving the information given by these methods will have value for exploratory data-analysis and characterization of relationships among variables. Future versions of the package may include additional functionalities as:

- Parallelizing the sensitivity calculations when workers registered to work in parallel are detected.

- Calculating the sensitivities of the output variables with regard to the output of hidden

neurons, in order to obtain the importance of each hidden neuron and helping to select the optimal network structure.

- Calculating the sensitivities of other neural network models such as probabilistic radial basis function network (PRBFN) or Recurrent Neural Network (RNN).

- Calculating the second order partial derivatives of an MLP model to analyze the effect of interactions between two input variables.

- Develop a statistic to determine if the relationship between the output and input is linear.

References

Allaire JJ, Chollet F (2021). **keras**: *R Interface to 'Keras'*. R package version 2.7.0, URL https://CRAN.R-project.org/package=keras.

Allaire JJ, Tang Y (2021). **tensorflow**: *R Interface to 'TensorFlow'*. R package version 2.7.0, URL https://CRAN.R-project.org/package=tensorflow.

Amasyali K, El-Gohary NM (2018). "A Review of Data-Driven Building Energy Consumption Prediction Studies." *Renewable and Sustainable Energy Reviews*, **81**, 1192–1205. doi:10.1016/j.rser.2017.04.095.

Beck MW (2018). "**NeuralNetTools** : Visualization and Analysis Tools for Neural Networks." *Journal of Statistical Software*, **85**(11). doi:10.18637/jss.v085.i11.

Bergmeir C, Benítez JM (2012). "Neural Networks in R Using the Stuttgart Neural Network Simulator: **RSNNS**." *Journal of Statistical Software*, **46**(7), 1–26. doi:10.18637/jss.v046.i07.

Butler KT, Davies DW, Cartwright H, Isayev O, Walsh A (2018). "Machine Learning for Molecular and Materials Science." *Nature*, **559**(7715), 547–555. doi:10.1038/s41586-018-0337-2.

Cannas B, Fanni A, See L, Sias G (2006). "Data Preprocessing for River Flow Forecasting Using Neural Networks: Wavelet Transforms and Data Partitioning." *Elsevier*, **31**, 1164–1171. doi:10.1016/j.pce.2006.03.020.

Cannon AJ (2017). **monmlp**: *Multi-Layer Perceptron Neural Network with Optional Monotonicity Constraints*. R package version 1.1.5, URL https://CRAN.R-project.org/package=monmlp.

Chambers JM, Hastie TJ (1992). "Classes and Methods: Object-Oriented Programming in S." In *Statistical Models in S*. Wadsworth & Brooks/Cole.

Cybenko G (1989). "Approximation by Superpositions of a Sigmoidal Function." *Mathematics of Control, Signals and Systems*, **2**(4), 303–314. doi:10.1007/bf02551274.

Dimopoulos I, Bourret P, Lek S (1995). "Use of Some Sensitivity Criteria for Choosing Networks with Good Generalization Ability." *Neural Processing Letters*, **2**, 1–4. `doi: 10.1007/bf02309007`.

Dimopoulos I, Chronopoulos J, Chronopoulou-Sereli A, Lek S (1999). "Neural Network Models to Study Relationships Between Lead Concentration in Grasses and Permanent Urban Descriptors in Athens City (Greece)." *Ecological Modelling*, **120**(2-3), 157–165. `doi: 10.1016/s0304-3800(99)00099-x`.

Dua D, Karra Taniskidou E (2017). "UCI Machine Learning Repository." URL `http://archive.ics.uci.edu/ml/`.

Engelbrecht AP, Cloete I, Zurada JM (1995). "Determining the Significance of Input Parameters Using Sensitivity Analysis." In G Goos, J Hartmanis, J Leeuwen, J Mira, F Sandoval (eds.), *From Natural to Artificial Neural Computation*, volume 930, pp. 382–388. Springer-Verlag, Berlin, Heidelberg. `doi:10.1007/3-540-59497-3_199`.

Friedman JH (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, **29**(5), 1189–1232. `doi:10.1214/aos/1013203451`.

Fritsch S, Guenther F, Wright MN (2019). **neuralnet**: *Training of Neural Networks*. R package version 1.44.2, URL `https://CRAN.R-project.org/package=neuralnet`.

Garson GD (1991). "Interpreting Neural-Network Connection Weights." *AI 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 Networks Models." *Ecological Modelling*, **160**, 249–264. `doi:10.1016/s0304-3800(02)00257-0`.

Goldstein A, Kapelner A, Bleich J, Pitkin E (2015). "Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation." *Journal of Computational and Graphical Statistics*, **24**(1), 44–65. `doi:10.1080/10618600.2014.907095`.

Greenwell BM (2017). "**pdp**: An R Package for Constructing Partial Dependence Plots." *The R Journal*, **9**(1), 421–436. `doi:10.32614/rj-2017-016`.

Hastie T, Tibshirani R, Friedman J (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer-Verlag, New York. `doi:10.1007/978-0-387-21606-5`.

Hornik K (1991). "Approximation Capabilities of Multilayer Feedforward Networks." *Neural Networks*, **4**(2), 251–257. `doi:10.1016/0893-6080(91)90009-t`.

Hornik K, Stinchcombe M, White H (1989). "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks*, **2**(5), 359–366. `doi:10.1016/0893-6080(89)90020-8`.

Hyndman R (2020). **fpp2**: *Data for "Forecasting: Principles and Practice" (2nd Edition)*. R package version 2.4, URL `https://CRAN.R-project.org/package=fpp2`.

Hyndman RJ, Khandakar Y (2008). "Automatic Time Series Forecasting: The **forecast** Package for R." *Journal of Statistical Software*, **26**(3), 1–22. `doi:10.18637/jss.v027.i03`.

Kuhn M (2008). *Building Predictive Models in R Using the **caret** Package.* `doi:10.18637/jss.v028.i05`.

LeDell E, Gill N, Aiello S, Fu A, Candel A, Click C, Kraljevic T, Nykodym T, Aboyoun P, Kurka M, Malohlava M (2022). ***h2o**: R Interface for 'H2O'.* R package version 3.36.0.4, URL `https://CRAN.R-project.org/package=h2o`.

Lek S, Delacoste M, Baran P, Dimopoulos I, Lauga J, Aulagnier S (1996). "Application of Neural Networks to Modeling Nonlinear Relationships in Ecology." *Ecological Modelling*, **90**, 39–52. `doi:10.1016/0304-3800(95)00142-5`.

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. `doi:10.1016/s1364-8152(99)00007-9`.

McCulloch WS, Pitts W (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity." *The Bulletin of Mathematical Biophysics*, **5**(4), 115–133. `doi:10.1007/bf02478259`.

Moral-Carcedo J, Vicéns-Otero J (2005). "Modelling the Non-Linear Response of Spanish Electricity Demand to Temperature Variations." *Energy Economics*, **27**(3), 477–494. `doi:10.1016/j.eneco.2005.01.003`.

Mosavi A, Salimi M, Faizollahzadeh Ardabili S, Rabczuk T, Shamshirband S, Varkonyi-Koczy AR (2019). "State of the Art of Machine Learning Models in Energy Systems, a Systematic Review." *Energies*, **12**(7), 1301. `doi:10.3390/en12071301`.

Muñoz A, Czernichow T (1998). "Variable Selection Using Feedforward and Recurrent Neural Networks." *Engineering Intelligent Systems for Electrical Engineering and Communications*, **6**(2), 91–102.

Nagy A (2014). ***neural**: Neural Networks.* R package version 1.4.2.2, URL `https://CRAN.R-project.org/package=neural`.

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. `doi:10.1016/j.ecolmodel.2004.03.013`.

Özesmi SL, Özesmi U (1999). "An Artificial Neural Network Approach to Spatial Habitat Modelling with Interspecific Interaction." *Ecological Modelling*, **116**(1), 15–31. `doi:10.1016/s0304-3800(98)00149-5`.

Pedersen TL (2021). ***ggforce**: Accelerating **ggplot2**.* R package version 0.3.3, URL `https://CRAN.R-project.org/package=ggforce`.

Pedersen TL, Benesty M (2021). ***lime**: Local Interpretable Model-Agnostic Explanations.* R package version 0.5.2, URL `https://CRAN.R-project.org/package=lime`.

Philip Chen CL, Zhang CY (2014). "Data-Intensive Applications, Challenges, Techniques and Technologies: A Survey on Big Data." *Information Sciences*, **275**, 314–347. `doi:10.1016/j.ins.2014.01.015`.

Portela J, Muñoz A, Pizarroso J (2022). **NeuralSens**: *Sensitivity Analysis of Neural Networks*. R package version 1.0.0, URL `https://CRAN.R-project.org/package=NeuralSens`.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. URL `https://www.R-project.org/`.

Ribeiro MT, Singh S, Guestrin C (2016). ""Why Should I Trust You?": Explaining the Predictions of Any Classifier." In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 1135–1144. ACM, New York. `doi:10.1145/2939672.2939778`.

Ripley BD (2022). **MASS**: *Support Functions and Datasets for Venables and Ripley's MASS*. R package version 7.3-55, URL `https://CRAN.R-project.org/package=MASS`.

Rojas R (1996). "Fast Learning Algorithms." In *Neural Networks*, pp. 183–225. Springer-Verlag, Berlin, Heidelberg. `doi:10.1007/978-3-642-61068-4_8`.

Rumelhart DE, Hinton GE, Williams RJ (1986). "Learning Representations by Back-Propagating Errors." *Nature*, **323**, 533–536. `doi:10.1038/323533a0`.

Scardi M, Harding LW (1999). "Developing an Empirical Model of Phytoplankton Primary Production: A Neural Network Case Study." *Ecological Modelling*, **120**(2-3), 213–223. `doi:10.1016/s0304-3800(99)00103-9`.

Smalley E (2017). "AI-Powered Drug Discovery Captures Pharma Interest." *Nature Biotechnology*, **35**(7), 604–605. `doi:10.1038/nbt0717-604`.

Sun Z, Sun L, Strang KD (2018). "Big Data Analytics Services for Enhancing Business Intelligence." *Journal of Computer Information Systems*, **58**(2), 162–169. `doi:10.1080/08874417.2016.1220239`.

Valduriez P, Mattoso M, Akbarinia R, Borges H, Camata J, Coutinho A, Gaspar D, Lemus N, Liu J, Lustosa H, Masseglia F, Nogueira da Silva F, Silva V, Souza R, Ocaña K, Ogasawara E, de Oliveira D, Pacitti E, Porto F, Shasha D (2018). "Scientific Data Analysis Using Data-Intensive Scalable Computing: The SciDISC Project." In *LADaS: Latin America Data Science Workshop*. URL `https://hal-lirmm.ccsd.cnrs.fr/lirmm-01867804`.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York.

Vu MAT, Adalı T, Ba D, Buzsáki G, Carlson D, Heller K, Liston C, Rudin C, Sohal VS, Widge AS, Mayberg HS, Sapiro G, Dzirasa K (2018). "A Shared Vision for Machine Learning in Neuroscience." *The Journal of Neuroscience*, **38**(7), 1601–1607. `doi:10.1523/jneurosci.0508-17.2018`.

White H, Racine J (2001). "Statistical Inference, the Bootstrap, and Neural-Network Modeling with Application to Foreign Exchange Rates." *IEEE Transactions on Neural Networks*, **12**(4), 657–673. `doi:10.1109/72.935080`.

Wickham H (2016). **ggplot2**: *Elegant Graphics for Data Analysis*. 2nd edition. Springer-Verlag, New York.

Xu D, Tian Y (2015). "A Comprehensive Survey of Clustering Algorithms." *Annals of Data Science*, **2**. doi:10.1007/s40745-015-0040-1.

Yeh IC, Cheng WL (2010). "First and Second Order Sensitivity Analysis of MLP." *Neuro-computing*, **73**(10-12), 2225–2233. doi:10.1016/j.neucom.2010.01.011.

Zhang Z, Beck MW, Winkler DA, Huang B, Sibanda W, Goyal H (2018). "Opening the Black Box of Neural Networks: Methods for Interpreting Neural Network Models in Clinical Applications." *Annals of Translational Medicine*, **6**(11). doi:10.21037/atm.2018.05.32.

Zurada JM, Malinowski A, Cloete I (1994). "Sensitivity Analysis for Minimization of Input Data Dimension for Feedforward Neural Network." In *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94*, volume 6, pp. 447–450. IEEE, London. doi:10.1109/iscas.1994.409622.

A. Examples of `SensAnalysisMLP()` 'numeric' method

The R packages **monmlp** (Cannon 2017) and **tensorflow** (Allaire and Tang 2021) is used in this section to illustrate how the 'numeric' method of `SensAnalysisMLP()` function can be applied to new models. The `simdata` dataset described in Section 3.2 is used to train the MLP models in this appendix.

```
R> library("monmlp")
R> set.seed(150)
R> monmlp_model <- monmlp.fit(x = data.matrix(simdata[, 1:3]),
+    y = data.matrix(simdata[, 4]), hidden1 = 5, iter.max = 250,
+    silent = TRUE)
```

The weights of the model are extracted and ordered as described in Equation 12:

```
R> W1 <- rbind(monmlp_model[[1]]$W1[4,], monmlp_model[[1]]$W1[1:3,])
R> W2 <- c(monmlp_model[[1]]$W2[6,], monmlp_model[[1]]$W2[1:5,])
R> wts <- c(as.vector(W1), as.vector(W2))
```

The activation functions of the hidden and output layers and their derivatives are provided in the 'Th', 'To', 'Th.prime' and 'To.prime' attribute of the model respectively. However, the derivative functions does not meet the condition of returning a square 'matrix' so it must be modified before using `SensAnalysisMLP()`:

```
R> Actfunc <- c("linear", attr(monmlp_model, "Th"),
+    attr(monmlp_model, "To"))
R> Deractfunc <- c("linear",
+    function(v) {diag(attr(monmlp_model, "Th.prime")(v))},
+    function(v) {diag(attr(monmlp_model, "To.prime")(v))})
```

The last information that must be passed to `SensAnalysisMLP()` is the neural structure:

```
R> mlpstruct <- c(3, 5, 1)
```

Since `monmlp.fit()` automatically scales the variables when training the model, the input variables must be scaled before calculating the sensitivities.

```
R> x <- data.matrix(simdata[, 1:3])
R> x.center <- attr(monmlp_model, "x.center")
R> x.scale <- attr(monmlp_model, "x.scale")
R> x <- sweep(x, 2, x.center, "-")
R> x <- sweep(x, 2, x.scale, "/")
R> x <- cbind(data.frame(x), Y = simdata[, 4])
```

Once all the information has been prepared, the 'numeric' method of `SensAnalysisMLP()` can be used to perform a sensitivity analysis of the model:

```
R> sens_monmlp <- SensAnalysisMLP(wts, trData = x, mlpstr = mlpstruct,
+    coefnames = c("X1", "X2", "X3"), output_name = "Y",
+    actfunc = Actfunc, deractfunc = Deractfunc, plot = FALSE)
R> summary(sens_monmlp)
```

```
Sensitivity analysis of 3-5-1 MLP network.

Sensitivity measures of each output:
$Y
              mean         std  meanSensSQ
X1 -1.741261e-02 1.972087796 1.971671603
X2 -4.828116e-01 0.010075281 0.482916667
X3 -7.807139e-05 0.005014291 0.005013646
```

`summary()` method shows the same relationships between input variables X1, X2 and X3 and output variable Y as in Sections 3.2 and 3.3.

Sensitivity analysis of a **tensorflow** MLP model can be performed extracting analogous information as in the previous example. As the popularity of this package is growing rapidly, a specific guide on how to extract the information seems necessary.

The following code is used to load the **tensorflow** and **keras** (Allaire and Chollet 2021) libraries and to train a MLP model with two hidden layers:

```
R> library("tensorflow")
R> library("keras")
R> keras_model <- keras_model_sequential() %>%
+    layer_dense(units = 16, activation = "relu", input_shape = 3) %>%
+    layer_dense(units = 8, activation = "relu") %>%
+    layer_dense(units = 1) %>%
+    compile(loss = "mse", optimizer = optimizer_rmsprop(),
+        metrics = list("mean_absolute_error"))
R> history <- keras_model %>% fit(data.matrix(simdata[, 1:3]),
+    array(simdata[, 4]), epochs = 500, verbose = 0)
```

Now that the model is trained, the weights and neural structure of the model can be obtained using the `get_weights()` function:

```
R> model_weights <- get_weights(keras_model)
R> wts <- c()
R> neural_struct <- c(nrow(model_weights[[1]]))
R> for (i in seq(2, length(model_weights), 2)) {
+    neural_struct <- c(neural_struct, dim(model_weights[[i]]))
+    lyr_wgts <- rbind(model_weights[[i]], model_weights[[i - 1]])
+    wts <- c(wts, unname(do.call(c, as.data.frame(lyr_wgts))))
+  }
```

Since all the activation functions are already implemented in **NeuralSens**, they can be defined as a 'character' 'vector'.

```
R> actfunc <- c("linear", "ReLU", "ReLU", "linear")
```

The `SensAnalysisMLP()` function can already be used with all the information obtained.

```
R> sens_keras <- SensAnalysisMLP(wts, trData = simdata,
+    mlpstr = neural_struct, coefnames = names(simdata)[1:3],
+    output_name = names(simdata)[4], actfunc = actfunc, plot = FALSE)
R> summary(sens_keras)

Sensitivity analysis of 3-16-8-1 MLP network.

Sensitivity measures of each output:
$Y
            mean        std meanSensSQ
X1 -0.000817390 1.96281649 1.96232590
X2 -0.509612746 0.05305515 0.51236568
X3 -0.003731768 0.03125266 0.03146691
```

Again, the `summary()` method shows the same relationships between input variables X1, X2 and X3 and output variable Y as in Sections 3.2 and 3.3.

Affiliation:

Jaime Pizarroso, Antonio Muñoz
Instituto de Investigación Tecnológica (IIT)
Escuela Técnica Superior de Ingeniería ICAI
Universidad Pontificia Comillas
Calle de Alberto Aguilera, 23
28015 Madrid, Spain
E-mail: Jaime.Pizarroso@iit.comillas.edu, Antonio.Munoz@iit.comillas.edu

José Portela
Instituto de Investigación Tecnológica (IIT)
Escuela Técnica Superior de Ingeniería ICAI *and*
Facultad de Ciencias Económicas y Empresariales ICADE,
Universidad Pontificia Comillas
E-mail: jose.portela@iit.comillas.edu