Computer Assisted Mechanics and Engineering Sciences, **18**: 291–302, 2011.

Bayesian neural networks and Gaussian processes in identification of concrete properties

Marek Słoński
Cracow University of Technology
Institute for Computational Civil Engineering
Warszawska 24, 31-155 Kraków, Poland
e-mail: mslonski@L5.pk.edu.pl

This paper gives a concise overview of concrete properties prediction using advanced nonlinear regression approach and Bayesian inference. Feed-forward layered neural network (FLNN) with Markov chain Monte Carlo stochastic sampling and Gaussian process (GP) with maximum likelihood hyperparameters estimation are introduced and compared. An empirical assessment of these two models using two benchmark problems are presented. Results on these benchmark datasets show that Bayesian neural networks and Gaussian processes have rather similar prediction accuracy and are superior in comparison to linear regression model.

Keywords: nonlinear regression, Bayesian methods, concrete, neural network, Gaussian process.

1. Introduction

So far, various standard neural networks (SNNs) models have been applied in concrete properties prediction problems, see for example [4, 6, 11, 18]. Presented in these papers results, have proved that SNNs are very useful. But in general, the complexity of the data and/or task may make the development of such models by hand impractical. This is because complex models may easily overfit the noisy data and fail to infer the true process which generated the data. So, there is a strong need for regularization and complexity control methods. Standard approach is to manually select by trial and error the number of model parameters and/or by automatic regularization techniques. In this context, Bayesian inference offers a "principled" and practical approach, based on well established theoretical and statistical foundations [1].

In recent years, there has been growing interest in probabilistic methods for solving nonlinear regression problems. Today, probabilistic regression techniques, especially those associated with Bayesian inference, are gaining more and more popularity as a convenient tools for robust solving prediction problems which arise also in the context of concrete quality properties identification. One of the Bayesian regression models which have been successfully applied for concrete properties prediction are Bayesian neural network and Gaussian process [7, 13–15].

The main objective of this study is to introduce and compare the Bayesian neural network and Gaussian process for identification of mechanical concrete quality properties such as compressive strength of high-performance concrete and plain concrete fatigue failure.

The paper starts with a problem formulation. Then, the considered models are presented. Next, the description and analysis of three benchmark dataset and the experimental tests are given. Finally, the results are presented and conclusions are stated.

2. Nonlinear models for regression

In this paper, we consider the problem of concrete properties prediction as a nonlinear regression task, assuming independent, identically distributed (i.i.d.) data. For regression, we generally assume that the target is a noisy output of some unknown process described using functional relationship which we want to estimate. This assumption is mathematically described as

$$t_n = y(\mathbf{x}_n, \mathbf{w}) + \epsilon_n, \tag{1}$$

where t_n – output (target) variable for the n-th pattern, $y(\mathbf{x}, \mathbf{w})$ – regression function (model), $\mathbf{x}_n$ – vector of input variables for the n-th pattern, $\mathbf{w}$ – vector of adjustable parameters, ϵ – noise process.

2.1. Linear model

In this section, feed-forward neural network and Gaussian process models are introduced. More detailed description of these two models can be found in [17]. However, it may be useful to start by considering first a linear regression model. This model is a linear function in parameters $\boldsymbol{w}$ and nonlinear function of the input vector $\boldsymbol{x}$. In general, linear model is defined as a linear combination of fixed, nonlinear basis functions of the input variables:

$$y(\boldsymbol{x}; \boldsymbol{w}) = \sum_{m=1}^{M-1} w_m \phi_m(\mathbf{x}) + w_0, \tag{2}$$

where $\phi_m(\mathbf{x})$ are called basis functions. Many possible choices for the basis functions are possible. For example, polynomial basis functions or 'Gaussian' basis function.

Given N training patterns $\{(\mathbf{x}_n, t_n)\}_{n=1}^N$, parameters of the linear model are computed with penalized least squares (PLS) method, see [1]:

$$\mathbf{w}_{\mathrm{PLS}} = (\mathbf{\Phi}^{\mathbf{T}}\mathbf{\Phi} + \lambda \mathbf{I})^{-1}\mathbf{\Phi}^T \mathbf{t}, \tag{3}$$

where $\mathbf{\Phi}$ is $N \times M$ design matrix with elements Φ_{nm} defined as $\phi_m(\mathbf{x}_n)$. The regularization parameter λ has to be estimated using validation set or without validation set by applying Bayesian inference and maximizing evidence of dataset $p(\mathbf{t}|\alpha, \beta)$ w.r.t. hyperparameters α and β, where $\lambda = \alpha/\beta$.

Linear regression models with fixed basis functions, have some useful properties but they have also some limitations. They can be overcome by allowing the basis functions to be adaptive. This approach leads to feed-forward neural network (FLNN) model, called also as multilayer perceptron (MLP), see [3].

2.2. Feed-forward neural network

FLNN with one hidden layer of H adaptive units (neurons) can be described in functional form using the following equation

$$y(\mathbf{x}; \mathbf{w}) = \sum_{h=1}^{H} w_h^{(2)} g\left(\sum_{j=1}^{D} w_{hj}^{(1)} x_j + w_{0j}^{(1)}\right) + w_0^{(2)}, \tag{4}$$

where D is number of input variables and the set of all weight and bias parameters have been grouped together into a weight vector $\mathbf{w}$. Nonlinear function $g(\cdot)$ is an activation function of the hidden units and $\mathbf{w}$ is the parameters vector with: $w_{hj}^{(1)}$ being the first layer weights from the jth input to the hth hidden unit and $w_h^{(2)}$ being the second layer weights from hth hidden unit to the output. Finally, $w_{0j}^{(1)}$ and $w_0^{(2)}$ are the bias parameters for the hidden and output unit respectively. In this paper we use 'tanh' activation function.

One of the classical methods for determining the values of the parameter vector $\mathbf{w}$ uses the sum-of-squares error function defined by

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\Big(y(\mathbf{x}_n,\mathbf{w}) - t_n\Big)^2. \tag{5}$$

Minimizing this error function with respect to $\mathbf{w}$ gives a least-square (LS) estimate $\mathbf{w}_{\mathrm{LS}}$, which is used to make a prediction for a new value of $\mathbf{x}$ by evaluating $y(\mathbf{x};\mathbf{w}_{\mathrm{LS}})$.

A well-known problem with complex and flexible neural network models like FLNN is that they can over-fit the training data. It may be avoided by adding a penalty term to the error function (5) for penalizing large values for the network weights. This approach is known as penalized least squares (PLS) and a common regularizer is given by the sum of the squares of the weights:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\Big(y(\mathbf{x}_n,\mathbf{w}) - t_n\Big)^2 + \frac{\lambda}{2}||\mathbf{w}||^2, \tag{6}$$

where λ is a regularization coefficient typically estimated by a validation procedure.

2.3. Bayesian inference for FLNN

The Bayesian approach to neural networks learning and prediction processes is based on the Bayesian inference. This means the integration of the parameters of a neural model instead of searching for a single vector of the parameters. In this approach all parameters are treated as random variables.

The prior distribution $p(\mathbf{w}|\alpha)$ over weights is firstly defined. It is generally assumed that the prior distribution is a spherical Gaussian distribution with the zero mean and inverse variance hyperparameter α. A Gaussian noise model $p(\epsilon_n|\beta)$ is also adopted with the inverse of variance parameter called precision and defined by $\beta = 1/\sigma^2$.

After observing the data set $\mathbf{t}$, Bayes' theorem is used to update the posterior probability distribution over weights

$$p(\mathbf{w}|\mathbf{t},\alpha,\beta) = \frac{p(\mathbf{t}|\mathbf{w},\beta)p(\mathbf{w}|\alpha)}{p(\mathbf{t}|\alpha,\beta)}, \tag{7}$$

where $p(\mathbf{t}|\mathbf{w},\beta)$ is the likelihood function, which for independent and identically distributed (i.i.d.) data set is defined as

$$p(\mathbf{t}|\mathbf{w},\beta) = \prod_{n=1}^{N} p(t_n|\mathbf{w},\beta) = \prod_{n=1}^{N}(2\pi\sigma^2)^{-1/2}\exp\left[-\frac{\{t_n - y(\mathbf{x}_n;\mathbf{w})\}^2}{2\sigma^2}\right], \tag{8}$$

where $p(\mathbf{t}|\alpha,\beta)$ is a normalizing factor (evidence for hyperparameters) of the form

$$p(\mathbf{t}|\alpha,\beta) = \int p(\mathbf{t}|\mathbf{w},\beta)p(\mathbf{w}|\alpha)d\mathbf{w}. \tag{9}$$

Taking the negative logarithm of (7), we can find the maximum a posteriori (MAP) estimate of $\mathbf{w}$, which is equivalent to minimizing (6) with a regularization coefficient given by $\lambda = \alpha/\beta$ [1].

For regression problems, what is of main interest is making the prediction of t_{N+1} for a new value of $\mathbf{x}_{N+1}$. In Bayesian approach, instead of point prediction, the predictive distribution over target variable t_{N+1} is computed applying the sum rule of probability and marginalizing out the weight vector $\mathbf{w}$. This leads to the following equation

$$p(t_{N+1}|\mathbf{x}_{N+1}, \mathbf{t}, \alpha, \beta) = \int p(t_{N+1}|\mathbf{x}_{N+1}, \mathbf{w}, \beta) p(\mathbf{w}|\mathbf{t}, \alpha, \beta) d\mathbf{w}, \tag{10}$$

assuming that the hyperparameters α and β are known.

In the fully Bayesian framework, also the uncertainty over the hyperparameters should be taken into account by defining the prior distributions $p(\alpha)$ and $p(\beta)$ which are called hyperpriors. Then the full posterior distribution is defined by

$$p(\mathbf{w}, \alpha, \beta|\mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{w}, \beta) p(\mathbf{w}|\alpha) p(\alpha) p(\beta)}{p(\mathbf{t})}. \tag{11}$$

Finally, the fully Bayesian prediction is computed evaluating the predictive distribution of the form

$$p(t_{N+1}|\mathbf{x}_{N+1}, \mathbf{t}) = \int p(t_{N+1}|\mathbf{x}_{N+1}, \mathbf{w}, \beta) p(\mathbf{w}, \alpha, \beta|\mathbf{t}) d\mathbf{w}\; d\alpha\; d\beta, \tag{12}$$

using the posterior distribution $p(\mathbf{w}, \alpha, \beta|\mathbf{t})$.

Because the Bayesian prediction is based on integrations in the parameter and hyperparameter space over all parameters and hyperparameters, in general it is analytically intractable and the approximation techniques have to be used.

2.3.1. The evidence approximation

The evidence approximation method is based on iterative algorithm for determining optimal hyperparameters α and β and weights $\mathbf{w}$. This procedure leads to the following formulae for finding the optimal values of the hyperparameters

$$\alpha^{new} = \frac{\gamma}{2E_W}, \qquad \beta^{new} = \frac{N-\gamma}{2E_D}, \qquad \gamma = \sum_{i=1}^{M} \frac{\lambda_i}{\lambda_i + \alpha}, \tag{13}$$

where γ is the number of well-determined parameters and λ_i is the eigenvalue of the Hessian $\mathbf{H}$ matrix, evaluated at $\mathbf{w} = \mathbf{w}_{MP}$. More details on the evidence approximation method can be found in [1].

2.3.2. Markov chain Monte Carlo sampling

In this approach, Markov chain Monte Carlo (MCMC) method is used to approximate the integrals (12) by the finite sum

$$p(t_{N+1}|\mathbf{x}_{N+1}, \mathbf{t}) \approx \frac{1}{m} \sum_{i=1}^{m} p(t_{N+1}|\mathbf{x}_{N+1}, \mathbf{w}_i), \tag{14}$$

where $\mathbf{w}_i$ are samples generated from the posterior distribution $p(\mathbf{w}|\mathbf{t})$. For BNN, the hybrid Monte Carlo (HMC) algorithm is most often applied [9, 10]. This method combines the Metropolis-Hastings algorithm with the dynamical approach to stochastic sampling based on Hamiltonian dynamics [1]. This allows us to incorporate gradient information from the posterior distribution, which means that exploration of the sample space is more effective than completely stochastic approach such as Metropolis-Hastings algorithm [8].

2.4. Gaussian process

Gaussian process (GP) model can be obtained by reformulation of linear model in terms of dual representation. In this approach, linear model is trained by minimizing a regularized error function, defined using Gram matrix $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$. The Gram matrix is $N \times N$ symmetric matrix with elements

$$K_{nm} = \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{\phi}(\boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m), \tag{15}$$

where $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$ is a kernel function. The design matrix is an $N \times M$ matrix with elements $\Phi_{nm} = \phi_m(x_n)$. The vector $\boldsymbol{k}(\boldsymbol{x})$ is defined with elements $k_n(\boldsymbol{x}) = k(\boldsymbol{x}_n, \boldsymbol{x})$.

The prediction for a new input $\boldsymbol{x}$ is obtained from

$$y(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T (\boldsymbol{K} + \lambda \boldsymbol{I}_N)^{-1} \boldsymbol{t}, \tag{16}$$

where $\boldsymbol{t} = (t_1, \ldots, t_N)^T$ is a vector of training target values, $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$ is the Gram matrix and $\boldsymbol{\Phi}$ is the design matrix. From the Bayesian point of view of linear model, dual representation approach leads to the Gaussian process model, where the kernel function is interpreted as a covariance function of the Gaussian process. Application of GP regression model for prediction allows to compute the predictive distribution of the target variable t_{N+1} for a new input vector $\boldsymbol{x}_{N+1}$. This requires evaluation of conditional distribution $p(t_{N+1}|\boldsymbol{t}_N)$, where $\boldsymbol{t}_N$ is a vector of training target values. This conditional distribution for the Gaussian processes is a Gaussian distribution with mean and covariance given by

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}, \tag{17}$$

$$\sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}, \tag{18}$$

where the $\boldsymbol{C}_N$ is the $N \times N$ covariance matrix with elements given by a sum of two terms: the covariance function $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ and the Gaussian noise component represented by a precision β

$$C(\boldsymbol{x}_n, \boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \beta^{-1}\delta_{nm}. \tag{19}$$

The vector $\boldsymbol{k}$ has elements $k(\boldsymbol{x}_n, \boldsymbol{x}_{N+1})$ and the scalar c is $k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$. From Eqs. (17) and (18) we see that the Gaussian process regression model is completely defined by the covariance function $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$. This function allows us to define the situation that for the nearby points $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ in the input space, the corresponding values $y(\boldsymbol{x}_n)$ and $y(\boldsymbol{x}_m)$ will be more strongly correlated than for dissimilar points [1].

The covariance function can be any function that will generate a non-negative definite covariance matrix for any ordered set of (input) vectors $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.It is usual to choose the covariance function to be stationary, i.e. such that the condition

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x} - \boldsymbol{x}') \tag{20}$$

holds. This means that the location of the points $\boldsymbol{x}$ and $\boldsymbol{x}'$ does not affect their covariance, just the vector joining them. In this paper we use squared exponential (SE) covariance function

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \theta_0 \exp\left(-\frac{1}{2}\sum_{i=1}^{d} \eta_i (x_{ni} - x_{mi})^2\right) + \theta_2, \tag{21}$$

which is the exponential of a weighted squared distance between points in $\mathbb{R}^d$. The SE covariance function has some free parameters which are called hyperparameters to emphasize that they are parameters of a non-parametric model. The term θ_2 controls the vertical offset of the GP model, while θ_0 controls the vertical scale of the process. The η_i hyperparameters allow a different distance measure for each dimension.

After defining the covariance function we can make predictions for the new input vectors but it is often necessary to learn the hyperparameters before making reliable prediction. The simplest approach is similar to the evidence approximation discussed above. For Gaussian process regression, we find the most probable hyperparameters of the covariance function by maximizing the log likelihood function given by

$$\ln p(\mathbf{t}|\theta) = -\frac{1}{2}\ln|\mathbf{C}_N| - \frac{1}{2}\mathbf{t}^T\mathbf{C}_N^{-1}\mathbf{t} - \frac{N}{2}\ln(2\pi) \tag{22}$$

using gradient-based optimization algorithms such as conjugate gradients. More details on Gaussian process models can be found in Bishop's book [1] and in the book by Rasmussen and Williams [12].

3. Hpc compressive strength prediction

This dataset was collected from various papers by Kasperkiewicz et al. and described in [6]. It contains the total number of $N = 346$ mix designs. Each experimental sample consists of a high-performance concrete (HPC) mix proportion and is defined by the amounts in kg/m^3 of six ingredients: cement (C), water (W), silica (S), superplasticizer (Su), fine aggregate (FA) and coarse aggregate (CA). These amounts are collected in the input vector $\mathbf{x} = \{C, W, S, Su, FA, CA\}$ and the corresponding target value is set to be a 28-day compressive strength of HPC $t = f'_c$, in MPa.

3.1. HPC dataset visualization and analysis

Before applying selected models, this database is visualized and analyzed for better understanding of the hidden relations between given input and output variables. In Table 1, the summary statistics such as range, median, mean and standard deviation for each variable are presented and Fig. 1 presents six scatter plots of the output variable against each input variable. Finally, Fig. 2 shows a five-number summary in the form of a box-and-whisker diagram.

Table 1. Statistical properties of input and output variables.

Number	Variable	Range	Median	Mean	St.dev.
1	C [kg/m^3]	94–1585	449	473	214
2	W [kg/m^3]	61–540	160	178	79
3	S [kg/m^3]	0–298	22.5	32.8	41.2
4	Su [kg/m^3]	0–38.1	7.6	9.1	7.6
5	FA [kg/m^3]	0–1760	750	769	267
6	CA [kg/m^3]	0–1443	1038	900	350
7	f'_c [MPa]	2.8–135	71.9	72.2	26.8

3.2. Numerical experiments

Computer simulations for FLNN standard neural network, true Bayesian (TBNN) neural network and Gaussian process (GP) were carried out using $L = 226$ learning examples and $T = 114$ testing patterns. They are presented in Fig. 1.

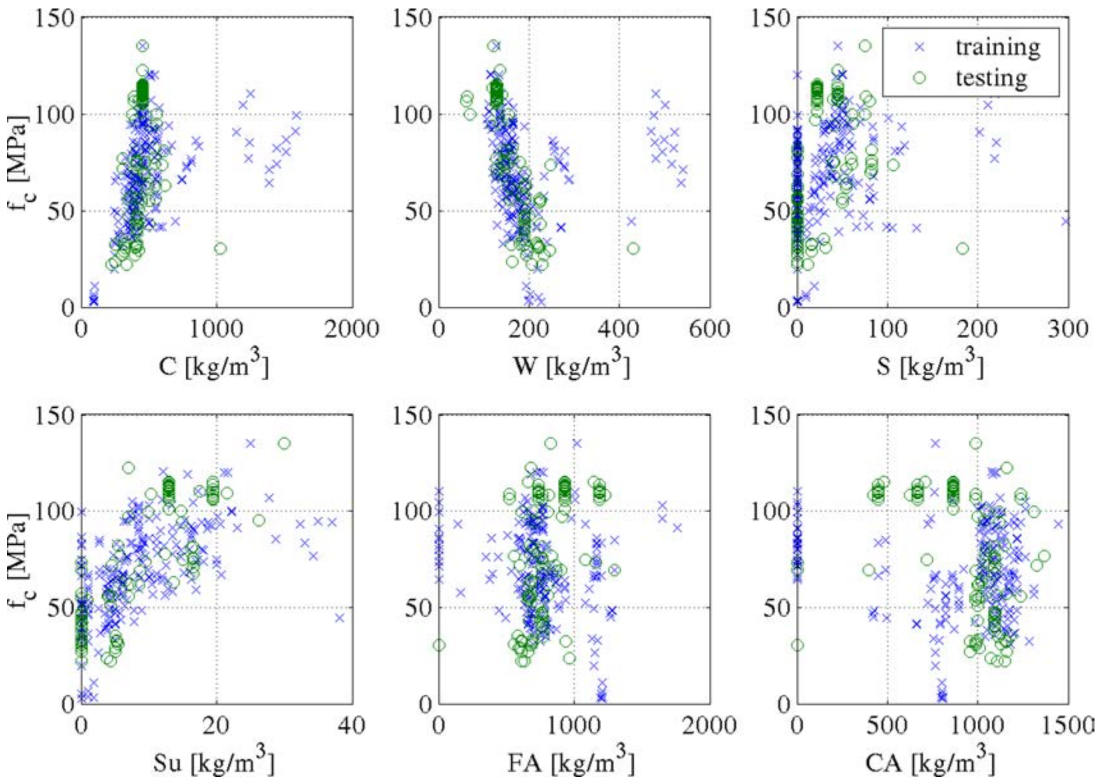


Fig. 1. A 28-day compressive strength f_c' vs. each input variable. From left to right and top to bottom: C, W, S, Su, CA and FA.

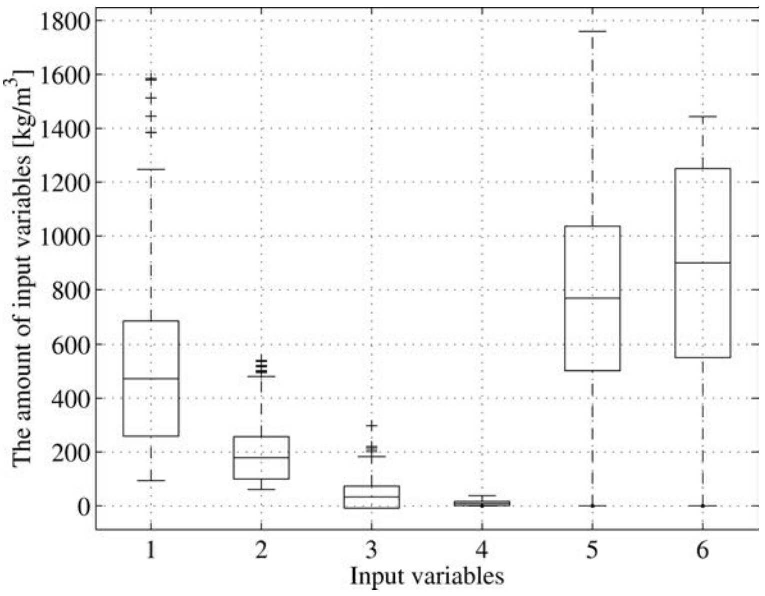


Fig. 2. A box-and-whisker plot for all input variables in HPC dataset.

3.3. Neural networks

The FLNN with single hidden layer of 10 hidden neurons was used to model the relationship between the inputs and the output. Two approaches to learning and prediction described in Sec. 2 were applied. Training of FLNN model was done within 20 epochs of scaled conjugate gradient (SCG) optimization method using Netlab toolbox for MATLAB [8].

True Bayesian neural network was defined having the same architecture as for SNN and isotropic Gaussian prior for weights was assumed. Also the noise model was defined as a normal distribution with the variance hyperparameter. Inference and prediction with TBNN model was done by using Gibbs sampler and 10000 steps of Hybrid Monte Carlo (HMC) algorithm, implemented in MCMCstuff toolbox for MATLAB [16]. The initial network weights were sampled from a zero mean spherical Gaussian distribution with variance one.

3.4. Gaussian process

A Gaussian process using squared exponential (SE) covariance function was defined and simulated with Netlab toolbox [8]. The hyperparameters of the covariance function were computed using maximum likelihood approach described in Sec. 2. The scaled conjugate gradient optimization algorithm was applied with only 4 iterations.

3.5. Results and discussion

In this section, the results for multiple linear regression (MLR), standard neural network (FLNN), true Bayesian neural network (TBNN) and Gaussian process (GP) models are presented and discussed. The models are compared on the base of the following error formulae:
– root-mean-squared (RMS) error

$$\mathrm{RMS} = \sqrt{\frac{1}{V}\sum_{n=1}^{V}(t_n - y_n)^2}; \tag{23}$$

– average percentage (AP) error

$$\mathrm{AP} = \frac{1}{V}\sum_{n=1}^{V}\left|\frac{t_n - y_n}{t_n}\right| \cdot 100\%. \tag{24}$$

Also the coefficient of correlation r was computed

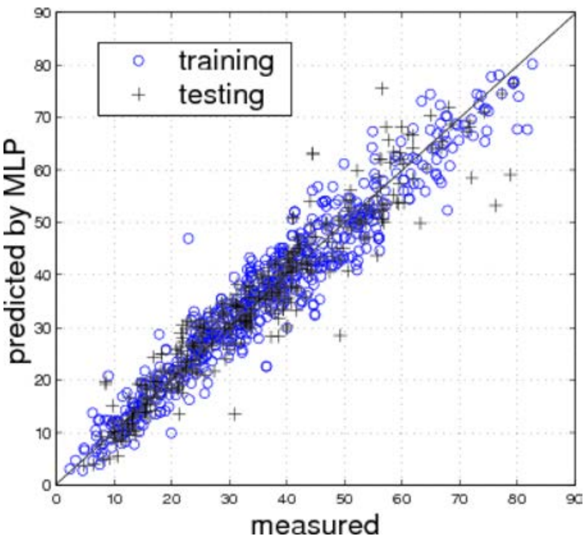
$$r = \frac{\sum\limits_{n=1}^{V}(t_n - \overline{t})(y_n - \overline{y})}{\sum\limits_{n=1}^{V}(t_n - \overline{t})\sum\limits_{n=1}^{V}(y_n - \overline{y})}, \tag{25}$$

where $\overline{t}$ and $\overline{y}$ are mean values of targets t_n and predicted values y_n, respectively.

In Table 2, the results for the considered models are shown. The first three columns contain information about models (shortened name, number of model parameters (NMPs) and number of model hyperparameters (NMHs)). In the fourth column, the CPU average learning time in seconds is given. These times where obtained using Matlab installed on the server equipped with two six core Intel Xeon X5650 processors. In the remaining columns, the root-mean-squared errors, average percentage errors and the Pearson's correlation coefficient for learning (L) and testing (T) patterns are shown. Comparing the predictive performance of TBNN and GP on testing dataset shown in Table 2, it can be seen that the best results are obtained for the true Bayesian neural network. In Fig. 3, the measured compressive strengths values are compared with the predicted values for both training and testing patterns. The plot on the left shows results for TBNN model and the right-hand plot shows results for GP model. Both plots are quite similar, showing that these models have similar prediction accuracy with respect to the training and the testing patterns, respectively.

Table 2. Results of learning (L) and testing (T) for considered models. From left to right: model name, number of model parameters (NMPs), number of model hyperparameters (NMHs), CPU time for learning, root-mean-squared error (RMSE), average percentage error (APE) and Pearson's correlation coefficient (r).

Model	NMPs	NMHs	CPUtime(L) [s]	RMS(L)	APE(L)	r(L)	RMS(T)	APE(T)	r(T)
MLR	7	0	0.003	12.92	15.7%	0.861	13.13	16.3%	0.860
FLNN	81	1	13.8	7.75	9.0%	0.953	9.57	10.8%	0.928
GP	0	9	0.145	**5.43**	6.6%	**0.978**	9.52	11.4%	0.930
TBNN	81	11	$19 \cdot 10^3$	5.51	**6.5%**	0.976	**8.34**	**10.7%**	**0.947**



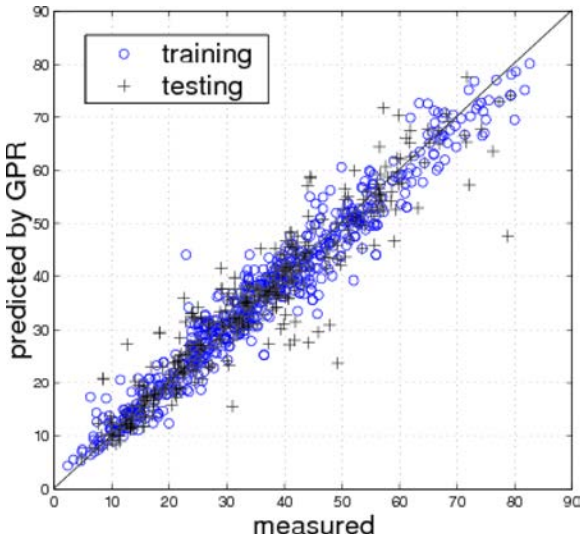


Fig. 3. Predicted HPC compressive strengths vs measured values; by the true Bayesian neural network (left) and using Gaussian process (right).

4. Concrete fatigue failure

Concrete fatigue failure can be defined as a number of loading cycles N causing fatigue damage of plain concrete specimens. The problem of predicting concrete fatigue failure was formulated as a Bayesian regression problem. We assumed that the fatigue failure is a sum of an underlying deterministic function $y(\boldsymbol{x})$ and a random variable ϵ. The input vector $\boldsymbol{x}$ consists of four variables, namely concrete static uniaxial compressive strength (f_c), ratio of minimal and maximal stress level in compressive cycle of loading ($R = \sigma_{\min}/\sigma_{\max}$), ratio of compressive fatigue and static strength of concrete, also called maximal compressive stress level ($\chi = f_{cN}/f_c$) and frequency of the loading cycle (f). The target variable is the scalar output $y = \log N$.

4.1. Dataset description and analysis

In Ref. [2] a wide experimental evidence was described and compiled, corresponding to more than 400 tests performed in 14 laboratories. The concrete specimens were subjected to cycles of compressive loadings and the numbers of cycles N which caused the specimens fatigue damage were measured. In this paper we used only $P = 218$ results (examples) of tests from 8 laboratories mentioned in [4] and [5]. For example, in Fig. 4 the results of two various fatigue tests on concrete samples are presented. In Table 3 the statistical parameters, namely minimal and maximal values, mean values and standard deviations for inputs and output variables are shown.

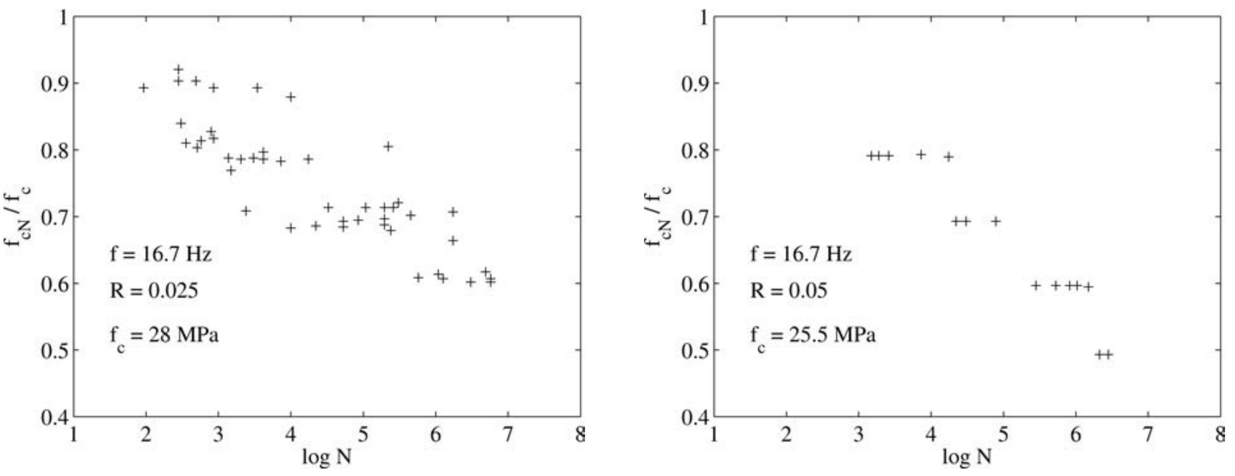


Fig. 4. Results of fatigue tests on concrete samples given by: left) Artim and McLaughlin (1959), right) Gray et al. (1961) taken from [2].

Table 3. Statistical parameters of input and output variables.

Variable	Min	Max	Mean	St.Dev.
f_c [MPa]	20.70	45.20	34.68	8.84
R [–]	0.00	0.88	0.14	0.18
f [Hz]	0.025	150.0	21.30	39.38
χ [–]	0.49	0.94	0.74	0.11
$\log N$ [–]	1.86	7.34	4.56	1.41

4.2. Numerical experiments

Numerical experiments using both standard (FLNN) and true Bayesian (TBNN) neural networks and Gaussian process (GP) for fatigue failure dataset are compared. The learning was performed using $L = 146$ learning examples. The generalization capability of these models for predicting concrete fatigue failure was estimated using $T = 72$ testing examples. In the analysis, only for FLNN neural network and Gaussian process, both the inputs and output variables were first standardized to zero mean and unit standard deviation.

4.3. Neural networks

A FLNN with a single hidden layer of hyperbolic tangent units (neurons) and linear output unit was used to model the relationship between the inputs and the output variables. On the basis of the preliminary analysis, neural network with only 7 hidden units was designed. This number of units gives the neural models sufficient flexibility for approximating the relationship.

Having defined the architecture of the neural network model, two approaches for learning and prediction described in Sec. 2 were applied. FLNN model was trained by minimization of the regularized error function given in (6), using scaled conjugate gradient (SCG) optimization method. In this approach, the value of the regularization parameter λ was estimated using the cross-validation. The learning process was stopped after only 20 epochs and the computations were performed using Netlab toolbox for MATLAB [8].

In the next step, true Bayesian neural network (TBNN), described shortly in Sec. 2 with the same architecture was applied. In experiments, spherical (isotropic) Gaussian prior distributions for weights were defined. The normal noise model was also defined with the variance hyperparameter.

Learning and prediction TBNN model was done using Gibbs sampling for the hyperparameters and 600 steps of Hybrid Monte Carlo (HMC) algorithm for the weights. In the Bayesian computations, MCMCstuff toolbox for MATLAB [16] was used.

4.4. Gaussian process

A Gaussian process with a squared exponential covariance function was applied using Netlab toolbox [8]. The hyperparameters of the covariance function were estimated using maximum likelihood approach described in Sec. 2 and the scaled conjugate gradient (SCG) optimization algorithm with only 2 iterations.

4.5. Results and discussion

In this section, the results for Gaussian process and true Bayesian neural network (TBNN) are presented and discussed. Table 4 shows the results for the applied models. Comparing the predictive performance of TBNN and GP on testing dataset shown in Table 4, it is seen that the best results are obtained for the true Bayesian neural network. These results are slightly better in comparison with GP model and standard neural network (SNN).

Table 4. Results of learning (L) and testing (T) for considered models. From left to right: model, number of model parameters (NMPs), number of model hyperparameters (NMHs), CPU time for learning, root-mean-squared error (RMSE), average percentage error (APE) and Pearson's correlation coefficient (r).

Model	NMP	NMH	CPUtime(L) [s]	RMS(L)	APE(L)	r(L)	RMS(T)	APE(T)	r(T)
LRM	5	0	0.003	0.75	14.7%	0.848	0.78	15.6%	0.828
FLNN	43	1	22	0.64	12.4%	0.894	0.72	14.0%	0.857
GP	0	7	0.132	**0.61**	**11.9%**	**0.904**	**0.70**	**13.3%**	0.866
TBNN	43	9	$15 \cdot 10^3$	0.64	12.4%	0.893	**0.70**	**13.3%**	**0.868**

5. Final remarks

In this paper, Bayesian neural networks and Gaussian processes were applied to the analysis of concrete properties identification. They were compared on the basis of two databases of experimental results. The first problem was to predict the 28-day compressive strength of high-performance concrete based on the mix proportions. In the second problem, the task was to identify the fatigue failure of plain concrete defined as a number of loading cycles causing fatigue damage of a specimen, with respect to the four input variables containing information about the loading cycles and static uniaxial compressive strength.

Results of the experiments show that both models give very similar prediction accuracy for both datasets but the computational cost of learning these two models is very different. In this context, Gaussian process model should be recommended as a better choice for solving the problem of concrete properties prediction.

Acknowledgements

Author would like to acknowledge support from the Polish Ministry of Science and Higher Education Grant "Applications of Bayesian machine learning methods in identification problems of experimental mechanics of materials and structures", No. N N506 250938.

References

[1] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[2] K. Furtak. Strength of the concrete under multiple repeated loads [in Polish]. *Arch. of Civil Eng.*, **30**, 1984.

[3] S. Haykin. *Neural Networks, a comprehensive foundation*. Prentice Hall, 1999.

[4] M. Jakubek and Z. Waszczyszyn. Neural analysis of concrete fatigue durability by the neuro-fuzzy FWNN. In L. Rutkowski, J. Siekmann, R. Tadeusiewicz, and Lotfi A. Zadeh [Eds.], *Artificial Intelligence and Soft Computing – ICAISC 2004*, Lecture Notes in Artificial Intelligence. Springer Berlin/Heidelberg, 2004.

[5] J. Kaliszuk, A. Urbańska, Z. Waszczyszyn, and K. Furtak. Neural analysis of concrete fatigue durability on the basis of experimental evidence. *Arch. of Civil Eng.*, **38**, 2001.

[6] J. Kasperkiewicz, J. Racz, and A. Dubrawski. HPC strength prediction using artificial neural network. *Journal of Computing in Civil Engineering*, **9**(4): 1–6, 1995.

[7] J. Lampinen and A. Vehtari. Bayesian approach for neural networks – review and case studies. *Neural Networks*, **14**(3): 7–24, April 2001. (Invited article).

[8] I.T. Nabney. *Netlab: Algorithms for Pattern Recognition*. Springer, London, 2002.

[9] R.M. Neal. Bayesian training of backpropagation networks by the hybrid Monte Carlo method. Technical Report CRG-TR-92-1, 1992.

[10] R.M. Neal. *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics 118. Springer, 1996.

[11] J.W. Oh, I.W. Lee, J.T. Kim, and G.W. Lee. Application of neural networks for proportioning of concrete mixes. *ACI Materials Journal*, **96**: 61–67, 1999.

[12] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2006.

[13] M. Słoński. Bayesian regression approaches on example of concrete fatigue failure prediction. *Computer Assisted Mech. Eng. Sci.*, **13**(4): 655–668, 2006.

[14] M. Słoński. HPC strength prediction using Bayesian neural networks. *Computer Assisted Mech. Eng. Sci.*, **14**(1), 2007.

[15] M. Słoński. A comparison of model selection methods for compressive strength prediction of high-performance concrete using neural networks. *Computers & Structures*, **88**(21–22): 1248–1253, 2010.

[16] A. Vehtari. MCMCstuff toolbox for MATLAB. User Manual, 2006.

[17] Z. Waszczyszyn and M. Słoński. Some problems of artificial neural networks design. In Z. Waszczyszyn [Ed.], *Advances of Soft Computing in Engineering*, volume 512 of *CISM Lectures and Notes*, pages 237–316. Springer Wien New York, 2010.

[18] I-Cheng Yeh. Design of high-performance concrete mixture using neural networks and nonlinear programming. *Journal of Computing in Civil Engineering*, **13**(1): 36–42, 1999.