Marginalized Neural Networks Mixtures for Large-Scale Regression

Miguel Lázaro-Gredilla and Aníbal R. Figueiras-Vidal, *Senior Member, IEEE*

Abstract

For regression tasks, traditional Neural Networks (NNs) have been superseded by Gaussian Processes, which provide probabilistic predictions (input-dependent error bars), improved accuracy, and virtually no overfitting. Due to their high computational cost, in scenarios with massive data sets, one has to resort to sparse Gaussian Processes, which strive to achieve similar performance with much smaller computational effort. In this context, we introduce a mixture of NNs with marginalized output weights that can both provide probabilistic predictions and improve on the performance of sparse Gaussian Processes, at the same computational cost. The effectiveness of this approach is shown experimentally on some representative large data sets.

Index Terms

Bayesian models, Gaussian processes, large data sets, multilayer perceptrons, regression

I. Introduction

In recent years there has been a growing interest in Gaussian Processes (GPs) as an alternative to Neural Networks (NNs) for regression tasks. GPs offer important advantages:

1) They provide full posterior probability density estimations. The posterior variance can be used to assess each prediction's uncertainty. Unlike traditional NNs, this predictive variance is not constant and depends on the test case being considered.
2) They use the "kernel trick", which reduces the modeling task to the selection of a moderate number of hyperparameters, which in turn reduces the risk of overfitting.

Authors are with the Department of Sig. Proc. and Comm., Univ. Carlos III de Madrid, Madrid, Spain. Contact e-mail: miguel@tsc.uc3m.es. This work has been partly supported by the Spanish government under grant TEC2008-02473/TEC, and by the Madrid Community under grant S-505/TIC/0223.

3) For some kernels, GPs correspond to predictors with NN structure in which all the weights have been integrated out (see Section II-A); therefore, all possible weight values are taken into account, in contrast to a single, best performing set of parameters, which is the standard, maximum likelihood form of work with NNs.

But all these advantages come at a cost: As we will see in Section II, GP model selection takes $\mathcal{O}(n^3)$ time (where n is the number of training samples). Probabilistic predictions can be made in $\mathcal{O}(n^2)$ time each, after some precomputations which take $\mathcal{O}(n^3)$ time. This cubic and quadratic scaling renders GPs impractical for real world applications with massive data sets, which are increasingly frequent.

To alleviate this problem, several approximations to GPs have been proposed. The current state-of-the-art approximation, denominated "Sparse Gaussian Processes using Pseudoinputs", was introduced in [10]. Later, it was renamed to Fully Independent Training Conditional (FITC) model in the unifying framework of [7]. This approximation achieves near full-GP performance at a much reduced cost: Computing time is linear in n for learning and constant (independent of n) for making probabilistic predictions. We will review this approach in Section II-B.

In this work, instead of simplifying a full GP, we try to achieve the advantages of GPs the other way around, applying Bayesian techniques to finite NNs: We marginalize out some of the parameters of an NN (the output weights) and then combine the predictions of several NNs. The resulting algorithm, Marginalized NN Mixture (MNNmix) has roughly the same computational cost as FITC. The idea of using priors on NN weights has been around for long, but most approaches do not marginalize them out analytically (e.g., [6]).

The rest of the paper is organized as follows: In Section II we will be reviewing the relevant portion of GPs for regression and their relation to infinite NNs and FITC. Marginalized NNs are introduced in Section III, and Section IV explains how to combine them. In Section V we will test their performance on some large data sets, comparing MNNmix results with those provided by FITC and a full GP. Finally we will provide a concluding discussion in Section VI.

II. Review of Gaussian Process Regression

Assume we are given a set of independent and identically distributed (i.i.d.) samples $\mathcal{D} \equiv \{\mathbf{x}_j, y_j | j = 1, \dots n\}$, where each D-dimensional input $\mathbf{x}_j$ is associated with a scalar output y_j.

The regression task goal is, given a new input $\mathbf{x}_*$, to predict the corresponding output y_* based on $\mathcal{D}$.

The GP regression model assumes that the outputs can be modeled as some noiseless latent function of the inputs plus an independent noise $y = f(\mathbf{x}) + \varepsilon$, and then sets a zero mean[1] GP prior on $f(\mathbf{x})$ and a Gaussian prior on ε_j:

$$f(\mathbf{x}) \sim \mathcal{GP}(\mathbf{0}, k_f(\mathbf{x}, \mathbf{x}')), \qquad \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

where $k_f(\mathbf{x}, \mathbf{x}')$ is the covariance function and σ^2 is a hyperparameter that specifies the noise power. The joint distribution of observed outputs $\mathbf{y}$ and some unknown output y_* is therefore also Gaussian, see [8, p. 16]:

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K_{ff}} + \sigma^2\mathbf{I}_n & \mathbf{k_{f*}} \\ \mathbf{k_{f*}}^\top & k_{**} + \sigma^2 \end{bmatrix}\right) \tag{1}$$

where $\mathbf{K_{ff}}$, $\mathbf{k_{f*}}$ and k_{**} are a matrix with elements $k_f(\mathbf{x}_i, \mathbf{x}_j)$, a vector with elements $k_f(\mathbf{x}_i, \mathbf{x}_*)$, and $k_f(\mathbf{x}_*, \mathbf{x}_*)$, respectively. $\mathbf{I}_n$ is used to denote the identity matrix of size n. From (1) and conditioning on the observed training outputs we obtain the desired predictive distribution

$$p_{\text{GP}}(y_*|\mathbf{x}_*, \mathcal{D}) = \mathcal{N}(y_*|\mu_{\text{GP}*}, \sigma^2_{\text{GP}*}) \tag{2a}$$

$$\mu_{\text{GP}*} = \mathbf{k_{f*}}^\top(\mathbf{K_{ff}} + \sigma^2\mathbf{I}_n)^{-1}\mathbf{y} \tag{2b}$$

$$\sigma^2_{\text{GP}*} = \sigma^2 + k_{**} - \mathbf{k_{f*}}^\top(\mathbf{K_{ff}} + \sigma^2\mathbf{I}_n)^{-1}\mathbf{k_{f*}} \tag{2c}$$

which is computable in $\mathcal{O}(n^3)$ time (this cost arises from the inversion of the $n \times n$ matrix $\mathbf{K_{ff}} + \sigma^2\mathbf{I}_n$), see [8], Ch. 8.

The key aspect of GP regression is the "kernel trick", which forces a parametric autocovariance (kernel) $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}')$ for $k_f(\mathbf{x}, \mathbf{x}')$. The corresponding hyperparameters $\{\boldsymbol{\theta}, \sigma\}$ are typically selected by Type-II Maximum Likelihood, using the marginal likelihood (also called evidence) of the observations:

$$\begin{aligned} \log p_{\text{GP}}(\mathbf{y}|\boldsymbol{\theta}, \sigma) &= -\frac{n}{2}\log(2\pi) - \frac{1}{2}|\mathbf{K_{ff}} + \sigma^2\mathbf{I}| \\ &\quad -\frac{1}{2}\mathbf{y}^\top\left(\mathbf{K_{ff}} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{y} \end{aligned} \tag{3}$$

[1]It is customary to subtract the sample mean from data $\{y_j\}_{j=1}^n$, which allows to assume a zero mean model.

Applicable search algorithms are described in standard textbooks such as [1], [8]. In case the analytical derivatives of (3) are available and conjugate gradient ascent is used for optimization, each step takes $\mathcal{O}(n^3)$ time.

A. Gaussian Processes as the infinite limit of NNs

NNs are a biologically inspired computational model that consists on a set of interconnected processing units or "neurons". Of the many existing types, we will consider feedforward NNs with one hidden layer and sigmoid activation function. It has been proved (see [4]) that this architecture is a universal approximator on a compact subset of $\mathbb{R}^D$. Their input-to-output mapping can be expressed, for N_H hidden neurons, as

$$f(\mathbf{x}) \;=\; \sum_{r=1}^{N_H} w_r \phi(\mathbf{u}_r; \mathbf{x}) \tag{4}$$

where $\{w_r\}_{r=1}^{N_H}$ are the output weights and $\phi(\mathbf{u}; \mathbf{x})$ is a sigmoid activation function, which in turn depends on input weights $\mathbf{u}$. It will be convenient to use $\phi(\mathbf{u}; \mathbf{x}) = \text{erf}(u_0 + \sum_{d=1}^{D} u_d x_d)$ for analytical tractability, where u_0 is regarded as the "bias" term of the neuron. We use the standard definition for the error function, which is $\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$.

Networks of this type are usually trained by selecting their weights $\{w, u_0, \mathbf{u}\}_r$ so as to minimize the squared error over the training set, i.e. $\sum_{j=1}^{n} (f(\mathbf{x}_j) - y_j)^2$. As we discussed before, this results in a number of problems, including overfitting for big N_H, convergence to poor local minima, having to resort to expensive cross-validation, etc. On the other hand, they are very cheap to train, taking only $O(N_H n)$ time for simple, gradient-based algorithms and $O(N_H^2 n)$ time for more sophisticated algorithms such as Levenberg-Marquardt.

A solution to the aforementioned problems is to follow a non-parametric Bayesian approach, which in this case involves using an infinite number of hidden neurons, placing a prior on all the weights of the network and integrating them out. Perhaps surprisingly, when independent, zero-mean Gaussian priors are used on all the network weights, the resulting model is analytically tractable. We set the prior variance to σ_w^2 / N_H for the output weights and to $\{\ell_d^{-2}\}_{d=0}^{D}$ for the input weights, where $d = 0$ denotes the bias. In this fashion, the first two moments of the output

are:

$$\begin{aligned}
\mathbb{E}[f(\mathbf{x})] &= 0 \\
\mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] &= \frac{1}{N_H}\sum_{r=1}^{N_H}\sigma_w^2\mathbb{E}[\phi(\mathbf{u}_r;\mathbf{x})\phi(\mathbf{u}_r;\mathbf{x}')] \\
&= \sigma_w^2\mathbb{E}[\phi(\mathbf{u};\mathbf{x})\phi(\mathbf{u};\mathbf{x}')] \\
&= \sigma_0^2\sin^{-1}\left(\frac{2\tilde{\mathbf{x}}^\top\Lambda^{-1}\tilde{\mathbf{x}}'}{\sqrt{1+2\tilde{\mathbf{x}}\Lambda^{-1}\tilde{\mathbf{x}}}\sqrt{1+2\tilde{\mathbf{x}}'\Lambda^{-1}\tilde{\mathbf{x}}'}}\right) \qquad (5)
\end{aligned}$$

where $\tilde{\mathbf{x}} = [1\ \mathbf{x}^\top]^\top$ is the augmented input vector, Λ is a diagonal matrix with elements$\{\ell_d^2\}_{d=0}^D$ and we have compacted $\sigma_0^2 = 2\sigma_w^2/\pi$. The first step follows trivially because all input weights $\mathbf{u}_r$ are i.i.d., but the second step is quite involved (see [11] for the complete proof).

Since the activation function is bounded, all the moments of $f(\mathbf{x})$ are bounded, and due to the Central Limit theorem, when number of hidden units N_H tends to infinity, $f(\mathbf{x})$ converges to a GP with the given mean and covariance function. The hyperparameters of this covariance function are GP power scaling σ_0^2 and lengthscales $\{\ell_d\}_{d=0}^D$. The lengthscales specify the scaling for each input dimension and bias. This covariance function is called Automatic Relevance Determination (ARD) NN because, when coupled with a model selection method, non-informative input dimensions can be removed automatically by growing the corresponding lengthscale.

Finally, it is possible to use this infinite NN to make predictions with finite computation by invoking eqs. (2b) and (2c), treating it as a GP and enjoying all its advantages (but raising computation time to $\mathcal{O}(n^3)$).

B. Sparse GPs: The FITC model

In order to reduce GPs' computation time from cubic to linear in the number of training samples, "Sparse GPs using Pseudo-inputs" are introduced in [10]. Although there have been previous attempts to achieve this goal, this method consistently outperforms them in most problems. In order to fit it into a systematic framework of sparse GP approximations, it was later renamed to FITC (see [7]), and we will maintain that name. In this section we will briefly describe it.

The key idea of FITC is to augment the existing training data set $\mathcal{D}$ with a noiseless pseudo-data set $\overline{\mathcal{D}} \equiv \{\overline{\mathbf{x}}_i, u_i\}_{i=1}^m$ and to assume that all output variables are conditionally independent

given the pseudo-data set. The pseudo-outputs can be integrated out, and the pseudo-inputs can be learned by maximizing the evidence of the model.

Conditioning on the pseudo data set, the distribution of a single latent function value f_j can be obtained:

$$p(f_j|\mathbf{x}_j, \overline{\mathcal{D}}) = \mathcal{N}(f_j|\mathbf{K}_{f_j\mathbf{u}}\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{u}, \mathbf{K}_{f_jf_j} - \mathbf{Q}_{f_jf_j})$$

where $\mathbf{Q}_{\mathbf{ab}} \equiv \mathbf{K}_{\mathbf{au}}\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{K}_{\mathbf{ub}}$ and $\mathbf{u} = [u_1,\ u_2, \ldots,\ u_m]^\top$. The FITC approximation arises when the joint conditional distribution of the latent values is replaced with the product of the marginal conditional distributions: $p(\mathbf{f}|\{\mathbf{x}_j\}_{j=1}^n, \overline{\mathcal{D}}) \approx \prod_{j=1}^n p(f_j|\mathbf{x}_j, \overline{\mathcal{D}})$.

The original prior over the latent variables $p(\mathbf{f}|\{\mathbf{x}_j\}_{j=1}^n) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K}_{\mathbf{ff}})$ is thus approximated by:

$$\begin{aligned} p(\mathbf{f}|\{\mathbf{x}_j\}_{j=1}^n, \{\overline{\mathbf{x}}_i\}_{i=1}^m) &= \int p(\mathbf{f}|\{\mathbf{x}_j\}_{j=1}^n, \overline{\mathcal{D}})p(\mathbf{u})d\mathbf{u} \\ &\approx \int \prod_{j=1}^n p(f_j|\mathbf{x}_j, \{\overline{\mathbf{x}}_i\}_{i=1}^m, \mathbf{u})p(\mathbf{u})d\mathbf{u} \\ &= \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{Q}_{\mathbf{ff}} + \mathrm{diag}(\mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}})) \end{aligned}$$

where $\mathrm{diag}(\cdot)$ sets all off-diagonal elements to zero.

This prior change amounts to replacing the original covariance matrix $\mathbf{K}_{\mathbf{ff}}$ with a low-rank matrix $\mathbf{Q}_{\mathbf{ff}}$ plus a diagonal matrix $\mathrm{diag}(\mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}})$. With this approximation, eqs. (2b), (2c) and (3) can be computed in $\mathcal{O}(m^2n)$ time, using the matrix inversion lemma. The pseudo-inputs and hyperparameters are found by maximizing the evidence.

III. Marginalized Neural Networks

Instead of integrating out all the parameters of an NN and then sparsify the resulting model, we propose to enforce sparsity by design, integrating out only the output weights. We will call this model Marginalized Neural Network (MNN).

An NN with mapping (4) can be expressed as $f(\mathbf{x}) = \mathbf{w}^\top\boldsymbol{\phi}(\mathbf{x})$, where $\mathbf{w} = [w_1,\ w_2,\ \ldots,\ w_m]^\top$ and $\boldsymbol{\phi}(\mathbf{x}) = [\phi(\mathbf{u}_1;\mathbf{x}),\ \phi(\mathbf{u}_2;\mathbf{x}),\ \ldots,\ \phi(\mathbf{u}_m;\mathbf{x})]^\top$ are the outputs of the hidden layer. In Section II-A the activation function was set to $\mathrm{erf}(\cdot)$ to allow for analytical tractability, but this is not needed for MNN, hence any activation function is valid. Throughout this work we will use the same activation function for both models for comparison purposes.

Placing independent zero-mean Gaussian priors of variance σ_0^2/N_H over the output weights, we have that $f(\mathbf{x})$ is a zero-mean GP, since it is a linear combination of zero-mean Gaussian random variables. The covariance function of this MNN GP is $k_{\text{MNN}}(\mathbf{x}, \mathbf{x}') = \frac{\sigma_0^2}{N_H}\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}')$.

"Design matrix" $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\mathbf{x}_1),\ \boldsymbol{\phi}(\mathbf{x}_2), \ldots,\ \boldsymbol{\phi}(\mathbf{x}_n)]$ of the training set can be used to express the joint distribution of the latent function compactly:

$$p_{\text{MNN}}(\mathbf{f}|\{\mathbf{x}_j\}_{j=1}^n) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \sigma_0^2/N_H \boldsymbol{\Phi}^\top \boldsymbol{\Phi}).$$

Again, we can use all the prediction and model selection equations from the GP framework by simply replacing $\mathbf{K_{ff}}$ with a low-rank matrix $\frac{\sigma_0^2}{N_H}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$. Model selection is then performed by jointly optimizing eq. (3) over the input weights and the signal and noise hyperparameters (σ_0^2 and σ^2) to find suitable values. This optimization must be performed over $(D+1)N_H + 2$ real values. This is roughly the same number of parameters used by FITC when the number of pseudo-inputs m coincides with the number of hidden neurons N_H.

Since the covariance matrix of this GP is low rank by construction, previous GP equations can be re-stated in a computationally cheaper form. Expanding (2) in terms of $\boldsymbol{\Phi}$ and $\boldsymbol{\phi}(\mathbf{x}_*)$ and using the matrix inversion lemma we have[2]:

$$p_{\text{MNN}}(y_*|\mathbf{x}_*, \mathcal{D}) = \mathcal{N}(y_*|\mu_{\text{MNN}^*}, \sigma^2_{\text{MNN}^*}) \tag{6a}$$

$$\mu_{\text{MNN}^*} = \boldsymbol{\phi}(\mathbf{x}_*)^\top \mathbf{A}^{-1}\boldsymbol{\Phi}\mathbf{y} \tag{6b}$$

$$\sigma^2_{\text{MNN}^*} = \sigma^2 + \sigma^2\boldsymbol{\phi}(\mathbf{x}_*)^\top \mathbf{A}^{-1}\boldsymbol{\phi}(\mathbf{x}_*) \tag{6c}$$

where $\mathbf{A} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \frac{N_H\sigma^2}{\sigma_0^2}\mathbf{I}_{N_H}$ is an $N_H \times N_H$ matrix, much smaller than the covariance matrix.

It is interesting to notice that the predictive mean can be expressed as the output of a traditional NN with the selected input weights and the output weights set to $\mathbf{w} = \mathbf{A}^{-1}\boldsymbol{\Phi}\mathbf{y}$. Unlike FITC, MNNs preserve the structure of traditional NNs.

In order to select the free parameters, the following computationally-efficient form of the evidence can be used:

$$\begin{aligned}\log p_{\text{MNN}}(\mathbf{y}|\{\mathbf{x}_j\}_{j=1}^n, \mathcal{M}) = &-\frac{1}{2\sigma^2}\left[\mathbf{y}^\top\mathbf{y} - \mathbf{y}^\top\boldsymbol{\Phi}_\mathbf{f}^\top\mathbf{A}^{-1}\boldsymbol{\Phi}_\mathbf{f}\mathbf{y}\right] \\ &- \frac{1}{2}\log|\mathbf{A}| + \frac{N_H}{2}\log\frac{N_H\sigma^2}{\sigma_0^2} - \frac{n}{2}\log 2\pi\sigma^2,\end{aligned} \tag{7}$$

[2]An analogous, alternative derivation can be found in [8, p. 8-11].

where $\mathcal{M} \equiv \{\sigma_0^2, \sigma^2, \mathbf{u}_1, \dots \mathbf{u}_{N_H}\}$ collects the signal and noise power hyperparameters, and the input weights, i.e., the free parameters of the model.

Equations (6b), (6c), and (7) can be computed in $\mathcal{O}(N_H^2 n)$, the same bound offered by FITC. Moreover, when $m = N_H$, MNNs are slightly faster because their low rank covariance matrix is simpler to compute. Further computational savings and numerical accuracy can be achieved by using the Cholesky factorization to code these equations.

We have thus achieved most of full GPs benefits while maintaining a computationally tractable model for large data sets.

IV. Combining Marginalized Neural Networks

Selecting $\mathcal{M}$ through evidence maximization poses some risk of overfitting. Fortunately, it is possible to reduce this risk by combining several MNNs in a way that is equivalent to approximately integrating out[3] $\mathcal{M}$.

First, consider the full Bayesian approach in which $\mathcal{M}$ is integrated out analytically. The predictive distribution of y_* at a new test point $\mathbf{x}_*$ is

$$p(y_*|\mathbf{x}_*, \mathcal{D}) = \int p_{\text{MNN}}(y_*|\mathbf{x}_*, \mathcal{M}, \mathcal{D}) p(\mathcal{M}|\mathcal{D}) \mathrm{d}\mathcal{M}. \tag{8}$$

Predictions (8) avoid overfitting, since all parameters are integrated out. However, since this integral is analytically intractable, we have to resort to numerical methods to compute it. Using Monte Carlo integration we have

$$p(y_*|\mathbf{x}_*, \mathcal{D}) \approx p_{\text{Mix}}(y_*|\mathbf{x}_*, \mathcal{D}) = \frac{1}{K} \sum_{k=1}^{K} p_{\text{MNN}}(y_*|\mathbf{x}_*, \mathcal{M}_k, \mathcal{D}), \tag{9}$$

where $\{\mathcal{M}_k\}_{k=1}^K$ are samples drawn from $p(\mathcal{M}|\mathcal{D})$. The Monte Carlo approximation converges to the exact value in the infinite limit. Therefore, for $K = \infty$, (9) would coincide with the exact Bayesian posterior (8) and no overfitting would be present. When only a few samples are used (small K), the approximation is more coarse, but nonetheless shows some overfitting resistance.

The set $\{\mathcal{M}_k\}_{k=1}^K$ must be sampled from the posterior probability $p(\mathcal{M}|\mathcal{D})$. This is not a trivial distribution and drawing samples from it usually requires to resort to expensive Markov Chain Monte Carlo methods (MCMC). Since we only require a few samples (as little as 4 in

[3] $\mathcal{M}$ can be integrated out with arbitrary accuracy by resorting to (expensive) methods such as hybrid Monte Carlo, see [5].

our experiments), it is possible to use a much simpler strategy, such as choosing the samples to be the local modes of $p(\mathcal{M}|\mathcal{D})$, i.e., high-probability samples according to our exploration of the distribution. Since the modes of $p(\mathcal{M}|\mathcal{D})$ correspond to the modes of the log-likelihood $\log p_{\text{MNN}}(\mathbf{y}|\{\mathbf{x}_j\}_{j=1}^n, \mathcal{M})$ for a flat prior over $\mathcal{M}$, it is possible to obtain $\{\mathcal{M}_k\}_{k=1}^K$ as a set of local maximum likelihood estimates, which is the approach followed here. With these considerations, $\{p_{\text{MNN}}(y_*|\mathbf{x}_*, \mathcal{M}_k, \mathcal{D})\}_{k=1}^K$ correspond to the posterior distributions of K independently trained MNNs, which we assume to have converged to different modes of the likelihood. The mean and variance of the exact Bayesian posterior (8) can be approximated using (9), yielding

$$\mu_{\text{MNNmix}*} = \frac{1}{K}\sum_{k=1}^{K} \mu_{\text{MNN}*k} \tag{10a}$$

$$\sigma^2_{\text{MNNmix}*} = \frac{1}{K}\sum_{k=1}^{K} \mu^2_{\text{MNN}*k} + \sigma^2_{\text{MNN}*k} - \mu^2_{\text{MNNmix}*} \tag{10b}$$

where $\mu_{\text{MNN}*k}$ and $\sigma^2_{\text{MNN}*k}$ are the predictive mean and variance provided by the k-th MNN, respectively.

Though using a different motivation, our proposal is akin to bagging [2] (known to reduce overfitting), but it introduces diversity by combining several local modes of the evidence, instead of different subsamples of training data. Equation (10) is also presented in the context of GP bagging in [3].

We will now provide an alternative derivation which yields the same predictive mean and variance given by (10), but corresponds to a proper (posterior) GP. The posterior over the output weights for the k-th MNN is

$$p(\mathbf{w}_k|\mathcal{D}) = \mathcal{N}(\mathbf{w}_k|\mathbf{b}_k, \mathbf{\Sigma}_k)$$

$$\mathbf{b}_k = \mathbf{A}_k^{-1}\mathbf{\Phi}_k\mathbf{y}$$

$$\mathbf{\Sigma}_k = \sigma_k^2\mathbf{A}_k^{-1}$$

where $\mathbf{A}_k$, $\mathbf{\Phi}_k$, and σ_k^2 correspond to the specific parametrization found for the k-th MNN after training. Output weights from different MNNs are obviously independent given data.

We can introduce dependencies among the MNNs by combining the independent posteriors

in a single joint posterior as follows:

$$\begin{aligned} p(\tilde{\mathbf{w}}|\mathcal{D}) \equiv & \mathcal{N}(\tilde{\mathbf{w}}|\mathbf{b}, \boldsymbol{\Sigma}) \\ \mathbf{b} = & [\mathbf{b}_1^\top,\ \mathbf{b}_2^\top, \ldots,\ \mathbf{b}_K^\top]^\top \\ \boldsymbol{\Sigma} = & \text{bd}(\sigma_1^2 \mathbf{A}_1^{-1} + \mathbf{b}_1 \mathbf{b}_1^\top, \ldots, \\ & \sigma_K^2 \mathbf{A}_K^{-1} + \mathbf{b}_K \mathbf{b}_K^\top) \cdot K - \mathbf{b}\mathbf{b}^\top \end{aligned}$$

where $\tilde{\mathbf{w}} = [\mathbf{w}_1^\top,\ \mathbf{w}_2^\top, \ldots,\ \mathbf{w}_K^\top]^\top$, and $\text{bd}(\cdot)$ arranges the matrices given as argument in block-diagonal form. This joint Gaussian posterior preserves the posterior means of the separate MNNs while introducing correlations among the weights belonging to different networks.

The output of the network mixture at any point $\mathbf{x}_*$ is defined as

$$y_{*\text{MNNmix}}(\mathbf{x}_*) = \frac{1}{K} \sum_{k=1}^{K} (\mathbf{w}_k^\top \boldsymbol{\phi}_k(\mathbf{x}_*) + \sqrt{K} \varepsilon_k(\mathbf{x}_*))$$

i.e., the average of the outputs of each MNN plus a white noise process. It is, therefore, a GP:

$$\begin{aligned} y_{*\text{MNNmix}}(\mathbf{x}_*) &\sim \mathcal{GP}(m_{\text{MNNmix}}(\mathbf{x}_*), k_{\text{MNNmix}}(\mathbf{x}_*, \mathbf{x}'_*)) \\ m_{\text{MNNmix}}(\mathbf{x}_*) &= \frac{1}{K} \mathbf{b}^\top \tilde{\boldsymbol{\phi}}(\mathbf{x}_*) \\ k_{\text{MNNmix}}(\mathbf{x}_*, \mathbf{x}'_*) &= \frac{1}{K^2} \left[\tilde{\boldsymbol{\phi}}(\mathbf{x}_*)^\top \boldsymbol{\Sigma} \tilde{\boldsymbol{\phi}}(\mathbf{x}'_*) + \delta_{\mathbf{x}_* \mathbf{x}'_*} K \sum_{k=1}^{K} \sigma_k^2 \right] \end{aligned}$$

where $\tilde{\boldsymbol{\phi}}(\cdot) = [\boldsymbol{\phi}_1(\cdot)^\top,\ \boldsymbol{\phi}_2(\cdot)^\top, \ldots,\ \boldsymbol{\phi}_K(\cdot)^\top]^\top$ is a column vector containing the hidden neurons outputs of all the networks. Note that the mean and variance of $y_{*\text{MNNmix}}(\mathbf{x}_*)$ at any point $\mathbf{x}_*$ are exactly those given by (10).

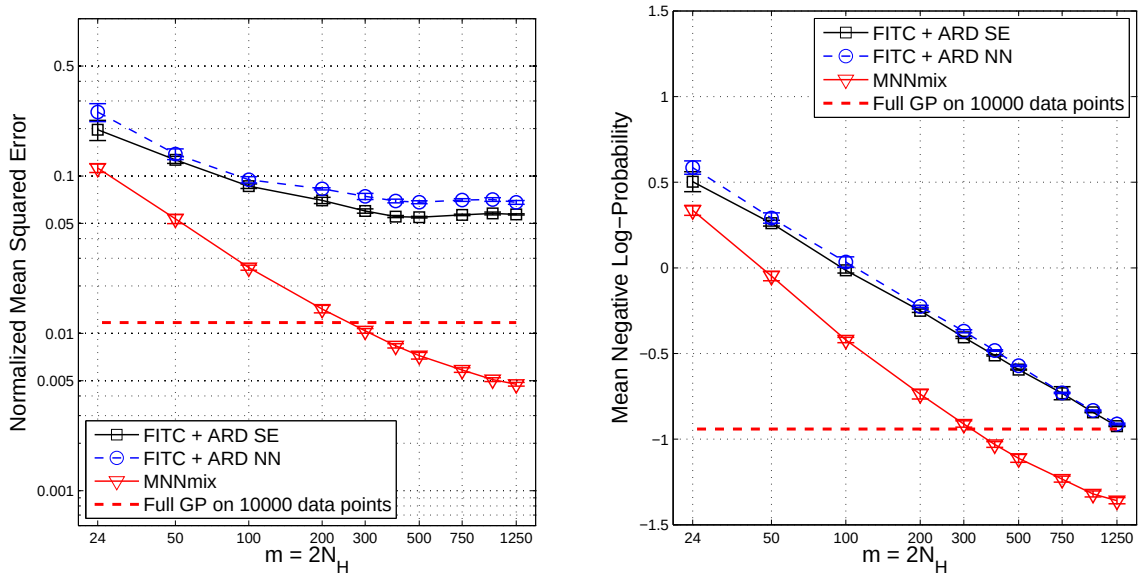
MNNmix preserves the traditional NN structure, using KN_H hidden neurons. Since predictions are obtained from a trivial combination of independent MNNs, the computational complexity of MNNmix is $\mathcal{O}(KN_H^2 n)$ time for training and $\mathcal{O}(KN_H^2)$ time for each probabilistic prediction.

V. Experiments

We will compare four regression methods: Full GP using the ARD SE covariance function[4], FITC both with ARD SE and ARD NN covariance functions, and MNNmix. The corresponding implementations have been made publicly available[5].

[4]ARD Squared Exponential: $k_{\text{SE}}(\mathbf{x}, \mathbf{x}') = \sigma_0^2 \exp\left[-\frac{1}{2}\mathbf{x}^\top \Lambda^{-1} \mathbf{x}\right]$

[5]Full GP: http://www.gaussianprocess.org/gpml/code, FITC+ARD SE: http://www.gatsby.ucl.ac.uk/~snelson/, FITC+ARD NN and MNNmix: http://www.tsc.uc3m.es/~miguel/.



(a) NMSE (log-log plot)

(b) MNLP (semilog plot)

Fig. 1. NMSE and MNLP for FITC using both SE and NN covariance functions, MNNmix and full GP, on the *Kin-40k* problem.

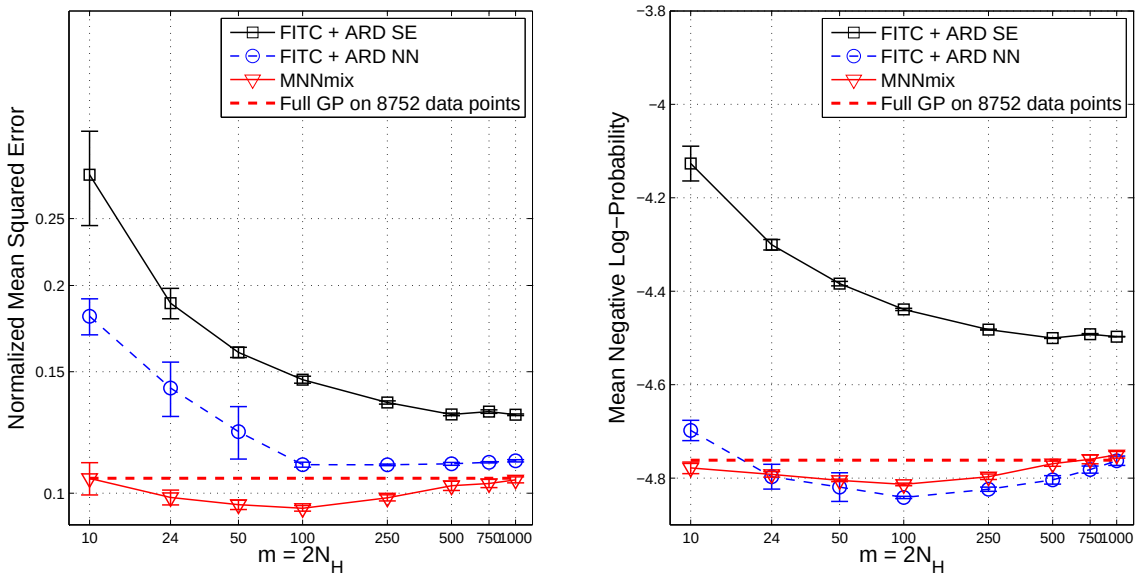
The full GP is expected to achieve state-of-the-art performance on the data sets, and its results are quoted only as a reference of desirable performance[6]. The huge amounts of computation time and memory necessary to obtain these results make full GPs impractical for big data sets.

In addition to the standard FITC implementation (using the ARD Squared Exponential covariance function), we have also implemented FITC with an ARD NN covariance function. This option had not been pursued yet, to the best of our knowledge, and it results very powerful.

By setting $K = 4$ and the number of pseudo-inputs in FITC to twice the number of hidden neurons in each network of MNNmix (i.e., $m = 2N_H$), the actual running time of both methods (regardless of the covariance function in use) is roughly the same, since their computational complexities are respectively $\mathcal{O}(m^2 n)$ and $\mathcal{O}(KN_H^2 n)$, with similar constant factors. The aim, therefore, is to improve on the performance figures of FITC.

Increasing K results into enhanced performance and higher computational cost (usually, this improvement is more notable in the predictive variance than the predictive mean).

[6] For *Kin-40k*, training a full GP can take around two days as opposed to the 10 minutes required by MNNmix with $N_H = 100$ (which achieves close to full GP performance).



(a) NMSE (log-log plot)

(b) MNLP (semilog plot)

Fig. 2. NMSE and MNLP for FITC using both squared exponential and NN covariance functions, MNNmix and full GP, on the *Elevators* problem.

A. Setup

We have evaluated the performance of the discussed regression methods on four large, highly non-linear, data sets. For the sake of brevity, we will only be presenting performance figures for two of them: *Kin-40k* (8 input dimensions, 10000 training samples, 30000 testing samples), and *Elevators* (17 input dimensions, 8752 training samples, 7847 testing samples). The other two considered data sets are *Pumadyn-32nm* (32 input dimensions, 7168 training samples, 1024 testing samples) and *Pole Telecomm* (26 input dimensions, 10000 training samples, 5000 testing samples). For these data sets, the obtained performance will only be discussed in the text. Data sets *Kin-40k* and *Pumadyn-32nm* have been used in previous works [9], [10] to assess the performance of sparse GPs. We have followed their preprocessing steps precisely and we have reproduced the experiments with FITC+ARD SE, obtaining the same results[7]. The task is to predict the dynamics of a robotic arm. *Elevators* and *Pole Telecomm* have been obtained from http://www.liaad.up.pt/~ltorgo/Regression/datasets.html, and are related to the control of

[7]Note that we quote NMSE and MNLP, whereas they only quote "mean squared error", which they define as one half the NMSE value.

the elevators of an F16 aircraft and a telecommunications problem, respectively.

Since FITC+ARD NN is not translation invariant, we have additionally centered the input data.

Hyperparameters are initialized as follows: $\sigma_0^2 = \frac{1}{n}\sum_{j=1}^{n} y_j^2$, $\sigma^2 = \sigma_0^2/4$, $\{\ell_d\}_{d=1}^{D}$ is set to half the range spanned by training data along each dimension, and $\ell_0^2 = \sum_{d=1}^{D} \ell_d^2$. For FITC, pseudo-inputs are initialized to a random subset of the training data, and for MNNmix input weights are initialized randomly from a zero-mean unit-variance Gaussian distribution. Their final values are selected by evidence maximization. Quoted performance figures are averaged over ten runs, with ± 1 standard deviation plotted around them. Results for the original splits are displayed. We have found no qualitative differences when resampling is introduced.

As quality measures, we will use the Normalized Mean Square Error (NMSE) and Mean Negative Log-Probability (MNLP) of each method's predictions on the test set:

$$\begin{aligned} \text{NMSE} &= \frac{\langle (y_{*l} - \mu_{*l})^2 \rangle}{\langle (y_{*l} - \langle y_j \rangle)^2 \rangle} \\ \text{MNLP} &= \frac{1}{2} \left\langle \left(\frac{y_{*l} - \mu_{*l}}{\sigma_{*l}} \right)^2 + \log \sigma_{*l}^2 + \log 2\pi \right\rangle \end{aligned}$$

where $\{y_j\}_{j=1}^{n}$ are the outputs (labels) of the training set, $\{y_l\}$ are the outputs (labels) of the test set, μ_{*l} and σ_{*l}^2 are the predictive mean and variance for the l-th test sample and $\langle \cdot \rangle$ averages over the corresponding set.

NMSE measures the accuracy of the predictive mean, ignoring the predictive variance. MNLP, on the other hand, takes into account the predictive variance, weighting the error by the predicted degree of certainty.

B. Results

Results for *Kin-40k* are reported in Fig. 1. FITC performs similarly for both covariance functions. For $m > 300$ there is almost no improvement in the NMSE, though the predictive variance improves steadily, as the MNLP plot reflects. MNNmix performs impressively well on this data set, largely outperforming FITC. To our surprise, even state-of-the-art, full-GP performance is beaten for $m > 300$.

For *Pole Telecomm* and considering the NMSE measure, MNNmix is overall superior (though the difference is less dramatic than in the previous case), and FITC+ARD NN performs better

than FITC+ARD SE, the difference being more significative for $m > 50$. For the MNLP measure, FITC+ARD NN is able to beat MNNmix slightly when $m > 50$.

Results for *Elevators* are shown in Fig. 2. Again, FITC+ARD NN performs better than FITC+ARD SE. MNNmix is the best-performing method, showing a large improvement over FITC regarding NMSE, and remaining similar to FITC+ARD NN with respect to MNLP.

For *Pumadyn-32nm*, only 4 out the 32 input dimensions are relevant to the regression task. It is therefore a good test of the ARD capabilities of the proposed algorithms. We follow [10] and use a full GP on a small subset of the training data (1024 data points) to obtain the initial length-scales for all methods. This allows the optimization procedure to converge to better, higher evidence optima. On this data set MNNmix performs best, closely followed by FITC+ARD NN. All compared methods do a good job selecting the relevant inputs, including MNNmix.

Overall, MNNmix seems the best of the compared methods on these four data sets, providing very accurate results, even with a small number of hidden neurons, and occasionally outperforming the full GP. As the MNLP figures show, the predictive variance is also estimated quite accurately.

In is also interesting to note in Fig. 2 how the problem is already well-modeled with $m = 10$, but using $m = 1000$ does not result in significant overfitting. This shows the effectiveness of marginalizing the output weights and combining (even a small amount of) networks.

VI. Discussion

In this work we have provided a practical model for real world, large-scale regression, which stands in between traditional NNs and full GPs, while trying to retain the advantages of both. Additionally, we have shown how existing sparse GP methods, such as FITC, can benefit from using the often-forgotten ARD NN covariance function.

In our approach we start from a simple NN and marginalize some, but not all, of their weights to get the key benefits of a GP without sacrificing performance. Since the resulting architecture is still that of a traditional NN, this approach can be seen as an alternative procedure to train an NN, with additional benefits, such as improved accuracy, overfitting resistance and providing predictive variances. Thanks to the computation of an *input-dependent*, specific uncertainty for each prediction, MNNmix can be used in applications, where "error bars" are needed, or even the main interest.

When put to test on standard large data sets, the proposed method turns out to be very effective, achieving close to full-GP performance with very few hidden neurons and often outperforming FITC, the current state-of-the-art, computationally-affordable approximation to GPs.

The encouraging results obtained in this work warrant further exploration of other types of MNN ensembles. We are also considering other applications of the proposed model, the most straightforward cases being classification and regression with non-Gaussian noise (including robust regression).

REFERENCES

[1] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press, 1995.

[2] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 26, pp. 123–140, 1996.

[3] T. Chen and J. Ren, "Bagging for Gaussian process regression," *Neurocomputing*, vol. 72, pp. 1605–1610, 2009.

[4] K. Hornik, "Some new results on neural network approximation," *Neural Networks*, vol. 6, pp. 1069–1072, 1993.

[5] R. M. Neal, "Bayesian training of backpropagation networks by the hybrid Monte Carlo method," University of Toronto, Tech. Rep. CRG-TR-92-1, 1992.

[6] S. J. Nowlan and G. E. Hinton, "Adaptive soft weight tying using Gaussian mixtures," in *Advances in Neural Information Processing Systems 4*. San Mateo, CA: Morgan Kauff., 1991, pp. 993–1000.

[7] J. Quiñonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[8] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge MA: MIT Press, 2006.

[9] M. Seeger, C. K. I. Williams, and N. D. Lawrence, "Fast forward selection to speed up sparse Gaussian process regression," in *Proc. of the 9th Int. Workshop on AI Stats*, Key West, FL, Jan. 2003, pp. 205–212.

[10] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," in *Advances in Neural Information Processing Systems 18*. Cambridge, MA: MIT Press, 2006, pp. 1259–1266.

[11] C. K. I. Williams, "Computing with infinite networks," in *Advances in Neural Information Processing Systems 9*. Cambridge, MA: MIT Press, 1997, pp. 1069–1072.