Deep Bayesian regression models

Aliaksandr Hubin *
Geir Storvik
Department of Mathematics, University of Oslo
and
Florian Frommlet
CEMSIIS, Medical University of Vienna

February 16, 2018

Abstract

Regression models are used for inference and prediction in a wide range of applications providing a powerful scientific tool for the researchers and analysts coming from different fields. In most of these fields more and more sources of data are becoming available introducing a variety of hypothetical explanatory variables for these models to be considered. Model averaging induced by different combinations of these variables becomes extremely important for both good inference and prediction. Not less important, however, seems to be the quality of the set of explanatory variables to select from. It is often the case that linear relations between the explanatory variables and the response are not sufficient for the high quality inference or predictions. Introducing non-linearities and complex functional interactions based on the original explanatory variables can often significantly improve both predictive and inferential performance of the models. The non-linearities can be handled by deep learning models. These models, however, are often very difficult to specify and tune. Additionally they can often experience overfitting issues. Latent Gaussian variables are also not incorporated in the existing deep learning approaches. In this paper we introduce a class of deep Bayesian regression models with latent Gaussian variables generalizing the classes of GLM, GLMM, ANN, CART, logic regressions and fractional polynomials into a powerful and flexible Bayesian framework. We then suggest algorithmic approaches for fitting them. In the experimental section we test some computational properties of the algorithm and show how deep Bayesian regression models can be used for inference and predictions in various applications.

Keywords: Bayesian deep learning; Deep feature engineering; Combinatorial optimization; Uncertainty in deep learning; Bayesian model averaging; Semi-parametric statistics; Automatic neural network configuration; Genetic algorithm; Monte Carlo Markov chains.

*The authors gratefully acknowledge *CELS project at the University of Oslo* for giving us the opportunity, inspiration and motivation to write this article.

1 Introduction

Regression models are an indispensible tool for answering scientific questions in almost all research areas. Traditionally scientists have been trained to be extremely careful in specifying adequate models and to include not too many explanatory variables. The orthodox statistical approach warns against blindly collecting data of too many variables and relying on automatic procedures to detect the important ones (see for example Burnham and Anderson, 2002). Instead, expert based knowledge of the field should guide the model building process such that only a moderate number of models are compared to answer specific research questions.

In contrast, modern technologies have lead to the entirely different paradigm of machine learning where routinely extremely large sets of input explanatory variables - the so called features - are considered. Recently deep learning procedures have become quite popular and highly successful in a variety of real world applications (Goodfellow et al., 2016). These algorithms apply iteratively some nonlinear transformations aiming at optimal prediction of response variables from the outer layer features. Each transformation yields another hidden layer of features which are also called neurons. The architecture of a deep neural network then includes the specification of the nonlinear intra-layer transformations (**activation functions**), the number of layers (**depth**), the number of features at each layer (**width**) and the connections between the neurons (**weights**). The resulting model is trained by means of some optimization procedure (e.g. stochastic gradient search) with respect to its parameters in order to fit a particular objective (like minimization of RMSE, or maximization of the likelihood, etc.).

Surprisingly it is often the case that such procedures easily outperform traditional statistical models, even when these were carefully designed and reflect expert knowledge (Refenes et al., 1994; Razi and Athappilly, 2005; Adya and Collopy, 1998; Sargent, 2001; Kanter and Veeramachaneni, 2015). Apparently the main reason for this is that the features from the outer layer of the deep neural networks become highly predictive after being processed through the numerous optimized nonlinear transformations. Specific regularization techniques have been developed for deep learning procedures to avoid overfitting of training data sets, however success of the latter is not obvious. Normally one has to use huge data sets to be able to produce generalizable neural networks.

The universal approximation theorems (Cybenko, 1989; Hornik, 1991) prove that all neural networks with sigmoidal activation functions (generalized to the class of monotonous bounded functions in Hornik, 1991) with at least one hidden layer can approximate any function of interest defined on a closed domain in $\mathbb{R}^n$. Successful applications typically involve huge datasets where such nonparametric methods can be efficiently applied. One

drawback of deep learning procedures is that, due to their complex nature, such models and their resulting parameters are difficult to interpret. Depending on the context this can be a more or less severe limitation. These models are densely approximating the function of interest and transparency is not a goal in the traditional applications of deep learning. However, in many research areas it might be desirable to obtain interpretable (nonlinear) regression models rather than just dense representation of them. Another problem is that fitting deep neural networks is very challenging due to the huge number of parameters involved and the non-concavity of the likelihood function. As a consequence optimization procedures often yield only local optima as parameter estimates.

This paper introduces a novel approach which combines the key ideas of deep neural networks and some other popular statistical learning approaches with Bayesian regression resulting in a flexible and broad framework that we call deep Bayesian regression. Compared to deep neural networks we do not have to prespecify the architecture but our deep regression model can adaptively learn the number of layers, the number of features within each layer and the activation functions. In a Bayesian model based approach potential overfitting is avoided through appropriate priors which strongly penalize engineered features from deeper layers. Furthermore deep Bayesian regression allows to incorporate correlation structures via latent Gaussian variables, which seems to be rather difficult to achieve within traditional deep neural networks.

Fitting of the deep Bayesian regression model is based on a Markov chain Monte Carlo (MCMC) algorithm for Bayesian model selection which is embedded in a genetic algorithm for feature engineering. A similar algorithm was previously introduced in the context of logic regression (Hubin et al., 2017). We further develop a reversible version of this genetic algorithm to obtain a proper Metropolis-Hastings algorithm. Efficient implementations of these algorithms to deal with the high dimensionality of the problem are based on adaptive proposals, which rely on marginal inclusion probabilities (Hubin and Storvik, 2016a). The latter allows to save a lot of computational effort (Roberts and Rosenthal, 2009) and leads to the algorithm's convergence to interpretable results within reasonable time in practice.

We will demonstrate that automatic feature engineering within regression models combined with Bayesian model averaging can improve predictive abilities of statistical models whilst keeping them reasonably simple, interpretable and transparent. We illustrate the potential of our approach to find meaningful non-linear models and infer on parameters of interest. As an example we will retrieve several ground physical laws from raw data. The predictive ability of deep Bayesian regression is compared with deep neural networks, CARTs, elastic networks, random forests, and other statistical learning

techniques under various scenarios.

The rest of the paper is organized as follows. The class of deep Bayesian regression models (DBRM) is mathematically defined in Section 2. In Section 3 we describe two algorithms for fitting DBRM, namely the genetically modified MJMCMC (GMJMCMC) and its reversible version (RGMJMCMC). In Section 4 these algorithms are applied to several real data sets. The first examples have the specific goal of retrieving an interpretable model, later examples are aiming at prediction where the performance of our approach is compared with various competing statistical learning algorithms. In the final Section 5 some conclusions and suggestions for further research are given. Additional pseudo-codes and proofs can be found in the Appendix.

2 The deep Bayesian regression model - DBRM

We model the relationship between m features and a response variable based on n samples from a training data set. Let Y_i denote the response data and $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{im})$ the m-dimensional vector of input covariates for $i \in \{1, ..., n\}$. The proposed model is within the framework of a generalized linear model with latent Gaussian variables, but extended to include a flexible class of non-linear transformations of the covariates (features) described in Section 2.1. This class includes a finite (though huge) number q of features, which can in principle be enumerated as $F_j(\boldsymbol{x}_i)$, $j \in \{1, ..., q\}$. With this notation we define the deep Bayesian regression model (DBRM), including (potentially) up to q features and r independent Gaussian latent variables $\delta_{i1}, ..., \delta_{ir}$:

$$Y_i|\mu_i \sim \mathfrak{f}(y|\mu_i;\phi), \quad i \in \{1, ..., n\} \tag{1}$$

$$\mathsf{h}(\mu_i) = \beta_0 + \sum_{j=1}^{q} \gamma_j \beta_j F_j(\boldsymbol{x}_i) + \sum_{k=1}^{r} \lambda_k \delta_{ik} \tag{2}$$

$$\boldsymbol{\delta_k} = (\delta_{1k}, ..., \delta_{nk}) \sim N_n(\mathbf{0}, \boldsymbol{\Sigma}_k). \tag{3}$$

Here $\mathfrak{f}(\cdot|\mu, \phi)$ is the density (mass) of a probability distribution from the exponential family with expectation μ and dispersion parameter ϕ, while $\mathsf{h}(\cdot)$ is a link function relating the mean to the underlying covariates (McCullagh and Nelder, 1989). The features can enter through an additive structure with coefficients $\beta_j \in \mathbb{R}, j \in \{1, ..., q\}$. In our formulation of DBRM equation (2) includes all possible $q+r$ components (that is features and latent Gaussian variables), using binary variables γ_j and λ_k indicating whether the corresponding variables are to be actually included into the model or not. The latent Gaussian variables with covariance matrices $\boldsymbol{\Sigma}_k$ allow to describe different correlation structures between individual observations. Our formulation is rather general and could

include for example covariance matrices obtained from classical random effects or from autoregressive models. The matrices are typically parametrized through a few parameters, so that in practice $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}_k(\boldsymbol{\psi}_k)$. Priors for the different parameters of the model are specified in Section 2.2.

2.1 Topology of the feature space

The feature space is constructed through a hierarchical procedure similar to the deep learning approach (LeCun et al., 2015; Goodfellow et al., 2016), but allowing for automatic construction of the architecture (whilst in deep neural networks the architecture has to be set in advance (LeCun et al., 2015)). We can also obtain posterior distribution of different architectures within the suggested approach.

Deep neural networks typically use one or several pre-specified activation functions to compute hidden layer values, where currently the rectified linear unit $\text{ReLU}(x) = \max\{0, x\}$ is the most popular. Configuration of where these functions are applied is however fixed. In contrast, in our approach the activation function g can be dynamically selected from a pre-specified **set** of non-linear functions $\mathcal{G}$, which could include, apart from the rectified linear unit, several other transformations, like for example $\exp(x)$, $\tanh(x)$, $\text{atan}(x)$, $\text{erf}(x)$, $\text{sigmoid}(x)$ and other functions adjustable to the particular application.

Let $\mathcal{G}$ denote a set of l non-linear functions $g_1(x), ..., g_l(x)$. Define the **depth** of the feature as the maximal number of $g(x) \in \mathcal{G}$ applied within it recursively. For example a feature $\sin(\cos(\log(x)) + \exp(x))$ has the depth of 3. Denote the space of features of depth d as $\mathcal{F}_d$, which will be of size q_d. We consider a maximal depth of D_{max}, which ensures a finite feature space of size $q = \sum_{d=0}^{D_{max}} q_d$. Define the vector $\boldsymbol{F}^d(\boldsymbol{x})$ by all features of maximal depth up to d. The inner layer of features consists of the original covariates themselves, $\mathcal{F}_0 := \{x_1, \ldots, x_m\}$, where we drop the index i for notational convenience. Then $q_0 = m$ and $\boldsymbol{F}^0(\boldsymbol{x}) = \boldsymbol{x} = (x_1, \ldots, x_m)$. A new feature $F_j(\boldsymbol{x}) \in \mathcal{F}_{d+1}$ is obtained by applying some non-linear transformation g on an affine transformation of $\boldsymbol{F}^d(\boldsymbol{x})$:

$$F_j(\boldsymbol{x}) = g(\boldsymbol{\alpha}^T \boldsymbol{F}^d(\boldsymbol{x}) + \alpha_0) \; . \tag{4}$$ eq:new_features

Equation (4) has the functional form most commonly used in deep neural networks (Goodfellow et al., 2016), though we allow for linear combinations of features from layers of different depths. The affine transformation is parameterized by the intercept $\alpha_0 \in \mathbb{R}$ and the coefficient vector of the linear combination $\boldsymbol{\alpha}$ which is typically very sparse but must include at least one non-zero coefficient for a feature from $\mathcal{F}_d$. Different features of $\mathcal{F}_{d+1}$ are distinguished only according to the pattern of non-zero entries of $\boldsymbol{\alpha}$ and the

vector of all features from the previous layer $\boldsymbol{F}^d(\boldsymbol{x})$. This is similar to the common notion in variable selection problems where models including the same variables but different non-zero coefficients are still considered to represent the same model. In that sense our features are characterized by the model topology and not by the exact values of the coefficients $\boldsymbol{\alpha}$ and α_0. For example for the vector of features $\boldsymbol{F}(\boldsymbol{x}) = (x_1, ..., x_m, \exp(x_1 + x_2))$ and $\mathcal{G} = \{\exp(x), x^2, x^3\}$ new features could be $\exp(\exp(x_1 + x_2))$, $(x_3 + 0.3\exp(x_1 + x_2))^2$, $(0.5x_3 + 0.3\exp(x_1 + x_2))^3$ and so on, giving an extremely flexible class.

To define the **width** of a feature we distinguish between a local width and a total width. The **local width** of a feature is defined by the number of non-zero coefficients of $\boldsymbol{\alpha}$ in equation (4) and corresponds to the width at the layer of the depth of the feature. However, features may inherit different widths from parental layers and we therefore define the **total width** of a feature recursively as the sum of all local widths of features contributing in equation (4). For example a feature based on a two layered neural network with three neurons at the first layer and one neuron at the second layer, a sigmoid activation function of a form $g(\boldsymbol{\alpha}^T\mathbf{F}(\boldsymbol{x})) = \text{sigmoid}(\boldsymbol{\alpha}_2^T(\text{sigmoid}(\boldsymbol{\alpha}_{1,1}^T\boldsymbol{x}), \text{sigmoid}(\boldsymbol{\alpha}_{1,2}^T\boldsymbol{x}), \text{sigmoid}(\boldsymbol{\alpha}_{1,3}^T\boldsymbol{x}))$ has the total depth of 2 and the total width of $|\boldsymbol{\alpha}_{1,1}^T| + |\boldsymbol{\alpha}_{1,2}^T| + |\boldsymbol{\alpha}_{1,3}^T| + |\boldsymbol{\alpha}_2^T|$, where $|\cdot|$ is a measure counting the number of non zero elements in the corresponding vector.

The number of features q_d from layer d is calculated recursively with respect to the number of features from the previous layer, namely

$$q_d = |\mathcal{G}| \left(2^{\sum_{t=0}^{d-1} q_t} - 1\right) - \sum_{t=1}^{d-1} q_t, \tag{5}$$ eq:num_features

where as discussed above $q_0 = m$ and $|\mathcal{G}|$ simply denotes the number of different functions included in $\mathcal{G}$. One can clearly see that at each new layer the number of features grows exponentially with respect to the number of features from the previous layer. Although the total number of features q is exploding rapidly with growing depth, it still remains finite for any given maximal depth D_{max}.

The feature space we have iteratively constructed with equation (4) thus is extremely rich and encompasses as particular cases features from numerous other popular statistical and machine learning models. If our set of non-linear functions consists of only one specific transformation, for example $\mathcal{G} = \{\text{sigmoid}(x)\}$, then our feature space includes all possible neural networks with the sigmoid activation function. Including more than one function in $\mathcal{G}$ provides quite interesting additional properties of the features. Assume for example that both $\log(x)$ and $\exp(x)$ are members of $\mathcal{G}$. Then multiplication of two (positive) features becomes a new feature with depth $d = 2$ via

$$F_k(\boldsymbol{x}) * F_l(\boldsymbol{x}) = \exp\left(\log(F_k(\boldsymbol{x})) + \log(F_l(\boldsymbol{x}))\right) \ . \tag{6}$$ eq:mult

Clearly also all fractional polynomials (Royston and Altman, 1997) are fully covered if $\{\log(x), \exp(x)\} \subset \mathcal{G}$. However, due to its importance we will include the multiplication operator between two features directly in our algorithmic approach described in Section 3.2 and treat it simply as an additional transformation with depth 1.

Bayesian logic regression (Hubin et al., 2017) is also fully covered by DBRM models. In particular for any binary features $F_k(\boldsymbol{x})$ and $F_l(\boldsymbol{x})$ the two logical operators **AND** and **OR** are simply expressed by $F_k(\boldsymbol{x}) \wedge F_l(\boldsymbol{x}) = F_k(\boldsymbol{x}) * F_l(\boldsymbol{x})$ and $F_k(\boldsymbol{x}) \vee F_l(\boldsymbol{x}) = F_k(\boldsymbol{x}) + F_l(\boldsymbol{x}) - F_k(\boldsymbol{x}) * F_l(\boldsymbol{x})$. Similarly all decision trees (Friedman, 2001) are represented by features within the described topology when using simply $g(x) = \mathrm{I}(x \geq 1)$ where intervals and higher dimensional regions can be defined through multiplications of such terms. A pair of functions $\sin(x)$ and $\cos(x)$ in turn would allow to have all Fourier series within the feature space of interest. Additionally these functions will allow to capture seasonality in various types of data.

It is also important to mention that the universal approximation theorem generalized by Hornik (1991) is applicable to the defined class of models provided that $\mathcal{G}$ contains at least one bounded monotonously increasing function. Hence the defined class of models is dense in terms of approximating any function of interest in the closed domain of $\mathbb{R}^{\mathrm{n}}$.

2.2 Bayesian model specifications

In order to put the model into a Bayesian framework we define priors for both the models and the parameters involved. The model space described by equation (2) is extremely large and includes in principal a total number of 2^{q+r} models, where q depends on the initial number of variables m, the maximal depth D_{max} and the number of nonlinear transformations $|\mathcal{G}|$, and r is the number of available latent Gaussian variables. To avoid potential overfitting by allowing too complex models the following prior definitions make two more restrictions on the model space by specifying an upper limit Q on the number of features which can be included in a model and an upper limit R on the number of latent Gaussian variables. Thus the total number of models with non-zero prior probability will be $\sum_{k=1}^{Q} \binom{q}{k} \times \sum_{l=1}^{R} \binom{r}{l}$. Additionally the priors are penalizing complexity of the features involved. Hence the features can only become deeper or wider when the data really forces them to do so.

Let $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_q)$ and $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_r)$ be the vectors of binary random variables which uniquely define a specific model $\mathfrak{m} = (\boldsymbol{\gamma}, \boldsymbol{\lambda})$ within the full model space. The

corresponding prior for the model structure is defined by

$$p(\mathfrak{m}) \propto \mathrm{I}\left(|\boldsymbol{\gamma}| \leq Q\right) \mathrm{I}\left(|\boldsymbol{\lambda}| \leq R\right) \prod_{j=1}^{q} a^{\gamma_j w_j c(F_j(\mathbf{x}))} \prod_{k=1}^{r} b^{\lambda_k \omega_k v(\delta_k)}. \quad (7)$$

eq:modelprior

Here $|\boldsymbol{\gamma}| = \sum_{j=1}^{q} \gamma_j$ is the number of features involved with Q being the maximum number allowed and $0 < a < 1$. Similarly, $|\boldsymbol{\lambda}| = \sum_{i=k}^{r} \lambda_k$ is the number of random effects involved with R being the maximum number allowed and $0 < b < 1$. The function $c(F_j(\mathbf{x})) \geq 0$ is a non-decreasing measure for the complexity of features $F_j(\mathbf{x})$). In the current implementation of DBRM we use the complexity $c(F_j(\boldsymbol{x}))$ of a feature $F_j(\boldsymbol{x})$ as the total number of nonlinear activations within it plus the total width of the feature. w_j are weights for the complexities of the corresponding features, which can be chosen depending on the application and give additional modeling flexibility. The function $v(\delta_k) \geq 0$ is a measure for the complexity of the latent Gaussian variable δ_k, which is assumed to be a non-decreasing function of the number of hyperparameters defining the distribution of the latent Gaussian variable. ω_k are weights on the complexities of different latent Gaussian variables. Hence for both types of components the prior will prefer simpler terms or combinations of simpler terms whereas more complex terms or more complex combinations are penalized stronger in order to avoid overfitting. Notice that if $\mathfrak{m}$ and $\mathfrak{m}'$ are two vectors only differing in one component with the same complexity with $\gamma_j' = 1 = 1 - \gamma_j$, then

$$\frac{p(\mathfrak{m}')}{p(\mathfrak{m})} = a^{c(F_j(\mathbf{x}))} < 1$$

showing that larger models are penalized more, which gives the basic intuition behind the chosen prior. This result easily generalizes to more complex cases with larger differences in terms of combinations of the addressed components and their joint complexities. Also notice that this prior will prefer a combination of simpler terms to the single complex terms, if the complexity of the latter is greater than common complexity of the combination of the simpler terms. This allows for parsimonious comparisons of models in the space of DBRM models.

Priors for the model parameters are denoted as follows:

$$\boldsymbol{\beta}_j \sim \pi_\beta(\boldsymbol{\beta}_j), \qquad \text{for } \gamma_j = 1 \quad (8)$$

$$\boldsymbol{\psi}_k \sim \pi_k(\boldsymbol{\psi}_k), \qquad \text{for } \lambda_k = 1 \quad (9)$$

latentprior0

$$\phi \sim \pi_\phi(\phi) \quad (10)$$

where independence between these distributions is assumed. Prior distributions on $\boldsymbol{\beta}_j$ and $\boldsymbol{\psi}_k$ (if latent Gaussian variables are present) are usually selected in a way to efficiently

compute marginal likelihoods of the models (by for example specifying conjugate priors) and should be carefully specified for the applications of interest. A thorough discussion on the choice of such priors is given in Hubin et al. (2017).

2.3 Bayesian inference

Posterior marginal probabilities for the model structure can be written using Bayes formula as

$$p(\mathfrak{m}|\mathbf{y}) = \frac{p(\mathfrak{m})p(\mathbf{y}|\mathfrak{m})}{\sum_{\mathfrak{m}' \in \mathcal{M}} p(\mathfrak{m}')p(\mathbf{y}|\mathfrak{m}')}, \qquad \texttt{PMP} \quad (11)$$

where $p(\mathbf{y}|\mathfrak{m})$ denotes the marginal likelihood of $\mathbf{y}$ given a specific model $\mathfrak{m}$. In the next section we will discuss how to estimate this marginal likelihood. The posterior distribution of any other statistic Δ one might be interested in (Δ can for example be predictions) becomes

$$p(\Delta|\mathbf{y}) = \sum_{\mathfrak{m} \in \mathcal{M}} p(\Delta|\mathfrak{m}, \mathbf{y})p(\mathfrak{m}|\mathbf{y}), \qquad \texttt{posterior\_quantile} \quad (12)$$

with corresponding expectation obtained via model averaging

$$\mathrm{E}[\Delta|\mathbf{y}] = \sum_{\mathfrak{m} \in \mathcal{M}} \mathrm{E}[\Delta|\mathfrak{m}, \mathbf{y}]p(\mathfrak{m}|\mathbf{y}). \qquad \texttt{expectation\_quantile} \quad (13)$$

For instance, let $\mathfrak{m}_k$ for any $k \in \{1, ..., q + r\}$ denote the indicator function of the k-th component (which could be a feature or a Gaussian latent variable). Then the posterior marginal inclusion probability $p(\mathfrak{m}_k = 1|\mathbf{y})$ can be expressed as

$$p(\mathfrak{m}_k = 1|\mathbf{y}) = \sum_{\mathfrak{m} \in \mathcal{M}} p(\mathfrak{m}|\mathbf{y})\mathrm{I}(\mathfrak{m}_k = 1) \ . \qquad \texttt{marginal\_inclusion} \quad (14)$$

This provides a well defined measure of the importance of an individual component for fitting DBRM models.

Uncertainty handling

Gal (2016) pin points to the fact that the main advantage of Bayesian deep learning against the classical one is the possibility to infer on uncertainty of the predictions. Currently the uncertainty only is associated to variance Bayesian deep learning (Gal, 2016). He also gives some ideas on the crude approximations for the latter based on variational Bayes for the models with Gaussian observations as well as some heuristics for the non Gaussian models.

With our approach one gets proper and well defined and justified mathematically uncertainty measures on any parameter of interest Δ, which can be naturally derived from

$$p(\Delta \in \Omega_\Delta|\mathbf{y}) = \sum_{\mathfrak{m}\in\mathcal{M}} p(\Delta \in \Omega_\Delta|\mathfrak{m},\mathbf{y})p(\mathfrak{m}|\mathbf{y}) = 1-\alpha. \quad \text{CIS1} \quad (15)$$

Resolving the equation above with respect to Ω_Δ for any fixed α will give the credible region of interest. For example in the univariate case when we want to infer on $\Delta \leq \Delta^\alpha_{ub}$ with probability $1-\alpha$ we get

$$\Delta^\alpha_{ub} = F^{-1}(1-\alpha|\mathbf{y}). \quad \text{CIS2} \quad (16)$$

Measuring variance of Δ is even simpler, since one only has to compute the variance across all of the models and then perform Bayesian model averaging of it as:

$$\mathrm{Var}[\Delta|\mathbf{y}] = \sum_{\mathfrak{m}\in\mathcal{M}} \mathrm{Var}[\Delta|\mathfrak{m},\mathbf{y}]p(\mathfrak{m}|\mathbf{y}), \quad \texttt{var\_quantile} \quad (17)$$

where $\mathrm{Var}[\Delta|\mathfrak{m},\mathbf{y}] = \mathrm{E}[\Delta^2|\mathfrak{m},\mathbf{y}] - \mathrm{E}^2[\Delta|\mathfrak{m},\mathbf{y}]$. Equivalently $\mathrm{Var}[\Delta|\mathbf{y}] = \mathrm{E}[\Delta^2|\mathbf{y}] - \mathrm{E}^2[\Delta|\mathbf{y}]$, where $\mathrm{E}[\Delta^2|\mathbf{y}] = \sum_{\mathfrak{m}\in\mathcal{M}} \mathrm{E}[\Delta^2|\mathfrak{m},\mathbf{y}]p(\mathfrak{m}|\mathbf{y})$.

3 Fitting of DBRM

In this section we will develop algorithmic approaches for fitting the DBRM model. The main tasks are to (i) calculate the marginal likelihoods $p(\mathbf{y}|\mathfrak{m})$ for a given model and (ii) to search through the model space $\mathcal{M}$. Concerning the first issue one has to solve the integral

$$p(\mathbf{y}|\mathfrak{m}) = \int_{\boldsymbol{\theta}} p(\mathbf{y}|,\boldsymbol{\theta}_\mathfrak{m},\mathfrak{m})p(\boldsymbol{\theta}_\mathfrak{m}|\mathfrak{m})d\boldsymbol{\theta}_\mathfrak{m} \quad \texttt{MarginalPosterior} \quad (18)$$

where $\boldsymbol{\theta}_\mathfrak{m}$ are all the parameters involved in model $\mathfrak{m}$. In general these marginal likelihoods are difficult to calculate (Friel and Wyse, 2012; Hubin and Storvik, 2016b). Depending on the model specification we can either use exact calculations when these are available (Clyde et al., 2011; Hubin et al., 2017) or numerical approximations based on simple Laplace approximations (Tierney and Kadane, 1986), the popular integrated nested Laplace approximations (INLA) approach (Rue et al., 2009) or MCMC based methods like Chib's or Chib and Jeliazkov's method (Chib, 1995; Chib and Jeliazkov, 2001). Some comparison of these methods are presented in Hubin and Storvik (2016b) and Friel and Wyse (2012).

The parameters $\boldsymbol{\theta}_{\mathfrak{m}}$ of DBRM for a specified model topology consist of $\boldsymbol{\beta}_{\mathfrak{m}}$ and $\boldsymbol{\psi}_{\mathfrak{m}}$, the regression coefficients for the features and the parameters of the latent Gaussian variables entering the model according to Equation (2). We are not including here the set of coefficients $\boldsymbol{\alpha}_{\mathfrak{m}}$ which encompasses all the parameters inside the features $F_j(\boldsymbol{x})$ of model $\mathfrak{m}$. These are considered simply as constants, used to iteratively generate features of depth d according to (4). Of course one could make the model more general and force $\boldsymbol{\alpha}_{\mathfrak{m}}$ to be a part of $\boldsymbol{\theta}_{\mathfrak{m}}$. However, in that case solving the integral (18) over the full set of parameters $\boldsymbol{\theta}_{\mathfrak{m}}$ including $\boldsymbol{\alpha}_{\mathfrak{m}}$ would be even more computationally demanding due to the complex non-linearity and the extremely high-dimensional integrals involved. Only very crude approximate solutions based of variational Bayes approach (Jordan et al., 1999) can be scalable in such a context (Barber and Bishop, 1998; Blundell et al., 2015). At the same time (MacKay, 1992; Denker and Lecun, 1991) tried Laplace's approximations to approximate marginal likelihood across all layers. This approach is also very demanding computationally and can not be easily combined with combinatorial search of the best architectures in a time friendly way. Neal (2012) suggests a Hamiltonian Monte Carlo to make proper Bayesian inference on Bayesian neural networks. Unfortunately his approach is even more computationally demanding and hence does not seem scalable to the ultra-high dimensional model selection. To reduce computational complexity of HMC and improve its scalability to large data sets Welling and Teh (2011) suggested to use stochastic estimates of the gradient of the likelihood instead of those over entire dataset. These approaches become natural extensions of what we have suggested here and once they become scalable (due to the development of either technologies or methodology) can be easily combined with our search procedure.

3.1 Feature engineering

To avoid computationally demanding marginalization of the likelihood across all layers (MacKay, 1992), we adopt a specific *feature engineering* procedure to define the values of $\boldsymbol{\alpha}_{\mathfrak{m}}$ for any engineered feature. In view of our algorithmic approach presented in Section 3.2 we introduce three different operators which take features of depth d as input and create new features of depth $d+1$.

$$F_j(\boldsymbol{x}) = \begin{cases} g(F_k(\boldsymbol{x})) & \text{for a } \textit{modification} \text{ of } F_k \in \mathcal{F}_d \\ F_k(\boldsymbol{x}) * F_l(\boldsymbol{x}) & \text{for a } \textit{crossover} \text{ between } F_k \in \mathcal{F}_d, F_l \in \mathcal{F}_s (s \leq d) \\ g(\boldsymbol{\alpha}^T \mathbf{F}^d(\boldsymbol{x}) + \alpha_0), & \text{for a nonlinear } \textit{projection} \text{ of all features of depth } d. \end{cases}$$

The operator called *projection* is simply the general feature generation mechanism described by formula (4), whereas the *modification* operator is the special case where $\boldsymbol{\alpha}$

has only one nonzero element $\alpha_k = 1$, and the *crossover* operator can be also seen as a special case in the sense of (6). It immediately follows that modifications have local width 1 and crossovers have local width 2. In general $\boldsymbol{\alpha}$ from the projection operator will be very sparse and its non-zero coefficients define which element $F_j(\boldsymbol{x})$ of the feature space $\mathcal{F}_{d+1}$ is generated. Note that the element $F_j(\boldsymbol{x})$ in fact represents an engineered feature of the form $\Phi_j(\boldsymbol{x}, \boldsymbol{\alpha_j})$, where

$$\Phi_j(\boldsymbol{x}, \boldsymbol{\alpha}_j) = g\left(\sum_{l=1}^{w} \alpha_{r_l,j} F_{r_l}(\boldsymbol{x})\right) .$$

Here $\boldsymbol{\alpha}_j = (\alpha_{r_1,j}, \ldots, \alpha_{r_w,j})$ denotes the non-zero coefficients of the projection where at least one of the corresponding features $F_{r_l}(\boldsymbol{x})$ has depth d.

Applying modifications and crossovers is particularly simple because the values of the corresponding coefficients are a priori set to 1. In contrast for projections one has to compute appropriate coefficients $\boldsymbol{\alpha}_j$. Let the generated projection based feature $F_j(\boldsymbol{x})$ have local width w. Our feature engineering approach allows to create only one representative of a projection based feature $F_j(\boldsymbol{x})$, where the corresponding coefficients are computed as the posterior modes of $p\left(\mathsf{h}(\sum_{l=1}^{w} \alpha_{r_l,j} F_{r_l}(\boldsymbol{x}))|\mathbf{y}\right)$, where h is the link function from (2). This means that only the coefficients at the outer level are estimated,whereas all coefficients corresponding to inner levels (like those from $F_{r_l}(\boldsymbol{x})$ or from features at levels even further down the hierarchy) will be kept fixed. The coefficients $\boldsymbol{\alpha}_j$ are obtained by using posterior modes of the model which includes only $F_{r_l}, r_l \in \{1, ..., w\}$ as covariates. This choice is made not only for computational convenience, but it also has some further important advantages. We observed that applying the non-linear transformation g after computing $\boldsymbol{\alpha}_j$ creates some functional bias of the feature, which often reduces collinearity between features of the same depth and provides additional regularization. Furthermore our procedure can easily be applied for functions g which are not differentiable, like for example the extremely popular rectified linear unit of neural networks or the characteristic functions for decision trees. Finally the suggested feature engineering approach always (provided that h is a canonical link) induces convex optimization problems with unique solutions for the posterior modes $\boldsymbol{\alpha}_j$. Potential ways to further improve the feature engineering step are discussed in Section 5.

3.2 Search algorithms

Having specified the feature engineering procedure and how to compute marginal likelihoods we can switch to task (ii), namely the development of algorithms for searching through the model space. In order to calculate $p(\mathfrak{m}|\mathbf{y})$ exactly we would have to iterate

through the space $\mathcal{M}$ including all potential models, which due to the combinatorial explosion (5) becomes computationally infeasible for even a moderate set of input variables and latent Gaussian variables. We therefore aim at approximating $p(\mathfrak{m}|\mathbf{y})$ by means of exploration of some subspace $\mathcal{M}^* \subset \mathcal{M}$ and approximating (11) by

$$\widehat{p}(\mathfrak{m}|\mathbf{y}) = \begin{cases} \frac{p(\mathfrak{m})p(\mathbf{y}|\mathfrak{m})}{\sum_{\mathfrak{m}' \in \mathcal{M}^*} p(\mathfrak{m}')p(\mathbf{y}|\mathfrak{m}')}, & \text{for } \mathfrak{m} \in \mathcal{M}^*; \\ 0, & \text{for } \mathfrak{m} \notin \mathcal{M}^*. \end{cases} \tag{19}$$

Estimates of marginal inclusion probabilities can then be derived from (19) through

$$\widehat{p}(\mathfrak{m}_k = 1|\mathbf{y}) = \sum_{\mathfrak{m} \in \mathcal{M}^*} \hat{p}(\mathfrak{m}|\mathbf{y}) \mathrm{I}(\mathfrak{m}_k = 1). \tag{20}$$

Low values of $p(\mathfrak{m})p(\mathbf{y}|\mathfrak{m})$ induce both low values of the numerator and small contributions to the denominator in (11), hence models with low $p(\mathfrak{m})p(\mathbf{y}|\mathfrak{m})$ will have no significant influence on posterior marginal probabilities for other models. On the other hand, models with high values of $p(\mathfrak{m})p(\mathbf{y}|\mathfrak{m})$ are important to be addressed. It might be equally important to include regions of significant mass in the model space where no single model has particularly large $p(\mathfrak{m})p(\mathbf{y}|\mathfrak{m})$ but there are many models giving a moderate contribution. Such regions of high posterior mass are particularly important for constructing a reasonable subspace $\mathcal{M}^* \subset \mathcal{M}$ and missing them can dramatically influence our posterior estimates.

The problem described above has been efficiently resolved for standard GLMM models in our previous work (Hubin and Storvik, 2016a) by means of the mode jumping MCMC algorithm - MJMCMC. The main ingredient in MJMCMC is the specification of (possibly large) moves in the model space. In the following two subsections we are suggesting an adaptation to DBRM models of the genetically modified MJMCMC algorithm called GMJMCMC which was originally introduced in the context of Bayesian logic regression (Hubin et al., 2017). Additionally we derive a fully reversible GMJMCMC algorithm (RGMJMCMC).

Genetically Modified MJMCMC

Following Hubin and Storvik (2016a), consider first a variable selection problem with $p = q + r$ potential components. Define $\mathfrak{m}_j$ to be 1 if the j-th component is to be included into the model and 0 otherwise. The general model space $\mathcal{M}$ is of size 2^p. If this discrete model space is multimodal in terms of model posterior probabilities then simple MCMC algorithms typically run into problems with staying for too long in the vicinity of local maxima. The mode jumping MCMC procedure (MJMCMC) was originally proposed

by Tjelmeland and Hegstad (1999) and recently extended to model selection settings by Hubin and Storvik (2016a). MJMCMC is a proper MCMC algorithm equipped with the possibility to jump between different modes within the discrete model space. The key to the success of MJMCMC is the generation of good proposals of models which are not too close to the current state. This is achieved by first making a large jump (changing many model components) and then performing local optimization within the discrete model space to obtain a proposal model. Within a Metropolis-Hastings setting a valid acceptance probability is constructed using symmetric backward kernels, which guarantees that the resulting Markov chain is ergodic and has the desired limiting distribution (see Hubin and Storvik, 2016a, for details).

The MJMCMC algorithm (Hubin and Storvik, 2016a) requires that all of the covariates (features) defining the model space are known in advance and are all considered at each iteration of the algorithm. For the DBRM models, the features are of a complex structure and a major problem in this setting is that it is quite difficult to fully specify the space $\mathcal{M}$ in advance (let alone storing all potential features in some data structure). To solve this problem, we first adopt an adaptive algorithm called Genetically Modified MJMCMC (GMJMCMC) for the DBRM class of models (a simplified version for fitting Bayesian **logic** regression was suggested in Hubin et al., 2017).

The very idea behind GMJMCMC is the introduction of a *population* of model components. At each iteration t only a subset $\mathcal{S}_t \subset \{1, ..., p\}$ of the model components (of fixed size s, where typically $s \ll p$) is considered. These subsets of model components are called populations. To be more specific, throughout our search we consider different populations $\mathcal{S}_1, \mathcal{S}_2, ...$ where each $\mathcal{S}_t$ is a set of s components. Each $\mathcal{S}_t$ then forms a separate *search space* for MJMCMC. Populations (and hence search spaces) dynamically evolve to allow MJMCMC explore different reasonable parts of the infeasibly large (although finite and countable in terms of the number of models) total search space.

The algorithm is initialized by first applying some procedure (for example based on marginal testing) which selects a subset of $q_0 \leq q$ input covariates and a subset of $r_0 \leq r$ latent Gaussian variables. We denote these preselected components by $\mathcal{S}_0 = \{X_1, ..., X_{q_0}; \delta_1, ..., \delta_{r_0}\}$ where for notational convenience we have ordered indices according to preselection which does not impose any loss of generality. Note that depending on the initial preselection procedure $\mathcal{S}_0$ might include a different number of components than all further populations $\mathcal{S}_t$. MJMCMC is then run for a given number of iterations N_{init} on $\mathcal{S}_0$ and the resulting $s_1 < s$ input components (covariates and latent Gaussian variables) with largest marginal inclusion probabilities will become the first s_1 members $\mathfrak{m}^0$ of population $\mathcal{S}_1$. The remaining $s - s_1$ members of $\mathcal{S}_1$ will be newly created

features from the neighborhood of the initial population $\mathcal{N}(\mathcal{S}_0)$, where the operators which define the neighborhood of a given population will be described in detail below. After $\mathcal{S}_1$ has been initialized, MJMCMC is performed for a fixed number of iterations N_{expl} before the next population $\mathcal{S}_2$ is generated. Members of the new population $\mathcal{S}_{t+1}$ for $t > 1$ are chosen from a neighborhood $\mathcal{N}(\mathcal{S}_t)$ of the previous population $\mathcal{S}_t$. This process is iterated for T_{max} populations $S_t \in \{1, ..., T_{max}\}$ and for the final population MJMCMC is run for an extended number of iterations N_{final} to obtain reliable posterior estimates for the features selected in the end.

The neighborhood of populations

The neighborhood $\mathcal{N}(\mathcal{S}_t)$ is defined to be the set of all components which can be generated by applying certain transformations to components of $\mathcal{S}_t$. This set of transformations to be applied includes the operators called *modification*, *crossover*, *projection*, *filtration*, *mutation* and *reduction*. The first three operators have been introduced already in Section 2.1 and have the purpose to generate new features with increasing depth. The other three operators are important to obtain an algorithm which efficiently scans through the search space, where the *filtration* operator deletes components from $\mathcal{N}(\mathcal{S}_t)$, the *mutation* operator allows to add simple input covariates and latent Gaussian variables, and the *reduction* operator removes sub-terms from the existing features.

To generate a new population $\mathcal{S}_{t+1}$ first some components from $\mathcal{S}_t$ with low marginal inclusion probabilities (below a threshold ρ_{min}) are removed using the $filtration$ operator. The removed components are then replaced, where each replacement is generated randomly by a *mutation* operator with probability P_m, by a *crossover* operator with probability P_c, by a *modification* operator with probability P_t, by a *projection* operator with probability P_p or by a *reduction* operator with probability P_r, where $P_c + P_m + P_t + P_p + P_r = 1$. A *reduction* is also applied if the obtained feature is larger than C_{max}. The six operators to generate components from the neighborhood $\mathcal{N}(\mathcal{S}_t)$ are formally defined as follows.

Filtration: Members of the current population $\mathcal{S}_t$ with marginal posterior probabilities below ρ_{min} are deleted with probability P_{del}. The algorithm offers the option that elements from $\mathfrak{m}^0$ (the input variables entering $\mathcal{S}_0$ during initialization) are always kept in the population throughout the search.

Mutation: A new F_{j_1} is selected within $\mathcal{F}_0 \cup \Delta_0$, where $\Delta_0 = \{\delta_{i1}, ..., \delta_{ir}\}$ with respect to the probability distribution $P_{M,t}$.

Crossover: A new covariate $F_{j_1} * F_{j_2}$ where both F_{j_1} and F_{j_2} are selected within $\mathcal{S}_t \cup \mathcal{F}_0$ with respect to the probability distributions $P_{p,t}$.

Modification: A new covariate $F_k = g(F_j)$ where F_j is selected from $\mathcal{S}_t \cup \mathcal{F}_0$ with respect to the probability distributions $P_{p,t}$ and $g(\cdot)$ from $\mathcal{G}$ is selected with respect to probability distribution $P_{g,t}$.

Projection: A new covariate $F_k = g\left(\boldsymbol{\alpha}^T \mathbf{F}^*\right)$ is selected in three steps. First each of the features $F_j(\mathbf{x})$ from the current search space $\mathcal{S}_t$ is selected to be included into $\mathbf{F}^*$ with probability $P_{\gamma_j,t}$. Then $\boldsymbol{\alpha}$ is computed as the posterior mode of $p\left(\mathsf{h}(\boldsymbol{\alpha}^T \mathbf{F}^*)|\mathbf{y}\right)$ and finally the non-linear transformation g is selected according to some probability distribution from the finite set $\mathcal{G}$.

Reduction: A new component is generated from an existing component $F_k \in \mathcal{S}_t$ (selected with probability distribution $P_{r,t}$) by deleting a subset of its terms (nonlinear function, additive or multiplicative terms), where each term has a probability of $P_{d_1,t}$ to be deleted. The separated terms (if the separation is present) are then combined in a single feature with an $*$ operator.

The reduction operator is automatically applied if any of the operators above results in a feature exceeding the limits $|\boldsymbol{\gamma}| \leq Q$ or $|\boldsymbol{\lambda}| \leq R$. For all features generated with any of these operators it holds that if either the newly generated feature is already present in $\mathcal{S}_t$ or it has linear dependence with the currently present features then it is not considered for $\mathcal{S}_{t+1}$. In that case a new replacement feature from the neighborhood $\mathcal{N}(\mathcal{S}_t)$ is proposed.

Reversible Genetically Modified MJMCMC

The GMJMCMC algorithm described above is not reversible and hence can't guarantee that the ergodic distribution of its Markov chain corresponds to the target distribution of interest (see Hubin et al. (2017) for more details). An easy modification based on performing swaps between populations can provide a proper MCMC algorithm in the model space of DBRM models. Specifically the populations of DBRM model components are used as auxiliary variables during these swaps. We consider a transition $\mathfrak{m} \to \mathcal{S}' \to \mathfrak{m}'_0 \to ... \to \mathfrak{m}'_k \to \mathfrak{m}'$ with a given probability kernel. Here $q(\mathcal{S}'|\mathfrak{m})$ is the proposal for the new population (which is simply a large jump in the language of MJMCMC), transitions $\mathfrak{m}'_0 \to ... \to \mathfrak{m}'_k$ are generated by MJMCMC within the model space induced by $\mathcal{S}'$, and the transition $\mathfrak{m}'_k \to \mathfrak{m}'$ is some randomization at the end of the procedure as described in the next paragraph. Notice that the models from steps $\mathfrak{m}'_0 \to ... \to \mathfrak{m}'_k \to \mathfrak{m}'$ as well

as those from similar backward steps are used further in the construction of $\mathcal{M}^*$ and are not just latent states in this sense. Let us also define $\pi(\mathfrak{m}) = p(\mathfrak{m}|\mathbf{y})$. Following arguments in Hubin and Storvik (2016a) the acceptance probability for such a procedure is $r_m = \min\{1, a_m\}$, which depends only on the randomization kernel and the posteriors of the models $\mathfrak{m}$ and $\mathfrak{m}'$, where

$$a_m = \frac{\pi(\mathfrak{m}')q(\mathfrak{m}|\mathfrak{m}_k)}{\pi(\mathfrak{m})q(\mathfrak{m}'|\mathfrak{m}'_k)}. \tag{21}$$

In order for this approach to have a sufficiently large acceptance ratio, the probability $q(\mathfrak{m}|\mathfrak{m}_k)$ in the reverse move has to be non-zero and reasonably high. Transitions within model spaces for this case are calculated exactly as described in Hubin and Storvik (2016a). Randomization is also performed as in Hubin and Storvik (2016a) and is based on swapping independently current components with a positive but small Bernoulli probability ρ_r. Then the ratio of the proposal probabilities can be written as $\frac{q(\mathfrak{m}|\mathfrak{m}_k)}{q(\mathfrak{m}'|\mathfrak{m}'_k)} = \rho_r^{d(\mathfrak{m},\mathfrak{m}_k)-d(\mathfrak{m}',\mathfrak{m}'_k)}$, where $d(\cdot,\cdot)$ is the Hamming distance.

Proposals $q(\mathcal{S}_{t+1}|\mathfrak{m})$ of the new population $t+1$ are taken from a neighborhood of the current model. This happens similarly to the case of the neighborhood of populations described in Section 3.2. Here we have suggested an adaptive proposal design such that first all variables that have $\mathfrak{m}_i = 0$ are not included into $\mathcal{S}_{t+1}$. Similarly to the standard GMJMCMC the deleted components are then replaced with the ones suggested by a by a *crossover* operator with probability P_c, by a *mutation* operator with probability P_c, by a *modification* operator with probability P_t, by a *projection* operator with probability P_p or by a *reduction* operator with probability P_r, where $P_c + P_t + P_p + P_r = 1$. The major difference is that for crossovers, reductions and modifications the features are selected from the set of all possible features $\{1, ..., q\}$ with $P_{M,t}$, $P_{r,t}$ and $P_{p,t}$ be proportional to the approximations of marginal inclusion probabilities of the corresponding features at time t. For the projection operator $\mathbf{F}^*$ is selected again not from $\mathcal{S}_t$, but from $\{1, ..., q\}$, where $P_{\gamma_j,t}$ now correspond to the current approximations of marginal inclusion probabilities. This is an adaptive proposal design strategy, the validity of which is explained in Section 3.3. We have tested also several non-adaptive strategies, none of which worked well in the simulations hence these strategies are not reported to avoid confusion and non-necessary complications.

Other transitions are generated by standard kernels for MJMCMC (Hubin and Storvik, 2016a). The following theorem shows the detailed balance equation for the suggested swaps between models.

Theorem 1. *Assume* $\mathfrak{m} \sim \pi(\cdot)$ *and* $(\mathcal{S}', \mathfrak{m}'_k, \mathfrak{m}')$ *are generated according to the large jump proposal distribution* $q_S(\mathcal{S}'|\mathfrak{m})q_o(\mathfrak{m}'_k|\mathcal{S}', \mathfrak{m})q_r(\mathfrak{m}'|\mathcal{S}, \mathfrak{m}'_k)$. *Assume further* $(\mathcal{S}, \mathfrak{m}_k)$ *are*

generated according to $q_S(\mathcal{S}|\mathfrak{m}')q_o(\mathfrak{m}_k|\mathcal{S},\mathfrak{m}')$. *Put*

$$\mathfrak{m}^* = \begin{cases} \mathfrak{m}' & \textit{with probability } \min\{1, a_m\}; \\ \mathfrak{m} & \textit{otherwise.} \end{cases}$$

where

$$a_m = \frac{\pi(\mathfrak{m}')q(\mathfrak{m}|\mathfrak{m}_k)}{\pi(\mathfrak{m})q(\mathfrak{m}'|\mathfrak{m}'_k)}. \tag{22}$$

Then $\mathfrak{m}^* \sim \pi(\cdot)$.

Proof. Define

$$\bar{\pi}(\mathfrak{m}, \mathcal{S}', \mathfrak{m}'_k) \equiv \pi(\mathfrak{m})q_S(\mathcal{S}'|\mathfrak{m})q_o(\mathfrak{m}'_k|\mathcal{S}', \mathfrak{m}).$$

Then by construction $(\mathfrak{m}, \mathcal{S}', \mathfrak{m}'_k) \sim \bar{\pi}(\mathfrak{m}, \mathcal{S}, \mathfrak{m}'_k)$. Define $(\mathfrak{m}', \mathcal{S}, \mathfrak{m}_k)$ to be a proposal within an ordinary Metropolis Hastings setting $q_r(\mathfrak{m}'|\mathcal{S}, \mathfrak{m}'_k)q_S(\mathcal{S}|\mathfrak{m}')q_o(\mathfrak{m}_k|\mathcal{S}, \mathfrak{m}')$. Then the Metropolis Hastings acceptance ratio becomes

$$\frac{\bar{\pi}(\mathfrak{m}', \mathcal{S}, \mathfrak{m}_k)q_r(\mathfrak{m}'|\mathcal{S}, \mathfrak{m}'_k)q_S(\mathcal{S}'|\mathfrak{m})q_o(\mathfrak{m}'_k|\mathcal{S}', \mathfrak{m})}{\bar{\pi}(\mathfrak{m}, \mathcal{S}', \mathfrak{m}'_k)q_r(\mathfrak{m}'|\mathcal{S}, \mathfrak{m}'_k)q_S(\mathcal{S}|\mathfrak{m}')q_o(\mathfrak{m}_k|\mathcal{S}, \mathfrak{m}')}$$

which reduces to α_m. □

From this theorem it follows that if the Markov chain is irreducible in the model space then it is ergodic and converges to the right posterior distribution. Within the described procedure marginally we are generating samples from the target distribution, i.e. the posterior model probability - $p(\mathfrak{m}|\mathbf{y})$. Therefore instead of using the approximation by (19) we can get frequency based estimates of the model posteriors $p(\mathfrak{m}|\mathbf{y})$. For a sequence of simulated models $\mathfrak{m}^1, ..., \mathfrak{m}^W$ from an ergodic MCMC algorithm with $p(\mathfrak{m}|\mathbf{y})$ as a stationary distribution it holds that

$$\widetilde{p}(\mathfrak{m}|\mathbf{y}) = \frac{\sum_{i=1}^{W} \mathrm{I}(\mathfrak{m}^{(i)} = \mathfrak{m})}{W} \xrightarrow[W\to\infty]{d} p(\mathfrak{m}|\mathbf{y}) \ . \tag{23}$$

Posterior marginal inclusion probabilities (14) for $j \in \{1, ..., q\}$ can then be estimated as

$$\widetilde{p}(\mathfrak{m}_j = 1|\mathbf{y}) = \frac{\sum_{i=1}^{W} \mathrm{I}(\mathfrak{m}_j^{(i)} = 1)}{W} \xrightarrow[W\to\infty]{d} p(\mathfrak{m}_j = 1|\mathbf{y}). \tag{24}$$

Both of these estimates will be consistent, however in general the estimates (19) and (20) will be more accurate as demonstrated by Hubin and Storvik (2016a).

3.3 Important computational tricks

To make the algorithms work sufficiently fast our implementation includes several tricks to be described below.

Delayed rejection

In order to make the computations more efficient and avoid making unnecessary backwards search we make use of the so called delayed acceptance approach. The most computationally demanding parts of the RGMJMCMC algorithms are the forward and backward MCMC searches (or optimizations). In many cases, the proposal generated through the forward search space leads to a very small value of $\pi(\mathfrak{m}')$ resulting in a low acceptance probability regardless of the way the backwards auxiliary variables are generated. In such cases, one would like to reject directly without the need for performing the backward exploration. This can be achieved by the delayed acceptance procedure (Christen and Fox, 2005; Banterle et al., 2015) which we can apply due to the following result:

Theorem 2. *Assume* $\mathfrak{m}$ *and* $\mathfrak{m}'$ *are generated according to the DGMJMCMC algorithm. Accept* $\mathfrak{m}'$ *if both*

1. $\mathfrak{m}'$ *is preliminarily accepted with a probability* $\min\{1, \frac{\pi(\mathfrak{m}')}{\pi(\mathfrak{m})}\}$
2. *and is finally accepted with a probability* $\min\{1, \frac{q_r(\mathfrak{m}|\mathfrak{m}'_k)}{q_r(\mathfrak{m}'|\mathfrak{m}_k)}\}$.

Then also $\mathfrak{m} \sim \pi(\cdot)$.

Proof. It holds that

$$\alpha^*_{mh}(\mathfrak{m}, \mathfrak{m}'; \mathfrak{m}'_k, \mathfrak{m}_k) = \alpha^1_{mh}(\mathfrak{m}, \mathfrak{m}'; \mathfrak{m}'_k, \mathfrak{m}_k) \times \alpha^2_{mh}(\mathfrak{m}, \mathfrak{m}'; \mathfrak{m}'_k, \mathfrak{m}_k)$$

where

$$\alpha^1_{mh}(\mathfrak{m}, \mathfrak{m}'; \mathfrak{m}'_k, \mathfrak{m}_k) = \frac{\pi(\mathfrak{m}')}{\pi(\mathfrak{m})}, \quad \alpha^2_{mh}(\mathfrak{m}, \mathfrak{m}'; \mathfrak{m}'_k, \mathfrak{m}_k) = \frac{q_r(\mathfrak{m}|\mathfrak{m}'_k)}{q_r(\mathfrak{m}'|\mathfrak{m}_k)}$$

Since $\alpha^j_{mh}(\mathfrak{m}, \mathfrak{m}'; \mathfrak{m}'_k, \mathfrak{m}_k) = [\alpha^j_{mh}(\mathfrak{m}', \mathfrak{m}; \mathfrak{m}_k, \mathfrak{m}'_k)]^{-1}$ for $j = 1, 2$, it follows by the general results in Banterle et al. (2015) that we obtain an invariant kernel with respect to $\bar{\pi}$. □

In general delayed acceptance results in a decreased total acceptance rate (Banterle et al., 2015, remark 1), but it is still worthwhile due to the computational gain by avoiding the backwards search step in case of preliminary rejection. Delayed acceptance is implemented in the RGMJMCMC algorithm of our R-package and is used in the examples of this paper.

Adaptive proposals

Another important trick consists of using the chain's history to approximate marginal inclusion probabilities and utilize the latter for the proposal of new populations. Using any measure based on the marginal inclusion probabilities is valid for the reversible algorithm by Theorem 1 from Roberts and Rosenthal (2007) for which two conditions need to be satisfied:

Containment: We have a finite number of models in the model space and hence their enumeration can be performed theoretically within a limited time provided irreducibility of the algorithm. Hence the first condition of the theorem is satisfied.

Diminishing adaptation: As long as the stationary marginal inclusion probabilities are obtained also the second condition is satisfied. This is again possible because of the irreducibility of the constructed Markov Chains in the finite model space.

Parallelization strategy

Due to our interest in exploring as many **unique** high quality models as possible and doing it as fast as possible, running multiple parallel chains is likely to be computationally beneficial compared to running one long chain. The process can be embarrassingly parallelized into B chains. If one is mainly interested in model probabilities, then equation (19) can be directly applied with $\mathcal{M}^*$ now being the set of unique models visited within all runs. An alternative and a more memory efficient approach is to utilize the following posterior estimates based on weighted sums over individual runs:

$$\tilde{P}(\Delta \mid Y) = \sum_{b=1}^{B} u_b \tilde{P}_b(\Delta \mid Y) \; . \qquad \texttt{weighted\_sum} \quad (25)$$

Here u_b is a set of arbitrary normalized weights and $\tilde{P}_b(\Delta \mid Y)$ are the posteriors obtained with either formula (19) or (23) from run b of GMJMCMC or RGMJMCMC. Due to the irreducibility of the GMJMCMC procedure it holds that $\lim_{k\to\infty} \tilde{P}(\Delta \mid Y) = P(\Delta \mid Y)$ where k is the number of iterations. Thus for any set of normalized weights the approximation $\tilde{P}(\Delta \mid Y)$ converges to the true posterior probability $P(\Delta \mid Y)$. Therefore in principle any normalized set of weights w_b would work, like for example $u_b = \frac{1}{B}$. However, uniform weights have the disadvantage to potentially give too much weight to posterior estimates from chains that have not quite converged. In the following heuristic improvement u_b is chosen to be proportional to the posterior mass detected by run b,

$$u_b = \frac{\sum_{M' \in \Omega_b^*} P(Y \mid M') P(M')}{\sum_{b=1}^{B} \sum_{M' \in \Omega_b^*} P(Y \mid M') P(M')} \; .$$

This choice indirectly penalizes chains that cover smaller portions of the model space. When estimating posterior probabilities using these weights we only need, for each run, to store the following quantities: $\tilde{P}_b(\Delta \mid Y)$ for all statistics Δ of interest and $s_b = \sum_{M' \in \Omega_b^*} P(Y \mid M')P(M')$ as a *'sufficient'* statistic of the run. There is no further need of data transfer between processes. The additional proof of that choice of weights to converge to the right results is given in Hubin et al. (2017).

4 Experiments

In this section we will first present two examples addressing prediction in the classification setting, where the performance of DBRM with GMJMCMC and RGMJMCMC is compared with nine competing algorithms. Then we present four examples of model inference after fitting deep regression models with GMJMCMC and RGMJMCMC. They include one simulation study, two real world examples with known data generating mechanisms and one exploratory example, where the underlying truth is not known.

4.1 Prediction

Evaluation measures that we address in the prediction driven examples include power of predictions (Precision), false positive rate (FPR) and false negative rate (FNR), defined as follows:

- Precision $= \frac{\sum_i^{n_p} \text{TP}_i}{n_p}$;
- FPR $= \frac{\sum_i^{n_p} \text{FP}_i}{\sum_i^{n_p} \text{FP}_i + \sum_i^{n_p} \text{TN}_i}$;
- FNR $= \frac{\sum_i^{n_p} \text{FN}_i}{\sum_i^{n_p} \text{TP}_i + \sum_i^{n_p} \text{FN}_i}$.

Here FP_i, FN_i, TP_i, and TN_i are correspondingly indicators of false positive, false negative, true positive, and true negative classifications for prediction $i \in \{1, ..., n_p\}$. And n_p is the size of the test data sample. In the experiments we report medians for these measures as well as the minimum and maximum for all of the algorithms across N runs.

In the two examples addressing binary classification scenarios we are going to address the NEO objects data from NASA Space Challenge 2016 https://2016.spaceappschallenge.org/ and breast cancer data set from ftp://ftp.cs.wisc.edu/math-prog/cpo-dataset/machine-learn/cancer/WDBC/ both publicly available. For both of the data sets we compare several algorithms. GMJMCMC and DGMJMCMC are fitting the DBRM based on

a Bayesian logistic regression with no latent Gaussian variables and independent observations $r = 0$, namely:

$$y_i = y|\rho_i \sim \text{Binom}(1, p_i) \tag{26}$$

$$\rho_i = \frac{e^{\beta_0 + \sum_{j=1}^{q} \gamma_j \beta_j F_j(\boldsymbol{x}_i)}}{1 + e^{\gamma_0 \beta_0 + \sum_{j=1}^{q} \gamma_j \beta_j F_j(\boldsymbol{x}_i)}} \tag{27}$$

Priors on the models and parameters are as follows:

$$p(\boldsymbol{\gamma}) \propto \prod_{j=1}^{q} \exp(-\gamma_j 2c(F_j(\mathbf{x}))) \tag{28}$$

gammodel2

$$p(\boldsymbol{\beta}|\boldsymbol{\gamma}) = |J_n^{\gamma}(\hat{\boldsymbol{\beta}})|^{\frac{1}{2}} , \tag{29}$$

where $c(F_j(\mathbf{x}))$ is the complexity of feature j and weights w_j are chosen to be 2 for all the models. Such a prior corresponds to a posterior having a penalty on the complexity similar to AIC model selection criterion, which is known (at least for the linear models) to be optimal in terms of prediction driven model selection and guarantees some optimality in terms of leave out one cross validation. Finally, $\hat{\boldsymbol{\beta}}$ are the maximum likelihood estimates of the coefficients and $|J_n^{\gamma}(\hat{\boldsymbol{\beta}})|$ is the determinant of the corresponding Fisher information matrix. The latter corresponds to the Jeffrey's prior. Laplace's approximations of the marginal likelihood (Hubin et al., 2017) are considered. Predictions based on DBRM are made as:

$$\hat{y}_i = \text{I}\left(\left(\sum_{\mathfrak{m} \in \mathcal{M}*} \hat{\rho}_{\mathfrak{m}} p(\mathfrak{m}|\mathbf{y})\right) \geq \Delta\right) \tag{30}$$

In the prediction driven examples we also address tree based and linear xGboost, elastic networks, neural networks with multiple hidden layers, random forest, naive Bayes, and simple *frequentest* logistic regressions. Tuning and hyperparameters of all of the algorithms are given in Table 11 in the Appendix of the article. In both of the examples we use a fixed split between training and test samples for all of the runs.

Example 1: Neo asteroids classification

The labeled dataset consists of 20766 well studied asteroids some of which are PHO (potentially hazardous objects), whilst others are not. The following explanatory variables are present in the data set: *Mean anomaly, Inclination Argument of perihelion Longitude of the ascending node, Rms residual, Semi major axis, Eccentricity, Mean motion, Absolute magnitude*. The allowed non-linearities in this example are: $\cos(x)$, $\text{sigmoid}(x)$, $\tanh(x)$, $\text{atan}(x)$, and $\text{erf}(x)$. The algorithms are compared on $N = 100$ replications

Algorithm	min.p	med.p	max.p	min.fn	med.fn	max.fn	min.fp	med.fp	max.fp
GMJMCMC	0.9949	**0.9998**	**1.0000**	**0.0001**	**0.0002**	0.0073	**0.0000**	0.0002	0.0032
RGMJMCMC	0.9939	0.9998	1.0000	0.0001	0.0002	0.0088	0.0000	0.0002	0.0072
LASSO	**0.9991**	0.9991	0.9991	0.0013	0.0013	0.0013	0.0000	**0.0000**	**0.0000**
RIDGE	0.9982	0.9982	0.9982	0.0026	0.0026	0.0026	0.0000	0.0000	0.0000
LXGBOOST	0.9980	0.9980	0.9980	0.0029	0.0029	0.0029	0.0000	0.0000	0.0000
NBAYESS	0.9963	0.9963	0.9963	0.0054	0.0054	0.0054	0.0000	0.0000	0.0000
MJMCMC	0.9943	0.9946	0.9947	0.0002	0.0002	**0.0002**	0.0159	0.0162	0.0172
DEEPNETS	0.8979	0.9728	0.9979	0.0018	0.0384	0.1305	0.0000	0.0000	0.0153
TXGBOOST	0.8283	0.8283	0.8283	0.0005	0.0005	0.0005	0.3488	0.3488	0.3488
RFOREST	0.6761	0.8150	0.9991	0.0003	0.1972	0.3225	0.0000	0.0162	0.3557
LR	0.6471	0.6471	0.6471	0.0471	0.0471	0.0471	0.4996	0.4996	0.4996

Table 1: Comparison of performance (Precision, FPR, FNR) of different algorithms for NEO objects data

Algorithm	Power	FNR	FPR
GMJMCMC	**0.9998** (0.9949,**1.0000**)	**0.0002** (**0.0001**,0.0073)	0.0002 (**0.0000**,0.0032)
RGMJMCMC	0.9998 (0.9939,1)	0.0002 (0.0001,0.0088)	0.0002 (0.0000,0.0072)
LASSO	0.9991 (-,-)	0.0013 (-,-)	**0.0000** (-,-)
RIDGE	0.9982 (-,-)	0.0026 (-,-)	0.0000 (-,-)
LXGBOOST	0.9980 (0.9980,0.9980)	0.0029 (0.0029,0.0029)	0.0000 (0,0.0000)
NBAYESS	0.9963 (-,-)	0.0054 (-,-)	0 (-,-)
MJMCMC	0.9946 (0.9943,0.9947)	0.0002 (0.0002,0.0002)	0.0162(0.0159,0.0172)
DEEPNETS	0.9728 (0.8979,0.9979)	0.0384 (0.0018,0.1305)	0.0000(0.0000,0.0153)
TXGBOOST	0.8283 (0.8283,0.8283)	0.0005 (0.0005,0.0005)	0.3488(0.3488,0.3488)
RFOREST	0.815 (0.6761,0.9991)	0.1972 (0.0003,0.3225)	0.0162 (0.0000,0.3557)
LR	0.6471 (-,-)	0.0471 (-,-)	0.4996 (-,-)

Table 2: Comparison of performance (Precision, FPR, FNR) of different algorithms for NEO objects data

with different seeds. We have used the training sample consisting of 64 objects (32 of which are NEO, whilst the other 32 are not) and the test sample consisting of all 20702 labeled asteroids that are present in the Nasa Space Challenge 2016 data set. We obtained and summarized in Table 2 the results on the given test and train sets. As one can see the train set seems to be extremely informative and all of the algorithms perform well. DBRM fitted with GMJMCMC and RGMJMCMC on average gives the highest scores and the lowest error rates, however the results on a given number of iterations exhibit some small variance.

Example 2: Breast cancer classification

The second example to be addressed for classification is the breast cancer data. Here the observations contain 357 benign, 212 malignant tissues. Ten real-valued features are computed for each cell nucleus: radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, fractal dimension. The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features in total. Train set was set to be 25% of the whole data set and was randomized for each of the $N = 100$ simulations. The rest was used as the test set. $N = 100$ runs of each of the addressed algorithms were performed. The allowed non-linearities in this example are: sigmoid(x), I$(x > 1)$, ReLu(x), $x^{\frac{1}{3}}$, and $x^{\frac{1}{5}}$. The results are reported in Table 3. As one can see performance of DBRM fitted with GMJMCMC and RGMJMCMC for this example is reasonably good in comparison to other popular algorithms.

Algorithm	min.p	med.p	max.p	min.fn	med.fn	max.fn	min.fp	med.fp	max.fp
RIDGE	**0.9742**	**0.9742**	0.9742	0.0592	0.0592	0.0592	0.0037	**0.0037**	**0.0037**
GMJMCMC	0.9437	0.9695	**0.9812**	0.0479	0.0536	0.1067	**0.0000**	0.0148	0.0361
DEEPNETS	0.9225	0.9695	0.9789	**0.0305**	0.0674	0.1167	0.0000	0.0074	0.0949
RGMJMCMC	0.9554	0.9683	0.9789	0.0479	0.0536	0.0809	0.0037	0.0148	0.0361
NAIVEBAYESS	0.9671	0.9671	0.9671	0.0479	0.0479	0.0479	0.0220	0.0220	0.0220
MJMCMC	0.9624	0.9624	0.9624	0.0756	0.0756	0.0756	0.0111	0.0111	0.0111
LASSO	0.9577	0.9577	0.9577	0.0756	0.0756	0.0756	0.0184	0.0184	0.0184
LXGBOOST	0.9554	0.9554	0.9554	0.0809	0.0809	0.0809	0.0184	0.0184	0.0184
TXGBOOST	0.9484	0.9531	0.9601	0.0536	0.0647	0.0756	0.0291	0.0326	0.0361
RFOREST	0.9038	0.9343	0.9624	0.0422	0.0914	0.1675	0.0000	0.0361	0.1010
LR	0.9272	0.9272	0.9272	0.0305	**0.0305**	**0.0305**	0.0887	0.0887	0.0887

Table 3: Comparison of performance (Precision, FPR, FNR) of different algorithms for breast cancer data

Moreover DBRM fitted by GMJMCMC achieves the best performance possible in the best of the simulations, showing that once it converges it can work better than the competing approaches. To prove this numerically we ran the GMJMCMC algorithm additionally for 100 times with the same tuning parameters but on 32 parallel threads for each simulation. The results are summarized in the Tables 4 and 5 of precision, FNR and FPR below. As one can clearly see in these Tables DBRM performance becomes stable and on average better than the performance of the compared above approaches. This shows that once DBRM is being fit for a sufficient amount of time, it becomes an extremely robust and powerful approach for making predictions.

Δ	min.p	med.p	max.p	min.fn	med.fn	max.fn	min.fp	med.fp	max.fp
0.6000	**0.9718**	**0.9765**	**0.9812**	0.0479	0.0592	0.0702	**0.0000**	**0.0000**	0.0074
0.5000	0.9671	0.9742	0.9812	0.0479	0.0479	0.0536	0.0000	0.0111	0.0220
0.7000	0.9648	0.9695	0.9742	0.0647	0.0756	0.0862	0.0000	0.0000	**0.0000**
0.4000	0.9577	0.9624	0.9671	0.0479	0.0479	0.0479	0.0220	0.0291	0.0361
0.8000	0.9554	0.9601	0.9648	0.0862	0.0966	0.1067	0.0000	0.0000	0.0000
0.3000	0.9507	0.9531	0.9601	0.0422	0.0479	0.0479	0.0361	0.0430	0.0464
0.2000	0.9413	0.9460	0.9531	0.0305	0.0422	0.0422	0.0498	0.0565	0.0632
0.9000	0.9366	0.9460	0.9531	0.1117	0.1264	0.1452	0.0000	0.0000	0.0000
0.1000	0.9272	0.9319	0.9413	**0.0245**	**0.0305**	**0.0305**	0.0729	0.0825	0.0918
1.0000	0.6268	0.6268	0.6268	0.5000	0.5000	0.5000	0.0000	0.0000	0.0000

Table 4: Comparison of performance (Precision, FPR, FNR) of parallel GMJMCMC for breast cancer data with different thresholds

Δ	min.p	med.p	max.p	min.fn	med.fn	max.fn	min.fp	med.fp	max.fp
0.5000	**0.9695**	**0.9765**	**0.9812**	0.0479	0.0479	0.0536	**0.0000**	0.0074	0.0184
0.6000	0.9695	0.9765	0.9789	0.0536	0.0592	0.0756	0.0000	**0.0000**	0.0037
0.7000	0.9671	0.9695	0.9742	0.0647	0.0756	0.0809	0.0000	0.0000	**0.0000**
0.4000	0.9577	0.9624	0.9695	0.0479	0.0479	0.0479	0.0184	0.0291	0.0361
0.8000	0.9554	0.9601	0.9648	0.0862	0.0966	0.1067	0.0000	0.0000	0.0000
0.3000	0.9507	0.9531	0.9577	0.0422	0.0479	0.0479	0.0361	0.0430	0.0464
0.2000	0.9413	0.9460	0.9507	0.0364	0.0422	0.0479	0.0498	0.0565	0.0632
0.9000	0.9366	0.9460	0.9531	0.1117	0.1264	0.1452	0.0000	0.0000	0.0000
0.1000	0.9249	0.9319	0.9390	**0.0245**	**0.0305**	**0.0305**	0.0729	0.0825	0.0918
1.0000	0.6268	0.6268	0.6268	0.5000	0.5000	0.5000	0.0000	0.0000	0.0000

Table 5: Comparison of performance (Precision, FPR, FNR) of parallel RGMJMCMC for breast cancer data with different thresholds

4.2 Model inference

For the examples 3-5 concerned with model inference we utilize DBRM models with conditionally independent Gaussian observations by letting $r = 0$:

$$Y_i|\mu_i \sim N(\mu_i, \sigma^2), \quad i \in \{1, ..., n\} \tag{31}$$ exomodel0

$$\mu_i = \beta_0 + \sum_{j=1}^{q} \gamma_j \beta_j F_j(\boldsymbol{x}_i) \tag{32}$$ exomodel1

We consider the set $\mathcal{G} = \{\text{sigmoid}(x), \tanh(x), \text{atan}(x), \sin(x), \cos(x), |x|^{\frac{1}{3}}\}$ of non-linear modifications and restrict the depth with $D_{max} = 5$ in these three examples. Priors are

specified according to

$$p(\boldsymbol{\gamma}) \propto \prod_{j=1}^{q} \exp\left(-2 \log n \gamma_j c(F_j(\mathbf{x}))\right) \ , \tag{33}$$

gammodel1

$$\pi(\sigma^2) = \sigma^{-2} \tag{34}$$

$$p(\boldsymbol{\beta}|\boldsymbol{\gamma}) = |J_n^{\gamma}(\hat{\boldsymbol{\beta}})|^{\frac{1}{2}}, \tag{35}$$

where $c(F_j(\mathbf{x}))$ is the complexity of feature j with weights $w_j = 2 \log n$ for all of the features, and $a = e^{-1}$ giving a BIC-like penalty for the joint complexity of the model. Finally, $\hat{\boldsymbol{\beta}}$ are the maximum likelihood estimates of the coefficients and $|J_n^{\gamma}(\hat{\boldsymbol{\beta}})|$ is the determinant of the corresponding Fisher information matrix. The latter corresponds to the Jeffrey's prior. Similarly to examples 1 and 2 Laplace's approximations of the marginal likelihood (Hubin et al., 2017) are considered. To evaluate the performance of algorithms we will report estimates for the power (Power), the proportion of false discoveries (FDP), and the expected number of false positives (FP), defined as follows:

- Power $= \frac{\sum_i^N \text{TP}_i}{N}$;
- FDP $= \frac{\sum_i^N \text{FP}_i}{\sum_i^N \text{TP}_i + \sum_i^N \text{FP}_i}$;
- FP $= \sum_i^N \text{FP}_i$.

Here N is the total number of simulation runs. FP_i and TP_i denote the number of false and true positives from the i-th simulation run.

The 6th example illustrates how DBRM would be used in practice on real data where the underlying data generating mechanism is unknown. Specifically we will try to find optimal combinations of the latent Gaussian variables which model the dependence structure between individuals. More details on the set of latent variables and their priors are given in the corresponding subsection below.

Only in case of the 3rd example comparable inference can be made via the logic regression approach (Hubin et al., 2017). Hence such a comparison is done. To our awareness no other statistical or machine learning approaches can be used for inference on the sophisticated functional relations between the variables in the closed forms. Hence no comparison with other approaches is possible.

Example 3: Simulated data

In this simulation study we generated $N = 100$ datasets with $n = 1000$ observations and $p = 50$ binary covariates. The covariates were assumed to be independent and were

simulated for each simulation run as $X_j \sim \text{Bernoulli}(0.5)$ for $j \in \{1, \ldots, 50\}$. In the first simulation study the responses were simulated according to a Gaussian distribution with error variance $\sigma^2 = 1$ and individual expectations specified as follows:

$$\begin{aligned} E(Y) &= 1 + 1.5X_7 + 1.5X_8 + 6.6X_{18} * X_{21} + 3.5X_2 * X_9 + 9X_{12} * X_{20} * X_{37} \\ &+ 7X_1 * X_3 * X_{27} + 7X_4 * X_{10} * X_{17} * X_{30} + 7X_{11} * X_{13} * X_{19} * X_{50} \end{aligned}$$

We compare the results of GMJMCMC, RGMJMCMC for DBRM with the GMJMCMC implementation for Bayesian logic regression (Hubin et al., 2017). All algorithms were run on 32 threads until the same number of models were visited after the last change of the model space. In particular, in each of the threads the algorithms were run until 20000 unique models were obtained after the last population of models had been generated at iteration 15000. Specification of the Bayesian Logic Regression model corresponds exactly to the one used in simulation Scenario 6 in Hubin et al. (2017). In this example a detected feature is only counted as a true positive if it exactly coincides with a feature of the data generating model. The results are summarized in Table 6. Detection in this example correspond to the features having marginal inclusion probabilities above 0.5 after the search is completed.

Both GMJMCMC and RGMJMCMC performed exceptionally well for fitting this DBRM with slight advantages of the former. The original GMJMCMC(LR) algorithm for fitting Bayesian Logic Regression in this case performed almost as well as GMJMCMC and RGMJMCMC, except for a significant drop in power in one of the four-way interactions. This is however not too surprising because the crossover operator of DBRM models perfectly fits the data generating model whereas the logic regression model focuses on general logic expressions and provides in that sense a larger chance to generate features which are closely related to the data generating four-way interaction Hubin et al. (2017).

Table 6: Results for the three simulation scenario. Power for individual expression, overall power, expected number of false positives (FP), and FDP are compared.

	GMJ	RGMJ(A))	GMJ(LR)
X_7	1.0000	1.0000	0.9900
X_8	1.0000	1.0000	1.0000
$X_2 * X_9$	1.0000	0.9600	1.0000
$X_{18} * X_{21}$	1.0000	1.0000	0.9600
$X_1 * X_3 * X_{27}$	1.0000	1.0000	1.0000
$X_{12} * X_{20} * X_{37}$	1.0000	1.0000	0.9900
$X_4 * X_{10} * X_{17} * X_{30}$	0.9900	0.9200	0.9100
$X_{11} * X_{13} * X_{19} * X_{50}$	0.9800	0.8900	0.3800
Overall Power	0.9963	0.9712	0.9038
FP	0.5100	1.1400	1.0900
FDP	0.0601	0.1279	0.1310

Example 4: Jupiter mass of the planet

The fourth and fifth examples will be based on data sets describing physical parameters of newly discovered exoplanets. The input covariates include planet and host star attributes, discovery methods, and dates of discovery. The data was originally collected and continues to be updated by Hanno Rein at the Open Exoplanet Catalogue Github repository https://github.com/OpenExoplanetCatalogue/open_exoplanet_catalogue. Here we will try to rediscover two basic physical laws from the given data set which involve sophisticated non-linearities. A subset of $n = 223$ planets from the full data set, which has no missing values, is used in these two examples.

In example 4 we consider the planetary mass as a function of its radius and density. It is common in astronomy to use the measures of Jupiter as units and a basic physical law gives the non-linear relation

$$m_p^a = \frac{m_p}{m_j} \approx \frac{4\pi}{3} \times \frac{R_p^3}{R_j^3} \times \frac{\rho_p}{\rho_j},$$

where m_p^a is *PlanetaryMassJpt*, the planetary mass m_p measured in units of Jupiter mass m_p. Similarly the radius of the planet R_p is measured in units of jupiter radius R_j and the density of the planet ρ_p is measured in units of jupiter density ρ_j. Hence in the data set the variable *RadiusJpt* refers to $\frac{R_p}{R_j}$, and *PlanetaryDensJpt* denotes $\frac{\rho_p}{\rho_j}$.

We will apply DBRM according to (32)-(33) to model *PlanetaryMassJpt* as a function of ten potential input variables, namely *TypeFlag*, *RadiusJpt*, *PeriodDays*, *SemiMajorAx-*

isAU, *Eccentricity*, *HostStarMassSlrMass*, *HostStarRadiusSlrRad*, *HostStarMetallicity*, *HostStarTempK*, *PlanetaryDensJpt*. In order to evaluate the algorithm and its capability to detect true signals for this example we run it $N = 100$ times on a given training data set, which is unique across the runs. To illustrate to which extent the performance of DBRM depends on the number of parallel runs we consider computations with 1, 4 and 16 threads, respectively. Like in the simulation study before in each of the threads the algorithms were run until 20000 unique models were obtained after the last population of models had been generated at iteration 15000. We compare GMJMCMC with RGMJMCMC and the results are summarized in Table 7.

Table 7: Power, False Positives (FP) and FDP based on the decision rule that the posterior probability of a feature is larger than 0.25. The feature $PlanetaryMassJpt^3$ *PlanetaryDensJpt* is counted as true positive, all other selected features as false positive.

	GMJMCMC			RGMJMCMC		
Effort	Power	FP	FDP	Power	FP	FDP
16 threads	1.00	0.00	0.00	0.97	0.06	0.058
4 threads	0.79	0.40	0.34	0.61	0.73	0.54
1 thread	0.43	1.21	0.74	0.32	1.67	0.84

Clearly the more resources become available the better DBRM performs. Both RGMJMCMC and GMJMCMC algorithms manage to find the correct model with rather large Power (reaching gradually one) and small FDP (reaching gradually zero). Note that once again the number of false positives grows with decreasing power which indicates that instead of the correct feature some potentially closely related features are selected.

Example 5: Kepler's third law

Kepler's third law says that the square of the orbital period P of a planet is directly proportional to the cube of the semi-major axis a of its orbit. Mathematically this can be expressed as

$$\frac{P^2}{a^3} = \frac{4\pi^2}{G(M+m)} \approx \frac{4\pi^2}{GM} = constant, \qquad \texttt{3kepl0} \quad (36)$$

where G is the gravitational constant, m is the mass of the planet, M is the mass of the corresponding hosting star and $M \gg m$.

In this example we want to model the semi-major axis of the orbits *SemiMajorAxisAU* as a function of the following 10 potential input variables: *TypeFlag*, *RadiusJpt*, *PeriodDays*, *PlanetaryMassJpt*, *Eccentricity*, *HostStarMassSlrMass*, *HostStarRadiusSlrRad*,

HostStarMetallicity, *HostStarTempK*, *PlanetaryDensJpt*. Equation (36) can be reformulated as

$$a = \left(\frac{GP^2}{4\pi^2}(M_h + m_p)\right)^{\frac{1}{3}} \approx \left(\frac{GM_s}{4\pi^2}\ P^2 M_h^a\right)^{\frac{1}{3}},$$

where a corresponds to $SemiMajorAxisAU$, M_h^a corresponds to *HostStarMassSlrMass* which is the mass of the host star measured in the unit of solar mass M_s, and P is *PeriodDays*. Hence we expect to have a high posterior probability of the feature $(HostStarMassSlrMass \times PeriodDays^2)^{\frac{1}{3}}$ or similar features.

In order to assess the ability of GMJMCMC and RGMJMCMC to detect the correct feature in this example we performed $N = 100$ runs with 1, 4, 16, 32 and 64 threads, using for each run a different seed. As previously the algorithms were run in each thread until 20000 unique models were obtained. It turned out that there are two features which are extremely correlated with the target feature, namely $(HostStarRadiusSlrRad \times PeriodDays^2)^{\frac{1}{3}}$ (with a correlation of 0.9999667) and $(HostStarTempK \times PeriodDays^2)^{\frac{1}{3}}$. This is not particularly surprising given that there exist certain power laws which relate the mass and the radius as well as the mass and the temperature of a star, although the relationship is never linear. At the same time we don't allow other fractional degrees than $x^{\frac{1}{3}}$ in our $\mathcal{G}$ hence other than these relations were not feasible in our simulations. In our evaluation of algorithmic results we anyways count a detection of any of these three highly correlated features as a true positive, other features are counted as false positives. The results for GMJMCMC and RGMJMCMC are presented in Table 8.

Table 8: Comparison of Results on Example 3 for GMJMCMC and RGMJMCMC using different number of threads. F_1, F_2 and F_3 refer to the number of times the specific features $(HostStarMassSlrMass \times PeriodDays^2)^{\frac{1}{3}}$, $(HostStarRadiusSlrRad \times PeriodDays^2)^{\frac{1}{3}}$ and $(HostStarTempK \times PeriodDays^2)^{\frac{1}{3}}$ had a posterior probability larger than 0.25. Power gives the percentage of runs where at least one of these three features was detected. FP counts the number of other features and FDP is the corresponding false discovery proportion.

	GMJMCMC						RGMJMCMC					
Effort	F_1	F_2	F_3	Power	FP	FDP	F_1	F_2	F_3	Power	FP	FDP
64 threads	81	71	1	1.00	0.02	0.013	78	75	2	0.99	0.03	0.019
32 threads	63	58	11	0.99	0.14	0.11	55	57	9	0.95	0.12	0.09
16 threads	34	41	32	0.84	0.46	0.30	31	38	18	0.79	0.68	0.44
4 threads	15	10	16	0.38	1.05	0.62	8	14	8	0.29	1.47	0.83
1 thread	6	5	3	0.13	1.46	0.82	6	4	2	0.12	1.81	0.94

Just like in Example 2 one can observe in the attempt to recover the 3rd Kepler's law from the raw data that with increasing computational effort power is converging to 1 and FDP is getting close to 0 both for GMJMCMC and RGMJMCMC. Note that both

in Example 2 and in Example 3 the fairly small sample size of $n = 223$ observations was used to obtain these results. Once again GMJMCMC is performing slightly better than RGMJMCMC.

Example 5: Interpretability of DBRM results

In order to show the importance of the suggested approach to transparency of the results we have additionally run the GMJMCMC algorithm with DBRM under 3 restricted scenarios for this example:

1. When $\mathcal{G} = \{\text{sigmoid}(x)\}$;
2. When $\mathcal{G} = \{\text{sigmoid}(x)\}$, $D_{max} = 300$, and $P_c = 0$;
3. When $\mathcal{G} = \{\text{sigmoid}(x)\}$, $D_{max} = 300$, and $P_c = 0$ and $p(\gamma_j) \propto 1$.

These examples do not allow to get the correct "true" model due to the restrictions. In the first scenario the true model is infeasible since the cubic root function is not a part of $\mathcal{G}$. In the second scenario additionally the crossovers are not allowed and a much larger depth is suggested. Finally in the third scenario we additionally remove penalization on the more complex features, so that all features get an uniform prior in the feature space and hence no regularization at all is present.

By universal approximation theorem (Hornik, 1991) the 3rd Kepler's law can still be approximated with the desired precision. Unfortunately the results become not transparent at all. We will now see the price of these approximations in terms of interpretability of the models and features. To simplify the reporting let us define *TypeFlag*, *RadiusJpt*, *PeriodDays*, *PlanetaryMassJpt*, *Eccentricity*, *HostStarMassSlrMass*, *HostStarRadiusSlrRad*, *HostStarMetallicity*, *HostStarTempK*, *PlanetaryDensJpt* as x_1-x_{10} correspondingly. Let us also use symbol g_σ for sigmoid functions. In the Table 9 we report the top most frequent detected features under these scenarios over $N = 100$ simulations.

Table 9: 10 most frequent features detected under scenarios 1, 2 and 3

Fq	Feature	Fq	Feature	Fq	Feature
99	x_3	100	x_3	100	x_3
98	x_3*x_3	72	g_σ(-10.33+0.24x_4-8.83x_8)	54	x_2
93	x_3*x_{10}	64	x_{10}	21	g_σ(-16.91-4.94x_2)
4	x_3*x_3*x_{10}	62	x_2	19	x_9
1	x_9*x_3	16	g_σ(0.21+0.01x_3+0.20x_7)	16	x_5
1	x_9*x_3*x_3	9	x_4	14	x_{10}
1	x_{10}*x_{10}*x_3	7	g_σ(-13.11-7.76x_8-3.33x_2+0.40x_{10})	10	g_σ(6.88×10^9-3.92x_2+ 3.44×$10^9$$g_\sigma$(-13.57-0.17$x_4-2.84x_2-7.66x_8$+0.54$x_{10}$) -13.76×$10^9$$g_\sigma$($g_\sigma$(-13.57-0.17$x_4-2.84x_2-7.66x_8$+ 0.54$x_{10}$)))
1	x_7*x_3*x_3	5	g_σ(-3.36+2.83x_3+0.21x_3-3.36x_9)	9	x_4
1	x_6*x_3*x_3	3	g_σ(g_σ(-10.33+0.24x_4)-8.83x_8)	8	g_σ(-13.57-0.17x_4-2.84x_2-7.66x_8+0.54x_{10})
1	x_3*x_3*x_3	3	g_σ(0.15+0.05x_4-0.01x_3+0.15x_7)	7	g_σ(0.21+0.21x_3)
0	Others	4	Others	> 300	Others

The ultimate conclusion is that more flexible feature space allows to achieve a way better models in terms of their transparency, which should be seen as one of the main selling points of the DBRM approach. A yet another conclusion is that in less flexible feature space one often requires extremely complex models to explain relatively simple phenomena. The latter leads not only to potential overfitting, but also to the fact that these models require significantly more space to be stored and significantly more computational power to be computed (if one for instance is interested in predictions based on them). Our approach, as we believe, will often construct architectures that reach state of the art performance and still remain relatively simple, hence representing sophisticated phenomena in the most dense way possible (under the chosen priors and set of nonlinear transformations).

Example 6: Epigenetic data with latent Gaussian variables

In this example we illustrate how DBRM models work in case of simultaneous feature engineering and choice of proper latent Gaussian variables. To this end we consider genomic and epigenomic data from Arabadopsis thaliana. Arabadopsis is an extremely well studies plant model organism for which plenty of genomic and epigenomic data sets

are publicly available (see for example Becker et al. (2011)).

In terms of epigenetic data we consider methylation markers. DNA locations with a nucleotide of type cytosine nucleobase (C) can be either methylated or not. Our focus will be on modeling the amount of methylated reads through different covariates including (local) genomic structures, gene classes and expression levels. The studied data was obtained from the NCBI GEO archive (Barrett et al., 2013). A sample of $T = 500$ base-pairs of a single plant from that data base is addressed in this example. We model the number of methylated reads $Y_t, \in \{1, ..., R_t\}$ per locus $t \in \{1, ..., T\}$ to be Poisson distributed with mean $\mu_t \in \mathbb{R}^+$. For the DBRM model (2) we use the logarithm as the canonic link function. The other input covariates are denoted by $X_t = \{X_{t,1}, ..., X_{t,p}\}, t \in \{1, ..., T\}$ and we include an offset defined by the total number of reads per location $R_t, \in \mathbb{N}$. The offset mentioned above is modeled as an additional component of the model and hence can be a matter of model choice. Furthermore we consider the following latent Gaussian variables to model spatial correlations:

***AR*(1) process:** an autoregressive process of order 1 with parameter $\rho \in \mathbb{R}$, namely $\delta_t = \rho\delta_{t-1} + \epsilon_t \in \mathbb{R}$ with $\epsilon_t \sim N(0, \sigma_\epsilon^2)$, $t \in \{1, ..., T\}$. For this process the priors on the hyper-parameters are defined as follows: we first reparametrize to $\psi_1 = \log \frac{1}{\sigma^2_{\epsilon_t}}(1 - \rho^2)$, $\psi_2 = \log \frac{1+\rho}{1-\rho}$ and assume $\psi_1 \sim \text{logGamma}(1, 5 \times 10^{-5})$, $\psi_2 \sim N(0, 0.15^{-1})$ (The R-INLA project, 2018).

***RW*(1) process:** a random walk process of order 1 based on the Gaussian vector $\delta_1, ..., \delta_T$, which is constructed assuming independent increments: $\Delta\delta_i = \delta_i - \delta_{i+1} \sim N(0, \tau^{-1})$. For this process the priors on the hyper-parameters are defined as follows: we reparametrize to $\psi = \log \tau$ and assume $\psi \sim \text{logGamma}(1, 5 \times 10^{-5})$ (The R-INLA project, 2018).

***OU* process:** an Ornstein-Uhlenbeck process (with mean zero), which is defined via the stochastic differential equation $\Delta\delta_t = -\phi\delta_t + \sigma dW_t$, where $\phi > 0$ and W_t is the Wiener process. This is the continuous time analogue to the discrete time $AR(1)$ model and the process is Markovian. Let $\delta_1, ..., \delta_T$ be the values of the process at increasing time-points $t_1, ..., t_T$, then the conditional distribution $\delta_i|\delta_1, ..., \delta_{i-1}$ is Gaussian with mean $\delta_{i-1}e^{-\phi z_i}$ and precision $\tau(1 - e^{-2\phi z_i})^{-1}$, where $z_i = t_i - t_{i-1}$ and $\tau = 2\phi/\sigma^2$. For this process the priors on the hyper-parameters are defined as follows: we first reparametrize to $\psi_1 = \log \tau$, $\psi_2 = \log \phi$ and assume $\psi_1 \sim \text{logGamma}(1, 5 \times 10^{-5})$, $\psi_2 \sim N(0, 0.2^{-1})$ (The R-INLA project, 2018).

***IG* process:** an independent Gaussian process $\delta_1, ..., \delta_T$ with $\delta_i \sim N(0, \tau^{-1})$. For this

process the priors on the hyper-parameters are defined as follows: we first reparametrize to $\psi = \log \tau$ and assume $\psi \sim \text{logGamma}(1, 5 \times 10^{-5})$ (The R-INLA project, 2018).

These different processes allow to model different spatial dependence structures of methylation rates along the genome as well as to account for the variance of observations which is not explained by the covariates. DBRM can be used to find the best pattern or combination of patterns for modeling this dependence in combination with deep feature engineering. The INLA approach (Rue et al., 2009) was used for obtaining marginal likelihoods in this example, hence the Gaussian priors

$$\boldsymbol{\beta}|\boldsymbol{\gamma} \sim N_{p_{\boldsymbol{\gamma}}}(\mathbf{0}, I_{p_\gamma} e^{\psi_{\beta_\gamma}}) \tag{37}$$

$$\psi_{\beta_\gamma} \sim \text{logGamma}(1, 5 \times 10^{-5}) \tag{38}$$

for the regression coefficients. We then use priors (33) for $\boldsymbol{\gamma}$. The priors for λ associated with selection of the latent Gaussian variables are of the form (39) giving equal prior probability to include any of the latent Gaussian variables

$$p(\boldsymbol{\lambda}) \propto \prod_{j=1}^{r} \exp(-2\lambda_j). \tag{39}$$

lammodel1

Here $\omega_j = -2 \forall j \in \{1, ..., r\}$, $b = e^{-1}$, and $v(\delta_j) = 1, \forall j \in \{1, ..., r\}$.

We consider $p = 14$ different covariates in addition to the intercept. Among these covariates we address a factor with 3 levels corresponding to whether a location belongs to a CGH, CHH or CHG genetic region, where H is either A, C or T and thus generating two covariates X_1 and X_2 corresponding to whether a location is CGH or CHH. The second group of factors indicates whether a distance to the previous cytosine nucleobase (C) in DNA is 1, 2, 3, 4, 5, from 6 to 20 or greater than 20 inducing the binary covariates $X_3 - X_8$. The third factor corresponds to whether a location belongs to a gene from a particular group of genes of biological interest, these groups are indicated as M_α, M_γ, M_δ or M_0 inducing 3 additional covariates $X_9 - X_{11}$. Finally, we have two cut offs a continuous predictor represented by an expression level for a nucleobase being either greater than 3000 or greater than 10000, defining binary covariates X_{12} and X_{13}. Finally X_{14} is the offset for the number of total bases per location offset(log(total.bases)).

As one can observe in Table 10 the offset for the total number of observations per location as well as two factors of whether the location is a CG or CHG are highly significant. Among the latent Gaussian variables only the random walk process of order one was found to be of importance. None of the engineered features were found of importance for this example. And this last result emphasizes the fact that even though the

Variable	Posterior
f(model="rw1")	1.00000e+00
offset(log(total.bases))	1.00000e+00
CG	9.990546e-01
CHG	9.515581e-01

Table 10: Features with posterior probability above 0.25 found by GMJMCMC with 16 threads for the epigenetic data example

defined feature space is higly non-linear the depth of the features is being adjusted up to a defined maximal depth and the suggested priors incorporate parsimonious penalization that only allow the features to go deeper in case the data drives them to do so.

5 Summary and discussion

In this article we introduced a new class of deep Bayesian regression models - DBRM to perform automated feature engineering and latent Gaussian variables selection in a Bayesian context of deep nonlinear models. The genetically modified MJMCMC approach (GMJMCMC) as well as its reversible modification (RGMJMCMC) are suggested for estimating posterior model probabilities and Bayesian model averaging and selection within the DBRM models. The algorithms combine the idea of keeping and updating the populations of highly predictive covariates combined with MJMCMC for the efficient exploration of these populations, however in the reversible version transitions between the populations are also constructed to satisfy the detailed balance equation. Based on several examples we have shown that the suggested approach can be efficient for both inference and prediction in various applications. The approach often requires significant resources in terms of computational time, hence embarrassing parallelization of GMJMCMC (RGMJMCMC) is suggested and used in some of the examples. It is in general recommended to use parallelization when the parallel resources are available. Memory efficient way to perform map and reduce steps within such parallelization is proposed. The *EMJMCMC* R-package is developed and currently available from the GitHub repository: http://aliaksah.github.io/EMJMCMC2016/. The developed package gives a user high flexibility in the choice of methods to obtain marginal likelihoods and model selection criteria for the class of DBRM models.

At the same time there are several important questions remaining for the general discussion. First of all, as we see it from the current perspective the suggested approach might well benefit from not restricting the model space by means of fixing values

of α and excluding them from the set of parameters. Exploring the full space of α vectors jointly with the models or marginalizing them out in a rigid way might well improve quality of the inference. The latter requires additional research and very likely extremely power intensive computations. This only seems feasible in case of effectively utilizing GPUs alongside with the CPUs, since extensive backpropagation steps are required for the computations of posterior modes of α vectors over all layers jointly. Since recently the latter is technically possible within R programming language due to adaptation of tensorflow package allowing for GPU computing, which is already available on CRAN (`https://cran.r-project.org/web/packages/tensorflow`). Recently a significant amount of research has been performed in the area of deep Bayesian learning (Pourzanjani et al.; Neal, 2012; Bhowmik et al.; Miller et al.) and people are beginning to see the advantages of proper statistical and probabilistic modeling. In most of these approaches different forms of variational Bayes (Blundell et al., 2015) are used to target the posterior distributions. Despite all of the VB theoretical limitations, it scales very well to extremely rich models and hence can also be used in our approach to jointly address α across all of the layers due to its computational feasibility.

Additionally, instead of making transitions between the populations within auxiliary states of large jumps of MCMC one might be interested in considering an extended model

$$\pi(\mathfrak{m}, \mathcal{S}) \equiv p(\mathfrak{m}|\mathbf{y})\pi(\mathcal{S}|\mathfrak{m})$$

where $\pi(\mathcal{S}|\mathfrak{m}) > 0$ only if $\mathfrak{m} \subset \mathcal{S}$. This extended distribution obviously has $p(\mathfrak{m}|\mathbf{y})$ as marginal for any choice of $\pi(\mathcal{S}|\mathfrak{m})$. The idea in this case is to switch between simulating $\mathfrak{m}|\mathcal{S}$ and $\mathcal{S}|\mathfrak{m}$.

Another important issue requiring discussion relies upon the fact that in machine learning applications one can be often interesting in minimizing a particular error measure $d(\mathbf{y}, \mathbf{x}|\mathfrak{m})$ of the model instead of getting the model averaged expectations of the quantile of interest. This measure is typically some prediction driven loss function in these applications. For such applications one can use some artificially defined distribution $\tilde{p}(\mathfrak{m}|\mathbf{y}) \propto \exp\left(-d(\mathbf{y}, \mathbf{x}|\mathfrak{m})\right)$ to be the target distribution. Exactly the same technique as the one described in (19) is then used to get the normalizing constant, however in these applications one is rather interested in the model that minimizes $d(\mathbf{y}, \mathbf{x}|\mathfrak{m})$ in the model space:

$$\mathfrak{m}^* = \underset{\mathfrak{m}\in\mathcal{M}}{\mathbf{argmax}}\ \tilde{p}(\mathfrak{m}|\mathbf{y}) = \underset{\mathfrak{m}\in\mathcal{M}}{\mathbf{argmin}}\ d(\mathbf{y}, \mathbf{x}|\mathfrak{m}). \tag{40}$$

Here $\mathfrak{m}^*$ is one best model of interest that minimizes the loss function of interest across the defined model space. Depending on the application the most popular loss functions include the mean squared error (MSE), the mean absolute error (MAE), area under curve

(AUC), precision (PREC) and others. The best model is normally approximated as $\underset{\mathfrak{m}\in\mathcal{M}^*}{\textbf{argmin}}\ d(\mathbf{y},\mathbf{x}|\mathfrak{m})$ in the explored part of the model space. Of course marginal inclusion probabilities within this approach only heuristically define feature importance in terms of the defined posterior distribution of the models. The same concerns all of the measures based on $\tilde{p}(\mathfrak{m}|\mathbf{y})$. Notice that this approach is perfectly possible within our framework (both theoretically and within the implementation), even though in the paper we were building a "proper" Bayesian approach based on the marginal likelihoods.

In addition, in the current notation the recurrent ANN structures are not exploited automatically unless the lags of interest (of both features and responses) are included manually in the set of input features. If one however extents the features to depend on all of the observations jointly, i.e. defining $F_j(\mathbf{x})$ instead of $F_j(\mathbf{x}_i)$ in equation (2), then with and $\text{lag}_p(x)$ operator, making a lag of the corresponding variable x, one is able to explore various recurrent structures in space in time automatically. The latter in combination with the latent Gaussian variables gives additional flexibility in modeling spatial-temporal relations. Whilst not to confuse the reader this more general notation is not introduced in (2), the approach is perfectly possible within our implementation if one uses one or several $\text{lag}_p(x)$ functions in $\mathcal{G}$.

Finally, an important issue left for discussion is how to manage large data samples (also known as Big Data) with the DBRM approach. In the loss function based approach discussed above one can simply rely upon bagging (Quinlan et al., 1996) idea when estimating loss based marginal probabilities of different models. However if one wants to be properly Bayesian, things turn to be slightly more sophisticated. Recently there have been many articles describing the possibility of sub-sampling combined with MCMC published (Quiroz et al., 2014, 2017, 2016; Flegal, 2012; Pillai and Smith, 2014). The aforementioned approaches suggest methods for estimating of the unbiased likelihood based on a sample of likelihoods based on subsamples of the whole large data set, which keep ergodicity and the desired limiting properties of the MCMC algorithm. These methods are not a part of the current implementation of DBRM, but our approach can relatively easily be adopted to allow sub-sampling MCMC techniques in the future.

SUPPLEMENTARY MATERIAL

R package: *R* package *EMJMCMC* for (R)(G)MJMCMC. (EMJMCMC_ 1.4.tar.gz; GNU zipped tar file)

Data and code: Data (simulated and real) and *R OOP* code for GMJMCMC algorithm, post-processing and creating figures wrapped together into a reference based

EMJMCMC class. (code-and-data.zip; zip file containing the data, code and a read-me file (readme.pdf))

Parameters pseudo codes: Tuning parameters for the addressed examples and pseudo codes for GMJMCMC and RJMCMCMC. (appendix.pdf)

ACKNOWLEDGMENTS

We would like to thank CELS project at the University of Oslo for giving us the opportunity, inspiration and motivation to write this article.

References

M. Adya and F. Collopy. How e! ective are neural networks at forecasting and prediction? a review and evaluation. **J. Forecasting**, 17:481–495, 1998.

M. Banterle, C. Grazian, A. Lee, and C. P. Robert. Accelerating metropolis-hastings algorithms by delayed acceptance. **arXiv preprint arXiv:1503.00996**, 2015.

D. Barber and C. M. Bishop. Ensemble learning in bayesian neural networks. **NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES**, 168:215–238, 1998.

T. Barrett, S. E. Wilhite, P. Ledoux, C. Evangelista, I. F. Kim, M. Tomashevsky, K. A. Marshall, K. H. Phillippy, P. M. Sherman, M. Holko, et al. NCBI GEO: archive for functional genomics data setsupdate. **Nucleic acids research**, 41(D1):D991–D995, 2013.

C. Becker, J. Hagmann, J. Müller, D. Koenig, O. Stegle, K. Borgwardt, and D. Weigel. Spontaneous epigenetic variation in the arabidopsis thaliana methylome. **Nature**, 480(7376):245–249, 2011.

A. Bhowmik, A. Adiga, C. Seelamantula, F. Hauser, J. Jacak, and B. Heise. Bayesian deep deconvolutional neural networks.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. **arXiv preprint arXiv:1505.05424**, 2015.

K. P. Burnham and D. R. Anderson. **Model Selection and Multimodel Inference: A Practical Information-Theoretical Approach (Second Edition)**. Springer-Verlag New York, Inc., 2002. ISBN 9780387224565.

S. Chib. Marginal likelihood from the gibbs output. **Journal of the American Statistical Association**, 90(432):1313–1321, 1995.

S. Chib and I. Jeliazkov. Marginal likelihood from the metropolis–hastings output. **Journal of the American Statistical Association**, 96(453):270–281, 2001.

J. A. Christen and C. Fox. Markov chain monte carlo using an approximation. **Journal of Computational and Graphical statistics**, 14(4):795–810, 2005.

M. A. Clyde, J. Ghosh, and M. L. Littman. Bayesian adaptive sampling for variable selection and model averaging. **Journal of Computational and Graphical Statistics**, 20(1):80–101, 2011.

G. Cybenko. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals and Systems**, 2(4):303–314, Dec 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL https://doi.org/10.1007/BF02551274.

J. S. Denker and Y. Lecun. Transforming neural-net output levels to probability distributions. In **Advances in neural information processing systems**, pages 853–859, 1991.

J. M. Flegal. Applicability of subsampling bootstrap methods in markov chain monte carlo. In **Monte Carlo and Quasi-Monte Carlo Methods 2010**, pages 363–372. Springer, 2012.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. **Annals of statistics**, pages 1189–1232, 2001.

N. Friel and J. Wyse. Estimating the evidence a review. **Statistica Neerlandica**, 66(3):288–308, 2012. ISSN 1467-9574.

Y. Gal. **Uncertainty in Deep Learning**. PhD thesis, University of Cambridge, 2016.

I. Goodfellow, Y. Bengio, and A. Courville. **Deep Learning**. MIT Press, 2016. http://www.deeplearningbook.org.

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

A. Hubin and G. Storvik. Efficient mode jumping MCMC for Bayesian variable selection in GLMM. Manuscript, 2016a.

A. Hubin and G. Storvik. Estimating the marginal likelihood with Integrated nested Laplace approximation (INLA), 2016b. arXiv:1611.01450v1.

A. Hubin, G. Storvik, and F. Frommlet. A novel algorithmic approach to bayesian logic regression. **arXiv preprint arXiv:1705.07616**, 2017.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. **Machine learning**, 37(2):183–233, 1999.

J. M. Kanter and K. Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In **Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on**, pages 1–10. IEEE, 2015.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. **Nature**, 521(7553):436–444, 2015.

D. J. MacKay. A practical bayesian framework for backpropagation networks. **Neural computation**, 4(3):448–472, 1992.

P. McCullagh and J. Nelder. **Generalized Linear Models. 2nd Edition**. Chapman and Hall, London, 1989.

D. Miller, L. Nicholson, F. Dayoub, and N. Sünderhauf. Dropout variational inference improves object detection in open-set conditions.

R. M. Neal. **Bayesian learning for neural networks**, volume 118. Springer Science & Business Media, 2012.

N. S. Pillai and A. Smith. Ergodicity of approximate mcmc chains with applications to large data sets. **arXiv preprint arXiv:1405.0182**, 2014.

A. A. Pourzanjani, R. M. Jiang, and L. R. Petzold. Improving the identifiability of neural networks for bayesian inference.

J. R. Quinlan et al. Bagging, boosting, and c4. 5. In **AAAI/IAAI, Vol. 1**, pages 725–730, 1996.

M. Quiroz, M. Villani, and R. Kohn. Speeding up mcmc by efficient data subsampling. **arXiv preprint arXiv:1404.4178**, 2014.

M. Quiroz, M. Villani, and R. Kohn. Exact subsampling mcmc. **arXiv preprint arXiv:1603.08232**, 2016.

M. Quiroz, M.-N. Tran, M. Villani, and R. Kohn. Speeding up mcmc by delayed acceptance and data subsampling. **Journal of Computational and Graphical Statistics**, 0(0):1–11, 2017. doi: 10.1080/10618600.2017.1307117. URL http://dx.doi.org/10.1080/10618600.2017.1307117.

M. A. Razi and K. Athappilly. A comparative predictive analysis of neural networks (nns), nonlinear

regression and classification and regression tree (cart) models. **Expert Systems with Applications**, 29(1):65–74, 2005.

A. N. Refenes, A. Zapranis, and G. Francis. Stock performance modeling using neural networks: a comparative study with regression models. **Neural networks**, 7(2):375–388, 1994.

G. O. Roberts and J. S. Rosenthal. Coupling and ergodicity of adaptive markov chain monte carlo algorithms. **J. Appl. Probab.**, 44(2):458–475, 03 2007. doi: 10.1239/jap/1183667414. URL http://dx.doi.org/10.1239/jap/1183667414.

G. O. Roberts and J. S. Rosenthal. Examples of adaptive mcmc. **Journal of Computational and Graphical Statistics**, 18(2):349–367, 2009. doi: 10.1198/jcgs.2009.06134. URL http://dx.doi.org/10.1198/jcgs.2009.06134.

P. Royston and D. G. Altman. Approximating statistical functions by using fractional polynomial regression. **Journal of the Royal Statistical Society: Series D (The Statistician)**, 46(3): 411–422, 1997.

H. Rue, S. Martino, and N. Chopin. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. **Journal of the Royal Statistical Sosciety**, 71 (2):319–392, 2009.

D. J. Sargent. Comparison of artificial neural networks with other statistical approaches. **Cancer**, 91 (S8):1636–1642, 2001.

The R-INLA project. Latent models, 2018. URL http://www.r-inla.org/models/latent-models.

L. Tierney and J. B. Kadane. Accurate approximations for posterior moments and marginal densities. **Journal of the american statistical association**, 81(393):82–86, 1986.

H. Tjelmeland and B. K. Hegstad. Mode jumping proposals in MCMC. **Scandinavian journal of statistics**, 28:205–223, 1999.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In **Proceedings of the 28th International Conference on Machine Learning (ICML-11)**, pages 681–688, 2011.

A Appendix

A.1 Tuning parameters of the algorithms used for predictions

In Table 11 we describe libraries and tuning parameters (different from the corresponding libraries' defaults) corresponding to the algorithms represented in Tables 2 and 3.

Algorithm	Library	Vectors of tuning parameters values
(R)GMJMCMC	emjmcmc	prior = aic.prior(), family = binomial(), n.models = 7000,unique = F, max.cpu = 4, max.cpu.glob = 4, create.table = F, create.hash = T, pseudo.paral = T, burn.in = 100, max.N.glob = 10, min.N.glob = 5, max.N = 3, min.N = 1, recalcmargin = 95, last.mutation=500, relations = c("cosi","sigmoid","tanh","atan","erf"), Nvars.max =15, relations.prob =c(0.1,0.1,0.1,0.1,0.1), mutationrate = 100, max.tree.size = 4, p.allow.replace=0.1,p.allow.tree=0.18, r=exp(-0.5)
MJMCMC	emjmcmc	prior = aic.prior(), family = binomial(), n.models = 450,unique = T, max.cpu = 4, max.cpu.glob = 4, create.table = F, create.hash = T, pseudo.paral = T, burn.in = 100, max.N.glob = 10, min.N.glob = 5, max.N = 3, min.N = 1, recalcmargin = 50
LXGBOOST	xgboost	objective = "binary:logistic", maxdepth = 15, evalmetric = "logloss", eta = 0.05, gamma = 0.005 subsample = 0.86, colsamplebytree = 0.92, colsamplebylevel = 0.9, minchildweight = 0, booster = "gblinear"
TXGBOOST	xgboost	objective = "binary:logistic", evalmetric = "logloss", booster = "gbtree", eta = 0.05, gamma = 0.005 subsample = 0.86, colsamplebytree = 0.92, colsamplebylevel = 0.9, minchildweight = 0, maxdepth = 15
LASSO	glmnet	family="binomial", alpha = 1
RIDGE	glmnet	family="binomial", alpha = 0
NBAYESS	h2o	-
DEEPNETS	h2o	hidden=c(200,200,200,200,200,200), distribution = "bernoulli"
RFOREST	h2o	stoppingmetric = "AUC", ntrees = 10000, stoppingrounds = 3, scoreeachiteration = T, ignoreconstcols = T
LR	h2o	family = "binomial", lambdasearch = F

Table 11: Different from default tuning parameters of the addressed in Tables 2 and 3 approaches.