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 (RVFL) networks 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 suboptimal, 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].

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 $\ell_2$ 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 nonconvex 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]–[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, we are interested in performing full BI, thus providing an entire distribution of the optimal weights. As one of the motivations behind this paper, Fig. 1 depicts a toy example where we consider a 1-D 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. 1(a), 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. 1(b), we perform a full Bayesian RVFL (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.

![Comparison of a standard RVFL network and a B-RVFL network 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.](image-url)

**A. Contributions of This 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].

**B. Organization of This Paper**

The rest of this paper is organized as follows. Section II introduces some basic concepts on RVFL networks (Section II-A), and BI (Section II-B). Sections III and 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 this 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 \( i \)th element of vector \( \mathbf{a} \), and \( A_{ij} \) the \((i,j)\) entry of the matrix \( \mathbf{A} \). The operator \( \| \cdot \|_p \) is the standard \( \ell_p \) 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 \( x \in \mathbb{R}^d \), the output of a generic RVFL network is computed as

\[
f(x) = \sum_{i=1}^{B} \beta_i h_i(x) = \beta^T h(x)
\]

(1)

where \( h_1(\cdot), \ldots, h_B(\cdot) \) 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 \( a \in \mathbb{R}^d \) and \( b \in \mathbb{R} \) drawn from a uniform distribution scoped in \([-\lambda, +\lambda] \), where \( \lambda > 0 \) is a positive real number, and

\[
h_i(x) = \frac{1}{1 + \exp(-[a^T x + b])}.
\]

(2)

In order to find the optimal weights \( \beta \) for a specific task, assume that we have available a dataset of \( N \) labeled samples \( D = \{(x_i, y_i) \mid i = 1, \ldots, N\} \), where \( y_i \) is the desired output on the \( i \)-th 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:

\[
\beta^* = \arg \min_{\beta \in \mathbb{R}^B} \left\{ \frac{1}{2} \| H \beta - y \|_2^2 + \frac{C}{2} \| \beta \|_2^2 \right\}
\]

(3)

where \( H \) is the hidden matrix built by stacking row-wise the vectors \( h(x_i) \), and similarly for \( y \), \( C \) 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

\[
\beta^* = (H^T H + C I)^{-1} H^T y.
\]

(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 an LASSO problem replacing the \( \ell_2 \) norm regularization term in (3) with a \( \ell_1 \) 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 \( \beta \). As we stated in Section I, 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 \( \beta \) as in (3), we express our beliefs on the optimal \( \beta \) using a prior density \( p(\beta) \), e.g., an isotropic Gaussian density. Second, instead of assuming a deterministic output, we define a probability distribution \( p(y \mid x, \beta) \) 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

\[
p(y \mid X, \beta) = \prod_{i=1}^{N} p(y_i \mid x_i, \beta)
\]

(5)

where \( X \) is computed similarly to \( H \). 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

\[
p(\beta \mid X, y) = \frac{1}{Z} p(y \mid X, \beta) p(\beta)
\]

(6)

where \( Z \) is a normalization constant and independent of \( \beta \). 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 \( \hat{x} \), the predictive distribution is computed as the expected value of (5) with respect to the posterior distribution

\[
p(y \mid X, y, \hat{x}) = \int_{\mathbb{R}^B} p(y \mid \hat{x}, \beta) p(\beta \mid X, y) d\beta.
\]

(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 \( \beta \), given by a mode of the posterior

\[
\beta_{MAP} = \arg \max_{\beta} p(y \mid X, \beta).
\]

(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 Section I, 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, Sec. 3.3], [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 \( \sigma^2 \), resulting in the following form for the terms in (5):

\[
p(y \mid x, \beta, \sigma^2) = \mathcal{N}(y \mid \beta^T h(x), \sigma^2)
\]

(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 \( \sigma^2 \) is given. Our belief that the optimal weights will be relatively close to zero can be expressed by another multivariate Gaussian with diagonal covariance

\[
p(\beta \mid y) = \mathcal{N}(\beta \mid 0, \gamma^{-1} I)
\]

(10)

where we use the precision parameter \( \gamma^{-1} \) 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 \( \Sigma \) given by
\[
\mathbf{m} = \frac{1}{\sigma^2} \Sigma \mathbf{H}^T \mathbf{y}
\] (11)
\[
\Sigma^{-1} = \gamma \mathbf{I} + \frac{1}{\sigma^2} \mathbf{H}^T \mathbf{H}
\] (12)
The predictive distribution is also Gaussian and can be evaluated by
\[
p(\mathbf{y} \mid \mathbf{X}, \mathbf{y}, \hat{\mathbf{x}}, \sigma^2, \gamma) = \mathcal{N}(\mathbf{m}^T \mathbf{h}(\hat{\mathbf{x}}), \phi(\hat{\mathbf{x}})^2)
\] (13)
where the variance is computed as
\[
\phi(\hat{\mathbf{x}})^2 = \sigma^2 + \mathbf{h}(\hat{\mathbf{x}})^T \Sigma \mathbf{h}(\hat{\mathbf{x}}).
\] (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 \( \sigma^2 \) and \( \gamma \). In order to fully leverage the power of BI, we can define additional hyper-prior distributions over these parameters, allowing to treat them equivalently to the RVFL parameters \( \beta \). To avoid confusion, in this case \( \sigma^2 \) and \( \gamma \) 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
\[
p(\gamma) = \text{Gamma}(\gamma \mid \alpha_1, \alpha_2)
\] (15)
\[
p(\sigma^2) = \text{Gamma}(\sigma^2 \mid \alpha_3, \alpha_4)
\] (16)
where \( \alpha_i, i = 1, \ldots, 4 \) are the parameters of the two Gamma distributions.\(^1\) By setting them with sufficiently small values, e.g., \( \alpha_i = 10^{-3} \), we can obtain an extremely broad hyper-prior, such that most information on \( \sigma^2 \) and \( \gamma \) will be inferred from the data. Unfortunately, the predictive distribution now requires the marginalization with respect to both \( \beta \) and \( \alpha_1, \ldots, \alpha_4 \), making it intractable. A common alternative is to fix \( \sigma^2 \) and \( \gamma \) to a single value by performing an MAP estimation on the posterior of the hyper-parameters, which is obtained by first marginalizing the likelihood with respect to \( \beta \) (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:
\[
\sigma^2_\ast, \gamma_\ast = \text{arg max}\left\{ \int_{\mathbb{R}^B} p(\mathbf{y} \mid \mathbf{X}, \mathbf{\beta}, \sigma^2) \times p(\mathbf{\beta} \mid \gamma)p(\gamma)p(\sigma^2)d\mathbf{\beta} \right\}
\] (17)

\(^1\)Remember that \( \Gamma(a \mid b) = \Gamma(a) b^a e^{-b} \), 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 \( \lambda_i^0, \ldots, \lambda_i^B \).
(i-3) Initialize \( \sigma^2 \) and \( \gamma \) to default values.

(ii) Posterior update:
(ii-1) Compute posterior mean \( \mathbf{m} \) as in (11).
(ii-2) Compute posterior covariance \( \Sigma \) 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 \( \gamma \) as in (18).
(iii-3) Update \( \sigma^2 \) as in (19).

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

It turns out that we can easily maximize (17) using an iterative procedure [13]. First, we initialize \( \sigma^2 \) and \( \gamma \) to some default values. Then, at every iteration, we compute the current mean and covariance according to (11) and (12). Second, we update the current estimate of \( \sigma^2 \) and \( \gamma \) according to the following formulas:
\[
\gamma = \frac{\delta + 2\alpha_1}{\|\mathbf{m}\|^2 + 2\alpha_2}
\] (18)
\[
\sigma^2 = \frac{\|\mathbf{y} - \mathbf{H} \beta\|^2 + \alpha_4}{N - \delta + 2\alpha_3}
\] (19)
where \( \delta \) is defined as
\[
\delta = \sum_{i=1}^B \frac{\lambda_i}{\gamma + \lambda_i}
\] (20)
with \( \lambda_i \) being the \( i \)-th eigenvector of \( (1/\sigma^2)^2 \mathbf{H}^T \mathbf{H} \). 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 \( \sigma^2 \). It can be shown that \( \sigma^2 \rightarrow \sigma^2_\ast \) and \( \gamma \rightarrow \gamma_\ast \). 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 B-RVFL.

Before concluding this section, we note that the framework described in this section can be easily customized to handle sparse weight vectors \( \beta \), 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
\[
p(\mathbf{\beta} \mid \gamma) = \prod_{i=1}^B \mathcal{N}(\beta_i \mid 0, \gamma_i^{-1})
\] (21)
where \( \mathbf{y} \) 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 matrix of the Gaussian. Empirically, it is found that several $\gamma_i$ 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 $y_i \in \{0, 1\}$. In a non-Bayesian approach, it is common to binarize the real-valued output $f(x)$ 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)

$$f(x) = \text{sigmoid}(f(x)) = \frac{1}{1 + \exp(-f(x))} \quad (22)$$

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

$$p(y_i | x_i, \beta) = f(x_i)^{y_i}(1 - f(x_i))^{1-y_i} \quad (23)$$

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 $p(\beta, \theta | X, y)$ a posterior distribution resulting from a generic combination of likelihood, prior, and hyper-priors distributions. From now onward, $\theta$ is a vector containing all hyper-parameters of our model. For instance, in B-RVFL we would have $\theta = \{\sigma^2, \gamma\}$, while for ARD-RVFL we would have $\theta = \{\sigma^2, \gamma_1, \ldots, \gamma_B\}$. 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 is handled easily by simple scalar transformations (see [28, Sec. 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 $q(\beta, \theta; \xi)$, which is parameterized by a set of adaptable coefficients $\xi$ known as variational parameters. In the simplest case, i.e., mean-field approximation, we approximate the posterior with a product of univariate Gaussian distributions

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

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 $\xi = \{\eta_1, \ldots, \eta_{B+K}, \delta_1, \ldots, \delta_{B+K}\}$ 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) divergence2 between the two distributions as defining our optimal variational approximation

$$\xi^* = \arg\min_{\xi \in \mathbb{R}^{2(B+K)}} \left\{ \text{KL} \left( q(\cdot) \parallel p(\beta, \theta | X, y) \right) \right\} \quad (25)$$

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

$$\text{KL} \left( q(\cdot) \parallel p(\beta, \theta | X, y) \right) = \mathbb{E}[\log(q(\cdot))] - \mathbb{E}[\log(p(\beta, \theta, X, y))] + \log(p(X, y)) \quad (26)$$

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,

2Remember 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].
thus effectively removing it [28]

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

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 reparameterization 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 $\beta_i$ to a standard Gaussian $v_i = S(\beta_i) = \exp((\omega_i)^{-1}(\beta_i - \eta_i))$, and similarly for the hyper-parameters $\theta$. The resulting variational density is given by

$$q(\nu) = \mathcal{N}(\nu | 0, I). \quad (28)$$

Using this transformation, we can rewrite our ELBO objective as

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

where $\mathbb{H}(q(\cdot))$ 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

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

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 $p(\mathbf{\beta}, \mathbf{\theta}, \mathbf{X}, y)$ with respect to our original parameters. The expectation of this gradient can be computed by drawing on Monte Carlo integration as

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

where we draw $S$ samples $v_i \sim q(\mathbf{\nu})$ 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 improvment 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 Theano6 as the underground symbolling modeling tool.

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 six 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

3https://bitbucket.org/ispamm/bayesian-rvfl
4http://scikit-learn.org/
5https://github.com/pymc-devs/pymc3
6http://deeplearning.net/software/theano/
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 $a$ and $b$ are extracted randomly from a uniform distribution over the interval $[-\lambda, +\lambda]$. Selecting a feasible $\lambda$ on each dataset is fundamental for a proper behavior of the algorithms. In order to select $\lambda$, 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 tenfold 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 an RVFL network trained using the least-squares procedure in (3). The regularization factor $C$ in this case is selected from the exponential interval $\{2^j\}, j = -10, \ldots, 10$ for each run by minimizing the MSE over a tenfold cross-validation of the training data. Given a generic test set $\mathcal{T} = \{(x_i, y_i) | i = 1, \ldots, T\}$, we compute the MSE and two additional accuracy measures. The first is the coefficient of determination $R^2$, given by

$$R^2 = 1 - \frac{\sum_{i=1}^{T} (y_i - f(x_i))^2}{\sum_{i=1}^{T} (y_i - \bar{y})^2}$$

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

$$E_{VAR} = 1 - \frac{\text{Var}_T(y_i - f(x_i))}{\text{Var}_T(y_i)}$$

where $\text{Var}_T(\cdot)$ 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

---

The results in Table II show a clear superiority of B-RVFL over standard RVFL, achieving 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 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 $\lambda$ 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 $\lambda$ 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. 3, 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 $\lambda$ outside such interval seriously degrades the performance. Specifically, for very small $\lambda$, 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 $\lambda$ 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 $\beta$, which is particularly helpful for hardware implementations (among others). In this case, we compare against an RVFL network optimizing a standard LASSO objective, which we denote as L-RVFL

$$
\beta_{LASSO}^1 = \arg \min_{\beta \in \mathbb{R}^B} \left\{ \frac{1}{2} \|H\beta - y\|_2^2 + C \sum_{i=1}^{B} |\beta_i| \right\}. \quad (34)
$$

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 $\beta$ 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, particularly Airfoil. If we optimize L-RVFL with a small
intervals, 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 \( \beta \), 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

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

(35)
where the additional parameter $\epsilon$ is called the degrees of freedom of the distribution. For $\epsilon \to \infty$ the distribution degenerates to a standard Gaussian distribution. We consider a Gaussian prior over $\beta$ 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 $\epsilon$ 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, 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. CONCLUSION

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 thoroughly 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].

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for the invaluable suggestions on improving this paper.

REFERENCES


Simone Scardapane (S’12–M’16) received the B.Sc. degree in computer engineering from Roma Tre University, Rome, Italy, in 2009, and the M.Sc. degree in artificial intelligence and robotics and the Ph.D. degree in 2016 from the “Sapienza” University of Rome, Rome.

He was a Software/Web Developer researching mainly in the field of distributed machine learning and adaptive audio processing with the “Sapienza” University of Rome, where he is currently a Post-Doctoral Fellow. He is an Honorary Research Fellow with theCogBiD Laboratory, University of Stirling, Stirling, U.K.

Dianhui Wang (M’03–SM’05) received the Ph.D. degree from the School of Information Science and Engineering, Northeastern University, Shenyang, China, in 1995.

From 1995 to 1997, he was a Post-Doctoral Fellow with the School of Electronic and Electrical Engineering, Nanyang Technological University, Singapore. Then, he was a Research Associate and a Research Fellow with Hong Kong Polytechnic University, Hong Kong, for three years until 2001. Since 2001, he has been with the Department of Computer Science and Computer Engineering, La Trobe University, Melbourne, VIC, Australia, where he is currently a Reader and an Associate Professor.

Dr. Wang is serving as the Editor-in-Chief for the International Journal of Machine Intelligence and Sensory Signal Processing, an Associate Editor for several international journals, including the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, the IEEE TRANSACTIONS ON CYBERNETICS, Information Sciences, Neurocomputing, and a Subject Editor for Applied Mathematical Modelling.

Aurelio Uncini (M’88) received the Laurea degree in electronic engineering from the University of Ancona, Ancona, Italy, in 1983 and the Ph.D. degree in electrical engineering from the University of Bologna, Bologna, Italy, in 1994.

He is currently a Full Professor with the Department of Information Engineering, Electronics and Telecommunications, “Sapienza” University of Rome, Rome, Italy, where he is teaching neural networks, adaptive algorithm for signal processing and digital audio processing, and the Founder and the Director of the “Intelligent Signal Processing and Multimedia” Group. His current research interests include adaptive filters, adaptive audio and array processing, machine learning for signal processing, blind signal processing, and multisensors data fusion.

Dr. Uncini is a member of the Associazione Elettrotecnica ed Elettronica Italiana, the International Neural Networks Society, and the Società Italiana Reti Neuroniche.