Bayesian Random Vector Functional-Link Networks for Robust Data Modeling

Simone Scardapane, *Member, IEEE,* Dianhui Wang, *Senior Member, IEEE,* and Aurelio Uncini, *Member, IEEE*

***Abstract*—Random vector functional-link (RVFLs) networks are randomized multi-layer perceptrons (MLPs) with a single hidden layer and a linear output layer, which can be trained by solving a linear modeling problem. In particular, they are generally trained using a closed-form solution of the (regularized) least-squares approach. This paper introduces several alternative strategies for performing full Bayesian inference (BI) of RVFL networks. Distinct from standard or classical approaches, our proposed Bayesian training algorithms allow to derive an entire probability distribution over the optimal output weights of the network, instead of a single pointwise estimate according to some given criterion (e.g., least-squares). This provides several known advantages, including the possibility of introducing additional prior knowledge in the training process, the availability of an uncertainty measure during the test phase, and the capability of automatically inferring hyper-parameters from given data. In this paper two BI algorithms for regression are firstly proposed that, under some practical assumptions, can be implemented by a simple iterative process with closed-form computations. Simulation results show that one of the proposed algorithms, B-RVFL, is able to outperform standard training algorithms for RVFL networks with a proper regularization factor selected carefully via a line search procedure. A general strategy based on variational inference is also presented, with an application to data modeling problems with noisy outputs or outliers. As we discuss in the paper, using recent advances in automatic differentiation this strategy can be applied to a wide range of additional situations in an immediate fashion.**

***Index Terms*—Random vector functional-link, Bayesian inference, Relevance vector machine, Variational inference**

I. Introduction

Random vector functional-link (RVFL) networks [1], [2], [3] are a powerful tool for solving data modeling problems with moderate complexity. In RVFL networks, the learning problem is recast as a linear modeling task by projecting the input to a high-dimensional space through a set of random basis functions, and the values of the randomized parameters in the basis functions are kept fixed during the training stage. Although a specific draw of these random bases might prove sub-optimal, RVFL networks perform soundly in practice, allowing them to achieve state-of-the-art results in several tasks including, but not limited to, conditional probability modeling [4], nonlinear control [5], image recognition [6], [7], and learning over sensor networks [8].

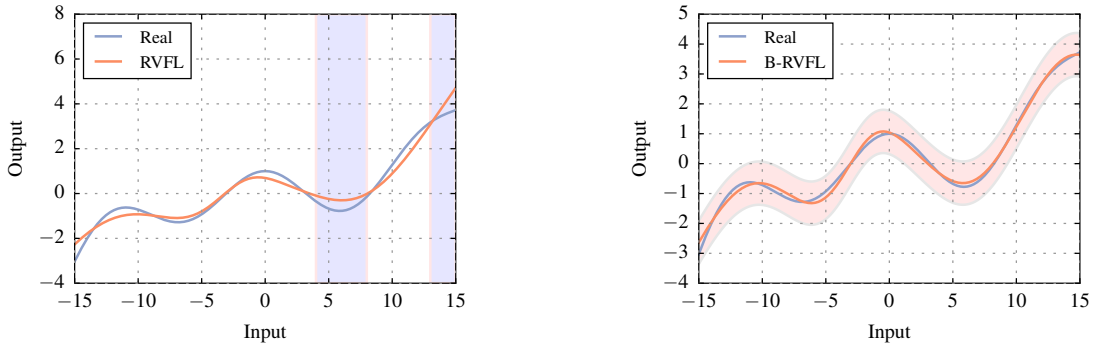
Simone Scardapane and Aurelio Uncini are with the Department of Information Engineering, Electronics and Telecommunications (DIET), "Sapienza" University of Rome, Via Eudossiana 18, 00184, Rome. Dianhui Wang is with the Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC 3086, Australia. Email: {simone.scardapane,aurelio.uncini}@uniroma1.it; dh.wang@latrobe.edu.au.

Corresponding Author: Simone Scardapane.

The major strength of RVFL networks is the possibility to immediately leverage over decades of research on linear regression and classification, granting them a simple closed-form solution when optimizing a squared loss function with ℓ_2 regularization, fast linear algebra libraries, and an immediacy of implementation. Due to this, in recent years we have witnessed an increasing number of works aiming at further improving the performance of these models. Among them, we can cite the use of sparse formulations [9], semi-supervised settings [10], and many more. An insightful editorial on randomized methods for training neural networks can be found in [11]. For a more complete exposition on random assignment of weights in neural networks, see also the recent survey in [12].

All works published on RVFL networks up to date are framed in classical supervised learning, where we generally search for a reasonably good estimate of its weights, satisfying some (generally convex) training criterion, such as regularized least-squares. In recent years, however, we are witnessing a renovated increase of interest in an alternative approach to learning, termed Bayesian inference (BI), where we look instead for an entire probability distribution over the weights of the learning model, by properly combining our beliefs on the model and the data using the Bayes theorem [13], [14]. BI, which has a long history in the machine learning field, provides several advantages with respect to classical training procedures, such as the possibility of introducing additional prior knowledge in the training process (in the form of our beliefs), the availability of an uncertainty measure during the test phase, and the capability of automatically inferring hyper-parameters from the data. The downside is that the optimization problems involved in BI are non-convex and (in most cases) analytically intractable, thus requiring approximate inference algorithms (e.g. variational inference [15]), or the use of expensive Monte Carlo sampling procedures. Classical models framed in BI are relevance vector machines (RVMs) [16], Latent Dirichlet algorithms [17], and Gaussian Processes [18], [19]. Over the last few years, BI has been applied increasingly for the training of deep networks, e.g., [20], [21], [22], [23]. This is a direct result of similar advancements in theoretical tools for performing large-scale approximate inference [17] and, more importantly, the recent availability of software tools for performing *automatic* inference on these models [24], [25], making BI an attractive set of algorithms in practice.

Among all the works mentioned above, only a single work provides a Bayesian interpretation of training RVFL networks [26], which was limited to a maximum *a posteriori* (MAP) pointwise estimate for a robust loss function. In this paper,



(a) Standard RVFL [MSE = 0.096]

(b) Bayesian RVFL [MSE = 0.013]

Fig. 1. Comparison of a standard RVFL network and a Bayesian RVFL network (B-RVFL) on a toy dataset. (a) Results of the standard RVFL, where we highlight in light blue two regions where the network is not generalizing correctly. (b) Results of B-RVFL, where we represent ± 1 standard deviation with a light red.

we are interested in performing full BI, thus providing an entire distribution of the optimal weights. As one of the motivations behind this study, Fig. 1 depicts a toy example, where we consider a $1D$ toy problem given by $f(x) = \cos(x/2) + 0.001x^3$. We train on a dataset of 5000 points randomly drawn in $[-15, 15]$, and we test on an independent set of 1000 elements. We also corrupt the training set with random Gaussian noise with variance $\sqrt{0.5}$. In Fig. 1a, we train a standard RVFL network with 15 hidden neurons using the classical least-squares procedure (see Section II-A), where the regularization coefficient is optimized following the line search procedure described in our experimental section and the random weights are extracted from the uniform distribution in $[-1, 1]$. As can be seen, the resulting mean-squared error (MSE) over the independent test set is far away from the optimal one, particularly in the regions highlighted in a light blue. In Fig. 1b, we perform a full B-RVFL according to the algorithm introduced in Section III. Not only is the MSE much lower, but the optimal amount of regularization is automatically inferred from the data, and the model also provides an uncertainty measure reflecting the noise in our sampling procedure (highlighted in light red). Interestingly, as we detail in Section III and validate experimentally later, the computational time elapsed for performing BI is comparable (and in many cases lower) to training a standard RVFL with the line search procedure. This is because, differently from line search, BI can quickly converge to a good set of hyper-parameters in a short number of iterations, consistently across all datasets we considered.

Contributions of the paper: In this paper, we provide two different approaches for performing BI on RVFL networks. The initial algorithm, presented in Section III, is based on an extension of known ideas from Bayesian ridge regression and RVMs, resulting in efficient closed-form computations. Nonetheless, it provides limited flexibility in specifying the initial beliefs on the data and the RVFL model. To provide a comprehensive treatment, in Section IV we describe an alternative solution based on recent advances in variational inference and automatic differentiation [27], [28]. Under this scheme, inference is performed automatically using a mean-field approximation, while the researcher is only required to specify the initial probabilistic model. In order to show the validity of this idea, in the experimental section we show a robust regression model based on the Student-t likelihood [29], which can obtain impressive results even when the dataset is contaminated by hundreds of outliers. This is a challenging problem for RVFL networks, which has become an important research direction lately, see in particular the works by Cao *et al.* [26], and Dai *et al.* [30].

Organization of the paper: The rest of the paper is organized as follows: Section II introduces some basic concepts on RVFL networks (Section II-A), and BI (Section II-B). Section III and Section IV describe two different approaches for performing BI of RVFL networks. Section V validates empirically all proposals on several regression datasets, with extensive comparisons to standard methods. We conclude this paper with some final remarks in Section VI. This paper is complemented by a comprehensive Python library allowing any researcher to exploit BI to train RVFL networks.

Notation: Throughout the paper, vectors are denoted by boldface lowercase letters, e.g., $\mathbf{a}$, while matrices are denoted by boldface uppercase letters, e.g., $\mathbf{A}$. All vectors are assumed column vectors. Symbol a_i denotes the ith element of vector $\mathbf{a}$, and A_{ij} the (i,j) entry of the matrix $\mathbf{A}$. The operator $\|\cdot\|_p$ is the standard ℓ_p norm on an Euclidean space. Other notation is introduced in the text when appropriate.

II. Preliminaries

A. Random vector functional-link networks

Given a real-valued input $\mathbf{x} \in \mathbb{R}^d$, the output of a generic RVFL network is computed as:

$$f(\mathbf{x}) = \sum_{i=1}^{B} \beta_i h_i(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{h}(\mathbf{x}), \tag{1}$$

where $h_1(\cdot), ..., h_B(\cdot)$ are randomly initialized basis functions defining a B-dimensional nonlinear mapping of the input vector. As an example, in this paper we consider the classical sigmoid nonlinearity with weights $\mathbf{a} \in \mathbb{R}^B, b \in \mathbb{R}$ drawn from

a uniform distribution scoped in $[-\lambda, +\lambda]$, where $\lambda > 0$ is a positive real number, and:

$$h_i(\mathbf{x}) = \frac{1}{1 + \exp\left(-\{\mathbf{a}^T\mathbf{x} + b\}\right)}\,. \tag{2}$$

In order to find the optimal weights $\boldsymbol{\beta}$ for a specific task, assume that we have available a dataset of N labeled samples $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}$, where y_i is the desired output on the ith example, depending on the problem, which can be a real value (regression) or a binary value (classification). Most commonly, RVFL training is formulated as a ridge regression problem:

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^B} \left\{ \frac{1}{2} \left\| \mathbf{H}\boldsymbol{\beta} - \mathbf{y} \right\|_2^2 + \frac{C}{2} \left\| \boldsymbol{\beta} \right\|_2^2 \right\}, \tag{3}$$

where $\mathbf{H}$ is the hidden matrix built by stacking row-wise the vectors $\mathbf{h}(\mathbf{x}_i)^T$, and similarly for $\mathbf{y}$, C is a regularization factor, and can be fine-tuned by the user according to the task. A solution of (3) can be given analytically as:

$$\boldsymbol{\beta}^* = \left(\mathbf{H}^T\mathbf{H} + C\mathbf{I}\right)^{-1} \mathbf{H}^T\mathbf{y}\,, \tag{4}$$

In this paper, we also consider a sparse variant with respect to the standard formulation. If one is interested in sparse representation, e.g., for hardware constraints, it is possible to solve a LASSO problem replacing the ℓ_2 norm regularization term in (3) with a ℓ_1 norm one to achieve the sparseness (while loosing a closed-form solution) [9].

B. Bayesian inference for supervised learning

The training procedure in the last section provides us with a single pointwise estimate of the optimal weights $\boldsymbol{\beta}$. As we stated in the introduction, BI is a principled way to obtain an entire probability distribution over the weights, by treating each quantity in the training problem as a random variable, and updating our beliefs according to the Bayes' rule. In this section, we review the general framework of BI, while in the next sections we will provide concrete examples of the inference procedure.

To begin with, instead of incorporating explicitly a regularization term on $\boldsymbol{\beta}$ as in (3), we express our beliefs on the optimal $\boldsymbol{\beta}$ using a prior density $p(\boldsymbol{\beta})$, e.g., an isotropic Gaussian density. Secondly, instead of assuming a deterministic output, we define a probability distribution $p(y \mid \mathbf{x}, \boldsymbol{\beta})$ over all outputs, e.g., another Gaussian centered around the RVFL output in (1). Since we assume that the examples in the dataset are independent and identically distributed, we can immediately write a likelihood distribution as:

$$p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta}) = \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i, \boldsymbol{\beta})\,, \tag{5}$$

where $\mathbf{X}$ is computed similarly to $\mathbf{H}$. The Bayes' rule allows us to combine our prior and likelihood beliefs to obtain a refined estimate on the weights' distribution, called the posterior distribution:

$$p(\boldsymbol{\beta} \mid \mathbf{X}, \mathbf{y}) = \frac{1}{Z}\, p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta})\, p(\boldsymbol{\beta})\,, \tag{6}$$

where Z is a normalization constant and independent of $\boldsymbol{\beta}$. The presence of Z (which is known as the marginal likelihood) is what makes (6) intractable in most practical cases. Finally, given a new input $\widehat{\mathbf{x}}$, the predictive distribution is computed as the expected value of (5) with respect to the posterior distribution (6):

$$p(y \mid \mathbf{X}, \mathbf{y}, \widehat{\mathbf{x}}) = \int_{\mathbb{R}^B} p(y \mid \widehat{\mathbf{x}}, \boldsymbol{\beta})\, p(\boldsymbol{\beta} \mid \mathbf{X}, \mathbf{y}) d\boldsymbol{\beta}\,. \tag{7}$$

The predictive distribution in (7) can be used to make pointwise predictions. The final important concept that we need to mention is the MAP estimation, which avoids the computation of the normalization constant Z by computing a single estimate for $\boldsymbol{\beta}$, given by a mode of the posterior:

$$\boldsymbol{\beta}_{\text{MAP}} = \arg\max_{\boldsymbol{\beta}} \left\{ p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta})\, p(\boldsymbol{\beta}) \right\}. \tag{8}$$

Even when the true posterior is intractable, maximization of (8) can be done by standard optimization procedures, and the resulting value provides both a good baseline for further evaluation, and a possible starting point for more elaborate inferential procedures such as the one introduced in Section IV. As stated in the introduction, MAP estimation of RVFL networks was considered briefly in [26] in order to provide a probabilistic interpretation of a robust training criterion.

III. Bayesian ridge regression for RVFL networks

In this section, we propose a simple BI algorithm for RVFL networks in the regression case, where the exact predictive distribution can be computed cheaply using an iterative procedure. To do so, we draw over basic results on the theory of Bayesian ridge regression, see [13, Section 3.3] and [16]. First, we make the classical assumption that our observations are deterministic functions of our input corrupted by simple, independently drawn white noise with variance σ^2, resulting in the following form for the terms in (5):

$$p(y \mid \mathbf{x}, \boldsymbol{\beta}, \sigma^2) = \mathcal{N}(y \mid \boldsymbol{\beta}^T\mathbf{h}(\mathbf{x}), \sigma^2)\,, \tag{9}$$

where $\mathcal{N}(y \mid a, b)$ denotes a Gaussian distribution over y of mean a and variance b, and for the moment we assume that the noise variance σ^2 is given. Our belief that the optimal weights will be relatively close to zero can be expressed by another multivariate Gaussian with diagonal covariance:

$$p(\boldsymbol{\beta} \mid \gamma) = \mathcal{N}(\boldsymbol{\beta} \mid \mathbf{0}, \gamma^{-1}\mathbf{I})\,, \tag{10}$$

where we use the precision parameter γ^{-1} instead of the variance to make it behave similarly to the regularization coefficient C in (3). Simple linear algebra calculations show that the posterior in this case is again Gaussian, with mean $\mathbf{m}$ and covariance $\boldsymbol{\Sigma}$ given by:

$$\mathbf{m} = \frac{1}{\sigma^2}\boldsymbol{\Sigma}\mathbf{H}^T\mathbf{y}\,, \tag{11}$$

$$\boldsymbol{\Sigma}^{-1} = \gamma\mathbf{I} + \frac{1}{\sigma^2}\mathbf{H}^T\mathbf{H}\,. \tag{12}$$

The predictive distribution is also Gaussian and can be evaluated by:

$$p(y \mid \mathbf{X}, \mathbf{y}, \widehat{\mathbf{x}}, \sigma^2, \gamma) = \mathcal{N}(y \mid \mathbf{m}^T\mathbf{h}(\widehat{\mathbf{x}}), \phi(\widehat{\mathbf{x}})^2)\,, \tag{13}$$

where the variance is computed as:

$$\phi(\widehat{\mathbf{x}})^2 = \sigma^2 + \mathbf{h}^T(\widehat{\mathbf{x}})\boldsymbol{\Sigma}\mathbf{h}(\widehat{\mathbf{x}})\,. \tag{14}$$

Note that both terms here have a very intuitive explanation. Due to the symmetry of the Gaussian distribution, the most probable prediction in (13) is given by considering the most probable set of parameters according to the posterior mean (11). The variance of the estimate in (14) is characterized by two terms, where the first term represents the noise level, and the second term is given by the uncertainty in the data itself. Although this already provides us with a confidence interval in the prediction, we still need to manually select σ^2 and γ. In order to fully leverage over the power of BI, we can define additional hyper-prior distributions over these parameters, allowing to treat them equivalently to the RVFL parameters $\boldsymbol{\beta}$. To avoid confusion, in this case σ^2 and γ are generally called 'hyper-parameters'. In order to make everything tractable, it is common to choose conjugate prior distributions, so that the resulting posterior is in the same family as the likelihood. Specifically, we can impose Gamma distributions on both the precision of the output and the precision of the prior weights:

$$p(\gamma) = \text{Gamma}(\gamma \mid \alpha_1, \alpha_2)\,, \tag{15}$$

$$p(\sigma^2) = \text{Gamma}(\sigma^{-2} \mid \alpha_3, \alpha_4)\,, \tag{16}$$

where α_i, $i = 1, \ldots, 4$ are the parameters of the two Gamma distributions.[1] By setting them with sufficiently small values, e.g., $\alpha_i = 10^{-5}$, we can obtain an extremely broad hyper-prior, such that most information on σ^2 and γ will be inferred from the data. Unfortunately, the predictive distribution now requires the marginalization with respect to both $\boldsymbol{\beta}$ and $\alpha_1, \ldots, \alpha_4$, making it intractable. A common alternative is to fix σ^2 and γ to a single value by performing a MAP estimation on the posterior of the hyper-parameters, which is obtained by first marginalizing the likelihood with respect to $\boldsymbol{\beta}$ (thus obtaining the so-called marginal likelihood), followed by multiplying with the hyper-priors. This is called the *evidence approximation*, or type-2 maximum likelihood, and it performs favorably in practice. Thus, we maximize the following index:

$$\sigma^2_*, \gamma_* = \arg\max \Bigg\{ \int_{\mathbb{R}^B} p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta}, \sigma^2) \cdot p(\boldsymbol{\beta} \mid \gamma) p(\gamma) p(\sigma^2) d\boldsymbol{\beta} \Bigg\}\,. \tag{17}$$

It turns out that we can easily maximize (17) using an iterative procedure [13]. Firstly, we initialize σ^2 and γ to some default values. Then, at every iteration, we compute the current mean and covariance according to (11) and (12). Secondly, we update the current estimate of σ^2 and γ according to the following formulas:

[1]Remember that $\text{Gamma}(s \mid a, b) = \Gamma(a)^{-1} b^a s^{a-1} e^{-bs}$, where $\Gamma(\cdot)$ is the gamma function.

Algorithm 1: Bayesian training of RVFL networks with evidence approximation

Data : Input matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$, desired outputs $\mathbf{y} \in \mathbb{R}^N$, hyper-prior parameters $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ (default to very small values for flat priors), number of random bases B.

(i) **Initialization**:

(i-1) Compute $\mathbf{H} \in \mathbb{R}^{N \times B}$ according to Sec. II-A.

(i-2) Store in memory $\mathbf{H}^T\mathbf{y}$, $\mathbf{H}^T\mathbf{H}$, and its eigenvalues $\lambda_1^0, \ldots, \lambda_B^0$.

(i-3) Initialize σ^2 and γ to default values.

(ii) **Posterior update**:

(ii-1) Compute posterior mean $\mathbf{m}$ as in (11).

(ii-2) Compute posterior covariance $\boldsymbol{\Sigma}$ as in (12).

(iii) **Hyper-parameters update**:

(iii-1) Compute updated eigenvalues $\lambda_i = \frac{1}{\sigma^2}\lambda_i^0$, $i = 1, \ldots, B$.

(iii-2) Update γ as in (18).

(iii-3) Update σ^2 as in (19).

(iv) Repeat steps (ii)-(iii) until a given convergence criterion.

$$\gamma = \frac{\delta + 2\alpha_1}{\|\mathbf{m}\|_2^2 + 2\alpha_2}\,, \tag{18}$$

$$\sigma^2 = \frac{\|\mathbf{y} - \mathbf{H}\boldsymbol{\beta}\|_2^2 + \alpha_4}{N - \delta + 2\alpha_3}\,, \tag{19}$$

where δ is defined as:

$$\delta = \sum_{i=1}^{B} \frac{\lambda_i}{\gamma + \lambda_i}\,, \tag{20}$$

with λ_i being the ith eigenvector of $\frac{1}{\sigma^2}\mathbf{H}^T\mathbf{H}$. These terms can be easily computed at every iteration by caching the eigenvalues of $\mathbf{H}^T\mathbf{H}$ at the beginning and multiplying them by the inverse of σ^2. It can be shown that $\sigma^2 \to \sigma^2_*$ and $\gamma \to \gamma_*$. These values can then be 'plugged-in' inside the predictive distribution (13). The overall algorithm is summarized in Algorithm 1, and we denote it as the Bayesian RVFL (B-RVFL).

Before concluding this section, we note that the framework described in this section can be easily customized to handle *sparse* weight vectors $\boldsymbol{\beta}$, similarly to what we can obtain with the use of the LASSO training criterion. Specifically, we can substitute our original prior (10) by providing each dimension with its own precision parameter:

$$p(\boldsymbol{\beta} \mid \boldsymbol{\gamma}) = \prod_{i=1}^{B} \mathcal{N}(\beta_i \mid 0, \gamma_i^{-1})\,, \tag{21}$$

where $\boldsymbol{\gamma}$ is now a B-dimensional vector of hyper-parameters. Its corresponding hyper-prior is modified in a similar fashion. The update equations in this case, originally developed by Tipping [16], are almost equivalent to the case under consideration in this section. In (21), each weight has a different precision parameter, denoted by the diagonal covariance ma-

trix of the Gaussian. Empirically, it is found that several γ_i will tend to very high values, particularly for redundant or not useful features, resulting in their probability distributions heavily centered around 0. Readers may refer to the original publication for more details. The prior in (21) is known as automatic relevance determination (ARD), while the resulting algorithm is known as the RVM. Following this, we refer to the RVFL network trained in this fashion as ARD-RVFL.

IV. Variational RVFL networks

The algorithms presented in the previous section are efficient, but they are weak in term of flexibility aspects, since any change in our choice of the likelihood or prior distributions results (in general) in posterior distributions which are no longer analytically tractable. As an example, consider a binary classification problem with $y_i = \{0, 1\}$. In a non Bayesian approach, it is common to binarize the real-valued output $f(\mathbf{x})$ to obtain the hard classification label, and train with a standard least-squares cost. A more elegant way to proceed is to limit the output of the network to $[0, 1]$ by applying an additional sigmoid nonlinearity to the output in (1):

$$\hat{f}(\mathbf{x}) = \text{sigmoid}\left(f(\mathbf{x})\right) = \frac{1}{1 + \exp\{-f(\mathbf{x})\}}, \tag{22}$$

and training the network by minimizing the cross-entropy loss function used in logistic regression. Nonetheless, this option is not common in the literature, as the possible gain in performance is almost never counterbalanced by the increased computational cost of having to apply an iterative optimization procedure to solve the resulting training problem. In the Bayesian case, a proper likelihood function (akin to the Bayesian treatment of logistic regression) would be a Bernoulli distribution:

$$p(y_i \mid \mathbf{x}_i, \boldsymbol{\beta}) = \hat{f}(\mathbf{x})^{y_i} \left(1 - \hat{f}(\mathbf{x})\right)^{1-y_i}, \tag{23}$$

which is parameterized in term of the normalized output in (22). Similarly, we might wish to consider more robust likelihoods for handling noise (an example of which is shown in the experimental section), more complex priors, such as mixture of Gaussian distributions as proposed in [21], or to combine the RVFL network as a component in a more complex probabilistic model, such as a graphical model [15]. Generally speaking, having to choose a specific form of our beliefs by reasoning only on the efficiency of the inference process does not exploit the entire potential of the BI framework.

In this section we briefly review a general algorithm for handling a large number of alternative choices for our distributions, which is based on the mean-field family of variational inference methods. To this end, let us denote by $p(\boldsymbol{\beta}, \boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y})$ a posterior distribution resulting from a generic combination of likelihood, prior, and hyper-priors distributions. From now onwards, $\boldsymbol{\theta}$ is a vector containing all hyper-parameters of our model. For instance, in B-RVFL we would have $\boldsymbol{\theta} = \left\{\sigma^2, \gamma\right\}$, while for ARD-RVFL we would have $\boldsymbol{\theta} = \left\{\sigma^2, \gamma_1, \ldots, \gamma_B\right\}$. For simplicity of discussion, we assume that the support of the prior distributions over the hyper-parameters is the set of reals $\mathbb{R}$. The case of hyper-parameters with bounded support of the prior distribution is handled easily by simple scalar transformations, e.g. see [28, Section 2.3], while discrete hyper-parameters cannot be used immediately. As an example, a generic variance b of a Gaussian is defined in the set of positive reals, but can be handled by transforming it to a different random variable as $c = \log(b)$, or similarly $c = \log(\exp(b) - 1)$. For simplicity of notation, we avoid working with these transformations, and refer to [28] and references therein for additional details.

Since computing the exact posterior is intractable, variational methods look for an approximation in a known family $q(\boldsymbol{\beta}, \boldsymbol{\theta}; \boldsymbol{\zeta})$, which is parameterized by a set of adaptable coefficients $\boldsymbol{\zeta}$ known as variational parameters. In the simplest case, i.e., mean-field approximation, we approximate the posterior with a product of univariate Gaussian distributions:

$$q(\boldsymbol{\beta}, \boldsymbol{\theta}; \boldsymbol{\zeta}) = \prod_{i=1}^{B} \mathcal{N}(\beta_i \mid \eta_i, \delta_i^2) \prod_{j=1}^{K} \mathcal{N}(\theta_j \mid \eta_{B+j}, \delta_{B+j}^2), \tag{24}$$

where K is the number of hyper-parameters, and we have $2(B + K)$ variational parameters given by the means and variances of the Gaussian distributions $\boldsymbol{\zeta} = \{\eta_1, \ldots, \eta_{B+K}, \delta_1, \ldots, \delta_{B+K}\}$ to be optimized. Note that the support of the standard deviation is the set of positive real numbers, which can be challenging during optimization. However, similarly to before, we can use a different parameter $\omega_i = \log(\delta_i)$ in order to remove this constraint. More complex approximations, such as a multivariate Gaussian distribution with a full covariance matrix, are generally harder to optimize, although they may show an interesting direction for future work [31].

There are several possibilities to define the closeness of two probability distributions. In our case, we consider the Kullback-Leibler (KL) divergence[2] between the two distributions as defining our optimal variational approximation:

$$\boldsymbol{\zeta}^* = \underset{\mathbb{R}^{2(B+K)}}{\arg\min} \left\{ \text{KL}\Big(q(\cdot) \parallel p(\boldsymbol{\beta}, \boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}) \Big) \right\}. \tag{25}$$

Expanding the KL divergence gives us the following (theoretical) objective (see Appendix B in [28] for the details of the computation):

$$\begin{aligned} \text{KL}\Big(q(\cdot) \parallel p(\boldsymbol{\beta}, \boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}) \Big) = \mathbb{E}\big[\log\left(q(\cdot)\right)\big] - \\ \mathbb{E}\big[\log\left(p(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}, \mathbf{y})\right)\big] + \log(p(\mathbf{X}, \mathbf{y})), \end{aligned} \tag{26}$$

where expectations are taken over the variational distribution. Interestingly, the intractable term of the posterior only appears as an additive term in the previous expression. This motivates the use of an approximate objective, which is called the evidence lower bound (ELBO) in the literature, given by minimizing the negative KL divergence plus the intractable term, thus effectively removing it [28]:

$$\text{ELBO}(\boldsymbol{\zeta}) = \mathbb{E}\big[\log\left(p(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}, \mathbf{y})\right)\big] - \mathbb{E}\big[\log\left(q(\cdot)\right)\big], \tag{27}$$

[2]Remember that the KL divergence is minimized if the two distributions are equal. Since the KL divergence is not symmetric, we can devise a different family of approximation methods by interchanging the order of the two distributions in (25). This is known as expectation-propagation (EP) [32].

Dataset	Features	Samples	Desired output	Reference
Boston	13	506	Median house value	[33]
Airfoil	6	1503	Sound pressure level	[34]
Communities	128	1994	Violent crimes per capita	NA
Concrete	9	1030	Concrete compressive strength	[35]
Abalone	8	4177	Number of rings	[36]

TABLE I
SCHEMATIC DESCRIPTION OF THE DATASETS USED FOR TESTING THE B-RVFL. SEE THE TEXT FOR MORE DETAILS ON THE PREPROCESSING.

which by construction is a lower bound for the divergence in (25). Optimizing the previous objective still requires the gradient of an expectation, which is not trivial. The last step is the use of a technical transformation, which in the literature is called interchangeably as the elliptical standardization, the re-parameterization trick, the coordinate transformation, or the invertible transformation [28]. The basic idea is to introduce a simple transformation such that the distribution with respect to which we take the expectation is independent of the variational parameters. Particularly, consider the transformation converting the variational transformation of β_i to a standard Gaussian $\nu_i = S(\beta_i) = \exp(\omega_i)^{-1}(\beta_i - \eta_i)$, and similarly for the hyper-parameters $\boldsymbol{\theta}$. The resulting variational density is given by:

$$q(\boldsymbol{\nu}) = \mathcal{N}(\boldsymbol{\nu} \mid \mathbf{0}, \mathbf{I}) \,. \tag{28}$$

Using this transformation, we can rewrite our ELBO objective as:

$$\text{ELBO}(\boldsymbol{\zeta}) = \mathbb{E}_{q(\boldsymbol{\nu})}\left[\log\left(p(S^{-1}(\boldsymbol{\beta}), S^{-1}(\boldsymbol{\theta}), \mathbf{X}, \mathbf{y})\right)\right] - \mathbb{H}\left(q(\cdot)\right) \,, \tag{29}$$

where $\mathbb{H}\left(q(\cdot)\right)$ is the entropy of $q(\cdot)$. At this point, we have a simple recipe for computing the gradients of the previous term. For the entropy term, we have analytical expressions that can be used immediately [37]. The gradient of the expectation (since the expectation is now defined with respect to a different distribution) can be written as the expectation of the gradient:

$$\nabla_{\boldsymbol{\zeta}} \mathbb{E}_{q(\boldsymbol{\nu})}\left[\log\left(p(\cdot)\right)\right] = \mathbb{E}_{q(\boldsymbol{\nu})}\left[\nabla_{\boldsymbol{\zeta}} \log\left(p(\cdot)\right)\right] \,. \tag{30}$$

By the chain rule of differentiation, the gradient in the right hand side can be computed as the gradient of our original joint distribution (with respect to our original parameters), times the gradient of the inverse elliptical transformation, which is one. Thus, the only thing we truly require is the gradient of our joint density $p(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}, \mathbf{y})$ with respect to our original parameters. The expectation of this gradient can be computed by drawing on Monte Carlo integration as:

$$\mathbb{E}_{q(\boldsymbol{\nu})}\left[\nabla_{\boldsymbol{\beta}} \log(p(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}, \mathbf{y}))\right] \approx \frac{1}{S}\sum_{i=1}^{S} \nabla_{\boldsymbol{\beta}} \log(p(S^{-1}(\boldsymbol{\nu}_i), \mathbf{X}, \mathbf{y})) \,, \tag{31}$$

where we draw S samples $\boldsymbol{\nu}_i \sim q(\boldsymbol{\nu})$ according to the normal distribution, and we have a similar expression for the hyper-parameters. This provides an unbiased estimation with variance $1/S$. The true strength of this approach is now visible, in that we can easily combine it with automatic differentiation tools, like those common in the deep learning community, resulting in what is called automatic differentiation variational inference (ADVI) [28]. In this scenario, the user can model the original prior, likelihood, and hyper-prior distributions in a symbolic fashion. As long as these distributions are differentiable, the compiler can automatically compute the gradients in (30) by drawing random samples from the normal distribution, and thus optimize the ELBO and the corresponding approximate posterior. Practically, it is found that a single draw of the gradient (i.e., $S = 1$) is enough for optimizing the ELBO term up to a very good precision [28]. We show an example of this technique in our experimental section, and we refer to RVFLs trained using this procedure as VAR-RVFLs. In all our experiments we use $S = 1$, since we found that the small improvement provided by $S > 1$ in some cases was not consistent, and it was offset by a larger computational time requested by the procedure. Before concluding, we briefly mention here that ADVI is not the only choice for automatic inference, and alternative frameworks are possible, such as black-box variational inference [24].

V. EXPERIMENTAL VALIDATION

In this section, we provide a comprehensive experimental evaluation of the methods we introduced. Section V-B compares B-RVFL with a standard RVFL trained using a regularized least-squares procedure with fine-tuned parameters. Section V-D evaluates ARD-RVFL in comparison to a standard LASSO training procedure, both in terms of accuracy and sparseness of the resulting output layers. Section V-E shows a possible use case for VAR-RVFL in situations of very high noise in the datasets. A general purpose library for repeating the experiments presented here is available on the web under open-source license.[3] The library is in Python 3.5, and all training algorithms (least-squares, LASSO, and the proposed BI ones) are compatible with the popular scikit-learn library.[4] For VAR-RVFL and performing ADVI, we use the PyMC3 library[5], which uses Theano[6] as the underground symboling modeling tool.

[3]https://bitbucket.org/ispamm/bayesian-rvfl
[4]http://scikit-learn.org/
[5]https://github.com/pymc-devs/pymc3
[6]http://deeplearning.net/software/theano/

Dataset	Algorithm	Accuracy			Tr. time [secs.]
		R^2	E_{VAR}	MSE	
Boston	RVFL	0.8743 ± 0.05180	0.8768 ± 0.0516	0.0052 ± 0.0026	6.51 ± 0.88
	B-RVFL	$\mathbf{0.8758 \pm 0.05680}$	$\mathbf{0.8781 \pm 0.0567}$	$\mathbf{0.0051 \pm 0.0028}$	1.59 ± 0.27
Airfoil	RVFL	0.7137 ± 0.0470	0.7157 ± 0.0471	0.0095 ± 0.0015	10.43 ± 1.55
	B-RVFL	$\mathbf{0.7843 \pm 0.044}$	$\mathbf{0.7859 \pm 0.044}$	$\mathbf{0.0071 \pm 0.0014}$	3.29 ± 0.74
Communities	RVFL	0.6590 ± 0.0510	0.6607 ± 0.0510	0.0184 ± 0.0032	16.53 ± 3.40
	B-RVFL	$\mathbf{0.6600 \pm 0.0515}$	$\mathbf{0.6618 \pm 0.0515}$	$\mathbf{0.0183 \pm 0.0031}$	1.44 ± 0.38
Concrete	RVFL	0.8318 ± 0.0356	0.8340 ± 0.0348	0.0071 ± 0.0012	7.70 ± 0.13
	B-RVFL	$\mathbf{0.8672 \pm 0.0302}$	$\mathbf{0.8687 \pm 0.0300}$	$\mathbf{0.0056 \pm 0.0010}$	1.88 ± 0.12
Abalone	RVFL	0.5630 ± 0.0433	0.5638 ± 0.04151	0.0058 ± 0.0008	27.82 ± 4.51
	B-RVFL	$\mathbf{0.5646 \pm 0.0415}$	$\mathbf{0.5654 \pm 0.04137}$	$\mathbf{0.0057 \pm 0.0007}$	3.74 ± 0.84

TABLE II
RESULTS (IN TERMS OF THREE ACCURACY MEASURES ON THE TEST SET AND TRAINING TIME) OF RVFL AND B-RVFL ON THE FIVE DATASETS. BEST RESULTS FOR EACH ACCURACY MEASURE ARE HIGHLIGHTED IN BOLD. TRAINING TIME FOR RVFL IS INCLUSIVE OF THE LINE SEARCH PROCEDURE. SEE THE TEXT FOR A DESCRIPTION OF THE ACRONYMS AND OF THE ERROR MEASURES.

A. Experimental setup

For our experiments, we consider five regression datasets, whose characteristics are schematically summarized in Table I. Boston (also known as Housing) concerns the prediction of a median house value starting from a set of characteristics of the house and of the neighborhood. In Airfoil, we need to predict a sound pressure level (in decibel) from 6 descriptions of an airfoil based on some simulations conducted by NASA. In Communities, we want to predict a rate of violent crimes starting from numerous data on a community, including socio-economical data and law data. For Concrete, we predict the compressive strength using 9 measurements describing the production process of a concrete sample. In Abalone we predict the age of an abalone from some physical measurements.

In all cases, input features are scaled in the range $[-1, +1]$, while output values are scaled in $[0, 1]$. Missing entries are handled by replacing them with the mean value of the corresponding feature in the dataset, while categorical features are transformed using a one-hot encoding. We consider RVFL networks with $B = 500$ nodes in the hidden layer, which is found to work relatively well in all the datasets we considered. Each node uses a sigmoid nonlinear function as in (22), whose parameters $\mathbf{a}$ and b are extracted randomly from a uniform distribution over the interval $[-\lambda, +\lambda]$. Selecting a feasible λ on each dataset is fundamental for a proper behavior of the algorithms. In order to select λ, we performed a preliminary robustness analysis on the different datasets, whose details are provided in Section V-C. Following that analysis, we found that $\lambda = 0.5$ provides good results on the five datasets under consideration, and we use it as the default choice in the rest of the experimental section. Testing accuracy is computed by performing a 10-fold cross-validation on the data, and the entire process is repeated 10 times, resulting in 100 different runs for each dataset.

B. Experimental comparison of B-RVFL

We begin by comparing B-RVFL with a RVFL network trained using the least-squares procedure in (3). The regularization factor C in this case is selected from the exponential interval $\left\{2^j\right\}, j = -10, \ldots, 10$ for each run by minimizing the mean-squared error (MSE) over a 3-fold cross-validation of the training data. Given a generic test set $\mathcal{T} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, T\}$, we compute the MSE and two additional accuracy measures. The first is the coefficient of determination R^2, given by:

$$R^2 = 1 - \frac{\sum_{i=1}^{T} (y_i - f(\mathbf{x}_i))^2}{\sum_{i=1}^{T} (y_i - \bar{y})^2}, \tag{32}$$

where $\bar{y}$ is the empirical mean of the desired output. The second is the explained variance E_{VAR} given by:

$$E_{VAR} = 1 - \frac{\text{Var}_{\mathcal{T}}(y_i - f(\mathbf{x}_i))}{\text{Var}_{\mathcal{T}}(y_i)}, \tag{33}$$

where $\text{Var}_{\mathcal{T}}(\cdot)$ computes the empirical variance with respect to $\mathcal{T}$. For both R^2 and E_{VAR} the best possible value is 1, while lower values are worse. The results of this set of experiments are presented in Table II, where the best results for each accuracy measure are highlighted in bold. For B-RVFL, we always select the prediction corresponding to the mean of the predictive distribution, which is in accordance with a squared loss criterion. We set $\alpha_i = 10^{-6}$ for $i = 1, \ldots, 4$.

The results show clearly a superiority of B-RVFL, which achieves slightly better results in three out of five cases (Boston, Communities and Abalone), and a definite increment in performance in the remaining two cases (Airfoil and Concrete). More in detail, we have a decrease of MSE of roughly 2% for Boston and Abalone, more than 20% for Concrete, and more than 25% for Airfoil. The three accuracy measures are always consistent in the five datasets. We also note that this comparison only takes into account the mean of the predictive

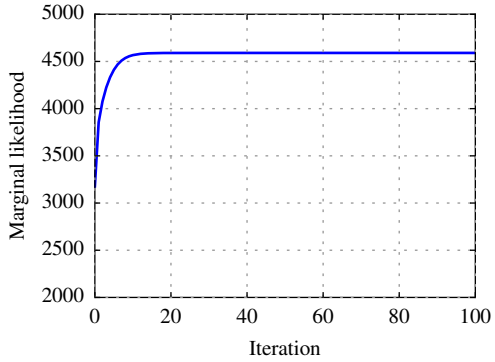


Fig. 2. Evolution of the marginal likelihood in (17) for the Boston dataset.

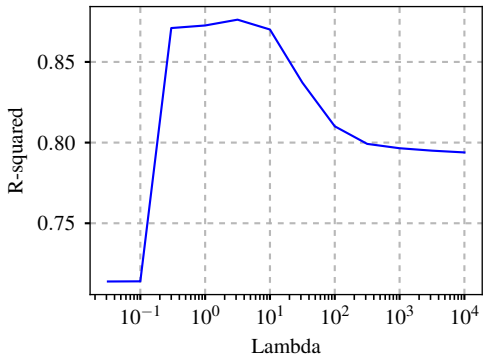


Fig. 3. Cross-validated R^2 score when varying λ on the Boston dataset.

distribution, while in practice B-RVFL also provides a variance that can be exploited during the test phase. With respect to the selected hyper-parameters, we have found that while the range of the selected values is similar, B-RVFL is able to obtain a finer precision on its choice, while the line search procedure is limited to the resolution selected by the user, which explains the improvement in performance found in Table II.

What is particularly surprising from Table II is that the improvement in performance comes with a definite reduction of training time (sixth column), where for fairness of comparison we consider the entire time spent for the line search procedure of the standard RVFL. To understand why, we plot a representative evolution of the marginal likelihood in (17) in Fig. 2. We see that a relatively small number of iterations are enough to achieve convergence. Since the computational time for a single iteration is roughly comparable to the training of a standard RVFL (due to the computation of (11), without considering the initial overhead for evaluating the eigenvalues), we see that this is by far more advantageous than doing the entire line search. The cost of the line search can be reduced in principle by decreasing the number of parameters under consideration, or by evaluating the accuracy only on a single holdout of the training dataset. In the former case, however, there is no principled way of selecting a smaller interval, while in the latter case, performance can be heavily degraded. Overall, we can conclude that B-RVFL is a good alternative for training RVFL networks in a regression setting, as it has a very good accuracy, low training time, and it does not require any additional hyper-parameter to be selected.

C. Selecting the scope for the random weights

Before proceeding to analyzing ARD-RVFL, we consider here the topic of selecting a proper λ for the uniform distribution when assigning the random parameters of the RVFL network. To this end, we run B-RVFL on the Boston dataset using the settings of the previous section, but we vary λ in the exponential interval 10^j, $j = -2, -1.5, \ldots, 4$, and we evaluate the cross-validated R^2 score. Results are presented in Fig. V-C, where the x-axis is shown with a logarithmic scale.

It can be seen that, although there is a large interval over which the performance is consistent, setting λ outside such interval seriously degrades the performance. Specifically, for very small λ, the sigmoid functions work mostly on a linear regime, and the results are similar to what would be obtained by a standard linear regressor. On the contrary, for large λ the sigmoid almost always saturate, making the algorithm less stable and more prone to numerical problems. Since all datasets were rescaled in the same interval, similar results are observed also for the other datasets, and are omitted here for brevity.

D. Experimental comparison of ARD-RVFL

Next, we compare the performance of ARD-RVFL for obtaining sparse weight vectors $\boldsymbol{\beta}$, which is particularly helpful for hardware implementations (among others). In this case, we compare against a RVFL network optimizing a standard LASSO objective, which we denote as L-RVFL:

$$\boldsymbol{\beta}^*_{\text{LASSO}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^B}{\arg\min} \left\{ \frac{1}{2} \|\mathbf{H}\boldsymbol{\beta} - \mathbf{y}\|_2^2 + C \sum_{i=1}^{B} |\beta_i| \right\}. \quad (34)$$

The problem in (34) is solved using a coordinate descent algorithm with a maximum of 2500 iterations. The coefficient C is again selected using a line search procedure. However, in this case selecting an optimal interval of values is not trivial. Specifically, if we optimize against the MSE and we allow very low values of C, in many cases (particularly for simpler datasets) the line search will select these smaller values, resulting in low error but very few zeros in the output layer. Conversely, limiting the interval in these situations can increase sparseness at the cost of a higher error. Due to this, we experiment with three different intervals for the exponent in $j = -I, \ldots, 10$, with $I = \{10, 15, 20\}$. In all cases, sparsity of $\boldsymbol{\beta}$ is defined as the relative number of weights whose absolute value is lower than 10^{-3}. Results are presented in Table III. Since all measures were consistent in the previous experiment, we report in the table only one of the three.

In terms of the prediction accuracy, ARD-RVFL outperforms all other algorithms in four out of five cases (i.e., Boston, Airfoil, Concrete and Abalone), while for the Communities dataset the results are comparable to the standard approach. The reduction in performance with respect to Table II is also negligible, except in two out of five datasets,

Dataset	Algorithm	E_{VAR}	Sparsity [%]	Tr. time [secs.]
Boston	L-RVFL ($C_{\max} = 10^{10}$)	0.7068 ± 0.1039	97.28 ± 0.51	5.23 ± 0.49
	L-RVFL ($C_{\max} = 10^{15}$)	0.8336 ± 0.0746	56.94 ± 2.16	13.60 ± 0.54
	L-RVFL ($C_{\max} = 10^{20}$)	0.8519 ± 0.0678	4.53 ± 11.22	26.54 ± 0.64
	ARD-RVFL	0.8611 ± 0.0627	17.51 ± 4.86	14.27 ± 0.52
Airfoil	L-RVFL ($C_{\max} = 10^{10}$)	0.4954 ± 0.0574	97.65 ± 0.45	19.81 ± 2.04
	L-RVFL ($C_{\max} = 10^{15}$)	0.6056 ± 0.0583	70.08 ± 1.49	41.58 ± 2.09
	L-RVFL ($C_{\max} = 10^{20}$)	0.6315 ± 0.0577	0.97 ± 0.38	77.44 ± 2.09
	ARD-RVFL	0.6623 ± 0.1023	42.64 ± 12.87	132.04 ± 3.17
Communities	L-RVFL ($C_{\max} = 10^{10}$)	0.6460 ± 0.0454	91.11 ± 0.89	16.73 ± 0.54
	L-RVFL ($C_{\max} = 10^{15}$)	0.6542 ± 0.0435	84.78 ± 3.92	52.29 ± 0.75
	L-RVFL ($C_{\max} = 10^{20}$)	0.6542 ± 0.0435	84.78 ± 3.92	100.60 ± 0.79
	ARD-RVFL	0.6378 ± 0.0480	72.81 ± 4.07	182.94 ± 59.63
Concrete	L-RVFL ($C_{\max} = 10^{10}$)	0.5942 ± 0.0520	98.02 ± 0.42	11.21 ± 0.63
	L-RVFL ($C_{\max} = 10^{15}$)	0.7403 ± 0.0462	56.00 ± 1.60	27.50 ± 0.64
	L-RVFL ($C_{\max} = 10^{20}$)	0.7663 ± 0.0406	0.71 ± 0.28	52.64 ± 0.60
	ARD-RVFL	0.8095 ± 0.0361	40.89 ± 9.89	53.61 ± 1.48
Abalone	L-RVFL ($C_{\max} = 10^{10}$)	0.4117 ± 0.0196	99.18 ± 0.06	28.00 ± 0.93
	L-RVFL ($C_{\max} = 10^{15}$)	0.5431 ± 0.0319	83.72 ± 0.43	86.67 ± 1.01
	L-RVFL ($C_{\max} = 10^{20}$)	0.5487 ± 0.0348	7.10 ± 0.74	185.30 ± 1.77
	ARD-RVFL	0.5668 ± 0.0347	15.24 ± 8.01	226.18 ± 4.47

TABLE III
RESULTS (IN TERMS OF E_{VAR}, SPARSITY OF $\boldsymbol{\beta}$ AND TRAINING TIME) OF ARD-RVFL AND L-RVFL TRAINED WITH THREE DIFFERENT INTERVALS FOR THE LINE SEARCH, ON THE FIVE DATASETS. SEE THE TEXT FOR A DESCRIPTION OF THE ACRONYMS AND THE ERROR MEASURE.

particularly Airfoil. If we optimize L-RVFL with a small interval, we end up with solutions with a very poor accuracy and almost all zeros in the output layer. Conversely, enlarging too much this interval results in solutions with a good accuracy but a small amount of sparsity with respect to $\boldsymbol{\beta}$, except in the Communities dataset where all algorithms are performing roughly similarly. ARD-RVFL tends to have the best overall accuracy, and a sparsity level which is intermediate among the different variants of L-RVFL, going over 40% two out of five times, and over 15% four out of five times. In the majority of cases, however, training time is comparable or higher than the line search with the largest interval. Overall, we can conclude from this set of experiments that ARD-RVFL is a good compromise if we want a moderately sparse RVFL without sacrificing too much accuracy. In addition, we are not required to specify an exact procedure for fine-tuning C and, as before, we can access the variance of the predictions for achieving better decisions.

E. Experimental comparison of VAR-RVFL for robust regression

For VAR-RVFL, we provide a simple use case in situations of very high noise in the datasets. Specifically, for each dataset we increase the amount of training data with an additional 25% of randomly generated examples. One possibility for increasing robustness of the BI procedure is to consider a Student-t likelihood instead of the Gaussian [29], which is given by:

$$p(y \mid \mathbf{x}, \boldsymbol{\beta}, \sigma^2, \epsilon) = \frac{\Gamma((\epsilon+1))/2}{\Gamma(\epsilon/2)\sqrt{\epsilon\pi}\sigma}\left(1 + \frac{(y_i - f(\mathbf{x}))^2}{\epsilon\sigma^2}\right)^{-\frac{(\epsilon+1)}{2}}, \tag{35}$$

where the additional parameter ϵ is called the degrees of freedom of the distribution. For $\epsilon \rightarrow \infty$ the distribution degenerates to a standard Gaussian distribution. We consider a Gaussian prior over $\boldsymbol{\beta}$ as in (9) with a small fixed precision $\gamma = 10^{-3}$ (as we found no improvement in performance in this case by including an hyper-prior), and a uniform hyper-prior over ϵ in $[1, 100]$. We initialize the ADVI solver with the MAP estimate in (8), and we optimize the ELBO objective with an Adagrad solver for 6000 iterations. We experimented with the use of a more complex Adam solver [38], but we found no improvement in convergence time. For simplicity, we make predictions by considering only the mean of the variational approximation. We compare with B-RVFL, and a standard RVFL trained using the procedure detailed in Section V-B. Results are provided in Table IV.

We can see that the performance of the standard RVFL is heavily degraded, except in the very simple Communities dataset. In the other four cases, B-RVFL is able to achieve a good performance despite the high level of noise in only one dataset. In the remaining three cases, however, also its performance is heavily affected by the noise. In these remaining situations,

Dataset	RVFL	B-RVFL	VAR-RVFL
Boston	0.4799 ± 0.1641	$\mathbf{0.8694 \pm 0.0602}$	0.7828 ± 0.0891
Airfoil	0.1964 ± 0.0306	0.5428 ± 0.0709	$\mathbf{0.6433 \pm 0.090}$
Communities	0.6302 ± 0.0380	$\mathbf{0.6366 \pm 0.0441}$	0.5456 ± 0.0722
Concrete	0.2628 ± 0.0452	0.7549 ± 0.0594	$\mathbf{0.8371 \pm 0.0365}$
Abalone	0.3341 ± 0.0227	0.4869 ± 0.0217	$\mathbf{0.5122 \pm 0.0214}$

TABLE IV
RESULTS (IN TERMS OF E_{VAR}) OF RVFL, B-RVFL AND VAR-RVFL TRAINED ON DATASETS WITH A VERY HIGH LEVEL OF OUTLIERS, ON THE FIVE DATASETS. BEST RESULTS FOR EACH DATASET ARE HIGHLIGHTED IN BOLD. SEE THE TEXT FOR A DESCRIPTION OF THE ACRONYMS AND THE ERROR MEASURE.

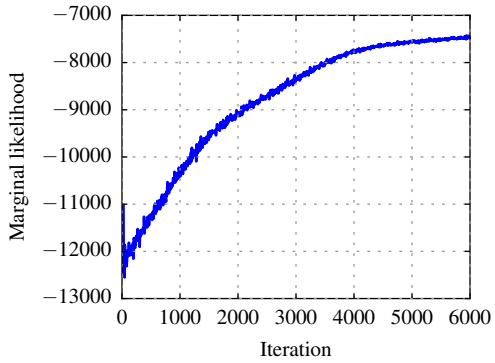


Fig. 4. Evolution of the ELBO objective in (27) for a representative dataset.

VAR-RVFL is the only algorithm which is able to achieve a satisfactory level of performance. For completeness, we show the evolution of the ELBO objective in (27) for a representative run in Fig. 4.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented several alternatives for training RVFLs using a BI framework, ranging from a simple iterative procedure exploiting the so-called evidence approximation, to a general algorithm framed in the variational inference field. BI has several advantages over standard training procedures, including flexibility, automatic inference of the hyper-parameters, and the possibility of exploiting uncertainty when making a prediction. As we show in our experimental results, B-RVFL is a very powerful alternative to standard least-squares training, both in terms of accuracy and training time. Similarly, ARD-RVFL is a good compromise when searching for RVFL networks having a sparse output layer. Finally, VAR-RVFL can be customized easily to a variety of situations exploiting the Python library we released, and we showed a simple use case involving robust regression.

Future works can consider the application to several large-scale problems, where we can leverage over multiple advances in variational inference [17]. We are also planning in investigating more thouroughly the performance of different choices for the likelihood function, the prior, and the hyper-priors. Overall, we believe the application of BI techniques to the training of RVFL networks is a promising research in order to further improve the performance on robust data modeling. Also, the proposed framework in this paper is being extended to an advanced randomized model, termed as stochastic configuration network [39].

ACKNOWLEDGMENTS

This work was supported in part by Italian MIUR, "*Progetti di Ricerca di Rilevante Interesse Nazionale*", GAUChO project, under Grant 2015YPXH4W_004. The authors would like to thank the anonymous referees for the invaluable suggestions on improving the manuscript.

REFERENCES

[1] Y. H. Pao and Y. Takefji, "Functional-link net computing: Theory, system architecture and functionalities," *IEEE Computer Journal*, vol. 25, no. 5, pp. 76–79, 1992.
[2] Y.-H. Pao, G.-H. Park, and D. J. Sobajic, "Learning and generalization characteristics of the random vector functional-link net," *Neurocomputing*, vol. 6, no. 2, pp. 163–180, 1994.
[3] B. Igelnik and Y.-H. Pao, "Stochastic choice of basis functions in adaptive function approximation and the functional-link net," *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 1320–1329, 1995.
[4] D. Husmeier and J. G. Taylor, "Neural networks for predicting conditional probability densities: Improved training scheme combining EM and RVFL," *Neural Networks*, vol. 11, no. 1, pp. 89–116, 1998.
[5] Q. Yang and S. Jagannathan, "Reinforcement learning controller design for affine nonlinear discrete-time systems using online approximators," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 377–390, 2012.
[6] J. Lu, J. Zhao, and F. Cao, "Extended feed forward neural networks with random weights for face recognition," *Neurocomputing*, vol. 136, pp. 96–102, 2014.
[7] L. Zhang and P. N. Suganthan, "Visual tracking with convolutional random vector functional link network," *IEEE Transactions on Cybernetics*, 2016, in press.
[8] S. Scardapane, D. Wang, M. Panella, and A. Uncini, "Distributed learning for random vector functional-link networks," *Information Sciences*, vol. 301, pp. 271–284, 2015.
[9] F. Cao, Y. Tan, and M. Cai, "Sparse algorithms of random weight networks and applications," *Expert Systems with Applications*, vol. 41, no. 5, pp. 2457–2462, 2014.
[10] S. Scardapane, D. Comminiello, M. Scarpiniti, and A. Uncini, "A semi-supervised random vector functional-link network based on the transductive framework," *Information Sciences*, vol. 364, pp. 156–166, 2016.
[11] D. Wang, "Editorial: Randomized algorithms for training neural networks," *Information Sciences*, vol. 364, pp. 126–128, 2016.
[12] S. Scardapane and D. Wang, "Randomness in neural networks: an overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 2, 2017.
[13] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer International, 2006.
[14] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT Press, 2012.

[15] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1-2, pp. 1–305, 2008.

[16] M. E. Tipping, "Sparse Bayesian learning and the relevance vector machine," *Journal of Machine Learning Research*, vol. 1, no. Jun, pp. 211–244, 2001.

[17] M. D. Hoffman, D. M. Blei, C. Wang, and J. W. Paisley, "Stochastic variational inference," *Journal of Machine Learning Research*, vol. 14, no. May, pp. 1303–1347, 2013.

[18] C. E. Rasmussen, *Gaussian processes for machine learning*. MIT Press, 2006.

[19] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 2, pp. 408–423, 2015.

[20] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012.

[21] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *32nd International Conference on Machine Learning*, 2015.

[22] Y. Lu, B. Huang, and S. Khatibisepehr, "A variational Bayesian approach to robust identification of switched ARX models," *IEEE Transactions on Cybernetics*, vol. 46, no. 12, pp. 3195–3208, 2016.

[23] Y. Zhao, A. Fatehi, and B. Huang, "Robust estimation of ARX models with time varying time delays using variational Bayesian approach," *IEEE Transactions on Cybernetics*, 2017, in press.

[24] R. Ranganath, S. Gerrish, and D. M. Blei, "Black box variational inference." in *AISTATS*, 2014, pp. 814–822.

[25] A. Kucukelbir, R. Ranganath, A. Gelman, and D. Blei, "Automatic variational inference in Stan," in *Advances in Neural Information Processing Systems*, 2015.

[26] F. Cao, H. Ye, and D. Wang, "A probabilistic learning algorithm for robust modeling using neural networks with random weights," *Information Sciences*, vol. 313, pp. 62–78, 2015.

[27] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *arXiv preprint arXiv:1601.00670*, 2016.

[28] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei, "Automatic differentiation variational inference," *Journal of Machine Learning Research*, vol. 18, no. 14, pp. 1–45, 2017.

[29] P. Jylänki, J. Vanhatalo, and A. Vehtari, "Robust Gaussian process regression with a Student-t likelihood," *Journal of Machine Learning Research*, vol. 12, no. Nov, pp. 3227–3257, 2011.

[30] W. Dai, Q. Liu, and T. Chai, "Particle size estimate of grinding processes using random vector functional link networks with improved robustness," *Neurocomputing*, vol. 169, pp. 361–372, 2015.

[31] E. Challis and D. Barber, "Gaussian Kullback-Leibler approximate inference." *Journal of Machine Learning Research*, vol. 14, no. Aug, pp. 2239–2286, 2013.

[32] T. P. Minka, "Expectation propagation for approximate Bayesian inference," in *17th Conference on Uncertainty in Artificial Intelligence*, 2001, pp. 362–369.

[33] D. A. Belsley, E. Kuh, and R. E. Welsch, *Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley & Sons, 2005.

[34] T. F. Brooks, D. S. Pope, and M. A. Marcolini, "Airfoil self-noise and prediction," Tech. Rep. NASA RP-1218, 1989.

[35] I.-C. Yeh, "Modeling of strength of high-performance concrete using artificial neural networks," *Cement and Concrete Research*, vol. 28, no. 12, pp. 1797–1808, 1998.

[36] S. Waugh, "Extending and benchmarking cascade-correlation," Ph.D. dissertation, Dept of Computer Science, University of Tasmania, 1995.

[37] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[38] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[39] D. Wang and M. Li, "Stochastic configuration networks: Fundamentals and algorithms," *arXiv preprint arXiv:1702.03180*, 2017.