Technical Note: An R package for fitting Bayesian regularized neural networks with applications in animal breeding[1]

P. Pérez-Rodríguez,*†[2] D. Gianola,†‡§ K. A. Weigel,† G. J. M. Rosa,†§ and J. Crossa#

*Colegio de Postgraduados, Km. 36.5 Carretera Mexico-Texcoco, C.P. 56230;
†Department of Animal Sciences, ‡Department of Dairy Science, and §Department of Biostatistics and Medical Informatics, University of Wisconsin, Madison, 53706; and #Biometrics and Statistics Unit, International Maize and Wheat Improvement Center (CIMMYT), Apdo. Postal 6-641, Mexico D.F., Mexico.

ABSTRACT: In recent years, several statistical models have been developed for predicting genetic values for complex traits using information on dense molecular markers, pedigrees, or both. These models include, among others, the Bayesian regularized neural networks (BRNN) that have been widely used in prediction problems in other fields of application and, more recently, for genome-enabled prediction. The R package described here (brnn) implements BRNN models and extends these to include both additive and dominance effects. The implementation takes advantage of multicore architectures via a parallel computing approach using openMP (Open Multiprocessing) for the computations. This note briefly describes the classes of models that can be fitted using the brnn package, and it also illustrates its use through several real examples.

Key words: animal model, Bayesian Regularized Neural Networks, dominance and additive effects, genomic selection, nonparametric models

 J. Anim. Sci. 2013.91:3522–3531
doi:10.2527/jas2012-6162

INTRODUCTION

Neural Networks (**NN**) are mathematical models that have been used in prediction problems in many fields of research (e.g., Minsky and Papert, 1972; Werbos, 1974; Ripley, 1996). However, the application of NN to prediction problems involving genotypic and phenotypic information is recent in animal breeding (Long et al., 2010; Gianola et al., 2011; Okut et al., 2011). Predictions derived from the fitted models can be used for example for selection individuals. It has been found that 2 classes of NN, Bayesian regularized neural networks (**BRNN**) and radial basis function neural networks, provide very flexible models that can be better than linear models in terms of predictive ability, and can be also better for cryptic underlying processes (e.g., González-Camacho et al., 2012; Pérez-Rodríguez et al., 2012). In the case of the BRNN, however, it is surprising that only a few existing software packages in either the public or commercial domains implement this type of model. Examples are the function *trainbr* in Matlab R2010b and the *fbm* software package of Neal (1996). Unfortunately, it is evident that these software packages lack flexibility for dealing with the types of data and models commonly used in animal and plant breeding.

R is a well known, popular, open-source software tool for statistical analysis and graphics (R Core Team, 2012), and it has been developed and maintained by a group of first-class statisticians and programmers. R is flexible, in the sense that it can be extended with new functionalities by writing packages that can use code written in R, C/C++, or Fortran (R Core Team, 2012). This software is becoming a tool of choice for data analysis with scientific or commercial objectives all over the world (The New York Times, January 7, 2009). There are also commercial extensions of this software for enterprise class users (see, for example, Revolution Analytics, www.revolutionanalytics.com). We have developed a package called **brnn** that

[1]The authors express their thanks to 2 anonymous referees and the Editor for their constructive comments, which helped to greatly improve the presentation of the original version of this document and the software. Financial support by the Wisconsin Agriculture Experiment Station and by AVIAGEN, Ltd. (Newbridge, Scotland) to Paulino Pérez-Rodríguez and Daniel Gianola is acknowledged.

[2]Corresponding author: perpdgo@gmail.com

Received December 7, 2012.
Accepted April 16, 2013.

implements BRNN, and we now extend its application, such that additive and dominance effects can be fitted for genome-enabled selection. The package is available at the CRAN site (Comprehensive R Archive Network, http://cran.r-project.org/mirrors.html).

In this application note, we discuss the approach used by brnn to fit models that include additive and dominance effects jointly. We begin by introducing linear and nonlinear models and the BRNN. Next, we discuss the types of algorithm used for fitting these models. Subsequently, the extension for dealing with additive and dominance effects is presented. Finally, some examples and concluding remarks are given.

METHODS

Linear, Nonlinear Models, and Neural Networks

Meuwissen et al. (2001) introduced the use of linear regression models in the context of what is called genomic selection (**GS**). The basic linear regression model for additive effects is

$$y_i = \mu + \sum_{j=1}^{p} x_{ij}\beta_j + \varepsilon_i, \qquad [1]$$

where y_i is a target trait measured on individual i; μ is an intercept; β_j is the allele substitution effect of marker j ($j = 1,2,\ldots, p$); x_{ij} is the jth marker genotype observed in individual i, and $\varepsilon_i \sim N(0,\sigma_e^2)$, where σ_e^2 is the residual variance. A more general regression model suitable for capturing nonlinear patterns can be written as

$$y_i = \mu + g(\mathbf{x}_i) + \varepsilon_i, \qquad [2]$$

where $g(\cdot)$ is a function that maps from the input space (p-dimensional) to the real line, representing a conditional (given $\mathbf{x}_i$) expectation function, where $\mathbf{x}_i$ is the set of genotypic codes observed on i and ε_i is as in Eq. [1]. It is well known that any nonlinear function can be exactly represented as a finite sum of smooth functions (Kolmogorov, 1957; Poggio and Girosi, 1990; Kurkova, 1992). This motivates the definition of a NN, where functions are approximated as sums of finite series of smooth functions. One of the most basic and well known NN is the Single Hidden Layer Feed Forward Neural Network (**SLNN**), also known as the perceptron, shown in Fig. 1. The predictions from this class of nets can be obtained in 2 steps: first the inputs are nonlinearly transformed in a hidden layer, and then the outputs from the hidden layer are combined linearly to obtain the predictions, as given by Gianola et al. (2011). The model is then:

$$y_i = \mu + \sum_{k=1}^{s} w_k g_k\left(b_k + \sum_{j=1}^{p} x_{ij}\beta_j^{[k]}\right) + \varepsilon_i \qquad [3]$$

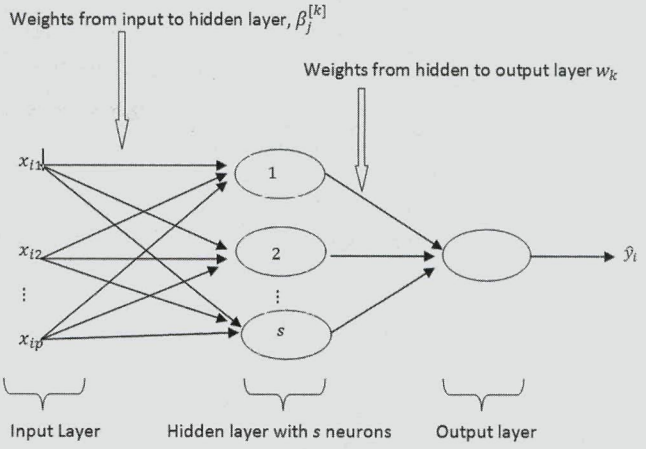

Figure 1. Illustration of Single layer feed forward neural network. The **x** values are values of p input variables, such as SNP genotypesv, and $\hat{y}_i$ is the value of y_i (phenotype) as predicted by the network.

where $(w_1,\ldots, w_s)$ are network weights; $(b_1,\ldots, b_s)'$ are called *biases* in machine learning; $(\beta_1^{[1]}, \ldots, \beta_p^{[1]};\ldots; \beta_1^{[s]},\ldots, \beta_p^{[s]})'$ are connection strengths, where $\beta_j^{[k]}$ denotes a parameter for input j in neuron $k = 1,\ldots, s$, and $g_k(\cdot)$ is the activation function, that is, a function that maps the inputs from the real line into the bounded open interval $(-1,1)$. For example, $g_k(x) = 2/[1 + \exp(-2x)] - 1$ is known as the *tanh* (tangent hyperbolic) activation function.

Bayesian Regularized Neural Networks

The SLNN described above is flexible enough to approximate any nonlinear function. When the number of inputs (p) and the number of neurons (s) increases, the number of parameters to estimate in the network also increases. This, together with the flexibility of the neural network, can lead to serious problems of overfitting. To avoid this issue, it is common to use penalized estimation methods implemented, for example, by using Bayesian approaches. Shrinkage is toward some prior distribution. MacKay (1992, 1994) developed algorithms used to obtain estimates of all parameters in a BRNN by using an Empirical Bayes (**EB**) approach. A brief description of it is provided below.

Let $\boldsymbol{\theta} = (w_1,\ldots, w_s; b_1,\ldots, b_s; \beta_1^{[1]},\ldots, \beta_p^{[1]},\ldots, \beta_1^{[s]},\ldots, \beta_p^{[s]})'$ be the vector of weights, biases, and connection strengths. Note that μ is not included in $\boldsymbol{\theta}$, as this parameter can be easily eliminated, for example, simply by centering the response vector. Let $p(\boldsymbol{\theta}|\sigma_\theta^2) = MN(\mathbf{0}, \sigma_\theta^2\mathbf{I})$ be a prior distribution, where MN stands for the multivariate normal distribution, and σ_θ^2 is a variance common to all elements of $\boldsymbol{\theta}$. In the EB approach, these 2 steps are repeated iteratively until convergence:

1. Obtain conditional posterior modes of the elements in θ, assuming σ_e^2 and σ_θ^2 are known. These modes are obtained by maximizing

$$p\left(\boldsymbol{\theta}|\boldsymbol{y},\sigma_e^2,\sigma_\theta^2\right)=\frac{p\left(\boldsymbol{y}|\boldsymbol{\theta},\sigma_e^2\right)p\left(\boldsymbol{\theta}|\sigma_\theta^2\right)}{p\left(\boldsymbol{y}|\sigma_e^2,\sigma_\theta^2\right)},$$

which is equivalent to minimizing the augmented sum of squares

$$F(\boldsymbol{\theta})=\frac{1}{2\sigma_e^2}\sum_{i=1}^{n}e_i^2+\frac{1}{2\sigma_\theta^2}\sum_{j=1}^{m}\theta_j^2,$$

where $e_i=y_i-\hat{y}_i$ is the prediction error.

2. Update the variance components σ_e^2 and σ_θ^2 by maximizing $p(\boldsymbol{y}|\sigma_e^2, \sigma_\theta^2)$. Because of nonlinearity, the marginal log-likelihood, $\log p(\boldsymbol{y}|\sigma_e^2, \sigma_\theta^2)$ is not expressible in closed form, but it can be approximated as

$$G(\alpha,\beta)=\log p\left(\boldsymbol{y}|\sigma_e^2,\sigma_\theta^2\right)\approx c+\frac{n}{2}\log\beta+\frac{m}{2}\log\alpha-\frac{1}{2}\log|\boldsymbol{\Sigma}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{map}}-F(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{map}},$$

where c is a constant; $\boldsymbol{\Sigma}=\nabla^2F(\boldsymbol{\theta})$ is a matrix of second derivatives of order $m \times m$; m is the order of $\boldsymbol{\theta}$, and **map** stands for *maximum a posteriori*.

Gianola et al. (2011) have shown that the standard additive infinitesimal model of quantitative genetics (Fisher, 1918) can be viewed as a NN with a single neuron. This can be motivated as follows, by letting

$$\boldsymbol{y}=\mathbf{u}+\boldsymbol{\varepsilon}, \qquad [4]$$

where $\mathbf{u} \sim MN(\mathbf{0},\sigma_u^2\mathbf{A})$ is a vector of additive effects, $\mathbf{A}$ is the additive relationship matrix computed from a pedigree, and $\boldsymbol{\varepsilon} \sim MN(\mathbf{0},\sigma_e^2\mathbf{I})$. The model given in Eq. [4] can be rewritten as

$$y=\boldsymbol{u}+\boldsymbol{\varepsilon}=\mathbf{L}\mathbf{z}\sigma_u+\boldsymbol{\varepsilon}=\mathbf{L}\mathbf{u}^*+\boldsymbol{\varepsilon}, \qquad [5]$$

where $\mathbf{L}=\{l_{ij}\}$ is the lower triangular matrix obtained from the Cholesky decomposition of $\mathbf{A}$, that is, $\mathbf{LL}'=\mathbf{A}$, $\mathbf{z} \sim MN(\mathbf{0},\mathbf{I})$, and $\mathbf{u}^*=\mathbf{z}\sigma_u \sim MN(\mathbf{0},\sigma_u^2\mathbf{I})$. It is clear that Eq. [5] is a special case of Eq. [3] obtained by taking $\mu = 0$ (y is centered at 0), $s = 1$, $w_1 = 1$, $b_1 = 0$, $x_{ij}=l_{ij}$ and $\beta_j^{[1]}=u_j^*$, $i,j=1,\ldots,n$ with g_1 being the identity operator. Gianola et al. (2011) pointed out that the predictive ability of this model can be potentially enhanced by including s neurons during the effecting of nonlinear transformations, which leads to the model given in Eq. [3]. In theory, this confers the model enormous flexibility, because the additive relationship information is used adeptly.

As the number of predictors increases, so does the number of parameters to estimate. Therefore, when dealing with high-dimensional genotypic data it is advisable to reduce the dimensionality of the problem. In the case of NN, this is straightforward because as shown above, the infinitesimal model can be viewed as a regression on pedigrees or on a genomic relationship matrix, as pointed out by de los Campos et al. (2009) and Gianola et al. (2011). For example, a genomic relationship matrix suggested by VanRaden (2008) as:

$$\mathbf{G}=\frac{\mathbf{XX}'}{2\sum_{j=1}^{p}p_j(1-p_j)}, \qquad [6]$$

where $\mathbf{X}=\{x_{ij}\} \in \{-1,0,1\}$ is the matrix of SNP codes of dimension $n \times p$ and p_j is the minor allele frequency for SNP j, $j = 1,\ldots,p$. In short, the basic NN given by Eq. [3] can be fitted by using as predictors: i) the incidence matrix $\mathbf{X}$, ii) the pedigree information ($\mathbf{A}$ matrix), or iii) a genomic relationship matrix (e.g., $\mathbf{G}$).

Typically, it is convenient to scale inputs to reside in [–1,1], because the behavior of NN is sensitive with respect to scale. In the approach used here, the training data are rescaled to reside in [–1, 1], the normalization information is retained so that once that the model is fitted, the predictions can be given in the original scale. The input information for the testing set (if present) is rescaled using the normalization information obtained from the training set, then the prediction is made and the output is rescaled so that it is mapped back to the original scale. This approach is used for example in Matlab R2010b (see functions premnmx, postmnmx, and tramnmx). Gianola et al. (2011) illustrate with a simple example how the rescaling can be done. We have developed routines to do the rescaling automatically, so that the user does not have to worry about these issues, the information is provided in the original scale and the predictions are obtained in the original scale.

A Neural Network with Additive and Dominance Effects

Several authors (e.g., Dagnachew et al., 2011; Wittenburg et al., 2011; Wellmann and Bennewitz, 2012) have pointed out that most of the models used in genomic selection focus on additive effects only. Wellmann and Bennewitz (2012) showed that including dominance effects in the statistical models can enhance the quality of the predictions. In this section we present an extension of Eq. [3] for including additive and dominance effects together in a NN. We also present the algorithm used for fitting this model by using the EB approach, with a similar strategy to that developed by MacKay (1992, 1994).

A NN model that includes additive and dominance effects jointly is given by:

$$y_i=\mu+\sum_{k=1}^{s_a}w_k^a g\left(b_k^a+\sum_{j=1}^{p}x_{ij}\beta_j^{a[k]}\right)+\sum_{k=1}^{s_d}w_k^d g\left(b_k^d+\sum_{j=1}^{p}z_{ij}\beta_j^{d[k]}\right)+\varepsilon_i \qquad [7]$$

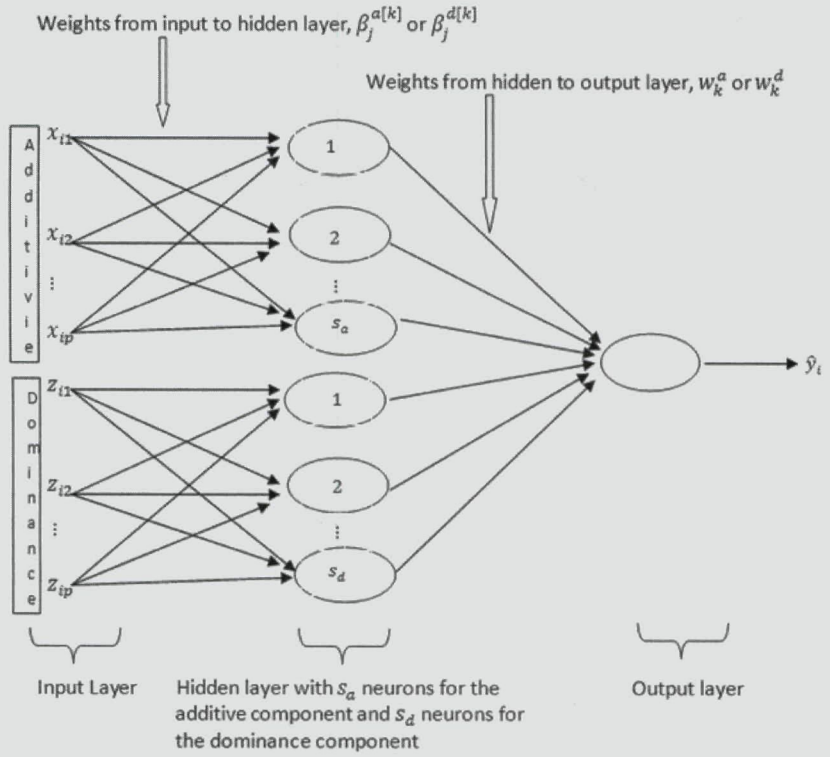

Figure 2. Graphical representation of the neural network given in Eq. [7]. In a marker based model, the x_{ij} and z_{ij} ($i = 1, ..., n; j = 1, ..., p$) inputs are the entries of additive and dominance matrices, respectively. Further, s_a and s_d are the numbers of neurons for the additive and dominance components in the hidden layer, respectively, and $\hat{y}_i$ is the value of y_i (phenotype) as predicted by the network.

where $\mathbf{Z} = \{z_{ij}\} \in \{0,1\}$ is the incidence matrix for dominance effects of dimension $n \times p$; if markers are used, $z_{ij} = 1$ if SNP j for individual i is heterozygous, and $z_{ij} = 0$ otherwise. Further, s_a and s_d are the numbers of neurons for the additive and dominance components in the hidden layers, respectively. The remaining parameters are as for Eq. [3]. Figure 2 shows a graphical representation of a NN with dominance and additive components. Note that we have disallowed connections between *additive* predictors and *dominance* neurons, and vice versa, so that the number of parameters to estimate is reduced considerably. We think that the universal approximation capability is not lost. Note also that the ratios

$$h_a^2 = \frac{1}{2\alpha} \Big/ \left(\frac{1}{2\alpha} + \frac{1}{2\delta} + \frac{1}{2\beta} \right)$$

and

$$h_d^2 = \frac{1}{2\delta} \Big/ \left(\frac{1}{2\alpha} + \frac{1}{2\delta} + \frac{1}{2\beta} \right)$$

can be interpreted as proportion of variances due to the additive and dominant component respectively (see Gianola et al., 2011). This kind of approach has been used in other fields, for example MacKay and Neal (1994); Neal (2001). The inputs are divided in groups; each group of inputs has its own noise level in a feed-forward network with several layers of unconnected neurons.

Empirical Bayes Estimation

For simplicity, assume that the parameter μ in Eq. [7] has been eliminated by centering the observations. Conditionally on t network parameters, the n phenotypes are assumed to be independently distributed with density:

$$p\left(\mathbf{y}|\boldsymbol{\varphi}, \sigma_e^2\right) = \prod_{i=1}^{n} N(y_i \mid \boldsymbol{\varphi}, \sigma_e^2),$$

where $\boldsymbol{\varphi}$ denotes a vector of dimension $t \times 1$ including all connection strengths, and coefficients for additive and dominance effects, as well as weights and biases; σ_e^2 is the residual variance.

Let $\boldsymbol{\varphi} = (\boldsymbol{\theta}_a' \boldsymbol{\theta}_d')'$, where $\boldsymbol{\theta}_a$ denotes the vector of dimension $m \times 1$ with strengths for additive effects, and $\boldsymbol{\theta}_d$ denotes the vector of dimension $q \times 1$ with strengths for the dominance effects. Note that $t = m + q$. Assuming a priori independent normal distributions for the connection strengths: $p(\boldsymbol{\varphi}|\boldsymbol{\psi}) = MN[\mathbf{0}, \text{bdiag}(\sigma_a^2 \mathbf{I}_{m \times m}, \sigma_d^2 \mathbf{I}_{q \times q})]$, where $\boldsymbol{\psi} = (\sigma_a^2, \sigma_d^2)'$ are the variances of connection strengths and weights for additive and dominance effects, and bdiag $(\cdot, \ldots, \cdot)$ represents a block diagonal matrix.

Conditional Posterior Mode

Assuming that the variance parameters are known, the posterior density of the connections strength is given by

$$p\left(\varphi \mid y,\psi,\sigma_e^2\right)=\frac{p(y|\varphi,\sigma_e^2)p(\varphi \mid \psi)}{p(y \mid \psi,\sigma_e^2)}, \qquad [8]$$

The denominator in Eq. [8] is the marginal density of the data, and it is given by:

$$p(y \mid \psi,\sigma_e^2) = p\left(y|\varphi,\sigma_e^2\right)p(\varphi \mid \psi)d\varphi$$
$$= C\times\int\exp\left\{\begin{array}{l}-\frac{1}{2\sigma_e^2}\sum_{i=1}^{n}e_i^2 - \\ \frac{1}{2\sigma_a^2}\theta_a'\theta_a - \frac{1}{2\sigma_d^2}\theta_d'\theta_d\end{array}\right\}d\varphi, \qquad [9]$$

where $C=\left(\frac{1}{2\pi\sigma_e^2}\right)^{\frac{n}{2}}\left(\frac{1}{2\pi\sigma_a^2}\right)^{\frac{m}{2}}\left(\frac{1}{2\pi\sigma_d^2}\right)^{\frac{q}{2}}$ and $e_i=y_i-\hat{y}_i$ is the prediction error.

As before, EB consists of 2 steps: first the conditional posterior modes of connection strengths are obtained assuming that the variance components are known, and then the variance components are reestimated in the second step. From Eq. [8] note that:

$$p\left(\varphi|\mathbf{y},\psi,\sigma_e^2\right)\propto p\left(\mathbf{y}|\varphi,\sigma_e^2\right)p(\varphi \mid \psi), \text{ so that}$$
$$\log p\left(\varphi|\mathbf{y},\psi,\sigma_e^2\right) \propto \log p\left(y \mid \varphi,\sigma_e^2\right)+\log p(\varphi \mid \psi)$$
$$= -\beta\sum_{i=1}^{n}e_i^2 - \alpha\theta_a'\theta_a - \delta\theta_d'\theta_d$$

where $\beta=\frac{1}{2\sigma_e^2}$, $\alpha=\frac{1}{2\sigma_a^2}$, $\delta=\frac{1}{2\sigma_d^2}$.

Assuming β, α, and δ are known, we can maximize this log-posterior to obtain $\varphi = \varphi^{map}$. Note that this is equivalent to minimizing:

$$Q(\varphi)=\beta\sum_{i=1}^{n}e_i^2 + \alpha\theta_a'\theta_a - \delta\theta_d'\theta_d \qquad [10]$$

***Tuning** β, α, δ*

For tuning the variance parameters we can maximize the marginal likelihood of the data given in Eq. [9]. The integral in that equation does not have a closed form because of nonlinearity, but it can be approximated using Laplace's method (Tierney and Kadane, 1986), that is:

$$\int\exp\{-nh(\varphi)\}\,d\varphi \approx \left(\frac{2\pi}{n}\right)^{t}|\Sigma|^{-1/2}\exp[-nh(\varphi)] \qquad [11]$$

where t is the number of elements in vector φ, n is the number of data points, and Σ is the Hessian matrix, that is, the matrix of second derivatives of $h(\varphi)$ with respect to φ.

Using Eq. [11] in Eq. [9], the integral $I = \int\exp[-Q(\varphi)]d\varphi$ can be approximated as follows:

$$I \approx (2\pi/n)^t|\Sigma|^{-1/2}\exp[-Q(\varphi)], \qquad [12]$$

where

$$\Sigma = \nabla^2 Q(\varphi) = \beta\nabla^2\sum_{i=1}^{n}e_i^2 + \alpha\nabla^2\theta_a'\theta_a + \delta\nabla^2\theta_d'\theta_d.$$

The Hessian matrix can be approximated using a procedure similar to that employed by Foresee and Hagan (1997). First note that

$$\beta\nabla^2\sum_{i=1}^{n}e_i^2 \approx 2\beta J'J$$

where **J** is the Jacobian for the training set errors [see Hagan et al. (2002), p. 12 to 21]; then $\alpha\nabla^2\theta_a'\theta_a$ = bdiag $(2\alpha\mathbf{I}_{m\times m}, \mathbf{0}_{q+2\times q+2})$, $\delta\nabla^2\theta_d'\theta_d$ = bdiag $(\mathbf{0}_{m\times m}, 2\delta\mathbf{I}_{q\times q}, \mathbf{0}_{2\times 2})$ so $\Sigma \approx 2\beta J'J + 2\text{bdiag}(\alpha\mathbf{I}_{m\times m}, \delta\mathbf{I}_{q\times q})$.

Substituting Eq. [12] into Eq. [9] and taking natural logarithms, one has:

$$G\left(\beta,\alpha,\delta,\psi=\psi^{map}\right)=\log p\left(y|\psi,\sigma_e^2\right)\approx$$
$$c+\frac{n}{2}\log\beta+\frac{m}{2}\log\alpha+\frac{q}{2}\log\delta - , \qquad [13]$$
$$\frac{1}{2}\log|\Sigma|_{\varphi=\varphi^{map}} - Q(\varphi)|_{\varphi=\varphi^{map}}$$

where c is a constant.

We can now take partial derivatives of $G(\cdot)$ with respect to each of the dispersion parameters, β, α, δ set to 0, and obtain expressions for updating these parameters. Fortunately, this problem has exactly the same solution as that found by MacKay (1992, 1994) in the context of "multiple regularization constants" (i.e., the assignment of different prior distributions to weights, biases, and the general bias in a NN). Doing this, one obtains the iteration, from right to left:

$$\alpha_{new} = \gamma_a/2\theta'_a\theta_a \text{ with } \gamma_a = m - 2\alpha\text{Trace}_a(\Sigma^{-1}),$$

$$\delta_{new} = \gamma_d/2\theta'_d\theta_d \text{ with } \gamma_d = q - 2\delta\text{Trace}_d(\Sigma^{-1}),$$

$$\beta_{new} = \frac{n-\gamma}{2\sum_{i=1}^{n}e_i^2},$$

The trace of the inverse Hessian is taken over the weights and bias for additive and dominance, respectively, and $\gamma = \gamma_a + \gamma_d$.

MacKay (1992, 1994) also gives approximations that do not require calculation of the Hessians: $\alpha_{new}=\frac{m}{4\theta_a'\theta_a}$, $\delta_{new}=\frac{q}{4\theta_d'\theta_d}$ and $\beta_{new}=\frac{n}{4\sum_{i=1}^{n}e_i^2}$.

Algorithm

The algorithm for fitting model [7] can be summarized as follows:

0. Initialize β, α, δ and the weights using the Nguyen and Widrow (1990) algorithm.

1. Take 1 step of the Levenberg-Marquardt algorithm to minimize the objective function $Q(\boldsymbol{\varphi})$ given in Eq. [10].
2. Update β, α, δ by maximizing Eq. [13] using the Nelder and Mead (1965) algorithm, or by using the update formula of MacKay (1992, 1994), as shown above.
3. Iterate Steps 1 and 2 until convergence.

The algorithm described above is a generalization of the algorithm presented in Foresee and Hagan (1997). A caveat pointed out by Carlin and Louis (2009) is that Laplace's approximation may not be sufficiently accurate for moderate to highly dimensional $\boldsymbol{\theta}$ (e.g., larger than 10).

The new algorithm can be extended to accommodate the situation where the entries of additive and dominance relationships are used as inputs in the network. Equation [4], extended to include dominance effect, becomes:

$$\boldsymbol{y} = \boldsymbol{u} + \mathbf{d} + \boldsymbol{\varepsilon}, \qquad [14]$$

where $\mathbf{d}$ is the vector of dominance effects, $\mathbf{d} \sim NM(\mathbf{0},\sigma_d^2\mathbf{D})$, $\mathbf{D}$ is a matrix of dominance relationships, and σ_d^2 is the dominance variance.

Using a representation analogous to that employed by Gianola et al. (2011), the model can be written as $\boldsymbol{y} = \mathbf{u} + \mathbf{d} + \boldsymbol{\varepsilon} = \mathbf{L}\mathbf{z}\sigma_u + \mathbf{W}\mathbf{v}\sigma_d + \boldsymbol{\varepsilon} = \mathbf{L}\mathbf{u}^* + \mathbf{W}\mathbf{d}^* + \boldsymbol{\varepsilon}$ with $\mathbf{W}\mathbf{W}^T = \mathbf{D}$ and $\mathbf{v} \sim NM(\mathbf{0},\mathbf{I})$, producing a special case of Eq. [7]. The genomic dominance relationship matrix $\mathbf{D}$ can be computed using the same arguments of Van Raden (2008) as:

$$\mathbf{D} = \frac{\mathbf{Z}\mathbf{Z}'}{2\sum_{j=1}^{p} p_j q_j (1-2p_j q_j)},$$

where $q_j = 1 - p_j$, $j = 1,\ldots,p$ and $\mathbf{Z} = \{z_{ij}\} \in \{0,1\}$ is the incidence matrix for dominance effects of markers of dimension $n \times p$.

Implementation

Algorithms that implement NN are complex, because it is necessary to perform time consuming matrix operations. This is a serious issue when the number of unknown parameters is very large. For example, for Eq. [3], the number of parameters to estimate is $m = 1 + s \times (2 + p)$, where s is the number of neurons, and p is the number of SNP. The Gauss-Newton algorithm used to minimize the "augmented sum of squares" requires solving systems of equations of order $m \times m$, which is an operation of order $O(m^3)$. MacKay (1992) pointed out that to maximize $G(\alpha,\beta)$, the marginal log likelihood, it is necessary to use methods that take into account the gradient of the function, which implies that the trace of $\boldsymbol{\Sigma}^{-1}$, of dimension $m \times m$, must be computed. This is an operation of order $O(m^3)$ and this can be problematic if m is huge and a single central processing unit (CPU) is used.

Nevertheless, many operations (including matrix manipulations) involved in the algorithm can be parallelized in modern multicore systems. We implemented the algorithms such that it was possible to take advantage of the multicore feature. We linked R against the Intel Math Kernel Library (intel-mkl, http://software.intel.com/en-us/articles/intel-mkl/) to perform matrix manipulations, and portions of the algorithms were coded in the C language using OpenMP (Chapman et al., 2007) to support multicore architectures in UNIX-like systems. The resulting code can be used with any standard R installation, but it works much better with R installations in UNIX-like systems linked against highly optimized matrix libraries, for example the intel-mkl or, in Apple systems, the Automatically Tuned Linear Algebra Software (**ATLAS**, http://math-atlas.sourceforge.net/). Additionally, it is possible to run the software in highly optimized versions of R, for example Revolution Analytics (www.revolutionanalytics.com/), and there are versions for Linux and Windows that are free for academic purposes.

The inversion of matrices for updating the variance components can be avoided by using methods that do not require derivatives, but the resulting algorithm may be numerically unstable. We also implemented a Monte Carlo version of the algorithm developed by Foresee and Hagan (1997) for fitting model [3] by using results about estimation of traces of the inverse of large scale matrices given in Bai et al. (1996) and Guo (2000). The algorithm was coded in C and parallelized using OpenMP (Chapman et al., 2007); for more information on the latter, see http://openmp.org.

EXAMPLES

Jersey Dataset

These data were used by Gianola et al. (2011), including milk production records for 3 traits (fat yield, milk yield and protein yield) in Jersey cows. The incidence matrices with marker codes for additive and dominance effects (**X** and **Z**, respectively) were derived from 33,262 SNP on each of 297 individually genotyped cows.

Example 1: Infinitesimal Model for Additive Effects as a Neural Net. This example shows how to use the software to fit the additive model described in Gianola et al. (2011). The necessary data are stored as R objects that can be accessed once the library `brnn` is loaded. The phenotypic information is stored in the data frame object `pheno`, and the additive genomic relationship matrix is stored in object `G`. The dataset also includes a vector (`partitions`) that assigns observations to 10 disjoint sets; this vector is later used to perform a 10 fold cross validation. We show next how to fit Eq. [3] using the entries of the $\boldsymbol{G}$ matrix as

inputs, and milk yield as response variable. The model is fitted using the formula interface, commonly used for specifying the response and explanatory variables in R, although it is also possible to supply the data as free-standing vector and matrices in the parent environment.

```
library(brnn)
data(Jersey)
#Create a data frame with the
#information necessary to fit the model
data = pheno
data$X = G
data$partitions = partitions
#Fit the model for the TESTING DATA
set.seed(456)
out = brnn(yield_devMilk~X,
data = subset(data,partitions! =
2),neurons = 2)
#Fitted values for the TRAINING SET
predict(out)
#Fitted values for the TESTING SET
predict(out,newdata
= subset(data,partitions = = 2))
```

The data are loaded into the workspace by using the function `data`. Once the data are loaded, a data frame is created to hold all the information necessary to fit the model. The model is fitted to the training data using the function `brnn`; here, 2 neurons are used as architecture. A complete list of arguments for the function `brnn` can be found in the manual of the package or by typing `?brnn` in the R command line. Once the model is fitted, results are stored in object `out`, which is an object of class `brnn`, mostly internal structure but has components:

- `$theta`: A list containing weights and biases. The list has *s* components that contain vectors with estimated parameters for the *k*th neuron, that is, $\left(\hat{w}_k, \hat{b}_k, \hat{\beta}_1^{[k]},\ldots,\hat{\beta}_p^{[k]}\right)$.
- `$message`: String that indicates the stopping criteria for the training process.
- `$alpha`: $\hat{\alpha} = (2\hat{\sigma}_\theta^2)^{-1}$.
- `$beta`: $\hat{\beta} = (2\hat{\sigma}_e^2)^{-1}$.
- `$gamma`: Estimated effective number of parameters ($\hat{\gamma}$).
- `$Ew`: The sum of the squares of the biases and weights.
- `$Ed`: The sum of the squares of the differences between observed and predicted values.

The predictions from the NN can be obtained by using the function predict that takes as arguments the output from function `brnn` and optionally a data frame in which to look for variables with which to predict. The function returns a vector with predictions.

The model was also fitted using the function `trainbr` in Matlab R2010b, as well as using the flexible Bayesian modeling package (Neal, 1996), which can be downloaded freely from www.cs.toronto.edu/~radford/fbm.software.html. The software developed by Neal fits neural network models using Markov Chain Monte Carlo (**MCMC**) techniques, and a comprehensive discussion can be found in Lampinen and Vehtari (2001). Inferences derived from Neal's model are based on 20,000 samples obtained after discarding 10,000 samples that were taken as burn-in, with the prior distributions being those described in Lampinen and Vehtari (2001).

Figure 3 shows the predictions obtained for milk yield from the 3 algorithms for training and testing set. As expected, the outputs from `brnn` and `trainbr` were quite similar but, there were some differences in results between these 2 methods and fbm, as the latter uses MCMC. The model was run using a Linux machine with an Intel Core i7–2620M CPU processor @2.70GHz with 8 GB of RAM memory. In the case of the fbm software, it took ~4 h to fit the model, whereas Matlab took ~2 min, and our R script required ~5 min.

These items should be noted:

1. The estimated effective number of parameters ($\hat{\gamma}$) can provide some insight about the number of neurons to be used in the neural network. For example if a model if fitted with 2 and 3 neurons and remains $\hat{\gamma}$ about the same in both cases the model with fewer neurons is preferred because the model has fewer parameters. A detailed discussion about the use of this parameter can be found in Foresee and Hagan (1997), Gianola et al. (2011) and Okut et al. (2011).
2. In UNIX-like systems it is possible to use more than 1 core for the computations. This can be done by setting for example `cores = k` in the list of arguments of the function `brnn`, where *k* is the number of cores requested. The function `detectCores` in the R package `parallel` can be used to attempt to detect the number of CPU in the machine that R is running, but not necessarily all the cores are available for the current user, because, for example, in multiuser systems it will depend on system policies. More details about how to know the number of cores and related issues can be found in the documentation for the `parallel` package in R. We refitted the model discussed above setting `cores = 2` in the `brnn` function, and our R script required about ~3 min to finish the task.

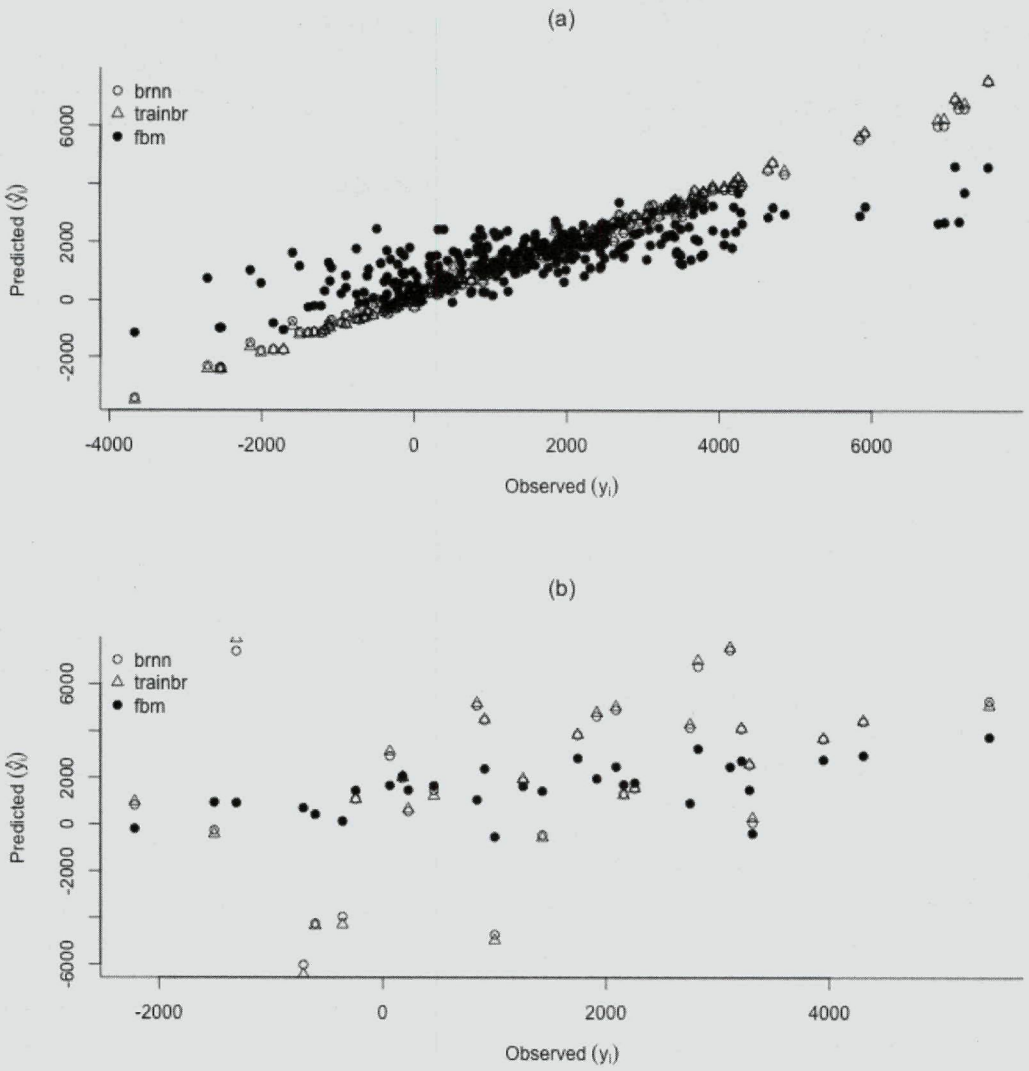

Figure 3. Predictions of milk yield for a) training and b) testing sets using the Bayesian regularization implemented in Bayesian regularized neural networks (brnn) library, trainbr function (Matlab), and fbm (Neal's package). $\hat{y}_i$ is the value of y_i (phenotype) as predicted by the network.

3. The model can also be fitted using a Monte Carlo algorithm for estimating the trace of the inverse of the Hessian matrix to update variance parameters. This can be done by setting `Monte_Carlo = TRUE` in the list of arguments of the function `brnn`.

4. The software also allows fitting the infinitesimal model given in Eq. [4] using the representation given in Eq. [5]. This can be done by replacing `data$X = G` with `data$X = t(chol(G))` in the code shown above. This may lead to overfitting in the training set, but it gives good predictions in the testing set. We notice that this also happens in Matlab with the `trainbr` function; it may be better to use directly the entries of **G** rather than those of its Cholesky decomposition.

Example 2: Including Additive and Dominance Effects Together in a Neural Net. In this example, Eq. [7] was fitted using the **G** and **D** matrices for the Jersey data. The model can be fitted using the function `brnn_extended`. The response variable and the predictors can be specified using the extended formula interface (Zeileis and Croissant, 2010). The function takes as arguments a formula with the response variable, the predictors separated with the symbol |, the number of neurons used to model dominance and additive effects (*neurons1* and *neurons2*, respectively). Additionally, it is also possible to supply the data as free-standing vector and matrices in the parent environment (type `?brnn_extended` in the R command line for more details). Next we show the R code used for fitting the model:

```
library(brnn)
data(Jersey)
data = pheno
data$X = G
data$Z = D

#Fit the model with Additive
#and Dominant effects
```

```
out = brnn_extended(yield_
devMilk ~ X | Z,
neurons1 = 2,neurons2 =
2,epochs = 100)
```

The results of the fitting process are stored in the R object out, which is an object of class `brnn_extended`, mostly internal structure but has components:

- `$theta1`: A list containing weights and biases for the additive component.
- `$theta2`: A list containing weights and biases for the dominance component.
- `$alpha`: $\hat{\alpha} = (2\hat{\sigma}_a^2)^{-1}$.
- `$beta`: $\hat{\beta} = (2\hat{\sigma}_e^2)^{-1}$.
- `$delta`: $\hat{\delta} = (2\hat{\sigma}_d^2)^{-1}$.
- `$E1`: The sum of squares of the parameters for the additive part.
- `$E2`: The sum of squares of the parameters of the dominance part.
- `$message`: String that indicates the reason for stopping the training.
- `$Ed`: The sum of squares of the differences between observed and predicted values.

Equation [7] cannot be fitted via the Matlab function `trainbr`, but it can be done with the fbm software, although in an indirect way that consists of using a feed-forward network with 2 layers of unconnected neurons, where each group of neurons has its own biases and weights (Neal, 2001).

Table 1 presents the results of the evaluation of the predictive power of the models including additive and dominance effects for a 10 fold cross-validation, using 2 neurons for the additive component and 2 neurons for dominance. The correlation between observed and predicted values for each of the folds was obtained as the average of 10 runs in the case of brnn software, to mitigate the effect of the starting values for all parameters in the net. The last 2 rows of Table 1 show the average correlation and the average root mean square error (**RMSE**). Inferences for Neal's model were based on 20,000 samples obtained after discarding 10,000 samples as burn-in; the prior distributions used were those described in Lampinen and Vehtari (2001). There was considerable of variability due to small sample size. On average, inclusion of the dominance component yielded slightly better predictions, even though milk yield is generally assumed to be an additive trait. Note that the predictions obtained by using the Neal program were much better than those obtained with the `brnn` function, perhaps because different prior distributions are assigned to weights, bias and connection strengths in the net or because the Laplace approximation that we are employing is not adequate.

Table 1. Correlations between observed and predicted values of milk yield in a testing set of Jersey cows[1]

	Additive		Additive + dominance	
fold	brnn-1–2–1	fbm-1–2–1	brnn-1–2–2–1	fbm-1–2–2–1
1	0.3455	0.4795	0.2715	0.4438
2	0.5211	0.5860	0.5546	0.4681
3	0.2553	0.4086	0.4264	0.5407
4	0.5241	0.6878	0.5051	0.7222
5	0.1118	0.2707	0.0989	0.3444
6	0.3821	0.5105	0.4211	0.5551
7	–0.1365	–0.1662	–0.0756	–0.0758
8	0.4533	0.6542	0.5387	0.7401
9	0.1361	0.0400	0.1974	–0.0800
10	0.4334	0.5411	0.4920	0.6106
Avg. correlation	0.3026	0.4012	0.3430	0.4269
Avg. RMSE	2498.17	1768.63	2392.10	1609.48

[1]brnn-1–2–1 = Bayesian regularized neural network with 2 neurons for the additive component; fmb-1–2–1 = Flexible Bayesian modeling (Neal's model) with 2 neurons for the additive component; brnn-1–2–2–1 = Bayesian regularized neural network with 2 neurons for each of the additive and dominance components; fbmnn-1–2–2–1 = flexible Bayesian modeling (Neal's model) with 2 neurons for each of the additive and dominance components; RMSE = root mean square error.

Concluding Remarks

We have developed open source software that allows fitting BRNN in R and that works in Windows and UNIX-like environments. This software takes advantages of multicore processors in UNIX-like systems by using OpenMP, and can take advantage of highly optimized linear algebra libraries to perform matrix computations. The software also uses state of the art algorithms to estimate the trace of the inverse of positive definite matrices that are needed for updating some parameters in the NN. On Windows platforms the parallel technology does not confer any advantage and the code is executed serially. The software is a generalization of that found in Matlab for fitting this type of model because it allows penalizing 2 types of inputs differently, which Matlab `trainbr` function does not allow.

The problem of fitting BRNN with the approach employed here and also with the MCMC developed by Neal (1996) can be time consuming. For genomic data for example, it is always desirable to reduce the dimensionality somehow; in this case it can be done computing the genomic relationship matrices whose number of rows and columns will be equal to the number of individuals (n). We have tested the algorithm in Linux machines @2.70 GHz with 4 and 16 cores with 8 and 16 Gb of RAM memory respectively and R linked against highly optimized matrix libraries described in the implementation section and the software was able to fit models with n = 500 individuals and employing the additive genomic relationship matrix (dimension $n \times n$) with up to 4 neurons. More efficient and scalable

programs can be written using parallel technology, for example we developed a program that is able to fit the additive model described in Gianola et al. (2011) using the C/C++ language and the ScaLAPACK libraries and the resulting program is able to fit a neural network with up to $n = 2500$ individuals and 4 neurons in the same 16 core machine described above. The resulting program is available on request from the first author.

LITERATURE CITED

Bai, Z. J., M. Fahey, and G. Golub. 1996. Some large-scale matrix computation problems. J. Comput. Appl. Math. 74(1–2):71–89.

Carlin, B. P., and T. A. Louis. 2009. Bayesian methods for data analysis. 3rd ed. CRC Press, Boca Raton, FL.

Chapman, B., G. Jost, and R. van van der Pas. 2007. Using OpenMP: Portable shared memory parallel programming (scientific and engineering computation). The MIT Press, Cambridge, MA.

Dagnachew, B. S., G. Thaller, S. Lien, and T. Adnoy. 2011. Casein SNP in Norwegian goats: Additive and dominance effects on milk composition and quality. Genet. Sel. Evol. 43:31.

de los Campos, G., D. Gianola, and G. J. M. Rosa. 2009. Reproducing kernel Hilbert spaces regression: A general framework for genetic evaluation. J. Anim. Sci. 87:1883–1887.

Fisher, R. A. 1918. The correlation between relatives on the supposition of Mendelian inheritance. Trans. R. Soc. Edinb. 52:399–433.

Foresee, F. D., and M. T. Hagan. 1997. Gauss-Newton approximation to Bayesian learning. International Conference on Neural Networks 3:1930–1935.

Gianola, D., H. Okut, K. A. Weigel, and G. J. M. Rosa. 2011. Predicting complex quantitative traits with bayesian neural networks: A case study with Jersey cows and wheat. BMC Genet. 12:87–100.

González-Camacho, J. M., G. de Los Campos, P. Pérez-Rodríguez, D. Gianola, J. Cairns, G. Mahuku, R. Babu, and J. Crossa. 2012. Genome-enabled prediction of genetic values using radial basis function neural networks. Theor. Appl. Genet. 25:759–771.

Guo, H. 2000. Computing traces of functions of matrices. Numerical mathematics (English Series) 2:204-215.

Hagan, M. T., H. B. Demuth, and M. H. Beale. 2002. Neural network design. PWS Publ., Boston.

Kolmogorov, A. N. 1957. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. Dokl. Akad. Nauk SSSR 114:953–956.

Kurkova, V. 1992. Kolmogorov theorem and multilayer neural networks. Neural Netw. 5:501–506.

Lampinen, J., and A. Vehtari. 2001. Bayesian approach for neural networks- review and case studies. Neural Netw. 14:257–274.

Long, N. Y., D. Gianola, G. J. M. Rosa, K. A. Weigel, A. Kranis, and O. Gonzalez-Recio. 2010. Radial basis function regression methods for predicting quantitative traits using SNP markers. Genet. Res. 92:209–225.

MacKay, D. J. C. 1992. A practical Bayesian framework for backpropagation networks. Neural Comput. 4:448–472.

MacKay, D. J. C. 1994. Bayesian non-linear modelling for the prediction competition. ASHRAE Trans. 100:1053–1062.

MacKay, D. J. C., and R. M. Neal. 1994. Automatic relevance determination for neural networks, Technical Report. University of Cambridge, MA.

Meuwissen, T. H. E., B. J. Hayes, and M. E. Goddard. 2001. Prediction of total genetic value using genome-wide dense marker maps. Genetics 157:1819–1829.

Minsky, M., and S. Papert. 1972. Perceptrons: An introduction to computational geometry. The MIT Press, Cambridge, MA.

Neal, R. M. 1996. Bayesian learning for neural networks. Vol. 118 of Lecture notes in statistics. Springer-Verlag, Berlin.

Neal, R. M. 2001. Survival analysis using a Bayesian neural network. Join Statistical Meeting, Atlanta, GA.

Nelder, J. A., and R. Mead. 1965. A simplex method for function minimization. Comput. J. 7:308–313.

Nguyen, D., and B. Widrow. 1990. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. IJCNN International Joint Conf. on neural networks 3:21–26.

Okut, H., D. Gianola, G. J. Rosa, and K. A. Weigel. 2011. Prediction of body mass index in mice using dense molecular markers and a regularized neural network. Genet Res. 93:1–13.

Pérez-Rodríguez, P., D. Gianola, J. M. González-Camacho, J. Crossa, Y. Manes, and S. Dreisigacker. 2012. Comparison between linear and non-parametric regression models for genome-enabled prediction in wheat. G3-Genetics 2:1595–1605.

Poggio, T., and F. Girosi. 1990. Networks for approximation and learning. Proc. IEEE 78:1481–1497.

R Core Team. 2012. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Ripley, B. D. 1996. Pattern recognition and neural networks. Cambridge Univ. Press, Cambridge, New York.

Tierney, L., and J. B. Kadane. 1986. Accurate approximations for posterior moments and marginal densities. J. Am. Stat. Assoc. 81(393):82–86.

VanRaden, P. M. 2008. Efficient methods to compute genomic predictions. J. Dairy Sci. 91:4414–4423.

Wellmann, R., and J. Bennewitz. 2012. Bayesian models with dominance effects for genomic evaluation of quantitative traits. Genet Res. 94:21–37.

Wittenburg, D., N. Melzer, and N. Reinsch. 2011. Including non-additive genetic effects in Bayesian methods for the prediction of genetic values based on genome-wide markers. BMC Genet. 12:74–87.

Werbos, P. J. 1974. Beyond regression: New tools for prediction and analysis in the behavioural sciences. Ph.D. thesis, Harvard University, Cambridge, MA.

Zeileis, A., and Y. Croissant. 2010. Extended model formulas in R: Multiple parts and multiple responses. J. Stat. Software 34(1). www.jstatsoft.org