Waldmann *Genet Sel Evol* (2018) 50:70
https://doi.org/10.1186/s12711-018-0439-1

GSE Genetics Selection Evolution

RESEARCH ARTICLE Open Access

Approximate Bayesian neural networks in genomic prediction

Patrik Waldmann*

Abstract

Background: Genome-wide marker data are used both in phenotypic genome-wide association studies (GWAS) and genome-wide prediction (GWP). Typically, such studies include high-dimensional data with thousands to millions of single nucleotide polymorphisms (SNPs) recorded in hundreds to a few thousands individuals. Different machine-learning approaches have been used in GWAS and GWP effectively, but the use of neural networks (NN) and deep-learning is still scarce. This study presents a NN model for genomic SNP data.

Results: We show, using both simulated and real pig data, that regularization is obtained using weight decay and dropout, and results in an approximate Bayesian (ABNN) model that can be used to obtain model averaged posterior predictions. The ABNN model is implemented in mxnet and shown to yield better prediction accuracy than genomic best linear unbiased prediction and Bayesian LASSO. The mean squared error was reduced by at least 6.5% in the simulated data and by at least 1% in the real data. Moreover, by comparing NN of different complexities, our results confirm that a shallow model with one layer, one neuron, one-hot encoding and a linear activation function performs better than more complex models.

Conclusions: The ABNN model provides a computationally efficient approach with good prediction performance and in which the weight components can also provide information on the importance of the SNPs. Hence, ABNN is suitable for both GWP and GWAS.

Background

Transformation of large quantities of data into valuable knowledge has become increasingly important in various fields of genomics and bioinformatics [1]. Machine-learning methods are flexible general-purpose approaches to automatically learn complex relationships and patterns from data, and they play a vital role in the analysis of big data [2–4]. The concept of genome-wide prediction (GWP) was introduced by Meuwissen et al. [5] and refers to the idea that regression coefficients of genomic markers, often single-nucleotide polymorphisms (SNPs), can be used to predict phenotypes of individuals. In order to identify markers that affect some phenotype of interest, state of the art genome-wide marker data comprise several thousands, sometimes millions of SNPs, scored in a number of individuals that is in the order of some hundreds to a few thousands [6]. There are plenty of examples of the successful use of machine-learning in genomic prediction, genome-wide association studies and in other forms of genomic sequence analysis [7, 8].

Among the most flexible methods in machine-learning are deep artificial neural networks, which have recently received large attention because of their outstanding prediction properties [9]. An artificial neural network (NN) connects the inputs (predictor variables) to an output (response variable), either directly or through one or several layers of interconnected computing units (neurons). The depth of an NN corresponds to the number of hidden layers and the width to the number of neurons in its layers. NN with larger numbers of hidden layers are called "deep networks". Training of a NN is accomplished with mathematical optimization algorithms that iteratively perform forward and backward passes (epochs) in order to minimize some loss (error) function and learn the weights (regression coefficients) and biases (intercepts) of the inputs. In the forward pass, the linear or non-linear

*Correspondence: Patrik.Waldmann@slu.se
Department of Animal Breeding and Genetics, Swedish University of Agricultural Sciences (SLU), Box 7023, 750 07 Uppsala, Sweden

activation functions are applied to the current values of the weights of the links to get the output at each layer. The final result of a forward pass is new predicted outputs. The backward pass starts by calculating the derivatives of the error function between the predicted outputs and the real outputs. Then, the derivatives are propagated backwards updating the weights and computing new error terms for that layer. This process is repeated for each layer until the input layer is reached again [10]. The number of epochs and the learning rate determine the amount of training and need to be evaluated against validation or test data in order to avoid over-fitting.

Gradient descent is a popular algorithm that is used to perform mathematical optimization and is one of the most common ways to perform learning in neural networks. The gradient is computed layer-wise using the chain rule for reverse-mode differentiation [11]. The computed gradient indicates by what amount the errors would increase or decrease if the weights are increased by a small amount. Then, the weight vector is adjusted in the opposite direction to the gradient vector, i.e. the negative gradient vector specifies the direction of the steepest descent towards the minimum of the loss function. The gradient descent algorithm is an efficient alternative in NN with many connections, but easily leads to overfitting. Two of the most important methods for regularization in NN are weight decay and dropout [12]. Weight decay is an old technique in which each weight decays towards zero at a rate that is proportional to its magnitude. Weight decay is closely related to ridge regression and can be interpreted as Bayesian Gaussian regularization [13]. Dropout is a recent innovation where a random proportion of the input weights are set to zero in each epoch [14]. Recently, it was shown that dropout can be interpreted as a Bayesian approximation to deep-learning with a near connection to Gaussian processes [15].

In genetics, there are some examples of the use of NN. Gianola et al. [16] introduced the feed-forward NN and showed how it could be interpreted in terms of standard regression models, and classical and molecular genetics. Moreover, they suggested Bayesian ridge regression based regularization to prevent overfitting in the NN and showed that this improved prediction accuracy of traits such as milk production in Jersey cows and yield of inbred lines of wheat compared to normal linear models. The same model improved prediction accuracy of body mass index in mice in an accompanying study [17]. Ehret et al. [18] replaced the computationally demanding Levenberg–Marquardt training algorithm in [16] with back-propagation. Glória et al. [19] showed how two approximate measures of variable contribution and importance could be used to assess marker effects in genomic NN. Deep-learning has also received attention in the related fields of bioinformatics and systems biology [20, 21].

The purpose of this study is to present a NN model for genomic SNP data that can be modified easily. We will also show that regularization is obtained using weight decay and dropout, and that it results in an approximate Bayesian model that can be used to obtain model-averaged posterior predictions. The predictive accuracy of the NN model is evaluated by using both simulated and real data, and we show that weight components can provide information on the importance of SNPs under some circumstances.

Methods

Neural network

The most basic model starts with a single hidden layer NN. Denote $\mathbf{W}_1$ and $\mathbf{W}_2$ the weight matrices that connect the input matrix $\mathbf{X}$ of dimension $n \times p$ to the output $\mathbf{y}$ of dimension $n \times 1$ through the hidden layer. The dimensions of $\mathbf{W}_1$ and $\mathbf{W}_2$ depend on the number of units in the hidden layer. Hence, for a hidden layer with k units $\mathbf{W}_1$ will be of dimension $p \times k$ and $\mathbf{W}_2$ of dimension $k \times 1$. Associated with $\mathbf{W}_1$ and $\mathbf{W}_2$ is a bias vector $\mathbf{b}$ of dimension k and an activation function $\sigma(\cdot)$ that performs element-wise linear or non-linear transformation of the inputs. A standard NN model now becomes:

$$\hat{\mathbf{y}} = \sigma(\mathbf{X}\mathbf{W}_1 + \mathbf{b})\mathbf{W}_2. \tag{1}$$

A model with one hidden layer containing one unit will be of dimension 1×1 and thus, $\mathbf{W}_2$ disappears. For regression purposes, the NN model is usually completed with the Euclidean squared loss function:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2n}\sum_{i=1}^{n} \|y_i - \hat{y}_i\|_2^2, \tag{2}$$

where $||\cdot||_2^2$ denotes the square of the Euclidean norm. For classification, one can use the logistic or cross-entropy loss.

If p is larger than n, training of the NN will lead to overfitting and it is necessary to regularize the NN parameters $\boldsymbol{\theta} = \{\mathbf{W}_1, \mathbf{W}_1, \mathbf{b}\}$. Therefore, regularization terms are added during optimization. One common approach is to add the ℓ_2 penalty [4] through weight decay parameters λ_i, which need to be tuned by cross-validation. The cost function to minimize is now:

$$\text{minimize}\left\{J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n} \|y_i - \hat{y}_i\|_2^2 + \lambda_1\|\mathbf{W}_1\|_2^2 + \lambda_2\|\mathbf{W}_2\|_2^2 + \lambda_3\|\mathbf{b}\|_2^2\right\}. \tag{3}$$

The above single hidden layer NN with the Euclidean loss is identical to a basis function regression model. With $k = 1$ and $\sigma(\cdot)$ chosen to be the linear identity

function, the model is closely related to ridge regression. Extending this simple NN model to multiple layers is straightforward and results in a more expressive deep NN model, but it may result in over-fitting if the relationship between the input and output is linear and lacks structure. Hence, it is important to evaluate NN with different architectures and possibly also different activation functions.

Activation functions and one-hot encoding

The most simple activation function is the linear identity function $\sigma(x) = x$, which has the derivative $\sigma'(x) = 1$ and range $(-\infty, \infty)$, and therefore is well adapted for linear regression purposes. In genetic terms, this function will infer additive effects of bi-allelic loci if the SNPs are coded 0, 1 and 2. In order to model non-linear effects, there are several activation functions that can be used. The tanh function is $\sigma(x) = \left(2/\left(1 + e^{-2x}\right)\right) - 1$ with derivative $\sigma^{'(x)} = 1 - \sigma(x)^2$ and range $(-1, 1)$. The rectifier function is defined as $\sigma(x) = \max(0, x)$ and has several related functions, for example the rectified linear unit (ReLU). However, these non-linear functions may not be so useful for the first layer if the input contains integer coding (e.g. the 0, 1 and 2 of SNP genotypes) and the weights are expected to be in both the positive and negative domains.

An alternative way of inferring non-linearity of integer coded input variables is one-hot encoding [22]. In this approach, indicator variables are formed for each level of the input variable. From a genetic perspective, it means that each genotype will be coded by one $(0, 1)$ variable, and both additive and dominance effects can be modelled straightforwardly.

Optimization algorithms

Gradient descent (GD) is a first order optimization algorithm where the cost function $J(\boldsymbol{\theta})$ is minimized by updating the parameters in the opposite direction of the gradient of the cost function $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. The learning rate η determines the size of the steps towards the minimum. The original gradient descent uses all training data per epoch t:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_{t-1}} J(\boldsymbol{\theta}_{t-1}), \quad \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \mathbf{g}_t, \tag{4}$$

and, thus, can be slow for large datasets and get stuck in local minima. In contrast, stochastic gradient descent (SGD) uses random single training samples that modifies the gradient to:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_{t-1}} J\left(\boldsymbol{\theta}_{t-1}; y_i; \mathbf{x}_i\right), \tag{5}$$

which makes it much faster per epoch, but also results in large fluctuations of the cost function. A compromise between GD and SGD is mini-batch GD (BSGD), where a batch of size b is used for every epoch:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_{t-1}} J\left(\boldsymbol{\theta}_{t-1}; y_{i\ldots i+b}; \mathbf{x}_{i\ldots i+b}\right), \tag{6}$$

which reduces the variance between updates and makes computations efficient. The number b needs to be chosen, usually between 50 and 250 depending on the number of observations in the training data.

Several suggestions on how to improve the convergence properties of BSGD [23] have been reported. Kingma and Ba [24] introduced the adaptive moment estimation (ADAM) algorithm that computes momentum components of the gradients:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t,\ \mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2, \tag{7}$$

where $\mathbf{m}_t$ is the moving average of the gradient and $\mathbf{v}_t$ is the moving average of the squared gradient. β_1 and β_2 are hyper-parameters that control the exponential decay of the moving averages. Furthermore, their bias-corrected estimates are calculated as:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \quad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}, \tag{8}$$

which leads to an update of the parameters with adaptive learning rate:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}, \tag{9}$$

where ϵ controls the effective stepsize. Kingma and Ba [24] suggest the following default hyperparameters, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. ADAM can be combined with weight decay, and then will perform ℓ_2 regularization. This results in parameter update:

$$\boldsymbol{\theta}_t = \left(1 - \frac{\eta\lambda}{b}\right)\boldsymbol{\theta}_{t-1} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}. \tag{10}$$

Dropout and its Bayesian interpretation

Dropout is applied by sampling of binary vectors $\mathbf{z}_1$ and $\mathbf{z}_2$ in each epoch from two Bernoulli distributions, i.e. $\mathbf{z}_1 \sim \text{Bernoulli}(p_1)$ and $\mathbf{z}_2 \sim \text{Bernoulli}(p_2)$, and setting $1 - p_1$ of the inputs and $1 - p_2$ of the outputs to zero. This leads to an extension of Eq. (1) as follows:

$$\hat{\mathbf{y}} = \sigma(\mathbf{X}(\mathbf{z}_1\mathbf{W}_1) + \mathbf{b})(\mathbf{z}_2\mathbf{W}_2). \tag{11}$$

The dropped weights of $\mathbf{z}_1\mathbf{W}_1$ and $\mathbf{z}_2\mathbf{W}_2$ are usually scaled by $1/p_1$ and $1/p_2$, respectively, to maintain constant output magnitude. The surviving nodes have to stand in for those that are omitted, which forms another form of regularization that has been shown to be effective in preventing over-fitting [25]. Dropout can be interpreted in several ways [26, 27]. Gal and Ghahramani [15] showed that dropout is mathematically equivalent to a variational approximation of a Bayesian deep Gaussian

process. The goal of Bayesian prediction is the posterior predictive distribution [28], which for the test input $\mathbf{X}^*$ is:

$$p(\hat{\mathbf{y}}^*|\mathbf{X}^*,\mathbf{X},\mathbf{y}) = \int p(\hat{\mathbf{y}}^*|\mathbf{X}^*,\boldsymbol{\omega})p(\boldsymbol{\omega}|\mathbf{X},\mathbf{y})\mathrm{d}\boldsymbol{\omega}, \tag{12}$$

where $p(\hat{\mathbf{y}}^*|\mathbf{X}^*,\boldsymbol{\omega})$ is the likelihood of the test output (response) and $\boldsymbol{\omega} = \{\mathbf{z}_1\mathbf{W}_1, \mathbf{z}_2\mathbf{W}_2, \mathbf{b}\}$. The posterior distribution of the training data $p(\boldsymbol{\omega}|\mathbf{X},\mathbf{y})$ is usually analytically intractable, but the variational distribution $q(\boldsymbol{\omega})$ can be defined as:

$$\mathbf{W}_i = \mathbf{M}_i \cdot \mathrm{diag}\left(\left[z_{i,j}\right]_{j=1}^{k}\right),$$
$$z_{i,j} \sim \mathrm{Bernoulli}(p_i),$$

where $\{i = 1, 2\}$, $\{j = 1, \ldots, k_{i-1}\}$, $\mathbf{M}_i$ is a random variational matrix and p_i are the dropout probabilities. p_1 and p_2 were set to 0.5 for all NN configurations in this study.

Then, the idea behind variational inference is to minimize the Kullback–Leibler (KL) divergence between $q(\boldsymbol{\omega})$ and $p(\boldsymbol{\omega}|\mathbf{X},\mathbf{y})$ through maximization of the log evidence lower bound:

$$\mathcal{L}_{VI} = \int q(\boldsymbol{\omega})\log p(\boldsymbol{\omega}|\mathbf{X},\mathbf{y})\mathrm{d}\boldsymbol{\omega} - \mathrm{KL}(q(\boldsymbol{\omega})p(\boldsymbol{\omega})), \tag{13}$$

which results in the approximate predictive distribution:

$$q(\hat{\mathbf{y}}^*|\mathbf{X}^*) = \int p(\hat{\mathbf{y}}^*|\mathbf{X}^*,\boldsymbol{\omega})q(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}. \tag{14}$$

Note that the loss function is equal to the negative log-likelihood, i.e. $\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) = -\log p(\mathbf{y}|\mathbf{X},\boldsymbol{\omega})$, and that the dropout model can be interpreted as Bayesian ridge regression with a spike-and-slab g-prior [29]. Sampling from Eq. (13) is straightforward. Start by sampling T sets of vectors $\left\{z_{i,j}^t\right\}_{t=1}^{T} \sim \mathrm{Bernoulli}(p_i)$, combine with $\mathbf{W}_i$ to obtain $\left\{\mathbf{W}_i^t\right\}_{t=1}^{T}$ and perform one gradient descent optimization per t with some predefined values of the weight decay. ADAM will automatically adapt the learning rate. Iterate until T to get predictions $\left\{\hat{\mathbf{y}}^{*t}\right\}_{t=1}^{T}$ and calculate MSE_t for each iteration:

$$\mathrm{MSE}_t = \frac{1}{ntest}\sum_{1}^{ntest}\left(\hat{\mathbf{y}}^{*t} - \mathbf{y}_{test}\right)^2. \tag{15}$$

Plot the MSE_t against the iteration number to determine at which iteration t_s the chain has converged. The first moments (expectations) of the parameters $\boldsymbol{\omega}$ are approximated as:

$$\mathbb{E}[\boldsymbol{\omega}] \approx \frac{1}{T - t_s}\sum_{t=t_s+1}^{T}\boldsymbol{\omega}^t, \tag{16}$$

whereas the predictive variances for the parameters are:

$$\mathrm{VAR}[\boldsymbol{\omega}] \approx \frac{1}{T - t_s}\sum_{t=t_s+1}^{T}\left(\boldsymbol{\omega}^t - \mathbb{E}\left[\boldsymbol{\omega}^t\right]\right)^2. \tag{17}$$

Moreover, the model's averaged MSE can be calculated based on the MSE from each t of the stationary part of the chain yielding:

$$\mathrm{MSE}_{\mathcal{M}} = \frac{1}{T - t_s}\sum_{t=t_s+1}^{T}\frac{1}{ntest}\sum_{1}^{ntest}\left(\hat{\mathbf{y}}^{*t} - \mathbf{y}_{test}\right)^2. \tag{18}$$

Data

Simulated data with dominance

The original data was produced for the QTLMAS2010 workshop and intended to mimic a real breeding livestock population [30]. The number of individuals is 3226, and these are structured in a pedigree with five generations. The pedigree is founded by 20 individuals (5 males and 15 females), and it was created assuming that each female mates once and gives birth to approximately 30 progeny. Five 100 Mbp long autosomal chromosomes were simulated. A neutral coalescent model was used to simulate the SNP data. The algorithm created 10,031 markers, including 263 monomorphic and 9768 biallelic SNPs. The mean LD (r^2) between adjacent SNPs with a minor allele frequency (MAF) higher than 0.05 is estimated at 0.100 (SD = 0.152).

The continuous quantitative trait was created from 37 quantitative trait loci (QTL), including nine controlled genes and 28 random genes. The QTL were modelled as additive effects, apart from two pairs of additive epistatic QTL and three paternal imprinting QTL. The controlled genes were selected based on their high polymorphism as well as their high linkage disequilibrium (LD) with markers. The additive effects of all controlled QTL were equal to +3 (i.e. half the difference between the means of the homozygotes). The random genes were selected from the simulated SNPs and then their effects were sampled from a truncated normal distribution, $N(0, 10)$, and accepted if the absolute value of the additive effect was less than 2. The resulting additive effects of the random genes varied between −1.98 and 1.93. The two epistatic pairs of QTL are on chromosomes 1 and 2, respectively, and determined by four controlled additive QTL with an additional epistatic effect of 4 for the lower homozygote pairs. Each simulated QTL was surrounded by 19 to 47 polymorphic SNPs (MAF > 0.05) positioned within a 1-Mb distance from the QTL. A total of 364 SNPs were in moderate to high LD with the QTL ($r^2 > 0.1$).

Furthermore, one dominance locus was positioned at SNP 9212 on chromosome 5 by giving the heterozygote (1) an effect of 5, and the upper homozygote (2) a value of 5.01 (for numerical reasons). One over-dominance locus was produced at SNP 9404 by assigning the heterozygote an effect of 5, the lower homozygote (0) an effect of −0.01, and the upper homozygote (2) an effect of 0.01. Finally, one under-dominance loci was created at SNP 9602 by assigning a value of −5 to the heterozygote, and giving the lower homozygote (0) an effect of −0.01 and the upper homozygote (2) an effect of 0.01. The values of the genotypes of these new SNPs were added to the original **y**-values. SNPs with a MAF lower than 0.01 were removed resulting in a final sample of 9723 SNPs.

Real data

Cleveland et al. [31] published a pig dataset comprising 3534 individuals with high-density genotypes, phenotypes, and estimated breeding values for five anonymous traits. Genotypes were obtained with the PorcineSNP60 chip, and after quality control, 52,842 SNPs remained. Missing genotypes were imputed using a probability score. SNPs with both known and unknown positions were included and imputed. The map order was randomized and the SNP identities were recoded. The number of SNPs was further reduced in this study by a more stringent MAF (< 0.01), which produced a final number of 50,276 SNPs.

Most of the genotyped animals were measured for all five purebred traits (phenotypes in a single nucleus line). Heritabilities ranged from 0.07 to 0.62. In this study, trait 3 with a heritability of 0.38 was used. The phenotypic data points were adjusted for environmental factors and rescaled by correcting for the overall mean. Individuals with missing phenotype data were removed which, at the end, resulted in 3141 observations.

Implementation

All NN models were implemented in the Python version of MXNet [32] using the ADAM optimizer with default settings. The Python code is available at https://github.com/patwa67/ABNN. Predictions were also obtained for Bayesian LASSO (BLASSO) and genomic best linear unbiased prediction (GBLUP) (using default settings in the R-package BGLR; [33]). For the simulated QTLMAS2010 datasets, individuals in generations 1 to 4 (2326 individuals) were used as training data and in generation 5 (900 individuals) as test data. Individuals in the real pig data were divided into different cross-validation (CV) sets with test datasets of sizes between 627 and 631. The MSE was averaged over these CV sets.

Results

Simulated data

The Monte Carlo Markov chains of the GBLUP and BLASSO analyses were run for 60,000 iterations, and a burn-in of 10,000 and thinning of 10 resulted in a final sample of 5000 iterations. The resulting testing set MSE was 88.42 and 89.22 for the GBLUP and BLASSO, respectively (Table 1). Initially, two NN were tested in which the first one was designed to have one hidden layer and one node, and the second to have two hidden layers with two and one node, respectively. The weight decay were varied between 1.0 and 1.5 for both NN. The ABNN analyses were run for 6000 iterations and the first 1000 were considered as burn-in. Overall, the model averaged test MSE and its standard deviation were smaller for the first NN than for the second NN (Table 1). This NN also produced the smallest test MSE with an estimate of 82.69 for a weight decay of 1.4.

The MSE quickly converged to a stationary phase for this NN (Fig. 1). One larger NN with three layers (3, 2 and 1 nodes, respectively) was run only for a weight decay of 1.4. The resulting test MSE was equal to 93.55 with a standard deviation (sd) of 1.424. In addition, the best model was evaluated with tanh and relu activation functions. The resulting test MSE were equal to 85.68 (sd = 0.166) and 83.72 (sd = 0.248), respectively.

In order to investigate the effect of possible outliers in the test predictions, we also calculated the model

Table 1 Test set MSE for GBLUP, BLASSO and ABNN evaluated on the simulated QTLMAS2010 data

GBLUP	88.42					
BLASSO	89.22					
ABNN	Weight decay λ_1					
	1.0	1.1	1.2	1.3	1.4	1.5
# units $k = 1$						
$MSE_{\mathcal{M}}$ (SD)	83.64 (0.272)	83.26 (0.216)	83.27 (0.243)	83.50 (0.256)	*82.69* (0.218)	83.51 (0.256)
# units $k = 2.1$						
$MSE_{\mathcal{M}}$ (SD)	88.40 (0.940)	87.94 (0.733)	89.31 (1.167)	88.12 (0.840)	87.42 (0.714)	88.01 (0.727)

Two architectures were evaluated for the ABNN were *k* refers to the number of units per hidden layer. $MSE_{\mathcal{M}}$ is the model-averaged MSE and SD is the standard deviation over iterations excluding burn-in. The best model MSE is indicated in italic characters

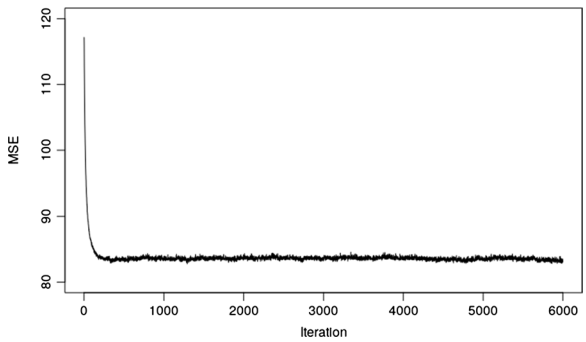

Fig. 1 Trace plot of test MSE for the best model (one-neuron, one-layer and linear activation function) on the simulated QTLMAS2010 data

averaged mean absolute error (MAE) for the most important models. MAE was equal to 7.640 and 7.635 for the BLASSO and GBLUP, respectively. The smallest MAE was 7.407 for the small ABNN model with a weight decay of 1.5. The MAE for the larger ABNN model was 7.675, which indicates overall the same pattern as for MSE.

The model's averaged weights $\mathbb{E}[\mathbf{W}_1]$ were calculated for the best model. The additive genetic effects were then obtained from the two homozygotes as $a = -\mathbb{E}[\mathbf{W}_{1,\text{Hom0}}] + \mathbb{E}[\mathbf{W}_{1,\text{Hom2}}]$ and are plotted in Fig. 2. It can be seen that the ABNN detects the two major additive loci on chromosome 3 (SNPs 4354 and 5327) and the additive part of the epistatic effect on chromosome 1 (SNP 931) as well as the additive part of the dominance locus (SNP 9212). Then, the dominance genetic effects were obtained from the heterozygote as $d = \mathbb{E}[\mathbf{W}_{1,\text{Het1}}]$ and are plotted in Fig. 3. It is clear that dominance, over-dominance and under-dominance loci were identified at SNP positions 9212, 9404 and 9602, respectively.

Real data

The analyses of the real data were run with the same number of iterations as for the simulated data. For the GBLUP and BLASSO, the MSE estimates were equal to 0.8759 and 0.8741, respectively (Table 2). In the ABNN analyses, weight decays were optimized for values between 21 and 25 based on the two smallest NN. Again, the model averaged MSE and its standard deviation were overall smaller for the first NN than for the second NN, with MSE estimates of 0.8653 and 0.9221, respectively, for a weight decay of 23 (Table 2). For the third NN, the test MSE was equal to 0.9236 (sd = 0.00447). The analyses of the best model with tanh and relu activation functions resulted in test MSE of 0.894 (sd = 0.000383) and 0.867 (sd = 0.000781), respectively.

MAE of 0.6863 and 0.6865 were obtained for the BLASSO and GBLUP models, respectively. The smallest MAE was 0.6811 for the small ABNN model with a weight decay set to 22. The second ABNN model yielded a MAE of 0.7153 for a weight decay of 22. Hence, the pattern of MAE is very similar to the MSE pattern for both datasets.

The model-averaged weights were extracted from the best model. Based on these, additive and dominance genetic effects were calculated and plotted (Figs. 4, 5), which shows that six SNPs have quite high dominance effects. The same SNPs were found to be important in an earlier study [34].

Discussion

Alternative designs to fully-connected feed-forward NN differ in the way neurons are arranged and activated, and the architecture needs to be tailored to

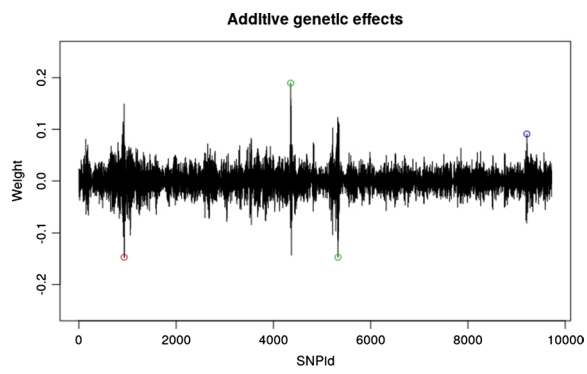

Fig. 2 Mode-averaged weight plots for the additive genetic effects for the best model (one-neuron, one-layer and linear activation function) on the simulated QTLMAS2010 data. The two major additive loci are SNP 4354 and 5327 (red circles) and the additive part of the epistatic effect at SNP 931 is indicated by a red circle) and the additive part of the dominance locus at SNP 9212 by a blue circle

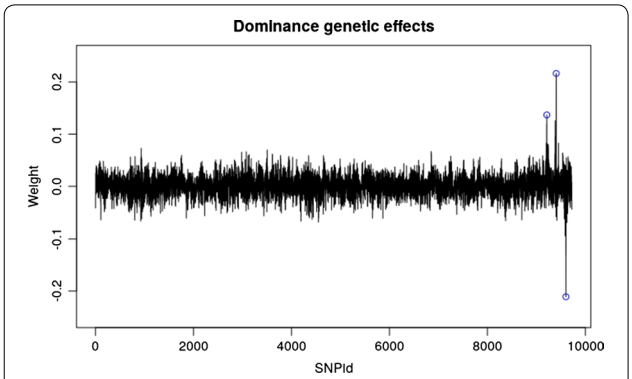

Fig. 3 Model-averaged weight plots for the dominance genetic effects for the best model (one-neuron, one-layer and linear activation function) on the simulated QTLMAS2010 data. The dominance, over-dominance and under-dominance loci were identified at SNPs 9212, 9404 and 9602, respectively (blue circles)

Table 2 Test set MSE for GBLUP, BLASSO and ABNN evaluated on the real Cleveland pig dataset

GBLUP	0.8759				
BLASSO	0.8741				
ABNN	Weight decay λ_1				
	21	22	23	24	25
# units $k=1$					
$MSE_{\mathcal{M}}$ (SD)	0.8688 (0.000796)	0.8675 (0.000790)	*0.8653* (0.000722)	0.8676 (0.000741)	0.8687 (0.000728)
# units $k=2.1$					
$MSE_{\mathcal{M}}$ (SD)	0.9230 (0.00432)	0.9233 (0.00440)	0.9221 (0.00399)	0.9233 (0.00430)	0.9235 (0.00439)

Two architectures were evaluated for the ABNN were k refers the number of units per hidden layer. $MSE_{\mathcal{M}}$ is the model-averaged MSE and SD is the standard deviation over iterations excluding burn-in. The best model MSE is indicated in italic characters

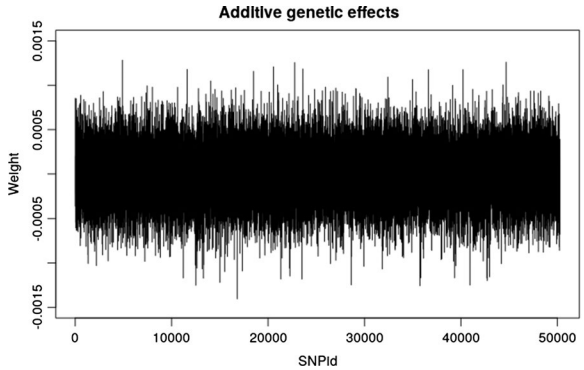

Fig. 4 Model-averaged weight plots for the additive genetic effects for the best model (one-neuron, one-layer and linear activation function) on the Cleveland pig dataset

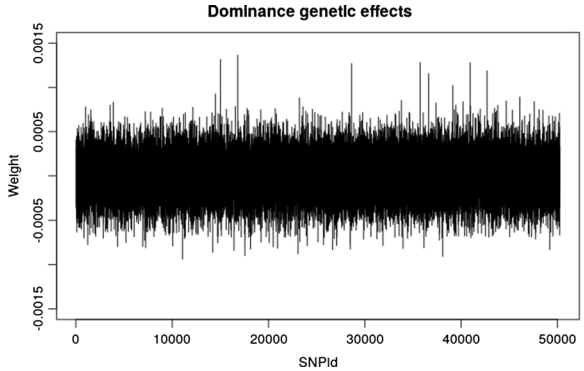

Fig. 5 Model-averaged weight plots for the dominance genetic effects for the best model (one-neuron, one-layer and linear activation function) on the Cleveland pig dataset

specific applications. Shallow NN have few layers and neurons, whereas deep NN consist of many layers, often with a large number of neurons, connected in various patterns. A one-layer and one-neuron NN with a linear activation function is equivalent to a standard multiple regression model. Many different architectures have been developed for deep-learning, the most important being convolutional neural networks (CNN), which are widely used for modelling images [35], and recurrent neural networks (RNN) for sequential data [36]. Restricted Boltzmann machines and autoencoders have been developed for unsupervised learning [37]. The number of software packages for deep-learning and computational sophistication have increased in recent years. However, application of NN in genetics is still relatively scarce, but tends to increase [8].

Here, we showed that regularization of genome-wide data can be obtained by a combination of weight decay and dropout in NN, and that this provides an efficient method, which can be used both for GWP and GWAS. Extensions to more complex NN resulted in overfitting and worse prediction accuracy on both simulated and real data. Hence, one can argue that it is more important to focus on efficient regularization and sparsity than on modelling of complex structures when genomic data consists of SNPs. Glória et al. [19] also found that more complex NN designs reduced the predictive accuracy compared to a simple one-layer one-node net when evaluated on simulated genotype/phenotype data.

A recommended strategy for optimization of the NN structure and the associated weight decays is to start with a simple model with one layer and one node, and monitor the MSE over a range of weight decays. Then, one can increase the complexity of the NN and evaluate if the test MSE decreases. Of course, cross-validation or some other form of test data is needed to obtain the minimum test MSE. The dropout probability can be set to a value between 0 and 1, but it turned out that the best result in terms of minimum test MSE was always obtained when setting this parameter to 0.5. Although, one should be aware that one of the dimension reduction properties of a NN structure is that once a variable from a layer is dropped, all the terms that are above it in the network also disappear [29]. Initially, it is important

to evaluate values of the weight decay and the dropout using a broad range of parameter values and then consecutively shorten down the interval.

A positive outcome of our results is that the weights can be interpreted as regression coefficients and therefore will be useful for the identification of important SNPs which is the main goal of GWAS. The one-hot encoding means that both additive and dominance effects can be detected as illustrated by the figures of both datasets. It is much more difficult to interpret weights for complex NN that include non-linear activation functions, pooling and feedback. Hence, the usefulness of deep-learning for GWAS is limited, although some techniques exist for variable importance analysis e.g. [19, 38].

Although SNPs are positioned on chromosomes, it is often sufficient to assume that the SNPs behave as independent data units due to recombination. However, it should be pointed out that the structures of SNP chip data and DNA sequence data differ. Some recent studies have tried to account for the structure at the DNA level in various prediction settings. Alipanahi et al. [39] introduced the DeepBind model based on deep CNN for the detection of protein binding sites in DNA sequences. The DeepBind model was shown to outperform other methods, to recover known and novel sequence motifs, and quantify the result of sequence alterations and detect functional single-nucleotide variants (SNVs). Quang and Xie [40] proposed DanQ, a novel hybrid convolutional and bi-directional long short-term memory recurrent (LSTM) neural network framework for predicting non-coding function de novo from DNA sequences. The idea behind DanQ is that the convolution layer captures regulatory motifs, while the recurrent layer takes care of long-term dependencies between the motifs. DanQ was shown to have outstanding prediction properties. It is likely that both CNN and RNN will become more important in the near future for GWP as sequence data becomes more abundant, but the computational demands for whole-genome analyses of large samples of individuals will be huge.

In future studies, it would be interesting to combine the ABNN with convolutional and recurrent structures in order to incorporate possible LD. Another option that would be worth testing would be to replace weight decay (i.e. ridge regularization) with the ℓ_1 penalty (i.e. lasso regularization) of the weight parameters. Further studies on other datasets are also needed before general conclusions can be drawn.

Conclusions

This study shows how the drop-out technique can be applied to neural networks and result in an approximate Bayesian (ABNN) model that provides a computationally efficient approach with good prediction performance. ABNN is suitable for prediction of unknown phenotypes using large-scale genome-wide SNP data, and as a tool for the detection of the SNPs that contribute information to the prediction when simple linear NN are favored. Our results show that, compared with the GBLUP and BLASSO methods on simulated data and real pig data, ABNN has lower prediction error and that a simple one-neuron one-layer network is preferred over deeper and more complex structures.

Authors' contributions

The author read and approved the final manuscript.

Acknowledgements

Financial support was provided by the Beijer laboratory for animal science, SLU, Uppsala.

Competing interests

The author declares that he has no competing interests.

Publisher's Note

Received: 6 August 2018 Accepted: 16 December 2018
Published online: 22 December 2018

References

1. Fan J, Han F, Liu H. Challenges of big data analysis. Nat Sci Rev. 2014;1:293–314.
2. Ghahramani Z. Probabilistic machine learning and artificial intelligence. Nature. 2015;521:452–9.
3. Jordan MI, Mitchell TM. Machine learning: trends, perspectives, and prospects. Science. 2015;349:255–60.
4. Theodoridis S. Machine learning: a Bayesian and optimization perspective. 1st ed. London: Academic Press; 2015.
5. Meuwissen THE, Hayes BJ, Goddard ME. Prediction of total genetic value using genome-wide dense marker maps. Genetics. 2001;157:1819–29.
6. de Los Campos G, Hickey JM, Pong-Wong R, Daetwyler HD, Calus MPL. Whole genome regression and prediction methods applied to plant and animal breeding. Genetics. 2013;2:327–45.
7. Okser S, Pahikkala T, Airola A, Salakoski T, Ripatti S, Aittokallio T. Regularized machine learning in the genetic prediction of complex traits. PLoS Genet. 2014;10:e1004754.
8. Libbrecht MW, Noble WS. Machine learning applications in genetics and genomics. Nat Rev Genet. 2015;16:321–32.
9. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521:436–44.
10. Goodfellow I, Bengio Y, Courville A. Deep learning. Cambridge: MIT Press; 2016.
11. Baydin AG, Pearlmutter BA, Radul AA, Siskind JM. Automatic differentiation in machine learning: a survey. J Mach Learn Res. 2018;18:1–43.
12. Schmidhuber J. Deep learning in neural networks: an overview. Neural Netw. 2015;61:85–117.
13. Nowlan SJ, Hinton GE. Simplifying neural networks by soft weight-sharing. Neural Comput. 1992;4:473–93.

14. Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov R. Improving neural networks by preventing co-adaptation of feature detectors. 2012. https://arxiv.org/pdf/1207.0580.
15. Gal Y, Ghahramani Z. Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In: Proceedings of the 33rd international conference on machine learning: 19–24 June 2016; New York; 2016.
16. Gianola D, Okut H, Weigel KA, Rosa GJM. Predicting complex quantitative traits with Bayesian neural networks: a case study with Jersey cows and wheat. BMC Genet. 2011;12:87.
17. Okut H, Gianola D, Rosa GJM, Weigel KA. Prediction of body mass index in mice using dense molecular markers and a regularized neural network. Genet Res (Camb). 2011;93:189–201.
18. Ehret A, Hochstuhl D, Gianola D, Thaller G. Application of neural networks with back-propagation to genome-enabled prediction of complex traits in Holstein-Friesian and German Fleckvieh cattle. Genet Sel Evol. 2015;47:22.
19. Glória LS, Cruz CD, Vieira RAM, de Resende MDV, Lopes PS, de Siqueira OHGB, et al. Accessing marker effects and heritability estimates from genome prediction by Bayesian regularized neural networks. Livest Sci. 2016;191:91–6.
20. Min S, Lee B, Yoon S. Deep learning in bioinformatics. Brief Bioinform. 2017;18:851–69.
21. Angermueller C, Pärnamaa T, Parts L, Stegle O. Deep learning for computational biology. Mol Syst Biol. 2016;12:878.
22. Potdar K, Pardawala TS, Pai CD. A comparative study of categorical variable encoding techniques for neural network classifiers. Int J Comp Appl. 2017;175:7–9.
23. Ruder S. An overview of gradient descent optimization algorithms. 2017. https://arxiv.org/pdf/1609.04747.pdf.
24. Kingma DP, Ba JL. ADAM: a method for stochastic optimization. 2015. https://arxiv.org/pdf/1412.6980.pdf.
25. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15:1929–58.
26. Baldi P, Sadowski P. The dropout learning algorithm. Artif Intell. 2014;210:78–122.
27. Helmbold DP, Long PM. Surprising properties of dropout in deep networks. Proc Mach Learn Res. 2017;65:1–24.
28. Gelman A, Meng XL, Stern H. Posterior predictive assessment of model fitness via realized discrepancies. Stat Sin. 1996;6:733–60.
29. Polson NG, Sokolov V. Deep learning: a Bayesian perspective. Bayesian Anal. 2017;12:1275–304.
30. Szydłowski M, Paczyńska P. QTLMAS 2010: simulated dataset. BMC Proc. 2011;5:S3.
31. Cleveland MA, Hickey JM, Forni S. A common dataset for genomic analysis of livestock populations. G3 (Bethesda). 2012;2:429–35.
32. Chen T, Li M, Li Y, Lin M, Wang N, Wang M. MXNet: a flexible and efficient library for deep learning. 2017. https://mxnet.incubator.apache.org/.
33. de los Campos G, Pérez P, Vazquez AI, Crossa J. Genome-enabled prediction using the BLR (Bayesian Linear Regression) R-package. Methods Mol Biol. 2013;1019:299–320.
34. Waldmann P. Genome-wide prediction using Bayesian additive regression trees. Genet Sel Evol. 2016;48:42.
35. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, et al. Handwritten digit recognition with a back-propagation network. In: Proceedings of the neural information processing systems conference 1989: 27–30 November 1989; Denver. 1990;396–404.
36. Williams RJ, Zipser D. A learning algorithm for continually running fully recurrent neural networks. Neural Comput. 1989;1:270–80.
37. Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science. 2006;313:504–7.
38. de Oña J, Garrido C. Extracting the contribution of independent variables in neural network models: a new approach to handle instability. Neural Comput Appl. 2014;25:859–69.
39. Alipanahi B, Delong A, Weirauch MT, Frey BJ. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nat Biotechnol. 2015;33:831–8.
40. Quang D, Xie X. DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. Nucl Acid Res. 2016;44:e107.