A Detailed Guide to the 'nnet' Package [R]

Spyros Orfanos

Concordia University, Summer 2019

---

1. Introduction

The nnet package is an R package used to train single layer feed-forward neural networks. The purpose of this paper is to provide further background information, insight, and documentation to the workings and usage of the 'neuralnet' package. An example is provided to demonstrate how to use various functions from this package. The results from this example are then validated with a feed-forward neural network that was trained using first principles.

2. 'nnet' Function

2.1. Training Algorithm

When training a neural network, the goal is to optimize the weights such that the loss function is minimized. This is typically done via gradient descent which involves moving towards a local minimum by taking steps in the direction opposite to the loss function's gradient. While gradient descent only considers the gradient (first-order derivatives) of the loss function to move towards a local minimum, Newton's Method considers its Hessian (second-order derivatives) in order to accelerate convergence to a local minimum. Newton's Method approximates the cost function with its second-order Taylor series expansion and yields the update rule $x_{k+1} = x_k - t_k \nabla^2 f(x_k) - \nabla f(x_k)$.

The 'nnet' function uses the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, which is a quasi-Newton numerical optimization method since it approximates the Hessian instead of calculating it directly[1]. The approximation of the Hessian, M_t, is then used to determine the direction of descent, ρ_t, where $\rho_t = M_t \cdot \nabla Loss(x_t)$. Finally, the update to the parameters of the neural network is given by $\theta_{t+1} = \theta_t + \epsilon^* \cdot \rho_t$, where ϵ^* is the step size as determined by a line search[2].

2.2. Inputs

Here is a list of inputs of the nnet function:

- *formula*: the regression formula, e.g.: y ∼ x1 + x2. Note: y can have more than one column (eg: y = cbind(y1, y2)).
- *data*: a data frame containing the variables in the above formula.
- *size*: number of units in the hidden layer. Can be 0 if skip layer.

- *weights*: (case) weights for each example – if missing defaults to 1. This controls the weight of each sample, so enter vector of length n.
- *Wts*: Can set the initial weights by entering a vector having the form:

$$c(w^1_{1,0}, w^1_{2,0}, ..., w^1_{size,0},$$
$$w^1_{1,1}, w^1_{2,1}, ..., w^1_{size,1},$$
$$..., ..., ...,$$
$$w^1_{1,p}, w^1_{2,p}, ..., w^1_{size,p},$$
$$w^2_{0,1}, w^2_{1,1}, ..., w^2_{size,1})$$

- *rang*: Initial random weights on [-rang, rang]. Value about 0.5 unless the inputs are large, in which case it should be chosen so that rang * max(—x—) is about 1.
- *linout*: T/F switch for linear output units. Default logistic output units.
- *entropy*: T/F switch for entropy (= maximum conditional likelihood) fitting. Default by least-squares.
- *linout*: T/F switch for linear output units. Default logistic output units.
- *softmax*: T/F switch for softmax (log-linear model) and maximum conditional likelihood fitting. Note: linout, entropy, softmax and censored are mutually exclusive.
- *decay*: parameter for importance of weight regularization term. Defualt is 0. $CostFunction = SSE + decay * W^T W$
- *abstol*: Stop if the fit criterion falls below abstol, indicating an essentially perfect fit.
- *reltol*: Stop if the optimizer is unable to reduce the fit criterion by a factor of at least 1 - reltol.
- *subset*: An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
- *maxit*: maximum number of iterations. Default is 100.
- *mask*: logical vector indicating which parameters should be optimized (default all).
- *na.action*: Extract information on the NA action used to create an object.

2.3. Outputs

The following results can be accessed directly from an object of type nnet, say myNN.

- $myNN\$n$: returns the structure of the neural network.
- $myNN\$nunits$: returns the number of units (includes input units and biases) .
- $myNN\$conn$: number of units in the hidden layer. Can be 0 if skip layer.
- $myNN\$decay$: decay term from input.
- $myNN\$entropy$: T/F depending on input.
- $myNN\$softmax$: T/F depending on input.
- $myNN\$censored$: T/F depending on input.
- $myNN\$value$: value of fitting criterion (eg: SSE) plus weight decay term ($decay * W^T W$).
- $myNN\$wts$: returns the calculated weights in the same order as the input Wts.
- $myNN\$convergence$: 1 if the maximum number of iterations was reached, otherwise 0.
- $myNN\$call$: returns the function (with the selected inputs) that was called.
- $myNN\$terms$: returns information about factors and response variables.
- $myNN\$coefnames$: returns the names of the factors.
- $myNN\$fitted.values$: vector of the fitted values for the training data.
- $myNN\$residuals$: vector of the residuals for the training data.

3. 'NeuralNetTools' package

3.1. plotnet

The plotnet function provides a visual representation of the neural network. Black lines represent positive weights, and grey lines represent negative weights. The thicker the line, the larger the magnitude of the weight. Some examples are illustrated below.

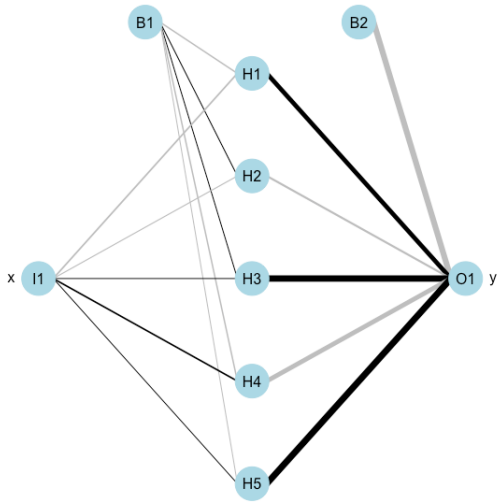

Figure 1: 1-5-1 ANN

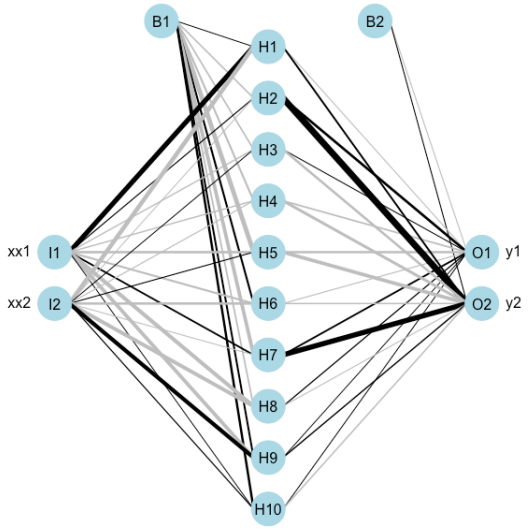

Figure 2: 2-10-2 ANN

3.2. Relative Importance

The garson function calculates the relative importance of each predictor variable in a neural network using Garson's algorithm. Garson's algorithm uses the weights of the neural network to calculate the relative importance [4]. Note: garson only applies to neural networks with one output node.

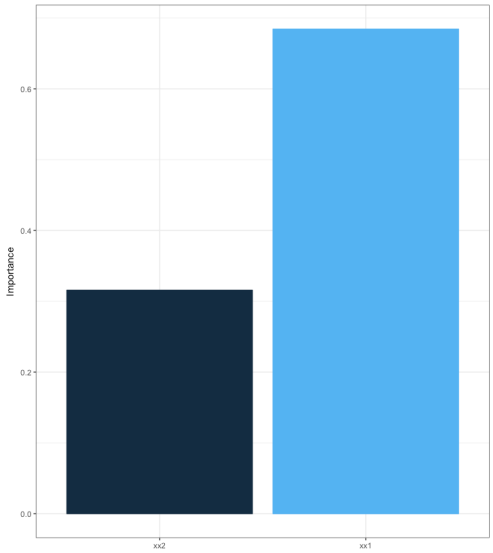

Figure 3: Garson's Relative Importance

4. An Example

Here, we compare the performance of the nnet package to the neuralnet package. Both neural networks have the same training data, hyper-parameters (hidden units: 8, layers: 1 hidden layer), and cost function (SSE, no decay); however, the nnet package uses the BFGS algorithm while the neuralnet package uses backpropagation. Training and testing data was generated from the function y = $x^3 + 4 \cdot sin(4 \cdot x) + Z$, where Z $\sim$ WN(0,1). As seen below, the nnet package yields a significantly lower test SSE in this example.

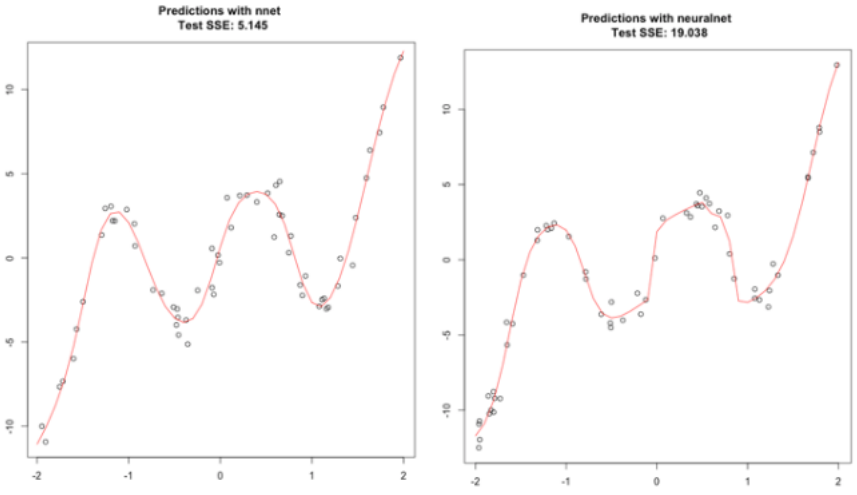

Figure 4: Garson's Relative Importance

5. References

[1] Póczos, B. and Tibshirani, R. (2013) CMU-10725: Convex Optimization - Quasi Newton Methods. Carnegie Mellon. https://www.stat.cmu.edu/ ryantibs/convexopt-F13/lectures/.

[2] Goodfellow, I., Bengio Y., and Courville, A. (2016) Deep Learning. MIT Press.

[3] Dennis. J.E., and Schabel, R.B. (1996) Numerical Methods for Unconstrained Optimization and Nonlinear Equations, chapter 9.

[4] Garson, D. G. (1991), 'Interpreting neural-network connection weights', Artificial Intelli- gence Expert 6(4), 46–51.