Sparse Bayesian Recurrent Neural Networks

Sotirios P. Chatzis(✉)

Department of Electrical Engineering, Computer Engineering and Informatics, Cyprus University of Technology, 33 Saripolou Str., 3036 Limassol, Cyprus
sotirios.chatzis@cut.ac.cy

Abstract. Recurrent neural networks (RNNs) have recently gained renewed attention from the machine learning community as effective methods for modeling variable-length sequences. Language modeling, handwriting recognition, and speech recognition are only few of the application domains where RNN-based models have achieved the state-of-the-art performance currently reported in the literature. Typically, RNN architectures utilize simple linear, logistic, or softmax output layers to perform data modeling and prediction generation. In this work, for the first time in the literature, we consider using a *sparse Bayesian* regression or classification model as the output layer of RNNs, inspired from the automatic relevance determination (ARD) technique. The notion of ARD is to continually create new components while detecting when a component starts to overfit, where overfit manifests itself as a *precision hyper-parameter posterior* tending to infinity. This way, our method manages to train sparse RNN models, where the number of effective ("active") recurrently connected hidden units is selected in a data-driven fashion, as part of the model inference procedure. We develop efficient and scalable training algorithms for our model under the stochastic variational inference paradigm, and derive elegant predictive density expressions with computational costs comparable to conventional RNN formulations. We evaluate our approach considering its application to challenging tasks dealing with both regression and classification problems, and exhibit its favorable performance over the state-of-the-art.

1 Introduction

Many naturally occurring phenomena such as music, speech, or human motion are inherently sequential. As a consequence, the problem of sequential data modeling is an important area of machine learning research. Recurrent neural networks (RNNs) [22] are among the most powerful models for sequential data modeling. As shown in [12], RNNs possess the desirable property of being universal approximations, as they are capable of representing any measurable sequence to sequence mapping to arbitrary accuracy. RNNs incorporate an internal memory module designed with the goal to summarize the entire sequence history in the form of high dimensional *state vector* representations. This architectural selection allows for RNNs to model and represent long-term dependencies in the observed data, which is a crucial merit in the context of sequential data modeling applications.

A. Appice et al. (Eds.): ECML PKDD 2015, Part II, LNAI 9285, pp. 359–372, 2015.
DOI: 10.1007/978-3-319-23525-7_22

A major challenge RNN-based architectures are confronted with concerns the fact that it is often the case that gradient-based optimization results in error signals either blowing up or decaying exponentially for events many time steps apart, rendering RNN training largely impractical [6,19]. A great deal of research work has been devoted to the amelioration of these difficulties, usually referred to as the problem of *vanishing and exploding gradients*. One first attempt towards this end consisted in coming up with special architectures, robust to the vanishing and exploding gradients problem. Long short-term memory (LSTM) [13] networks constitute the most successful architecture developed for this purpose, having been shown to yield the state-of-the-art performance in speech and handwriting recognition tasks [11]. In a different vein, [15] proposed the echo-state network (ESN) architecture, which consists in completely abandoning gradient-based training of the recurrent connection weights (which gives rise to the vanishing and exploding gradients problem). As such, ESNs solely rely on sensible initializations of the recurrent connection weights, and limit training to the connection weights of the output (*readout*) layer of the network. Finally, a recent breakthrough in the literature of RNNs has been accomplished in the landmark publication of [17], where it was shown that even standard RNN architectures can be successfully trained with the right optimization method. While a sophisticated Hessian-free optimizer for RNNs was developed therein, further research has shown that carefully designed conventional first-order methods can find optima of similar or slightly worse quality in the context of RNN training [24].

Despite these advances in the field of RNN research, a problem that has not been tackled by the machine learning community concerns data-driven model selection. The problem of model selection consists in coming up with model treatments allowing for an RNN model to automatically infer the number of necessary hidden units, as part of its training procedure. Our work constitutes the first attempt towards addressing these shortcomings. To achieve our goals, we introduce the concept of training RNN models under a Bayesian inferential procedure. Specifically, we consider imposing appropriate *sparsity-promoting prior distributions on the output connection weights* of RNN models. Under this construction, we essentially give rise to a Bayesian inference procedure for RNNs that yields sparse posterior distributions over the output connection weights, and associated predictive posteriors that characterize the output variables of the model. Under a different regard, our approach can be viewed as introducing a *sparse Bayesian* regression or classification model as the *output layer* of RNNs, resulting in a sparse *Bayesian treatment* of the whole RNN architecture.

Sparsity in the context of our model is induced by adopting a prior model configuration inspired from an inference technique widely known as automatic relevance determination (ARD) [10]. The notion of ARD is to continually create new model components while detecting when a component starts to overfit. Overfit manifests itself as a *precision hyperparameter posterior* tending to infinity, indicating that a single data value is being modeled by the component. In the case of the proposed model, the ARD mechanism is implemented by imposing an

appropriate hierarchical prior over the weights of the output layer connections, which results in an efficient mechanism for *automatically inferring the effective number* of ("*active*") recurrently connected hidden units, in a data-driven fashion. We derive an efficient and scalable training algorithm for our model under the *stochastic variational* inference (SVI) paradigm [14], exploiting the most recent advances on gradient-based RNN training algorithms. We dub our approach sparse Bayesian RNN (SB-RNN).

The layout of the paper is as follows. In Section 2, we briefly introduce the concept of RNNs, and discuss modern RNN training algorithms that obviate the vanishing and exploding gradients problem. In Section 3, we present our method and derive efficient model training and inference algorithms. In Section 4, we perform an extensive experimental evaluation of our approach considering several challenging benchmark tasks; we compare the obtained performance of our approach to related state-of-the-art approaches. Finally, in Section 5 we summarize our results, discussing the shortcomings and advantages of the proposed model, and conclude this work.

2 Recurrent Neural Networks

RNNs are neural network architectures designed for modeling sequential data with long temporal dynamics. RNNs operate by simulating a discrete-time dynamical system presented with M-dimensional inputs $\{\boldsymbol{x}_t\}_{t=1}^{T}$, with corresponding N-dimensional outputs $\{\boldsymbol{y}_t\}_{t=1}^{T}$, where T is the length (time-duration) of the observed sequences. The postulated dynamical system is defined by

$$\boldsymbol{y}_t = \phi_0(\boldsymbol{V}\boldsymbol{h}_t) \tag{1}$$

where ϕ_0 is the activation function of the output units, $\boldsymbol{V} \in \mathbb{R}^{N\times H}$ is the parameters (weights) matrix of the connections of the output layer of the model, and $\boldsymbol{h}_t \in \mathbb{R}^H$ is the hidden state vector of the model. In essence, $\boldsymbol{h}_t$ is the vector of the activations of the hidden units of the network, which encodes the history of observations presented to the system up to time t in the form of a high-dimensional data point representation. Typically, the expressions of the hidden state vectors of the postulated dynamical system are considered to be given by the following expression:

$$\boldsymbol{h}_t = \begin{cases} \phi_h(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{\Omega}\boldsymbol{x}_t), t > 0 \\ \mathbf{0}, t = 0 \end{cases} \tag{2}$$

In Eq. (2), ϕ_h is the activation function of the hidden (recurrently connected) units that capture the temporal dynamics in the modeled data, $\boldsymbol{W} \in \mathbb{R}^{H\times H}$ is the recurrent connections weights matrix of the model, and $\boldsymbol{U} \in \mathbb{R}^{H\times M}$ is the input connections weights matrix. Regarding selection of the hidden and output unit activation functions, typically a saturating nonlinear function is used, such as a logistic sigmoid function or a hyperbolic tangent function.

RNN model training, i.e. optimal estimation of the parameter values of the model given a set comprising S training sequences, $D = \{(\boldsymbol{x}_1^s, \boldsymbol{y}_1^s), \ldots, (\boldsymbol{x}_{T_s}^s, \boldsymbol{y}_{T_s}^s)\}_{s=1}^S$, can be performed by minimization of the cost function

$$J(\Theta) = \frac{1}{S} \sum_{s=1}^{S} \sum_{t=1}^{T_s} d(\boldsymbol{y}_t^s, \phi_0(\boldsymbol{V}\boldsymbol{h}_t^s)) \tag{3}$$

where Θ is the parameters set of the RNN, $\Theta = \{\boldsymbol{V}, \boldsymbol{W}, \boldsymbol{U}\}$, $\boldsymbol{h}_t^s = \phi_h(\boldsymbol{W}\boldsymbol{h}_{t-1}^s + \boldsymbol{U}\boldsymbol{x}_t^s)$, and $d(\boldsymbol{a}, \boldsymbol{b})$ is a suitable divergence measure, appropriate for the learning problem at hand (e.g., a Euclidean distance function when dealing with regression problems, and a cross-entropy function in cases of classification problems).

To effect the minimization task of the RNN objective function $J(\Theta)$ in a scalable manner, stochastic gradient descent (SGD) algorithms are typically used, with the gradient of the cost function in Eq. (3) computed by means of backpropagation through time (BPTT) [22]. However, as we already discussed in the introduction of this paper, conventional BPTT-based RNN training is often confronted with the problem of the obtained error signals either blowing up or decaying exponentially for events many time steps apart, with detrimental effects to the outcome of the model training algorithm. The effort to mitigate these hurdles has recently triggered a significant corpus of new research in the field of RNN methods. Among this large corpus of works, we here focus on a very recent research result showing that first-order optimizers can, indeed, avoid the problem of exploding and vanishing gradients by: (i) performing an appropriate initialization of the model weights based on the principles of ESNs; and (ii) using Nesterov's Accelerated Gradient (NAG) [18] to perform model training instead of conventional SGD [24].

As discussed in Section 1, ESNs are RNN-based architectures where the recurrent connections weights matrix $\boldsymbol{W}$ is not trained but merely properly initialized. Specifically, ESN theory stipulates that the initialization of the matrix $\boldsymbol{W}$ should be performed in a way ensuring that the *largest absolute eigenvalue of the determinant* $|\boldsymbol{W}|$ (*spectral radius*) be close to one. This way, the dynamics of the network can be shown to become oscillatory and chaotic, allowing it to generate responses that are varied for different input histories. At the same time though, the gradients are no longer exploding (and if they do explode, then only "slightly so"), so learning may be possible for even the hardest sequential data modeling problems that conventional RNNs fail to address [15].

On the other hand, NAG has been the subject of much recent attention by the convex optimization community [9,16]. Like SGD, NAG is a first-order optimization method with better convergence rate guarantee than conventional SGD. NAG has been shown to be closely related to classical momentum-based SGD variants (e.g., [20]), with the only difference lying in the precise update expression of the velocity vector of the algorithm. These differences, although slight, can be of major significance to the asymptotic rates of local convergence of the optimization algorithm, as discussed in [24].

Inspired from these merits, in developing the training algorithm of our proposed model we shall rely on both performing an ESN-inspired initialization,

and using NAG instead of mainstream SGD optimization algorithms. We shall introduce our model and elaborate on our selected training strategies in the following section.

3 Proposed Approach

We differentiate SB-RNN model formulation between regression and (multiclass) classification tasks, to allow for better handling the intricacies of each type of problems. In the following, we elaborate on our modeling strategies in both cases, and derive efficient training algorithms under the SVI paradigm.

3.1 Regression SB-RNN

Let us consider that the modeled output variables $\boldsymbol{y}_t$ are N-dimensional real vectors, i.e. $\boldsymbol{y}_t \in \mathbb{R}^N$. In this case, we define a likelihood function for our model of the form

$$p(\boldsymbol{y}_t|\boldsymbol{x}_t; \boldsymbol{V}, \beta) = \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{V}\boldsymbol{h}_t, \beta^{-1}\boldsymbol{I}) \tag{4}$$

where β is the precision (inverse variance) of a simple white noise model adopted in the context of our method, and $\boldsymbol{h}_t$ is the *state vector* of our model that encodes the history of past observations $\{\boldsymbol{x}_\tau\}_{\tau=1}^{t-1}$. In essence, $\boldsymbol{h}_t$ consists the vector of the activations of the (recurrently connected) hidden units of our model; we consider that it is expressed by a discrete-time dynamical system of the form (2). In the same vein, $\boldsymbol{V}$ can essentially be perceived as the weight matrix of the output connections of our model.

Further, we introduce an appropriate prior distribution over the weights matrix $\boldsymbol{V}$; we consider a zero-mean Gaussian prior of the form

$$p(\boldsymbol{V}|\boldsymbol{\alpha}) = \prod_{n=1}^{N}\prod_{u=1}^{H}\mathcal{N}(v_{nu}|0, \alpha_u^{-1}) \tag{5}$$

where α_u is the precision of the weights pertaining to the uth hidden unit, $\{v_{nu}\}_{n=1}^{N}$, and $\boldsymbol{\alpha} = [\alpha_u]_{u=1}^{H}$. Finally, we impose a Gamma hyper-prior over the precision hyperparameters α_u, yielding

$$p(\alpha_u) = \mathcal{G}(\alpha_u|\eta_1, \eta_2) \tag{6}$$

This hierarchical prior configuration of our model essentially gives rise to an ARD mechanism that allows for data-driven inference over the number of hidden units, H. As we have already discussed, the notion of ARD is to continually create new components while detecting when a component starts to overfit. Overfit manifests itself as a *precision hyperparameter posterior* tending to infinity, indicating that a single data value is being modeled by the component. Hence, in the case of our SB-RNN model, the ARD mechanism is implemented by imposing a hierarchical prior over the output weights matrix $\boldsymbol{V}$, to discourage large weight values, with the width of each prior being controlled by a *Gamma distributed*

precision hyperparameter, α_u, as illustrated in Eqs. (5)-(6). If one of these precisions tends to infinity, $\alpha_u \to \infty$, then the outgoing weights will have to be very close to zero in order to maintain a high likelihood under this prior. This in turn leads the model to ignore the likelihood contribution of the corresponding hidden unit, which is effectively 'switched off' of the model.

We underline that this approach is substantially different from dropout [5,23], where, on each iteration of the training algorithm, *different* hidden units are *randomly* ignored, while *all* hidden units are used to perform prediction.

Model Training. To perform training of our model in a way scalable to massive datasets, we resort to the SVI paradigm [14]. SVI is an iterative stochastic optimization algorithm for mean-field variational inference that approximates the posterior distribution of a probabilistic model, and can handle massive datasets of observations. Indeed, SVI renders Bayesian inference scalable to massive datasets by splitting the observed data into small batches, and letting the inference algorithm operate only on one batch on each iteration.

SVI yields a lower bound to the log-evidence of the treated model (evidence lower bound, ELBO) expressed as a function of an approximate (variational) posterior distribution it seeks to optimally determine. Then, inference for a treated model consists in maximizing the corresponding ELBO over the variational posterior and the estimates of the associated hyper-parameters. For this purpose, SVI computes on each iteration a set of noisy estimates of the *natural* gradient [2] of the model's ELBO, and uses them in the context of a stochastic optimization scheme.

In the following, we denote as $\langle \cdot \rangle$ the posterior expectation of a quantity; the analytical expressions of these posteriors can be found in the Appendix. Let us consider that the training set $D = \{(\boldsymbol{x}_1^s, \boldsymbol{y}_1^s), \ldots, (\boldsymbol{x}_{T_s}^s, \boldsymbol{y}_{T_s}^s)\}_{s=1}^S$ comprises S sequences, and that we split this dataset into batches comprising S_b sequences each. Then, the SVI algorithm for the proposed SB-RNN regression model yields the following posterior over the output weights matrix $\boldsymbol{V}$:

$$q(\boldsymbol{V}) = \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{v}_n | \bar{\boldsymbol{v}}_n, \bar{\boldsymbol{\Sigma}}) \tag{7}$$

where the posterior hyperparameters $\bar{\boldsymbol{v}}_n$ and $\bar{\boldsymbol{\Sigma}}$ are updated according to

$$\bar{\boldsymbol{\Sigma}}^{-1} \leftarrow (1-\rho_k)\bar{\boldsymbol{\Sigma}}^{-1} + \rho_k \left[\langle \boldsymbol{A} \rangle + (\beta \tilde{\boldsymbol{H}}^T \tilde{\boldsymbol{H}}) \frac{S}{S_b} \right] \tag{8}$$

$$\bar{\boldsymbol{v}}_n \leftarrow (1-\rho_k)\bar{\boldsymbol{v}}_n + \rho_k \frac{S}{S_b} \beta \bar{\boldsymbol{\Sigma}} \tilde{\boldsymbol{H}}^T \tilde{\boldsymbol{y}}_n \tag{9}$$

where $\langle \boldsymbol{A} \rangle = \mathrm{diag}(\langle \boldsymbol{\alpha} \rangle)$. In Eqs. (8)-(9), $\tilde{\boldsymbol{H}}$ denotes the matrix of the network state vectors pertaining to the sequences included in the current batch, while $\tilde{\boldsymbol{y}}_n$ denotes the (target) values of the nth model output pertaining to the sequences included in the current batch. On the other hand, ρ_k is the *learning rate* of

the developed stochastic updating algorithm on the current (say, kth) iteration. Following common practice (e.g., [14]), in this work the learning rates ρ_k are updated on each algorithm iteration according to the rule

$$\rho_k = (k + \kappa)^{-f} \tag{10}$$

where the delay κ satisfies $\kappa \geq 0$, and the forgetting rate f satisfies $f \in (0.5, 1]$.

Further, the precision hyperparameters $\boldsymbol{\alpha}$ yield the hyperposteriors:

$$q(\boldsymbol{\alpha}) = \prod \mathcal{G}(\alpha_u | \bar{\eta}_{1u}, \bar{\eta}_{2u}) \tag{11}$$

where

$$\bar{\eta}_{1u} = \eta_1 + N\frac{S}{2} \tag{12}$$

$$\bar{\eta}_{2u} \leftarrow (1 - \rho_k)\bar{\eta}_{2u} + \rho_k \left[\eta_2 + \frac{S}{2S_b} \sum_{n=1}^{N} \left\langle v_{nu}^2 \right\rangle \right] \tag{13}$$

Interestingly, note that the updates of $\bar{\eta}_{1u}$ do not depend on the training data points of each batch; as such, the value of $\bar{\eta}_{1u}$ need not be updated on each algorithm iteration.

Finally, the input weights matrix $\boldsymbol{\Omega}$ and the recurrent connection weights matrix $\boldsymbol{W}$ of our model are updated as model hyperparameters, yielding point-estimates. This is effected by optimization of the ELBO of our model by application of a NAG-type optimization procedure, yielding:

$$\boldsymbol{W} \leftarrow (1 - \rho_k)\boldsymbol{W} + \rho_k \beta \delta_{\boldsymbol{W}} + \boldsymbol{\mu}_{\boldsymbol{W}} \tag{14}$$

$$\boldsymbol{\Omega} \leftarrow (1 - \rho_k)\boldsymbol{\Omega} + \rho_k \beta \delta_{\boldsymbol{\Omega}} + \boldsymbol{\mu}_{\boldsymbol{\Omega}} \tag{15}$$

In these equations, $\delta_{\boldsymbol{W}}$ and $\delta_{\boldsymbol{\Omega}}$ are the updates of the weights matrices $\boldsymbol{W}$ and $\boldsymbol{\Omega}$ obtained by application of *conventional BPTT [22]*, by setting the value of $\boldsymbol{V}$ equal to its posterior expectation $\bar{\boldsymbol{V}} = [\bar{\boldsymbol{v}}_n]_{n=1}^N$. In addition, $\boldsymbol{\mu}_{\boldsymbol{W}}$ and $\boldsymbol{\mu}_{\boldsymbol{\Omega}}$ are the momentum-type terms introduced by adoption of the NAG optimization scheme (c.f. [24]), as discussed in Section 2. Initialization of the recurrent connection weights matrix $\boldsymbol{W}$ is performed by adopting the principles of ESN architectures, as also discussed in Section 2.

Predictive Density. Having found estimates of the model hyperparameters and parameter posteriors, we can now proceed to derive the expression of its predictive distribution over the output variables $\boldsymbol{y}_t$ for a new input $\boldsymbol{x}_t$, with corresponding observation history $\{\boldsymbol{x}_\tau\}_{\tau=1}^{t-1}$ and state vector $\boldsymbol{h}_t$. We have

$$q(\boldsymbol{y}_t | \{\boldsymbol{x}_\tau\}_{\tau=1}^{t}) = \mathcal{N}(\boldsymbol{y}_t | \bar{\boldsymbol{V}}\boldsymbol{h}_t, \sigma^2(\boldsymbol{x}_t)\boldsymbol{I}) \tag{16}$$

where

$$\sigma^2(\boldsymbol{x}_t) = \beta^{-1} + \boldsymbol{h}_t^T \bar{\boldsymbol{\Sigma}} \boldsymbol{h}_t \tag{17}$$

It is worthwhile to underline here a significant difference between our approach and conventional RNN formulations when it comes to prediction generation: Conventional RNNs only provide an estimate of the target (output variables); instead, our SB-RNN approach, apart from this estimate (taken as the mode $\hat{\boldsymbol{y}} = \bar{\boldsymbol{V}}\boldsymbol{h}_t$ of the predictive distribution) does also yield a *predictive variance* estimate, given by $\sigma^2(\boldsymbol{x}_t)$. The obtained predictive variance is in essence a measure of the confidence of the model in the obtained predictions $\hat{\boldsymbol{y}}$, and can be utilized to provide error bars (or a reject option in safety-critical applications).

3.2 Classification SB-RNN

We now turn to the case of modeling a multiclass classification problem using our SB-RNN approach. Let us denote as $\boldsymbol{y}_t \in \{0,1\}^N$ the output variables of the addressed problem. In this case, the nth component of vector $\boldsymbol{y}_t$ indicates whether class $n \in \{1,\ldots,N\}$ is on or off at time t. On this basis, to obtain a suitable construction for our model, we postulate a standard Multinomial likelihood assumption of the form:

$$p(\boldsymbol{y}_t|\boldsymbol{x}_t;\boldsymbol{V}) = \prod_{n=1}^{N} \phi_0(\boldsymbol{v}_n^T\boldsymbol{h}_t)^{y_{tn}} \tag{18}$$

where $\boldsymbol{v}_n$ is the nth row of matrix $\boldsymbol{V}$, and ϕ_0 is a *sigmoid* activation function. We impose the same hierarchical prior over the output weights matrix $\boldsymbol{V}$ as in the previously examined regression setting, given by Eqs. (5)-(6), to introduce the ARD mechanism into our model.

Model Training. To perform SB-RNN model training in the classification setting, we can again resort to the SVI inference paradigm. However, a major obstacle to the application of SVI in this setting concerns the fact that the imposed likelihood (18) does not yield a conjugate model formulation. This in turn prohibits obtaining closed-form analytical expressions for the (variational) posterior distribution over the weights matrix $\boldsymbol{V}$. Specifically, we have

$$\log q(\boldsymbol{v}_n) \propto \sum_t y_{tn} \log\phi_0(\boldsymbol{v}_n^T\boldsymbol{h}_t) + \langle \log p(\boldsymbol{v}_n|\boldsymbol{\alpha})\rangle \;\; \forall n \tag{19}$$

To resolve these issues, in this work we resort to a Laplace approximation of the intractable posteriors $q(\boldsymbol{v}_n)$. Laplace approximation consists in taking the second order Taylor expansion of $\log q(\boldsymbol{v}_n)$ around its mode, resulting in the considered posterior distribution being conveniently approximated by a Gaussian. Specifically, our model yields

$$q(\boldsymbol{v}_n) \approx \mathcal{N}(\boldsymbol{v}_n|\bar{\boldsymbol{v}}_n, \bar{\boldsymbol{\Sigma}}_n) \tag{20}$$

where

$$\bar{\boldsymbol{\Sigma}}_n^{-1} \leftarrow (1-\rho_k)\bar{\boldsymbol{\Sigma}}_n^{-1} + \rho_k \left[\langle \boldsymbol{A}\rangle + (\tilde{\boldsymbol{H}}^T \tilde{\boldsymbol{B}}_n \tilde{\boldsymbol{H}})\frac{S}{S_b}\right] \tag{21}$$

$$\bar{\boldsymbol{v}}_n \leftarrow (1-\rho_k)\bar{\boldsymbol{v}}_n + \rho_k \frac{S}{S_b} \bar{\boldsymbol{\Sigma}}_n \tilde{\boldsymbol{H}}^T \tilde{\boldsymbol{B}}_n \tilde{\boldsymbol{y}}_n \tag{22}$$

and $\tilde{\boldsymbol{B}}_n$ is the diagonal matrix of the set of quantities $\phi_0(\boldsymbol{v}_n^T \boldsymbol{h}_t)$ corresponding to the sequences in the current batch.

On the basis of the derivations (20)-(22), the updates of the hyperposteriors $q(\boldsymbol{\alpha})$, as well as the updates of the model weight matrices $\boldsymbol{W}$ and $\boldsymbol{\Omega}$, yield exactly the same expressions as in (11)-(15).

Predictive Density. Having obtained the training algorithm expressions of the SB-RNN model for the case of dealing with classification tasks, we now turn to deriving the corresponding predictive density expressions. Based on the preceding discussions, the predictive density of our model yields:

$$q(y_{tn} = 1 | \{\boldsymbol{x}_\tau\}_{\tau=1}^t) \propto \left\langle \phi_0(\boldsymbol{v}_n^T \boldsymbol{h}_t) \right\rangle \tag{23}$$

where the state vectors $\boldsymbol{h}_t$ are given by (2). Note that the posterior expectations in (23) cannot be computed analytically due to the nonlinear nature of the activation function $\phi_0(\cdot)$. For this reason, we resort to a Monte Carlo sampling-based approximation, yielding:

$$\left\langle \phi_0 \left(\boldsymbol{v}_n^T \boldsymbol{h}_t \right) \right\rangle \approx \frac{1}{Z} \sum_{\zeta=1}^{Z} \phi_0 \left((\boldsymbol{v}_n^\zeta)^T \boldsymbol{h}_t \right) \tag{24}$$

where Z is the number of samples $\boldsymbol{v}_n^\zeta$ drawn i.i.d. from the posterior $q(\boldsymbol{v}_n)$, approximated by (20).

4 Experiments

We experimentally evaluate our approach in both regression and classification tasks. In all cases, we manually tune the hyperparameters of the learning rate schedule (10) for each dataset, as well as the hyperparameters of the momentum terms, similar to [24]. We developed our source codes in Python, using the Theano library [4][1]. We run our experiments on an Intel Xeon 2.5GHz Quad-Core server with 64GB RAM and an NVIDIA Tesla K40 GPU.

4.1 Human Motion Modeling

We begin by evaluating our method in a *regression* task. For this purpose, we use a publicly available benchmark, namely walking sequences from the CMU motion capture (MoCap) dataset [1]. The considered training sequences correspond to several different subjects included in the CMU MoCap database, following the experimental setup of [8]. After training, we use the obtained models to generate

[1] The source codes will be made available through our website, to allow for easier reproducibility of our results.

the human pose information in a different set of walking sequence videos, namely videos 35-03, 12-02, 16-21, 12-03, 07-01, 07-02, 08-01, and 08-02 of the same database[2]. The inputs presented to the evaluated algorithms are the positions of the tracked human joints, and their output is the predicted joint positions at a time point of interest. The dimensionality of the input space is equal to 62, similar to the output space.

To obtain some comparative results, apart from our method we also evaluate conventional RNNs trained as described in Section 2; we also cite the performance of ESNs, an ESN-driven formulation of Gaussian processes dubbed the ESGP method [8], and the Dynamic GP method [25]. In Table 1, we provide the RMSEs obtained by each one of the considered methods. These results correspond to optimal numbers of hidden units for the evaluated recurrent network-based methods; interestingly, this optimal model size turned out to be equal to 100 hidden units in all cases, as also observed in [8]. As we illustrate in Table 1, the SB-RNN method outperforms all the rest of the evaluated methods, both in average and in each single individual experimental case considered here.

Table 1. Human Motion Modeling: Obtained missing frames RMSEs.

Video ID	Dynamic GP	ESN	ESGP	RNN	SB-RNN
35-03	49.68	62.55	32.59	35.11	32.28
12-02	54.96	63.14	45.32	42.88	39.58
16-21	78.05	98.74	59.03	51.17	48.02
12-03	63.63	72.12	46.25	47.09	44.14
07-01	84.12	121.47	77.34	76.18	75.69
07-02	80.77	100.94	73.88	75.37	72.87
08-02	95.52	120.45	101.54	95.66	94.66
08-01	82.66	152.44	118.0	97.54	93.05
Average	73.67	98.98	69.24	65.13	62.54

4.2 Acoustic Novelty Detection

Further, in this experiment we perform evaluation of our approach in the context of a *classification* task, and under a setup that also allows for evaluating the *quality* of the obtained *predictive distributions.* Specifically, we consider the problem of novelty detection in acoustic signals. For the purposes of this experiment, we use a dataset composed of around three hours of recordings of a home environment, taken from the PASCAL CHiME speech separation and recognition challenge dataset [3]. Our dataset corresponds to a typical in-home scenario (a living room), recorded during different days and times; the inhabitants are two adults and two children that perform common actions, namely *talking*, *watching*

[2] All videos have been downsampled by a factor of 4, following the experimental setup of [8].

television, playing, and *eating*. On this basis, we use randomly chosen sequences to compose 100 minutes of background for training set, around 40 minutes for validation set, and another 30 minutes for test set. The validation and test sets were generated by randomly adding in the available sequences different kinds of sounds, namely *screams*, *alarms*, *falls*, and *fractures*. The total duration of *each novel* type is equal to 200 s.

Our experimental setup is the following: Initially, we train our model considering as input variables the auditory spectral features (ASF) computed by means of the short-time Fourier transformation (STFT); we use a frame size of 30 ms and a frame step of 10 ms. Each STFT yields the power spectrogram of the signal, which is eventually converted to the Mel-Frequency scale using a filter-bank with 26 triangular filters; we use a logarithmic representation of these features, to match the human perception of loudness. Finally, we also include the frame energy in our feature vectors, following standard practices in the literature.

Subsequently, we use the trained model to predict the class corresponding to each frame in the validation set. Since some frames correspond to *novel* classes which the model has not been trained to recognize, we are interested in examining how certain the model is for its predictions when these novel classes are actually the ones that appear in the data. Indeed, one would expect that the model should yield low predictive probability values for the *winner* class in cases where the actual class belongs to the set of *novel* ones. To examine whether this assumption does actually hold, we use the results obtained from our validation set to determine a *novelty threshold* for our model: if the predictive probability pertaining to the winner class is lower than this threshold, we consider that the current data frame actually belongs to a *novel class*. Determination of this threshold is performed on the basis of two different criteria: (i) maximization of the *precision* of the model in the task of novelty detection; (ii) maximization of the *recall* of the model in the task of novelty detection. Eventually, we utilize the so-obtained thresholds to measure the novelty detection precision and recall of our model using the available test set.

To obtain some comparative results, apart from our method we also evaluate conventional RNNs (trained as described in Section 2), and the state-of-the-art I/O-RNN-RBM and I/O-RNN-NADE methods presented in [7], under the same experimental setup. Our results are depicted in Table 2; these results our obtained for the best-performing model size in all cases. We observe that our method yields a very competitive result both in terms of the obtained precision and the yielded recall on the test set.

Table 2. Acoustic novelty detection: Precision and recall (%) of the evaluated models.

Model	Size	Precision	Recall
RNN	600	90.12	86.21
I/O-RNN-RBM	400	91.87	87.55
I/O-RNN-NADE	400	92.15	88.03
SB-RNN	600	92.30	88.56

4.3 Computational Complexity

Let us now turn to an analysis of the computational complexity of our method, and how it compares to conventional RNNs (trained as discussed in Section 2). We begin with the case of regression tasks: From the computational complexity perspective, the main difference between our approach and conventional RNNs concerns the fact that our approach also computes the quantities $\bar{\boldsymbol{\Sigma}}$ and $\bar{\eta}_{2u},\ \forall u$. However, the expressions of these approaches can be computed in time linear to the number of hidden units, as they do not entail any tedious calculations. Similar is the case when it comes to classification tasks. As such, one can expect that our method and conventional RNNs should share same computational complexity. To conclude, to provide some empirical evidence towards this direction, we here report the total training time of our method and conventional RNNs in the case of the acoustic novelty detection task (similar results can be obtained for the rest of our experimental scenarios). In our implementation, conventional RNNs took 6,855 sec to train, while our approach took 7,169 sec, that is a mere 4.58% extra computational time. Prediction generation took identical time in both cases. As such, we deduce that our approach offers a favorable performance/complexity trade-off over existing RNN formulations.

5 Conclusions and Future Work

In this paper, we proposed a sparse Bayesian formulation of RNNs, based on the introduction of a sparsity-inducing hierarchical prior over the output connection weights of the model. As we discussed, this model formulation introduces the ARD mechanism into the inferential procedures of RNNs, which allows for data-driven determination of the *effective* number of hidden (recurrently connected) units. We provided two alternative formulations of our model: one with likelihood function properly selected for handling regression tasks, and one designed for handling classification tasks. We devised simple and efficient inference algorithms for our model, scalable to massive datasets, for both the regression and (multi-class) classification settings. For this purpose, we resorted to the SVI paradigm.

To empirically evaluate the efficacy of our approach and how it compares to the competition, we conducted a number of experimental investigations dealing with human motion modeling using MoCap data and novelty detection in acoustic signals. In all cases, we used benchmark datasets in our experiments, and compared the performance of our method to state-of-the-art methods in the corresponding domains. As we observed, our approach yields a clear modeling performance advantage over the competition, without inducing notable overheads in terms of computational complexity.

In this work, posterior inference was conducted only for the output connection weights of the postulated RNNs, and the associated precision hyperparameters. In contrast, for the input and recurrent connection weight matrices of the model we obtained point-estimates, by maximization of the ELBO of the model over them. As such, one direction for future research concerns obtaining a fully Bayesian treatment of RNNs, with appropriate priors imposed over all

the weight matrices of the model, and associated posterior distributions obtained during model inference. A challenge we expect to encounter working towards this direction concerns the nonlinear nature of the activation functions of the hidden units $\phi_h(\cdot)$, which may prevent us from obtaining closed-form expressions of the associated posteriors. Employing the black-box variational inference framework proposed in [21] to train our model might be a suitable possible candidate solution towards the amelioration of these issues.

Appendix

We have

$$\langle \boldsymbol{\alpha} \rangle = \left[\frac{\bar{\eta}_{1u}}{\bar{\eta}_{2u}} \right]_{u=1}^{H} \tag{25}$$

and

$$\left\langle v_{nu}^2 \right\rangle = \left[\left\langle \boldsymbol{v}_n \boldsymbol{v}_n^T \right\rangle \right]_u \tag{26}$$

where $[\cdot]_u$ stands for the uth element of a vector, and it holds

$$\left\langle \boldsymbol{v}_n \boldsymbol{v}_n^T \right\rangle = \begin{cases} \bar{\boldsymbol{v}}_n \bar{\boldsymbol{v}}_n^T + \bar{\boldsymbol{\Sigma}}, & \text{for regression tasks} \\ \bar{\boldsymbol{v}}_n \bar{\boldsymbol{v}}_n^T + \bar{\boldsymbol{\Sigma}}_n, & \text{for classification tasks} \end{cases} \tag{27}$$

Finally, the expression of $\langle \log p(\boldsymbol{v}_n | \boldsymbol{\alpha}) \rangle$ yields (ignoring constant terms)

$$\langle \log p(\boldsymbol{v}_n | \boldsymbol{\alpha}) \rangle = -\frac{1}{2} \sum_{u=1}^{H} \left\langle v_{nu}^2 \right\rangle \langle \alpha_u \rangle + \frac{1}{2} \sum_{u=1}^{H} \langle \log \alpha_u \rangle \tag{28}$$

where

$$\langle \log \alpha_u \rangle = \psi(\bar{\eta}_{1u}) - \log \bar{\eta}_{2u} \tag{29}$$

and $\psi(\cdot)$ is the Digamma function.

Acknowledgments. We gratefully acknowledge the support of NVIDIA Corporation with the donation of one Tesla K40 GPU used for this research.

References

1. The CMU MoCap database. http://mocap.cs.cmu.edu/
2. Amari, S.: Natural gradient works efficiently in learning. Neural Computation **10**(2), 251–276 (1998)
3. Barker, J., Vincent, E., Ma, N., Christensen, H., Green, P.: The Pascal Chime speech separation and recognition challenge. Computer Speech & Language **27**(3), 621–633 (2013)
4. Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I.J., Bergeron, A., Bouchard, N., Bengio, Y.: Theano: new features and speed improvements. In: Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop (2012)

5. Bayer, J., Osendorfer, C., Korhammer, D., Chen, N., Urban, S., van der Smagt, P.: On fast dropout and its applicability to recurrent networks. In: Proc. ICLR (2014)
6. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks **52**(2), 157–166 (1994)
7. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: High-dimensional sequence transduction. In: Proc. ICASSP, pp. 3178–3182 (2013)
8. Chatzis, S., Demiris, Y.: Echo state Gaussian process. IEEE Transactions on Neural Networks **22**(9), 1435–1445 (2011)
9. Cotter, A., Shamir, O., Srebro, N., Sridharan, K.: Better mini-batch algorithms via accelerated gradient methods. In: Proc. NIPS (2011)
10. Fokoue, E.: Stochastic determination of the intrinsic structure in Bayesian factor analysis. Tech. Rep. TR-2004-17, Statistical and Applied Mathematical Sciences Institute (2004)
11. Graves, A., Mohamed, A., Hinton, G.: Speech recognition with deep recurrent neural networks. In: Proc. ICASSP (2013)
12. Hammer, B.: On the approximation capability of recurrent neural networks. Neurocomputing **31**(1), 107–123 (2000)
13. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
14. Hoffman, M., Blei, D.M., Wang, C., Paisley, J.: Stochastic variational inference. Journal of Machine Learning Research **14**(5), 1303–1347 (2013)
15. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks. Tech. Rep. 148, German National Research Center for Information Technology, Bremen (2001)
16. Lan, G.: An optimal method for stochastic composite optimization. Mathematical Programming, 1–33 (2010)
17. Martens, J., Sutskever, I.: Learning recurrent neural networks with hessian-free optimization. In: Proc. ICML (2011)
18. Nesterov, Y.: A method of solving a convex programming problem with convergence rate $o(1/sqr(k))$. Soviet Mathematics Doklady **27**, 372–376 (1983)
19. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: Proc. ICML (2013)
20. Polyak, B.: Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics **4**(5), 1–17 (1964)
21. Ranganath, R., Gerrish, S., Blei, D.M.: Black box variational inference. In: Proc. AISTATS (2014)
22. Rumelhart, D., Hinton, G., Williams, R.: Learning internal representations by error propagation. In: Parallel Dist. Proc., pp. 318–362. MIT Press (1986)
23. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. J. Machine Learning Research **15**(6), 1929–1958 (2014)
24. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proc. ICML (2013)
25. Wang, J.M., Fleet, D.J., Hertzmann, A.: Gaussian process dynamical models for human motion. IEEE Transactions on Pattern Analysis and Machine Intelligence **30**(2), 283–298 (2008)