Sparse Spectral Sampling Gaussian Processes

Miguel Lázaro-Gredilla
Department of Signal Processing & Communications
Universidad Carlos III de Madrid, Spain
miguel@tsc.uc3m.es

Joaquin Quiñonero-Candela
Microsoft Research
CB3 9LB Cambridge, UK
joaquinc@microsoft.com

Aníbal Figueiras-Vidal
Department of Signal Processing & Communications
Universidad Carlos III de Madrid, Spain
arfv@tsc.uc3m.es

November 2007
Technical Report
MSR-TR-2007-152

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

Abstract

In this work we introduce Power Spectral Density Sampling (PSDS), a new method for building a reduced rank approximation to a stationary kernel matrix. We apply this method in the framework of Gaussian Processes (GP). The resulting model can be trained in an online fashion: As each training sample arrives, we can include it in the model using only $\mathrm{O}(r^2)$ time and storage, where r is the selected number of spectral samples used in the approximation. For n training points, the overall cost is $\mathrm{O}(nr^2)$ time and $\mathrm{O}(r^2)$ storage. The resulting GP is properly defined and model hyperparameters can be selected by evidence maximization. Though it is especially well-suited for low dimensional problems, it can reach and even outperform other (batch) state-of-the-art sparse GP methods on higher dimensional datasets if allowed to work in batch mode, learning the locations of the spectral samples. We check this possibility on some large datasets.

1 Introduction and previous work

There has been a lot of interest in the past decade in the application of GPs to machine learning problems. They show state of the art performance in regression and classification tasks, within a probabilistic framework. However, their applicability is limited to problems with up to a few thousand samples, since the kernel matrix must be stored ($\mathrm{O}(n^2)$ real values) and inverted ($\mathrm{O}(n^3)$ computation time) during training. They are also most often presented in a batch setting, where all training samples are available at training time (as opposed to the online setting [1], where samples are included in the model as they become available, and inference can be done at any time using the current model). For a complete treatment of GPs for machine learning see [2].

There have been several proposals to alleviate this problem, bringing computation time down to $\mathrm{O}(nr^2)$ [3, 4, 5, 6, 7], an overview of which is given in [8]. Most approaches are based on selecting a subset of the training data using some information criterion, and then projecting the contribution of the whole dataset on them. The current state-of-the-art method, the Sparse Pseudo-input GP (SPGP) [9] uses the concept of pseudo-inputs, which are not part of the training dataset, but are learned maximizing the log marginal likelihood. In this work we propose an online method based on approximately reconstructing the kernel using the inversion formula from the spectral domain, taking samples at different frequencies. Then, we learn these frequencies to further improve the model. Though our approach is different, it can learns pseudo-frequencies in a "dual" way to the SPGP pseudo-inputs learning.

The rest of this paper is organized as follows. Section 2 introduces PSDS for a general stationary kernel. Section 4 develops the formulation for the case of the squared exponential kernel and presents the corresponding predictive equations for GPs. Section 5 demonstrates the performance of the method in an experimental setting, comparing it with other sparse models, both on a toy

problem and larger datasets.

2 Power Spectral Density Sampling (PSDS)

A zero-mean Gaussian process prior on a possibly complex function $f(\mathbf{x})$ is wholly defined by its covariance function $k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[f(\mathbf{x})f^*(\mathbf{x}')]$ where * denotes complex conjugation, and the indexing inputs vectors $\mathbf{x}$ and $\mathbf{x}'$ are D-dimensional. Stationary covariance functions depend only on the difference between the inputs $\boldsymbol{\chi} = \mathbf{x} - \mathbf{x}'$. Bochner's theorem grants that $k(\boldsymbol{\chi})$ can be represented as the Fourier transform of a positive finite measure μ.

The spectral density § of μ is a normalized version of the spectrum $\tilde{S}(\mathbf{s})$ of the stationary kernel. Being Fourier duals, $k(\boldsymbol{\chi})$ and $\tilde{S}(\mathbf{s})$ related by

$$\tilde{S}(\mathbf{s}) = \int_{\mathbb{R}^D} e^{-2\pi i \mathbf{s}\cdot\boldsymbol{\chi}} k(\boldsymbol{\chi}) d\boldsymbol{\chi} \quad \text{and} \quad k(\boldsymbol{\chi}) = \int_{\mathbb{R}^D} e^{2\pi i \mathbf{s}\cdot\boldsymbol{\chi}} \tilde{S}(\mathbf{s}) d\mathbf{s} \,. \tag{1}$$

We can make explicit the contributions of $\mathbf{x}$ and $\mathbf{x}'$ and express them as factors inside the integral:

$$k(\mathbf{x}, \mathbf{x}') = k(\boldsymbol{\chi}) = \int_{\mathbb{R}^D} e^{2\pi i \mathbf{s}\cdot(\mathbf{x}-\mathbf{x}')} S(\mathbf{s}) d\mathbf{s} = \int_{\mathbb{R}^D} e^{2\pi i \mathbf{s}\cdot\mathbf{x}} \left(e^{2\pi i \mathbf{s}\cdot\mathbf{x}'}\right)^* S(\mathbf{s}) d\mathbf{s} \tag{2}$$

Spectral densities are often used to illustrate frequency properties of a given kernel. However, kernel evaluations are most often performed in the original domain. We aim at exploring the possibility of not evaluating the kernel $k(\mathbf{x}, \mathbf{x}')$ directly, but of using a computationally attractive approximation to expression (2) to evaluate it.

Stationary covariance functions have constant diagonals, or variances. Without loss of generality for the purpose of our approximation, we can choose for simplicity to impose $k(\mathbf{0}) = 1$, which implies that the spectrum is equal to the spectral density $\tilde{S}(\mathbf{s}) = S(\mathbf{s})$, and therefore has the properties of a multivariate probability denisity in $\mathbf{s}$. With this interpretation, we can choose to consider eq. (1) and (2) to be expectations and write

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}_S \left[e^{2\pi i \mathbf{s}\cdot\mathbf{x}} \left(e^{2\pi i \mathbf{s}\cdot\mathbf{x}'}\right)^* \right] \tag{3}$$

where $\mathbb{E}_S$ denotes expectation wrt $S(\mathbf{s})$. Thus we are left with the problem of evaluating the mean of a product of complex exponentials. The exact evaluation of this average would bring us back to the original expression of the kernel. Instead, we propose to approximate it by the sample mean.

If we draw a set of r spectral samples $\mathbf{s}_l$, $l \in \{1, \ldots, r\}$ in $\mathbb{R}^D$ from $S(\mathbf{s})$, at which we evaluate the product of the complex exponentials, we obtain r samples and can then approximately reconstruct the covariance function as a finite sum

$$k(\mathbf{x}, \mathbf{x}') \approx \hat{k}(\mathbf{x}, \mathbf{x}') = \frac{1}{r} \mathrm{Re} \left[\sum_{l=1}^{r} e^{2\pi i \mathbf{s}_l\cdot\mathbf{x}} \left(e^{2\pi i \mathbf{s}_l\cdot\mathbf{x}'}\right)^* \right] , \tag{4}$$

where $\mathrm{Re}\,[\cdot]$ denotes the real part of a complex number. The approximation becomes exact as $r \to \infty$, with the imaginary part of the summation tending to zero.

It is convenient to introduce matrix notation at this point. If we have n available samples $\mathbf{x}_j$, $j \in \{1, \ldots, n\}$, we can define the usual $n \times n$ kernel matrix $\boldsymbol{K}$ with elements $k_{jj'} = k(\mathbf{x}_j, \mathbf{x}_{j'})$, and a $n \times r$ sampling matrix $\hat{\boldsymbol{\Phi}}$ with elements $\phi_{jl} = e^{2\pi i \mathbf{s}_l \mathbf{x}_j}$. The previous equation can be re-stated as

$$\boldsymbol{K} \approx \hat{\boldsymbol{K}} = \frac{1}{r}\mathrm{Re}\left[\hat{\boldsymbol{\Phi}}\hat{\boldsymbol{\Phi}}^H\right] = \frac{1}{r}\left[\mathrm{Re}[\hat{\boldsymbol{\Phi}}]\ \mathrm{Im}[\hat{\boldsymbol{\Phi}}]\right]\left[\mathrm{Re}[\hat{\boldsymbol{\Phi}}]\ \mathrm{Im}[\hat{\boldsymbol{\Phi}}]\right]^T \tag{5}$$
$$\text{with} \quad \mathrm{Re}[\phi_{jl} = e^{2\pi i \mathbf{s}_l \mathbf{x}_j}] = \cos(2\pi \mathbf{s}_l \mathbf{x}_j)$$
$$\text{and} \quad \mathrm{Im}[\phi_{jl} = e^{2\pi i \mathbf{s}_l \mathbf{x}_j}] = \sin(2\pi \mathbf{s}_l \mathbf{x}_j)$$

where ^{H} denotes Hermitian conjugate. To avoid working with complex numbers, we have expanded the complex exponentials in cosines and in sines. Note that the approximate kernel matrix has at most rank $2r$, which should be chosen to be less than n.

What we have obtained is an explicit, finite dimensional approximation to the function that maps points from the input space to the transformed space implied by the kernel. We can then use linear regression in this transformed space, of dimension $2r$. The kernel matrix can be inverted in $\mathrm{O}(nr^2)$ time (the dominating cost of the training process), and the cost of computing the predictive mean and variance is respectively $\mathrm{O}(r)$ and $\mathrm{O}(r^2)$. There is no need for active set selection (unlike is the case for example for the three methods in [3]). Furthermore as we detail in [REF SECTION], an online scheme can be easily devised where computation of the log marginal likelihood and any of its derivatives wrt kernel parameters can be achieved in $\mathrm{O}(r^2)$ time and space (assuming the hyperparameters are known). We call the application of this representation in the GP framework PSDS-GP. Despite being an approximation, $\hat{k}$ constitutes a valid covariance function by construction. This means that the resulting GP is well-defined and does not need to resort to "augmentation" to heal its predictive variance [8].

2.1 How many inducing inputs?

Most sparse models, as PSDS-GP, can be seen as a feature projection of the input points into a transformed space of dimension lower than n where the problem can be solved linearly. Such is the case of the three methods proposed in [3], with the size of the active set defining the dimension of this space. For SPGP, proposed in [9], the dimension of the transformed space would be the number of pseudo-inputs. In the unifying framework of [8], for a given sparse approximation, the number of *inducing inputs* would determine the dimension of the transformed space.

This dimension is reasonably used as a basis for comparisons among different approximations, and it is directly related to the amount of computation required to train the corresponding sparse model. However, for our proposed model the

fair benchmark is unclear: according to the dimension, one could simply state that the output dimension of the proposed transformation is $2r$, or could argue that the real, latent dimension is r since there we part from r real numbers and then apply *sin* and *cos* functions to obtain the $2r$ dimensions. This would take us to the more intuitive conclusion that the fair comparison for our r spectral samples model is an r inducing inputs model. According to the computational cost, we could again naively see that it is equivalent to having $2r$ inducing inputs.

In the light of this ambiguity, wherever possible we will compare our model with r spectral samples against alternative sparse GPs that both use r and $2r$ inducing inputs. In the remaining cases we will give the alternative model the benefit of the doubt and $2r$ inducing inputs.

3 Choosing the sampling points

So far we have assumed the sampling points $\mathbf{s}_l$ as given. Indeed, any set of points distributed according to $S(\mathbf{s})$ is valid and will work, and different methods for sampling from a given distribution can be applied (for instance, MCMC). Particularly simple is the case of the squared exponential covariance function, which has a probability density with the same form.

For our implementation, we will use sequences with better space-filling properties, and which, therefore, are preferred for numerical integration: quasirandom sequences. Without exhaustively verifying the behavior of every proposed quasirandom sequence, we have selected for our implementation the well-known Hammersley sequence. For details on the Hammersley sequence and quasi-montecarlo we refer the reader to [10] and [11]. These sequences are uniformly distributed over interval $[0, \ldots, 1]^D$. In our case we need the same property for another distribution, $S(\mathbf{s})$. This can be accomplished finding a set of functions $h_1 = g_1(\mathbf{s}),\ h_2 = g_2(\mathbf{s}), \ldots, h_D = g_D(\mathbf{s})$, such that their Jacobian is $S(\mathbf{s})$. If this condition is met, then $\{g_d\}$ constitute a transformation that takes samples from density $S(\mathbf{s})$ to a uniform density, and inverting them (i.e., obtaining $\mathbf{s}$ as a function of $\{h_d\}$), yields a transformation from a uniform density quasirandom sequence to the desired $S(\mathbf{s})$ density sampling points.

For the case in which $S(\mathbf{s})$ can be expressed as a product of the type $S(\mathbf{s}) = S_1(s_1)S_2(s_2)\ldots S_D(s_D)$ (where s_d are the elements of $\mathbf{s}$), the probability *distribution* functions associated to each $S_d(d)$ are a set of functions that fulfill the above condition.

Deterministically selecting the sampling points in low dimensional problems fills the space so well that it is difficult to improve the accuracy by moving their locations. However, when working on higher dimensional problems, we can use that selection as a starting point, and then learn their positions so that they minimize the negative log marginal likelihood of the training set. A disadvantage of doing this (appart from having a minimization problem in rD dimension) is that the algorithm loses its online ability: Moving the sampling points implies a new transformation function, and therefore, recomputing the

nonlinear transformation of past points. The same happens when pseudo-inputs are learned. It should also be noted that moving the sampling points away from their initial positions may lead to another effective stationary covariance function, since the overall distribution may change.

4 Application of PSDS to the squared exponential covariance function

We will now develop above formulation for the most widely used stationary covariance function, the anisotropic Gaussian, also called ARD (Automatic Relevance Determination) squared exponential:

$$k(\mathbf{x}^{(j)}, \mathbf{x}^{(j')}) = \sigma_0^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D} \theta_d^2 ||x_d^{(j)} - x_d^{(j')}||^2\right).$$

In this expression, $x_d^{(j)}$ is the dth component of the jth sample. To obtain the corresponding spectrum $S_{\text{ARD}}(\mathbf{s})$, we first set $\sigma_0^2 = 1$ (so that $k(\mathbf{x}_j, \mathbf{x}_j) = 1$), and then conveniently[1] define

$$\overline{\mathbf{x}}^{(j)} = \begin{bmatrix} x_1^{(j)}\theta_1 & x_2^{(j)}\theta_2 & \dots & x_D^{(j)}\theta_D \end{bmatrix}^T; \quad \boldsymbol{\chi}' = \frac{\overline{\mathbf{x}}^{(j)} - \overline{\mathbf{x}}^{(j')}}{2\pi},$$

so that we obtain

$$k(\mathbf{x}^{(j)}, \mathbf{x}^{(j')}) = \exp\left(-\frac{1}{2}\frac{||\boldsymbol{\chi}'||^2}{\left(\frac{1}{2\pi}\right)^2}\right); \tag{6}$$

$$S_{\text{ARD}}(\mathbf{s}) = \mathbb{F}\left\{\exp\left(-\frac{1}{2}\frac{||\boldsymbol{\chi}'||^2}{\left(\frac{1}{2\pi}\right)^2}\right)\right\} = \frac{1}{(2\pi)^{\frac{D}{2}}}\exp\left(-\frac{1}{2}||\mathbf{s}||^2\right). \tag{7}$$

Using our definition of $\boldsymbol{\chi}'$, eq. (1), and adding signal variance σ_0^2, we get

$$k(\mathbf{x}^{(j)}, \mathbf{x}^{(j')}) = \sigma_0^2 \int_{\mathbb{R}^D} e^{i\mathbf{s}\cdot\overline{\mathbf{x}}^{(j)}} e^{-i\mathbf{s}\cdot\overline{\mathbf{x}}^{(j')}} S_{\text{ARD}}(\mathbf{s}) d\mathbf{s}; \quad \hat{\boldsymbol{K}} = \frac{\sigma_0^2}{r}\text{Re}\left[\hat{\boldsymbol{\Phi}}_{n\times r}\hat{\boldsymbol{\Phi}}_{n\times r}^H\right].$$

As noted before, we can expand the sampling matrix in sines and cosines, $\boldsymbol{\Phi}_E = \frac{\sigma_0}{\sqrt{r}}\left[\text{Re}[\hat{\boldsymbol{\Phi}}]\ \text{Im}[\hat{\boldsymbol{\Phi}}]\right]$, so that we can write $\hat{\boldsymbol{K}} = \boldsymbol{\Phi}_E\boldsymbol{\Phi}_E^T$, with the constant absorbed.

The predictive equations of a GP using this approximate covariance matrix are

$$\mathbb{E}[f^*] = \boldsymbol{\phi}_{E*}\boldsymbol{\Phi}_E^T\left[\boldsymbol{\Phi}_E\boldsymbol{\Phi}_E^T + \sigma_n^2 I\right]^{-1}\boldsymbol{y}$$

$$\mathbb{V}[f^*] = \boldsymbol{\phi}_{E*}\boldsymbol{\phi}_{E*}^T - \boldsymbol{\phi}_{E*}\boldsymbol{\Phi}_E^T\left[\boldsymbol{\Phi}_E\boldsymbol{\Phi}_E^T + \sigma_n^2 I\right]^{-1}\boldsymbol{\Phi}_E\boldsymbol{\phi}_{E*}^T$$

[1] This will eliminate the dependence of $S(\mathbf{s})$ on θ_d and will set its variance to unity.

where $\boldsymbol{\phi}_{E*}$ is the extended sampling vector corresponding to test point $\mathbf{x}_*$. Now we can rearrange the equations by making use of the matrix inversion lemma and with some algebra:

$$\mathbb{E}[f^*] = \boldsymbol{\phi}_{E*} \left(\boldsymbol{\Phi}_E^T \boldsymbol{\Phi}_E + \sigma_n^2 I\right)^{-1} \boldsymbol{\Phi}_E^T \boldsymbol{y} \tag{8}$$

$$\mathbb{V}[f^*] = \boldsymbol{\phi}_{E*} \left(\sigma_n^{-2} \boldsymbol{\Phi}_E^T \boldsymbol{\Phi}_E + I\right)^{-1} \boldsymbol{\phi}_{E*}^T \tag{9}$$

The dependence of these equations on training data is restricted to matrix $\boldsymbol{\Phi}_E^T \boldsymbol{\Phi}_E$ of size $2r \times 2r$ and vector $\boldsymbol{\Phi}_E^T \boldsymbol{y}$ of size $2r \times 1$. Computing them for the first test point takes $\mathrm{O}(nr^2)$. Evaluation at additional test points takes $\mathrm{O}(r)$ and $\mathrm{O}(r^2)$ for the mean and the variance, respectively.

4.1 Obtaining the sampling points and the hyperparameters

Sampling points $\mathbf{s}_l$ are selected as described in Section 3. The squared exponential covariance matrix is factorizable, so we can use as $\{g_d\}$ the inverse of the probability distribution $F(s_d)$ of each factor:

$$F(s_d) = \int_{-\infty}^{s_d} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right) dt = \frac{1 + \mathrm{erf}(s_d/\sqrt{2})}{2};$$
$$s_d = F^{-1}(h_d) = \sqrt{2}\mathrm{erf}^{-1}\left(2h_d - 1\right)$$

If h_d is the dth-component of a point $\mathbf{h}$ in a uniform quasirandom sequence in interval $[0, 1)$, then we can obtain the corresponding sampling point $\mathbf{s}$ applying the previous to all components. If we do this at all $\{\mathbf{h}_l\}$ points in the quasirandom sequence, we obtain the desired $\{\mathbf{s}_l\}$ sampling points, distributed according to a zero mean, unit variance Gaussian.

The estimation of the hyperparameters of a GP model is usually accomplished by minimizing the negative log marginal likelihood of the training set. In most minimizations schemes, derivatives with respect to these parameters are needed. Also, as noted in subsection 3, we can learn the sampling points, so derivatives wrt their locations are also needed.

There are a total of D+2 hyperparameters and rD sampling point components to optimize. The whole vector of derivatives can be computed in $\mathrm{O}(nr^2D)$.They can also be computed online, but now also matrix $\boldsymbol{\Phi}_E^T \frac{\partial \boldsymbol{\Phi}_E}{\partial \theta_i}$ of size $2r \times 2r$ and vector $\frac{\partial \boldsymbol{\Phi}_E}{\partial \theta_i} \boldsymbol{y}$ of size $2r \times 1$ must be stored and updated for each new sample, which can be done exactly as we did with $\boldsymbol{\Phi}_E^T \boldsymbol{\Phi}_E$ and $\boldsymbol{\Phi}_E^T \boldsymbol{y}$. The final expresions of these derivatives are lengthy and ommited here for brevity.

4.2 Online predictive equations

If we call $\boldsymbol{\phi}_E^{(j)}$ to the jth row of $\boldsymbol{\Phi}_E$, we can express the needed matrices as

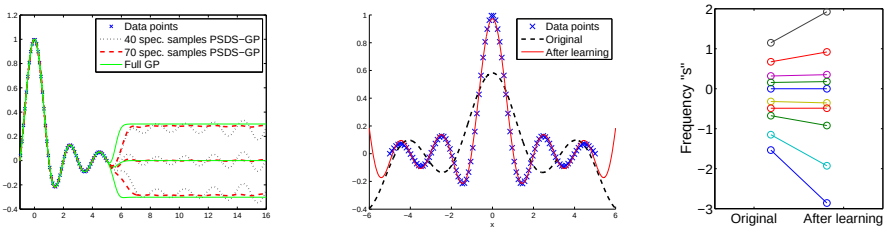

Figure 1: Learning the sinc function (a) full GP and PSDS-GP, (b) effect of learning the sampling points in PSDS-GP, (c) original and learned sampling points.

$$\boldsymbol{\Phi}_E^T \boldsymbol{\Phi}_E = \sum_{j=1}^{n} \boldsymbol{\phi}_E^{(j)T} \boldsymbol{\phi}_E^{(j)} \quad \text{and} \quad \boldsymbol{\Phi}_E^T \boldsymbol{y} = \sum_{j=1}^{n} \boldsymbol{\phi}_E^{(j)T} y_j,$$

and since $\boldsymbol{\phi}_E^{(j)}$ only depends on $\mathbf{x}_j$, this yields an online algorithm. When each data sample $\{\mathbf{x}_j, y_j\}$ arrives, both matrices can be updated in $\mathrm{O}(r^2)$. Using the online method to add new samples instead of directly applying batch eqs. (8) and (9) has therefore the same computational cost $\mathrm{O}(nr^2)$, and needs only $\mathrm{O}(r^2)$ storage, independent of the number of samples. Predictions with the model constructed at each step of the online update can be made at $\mathrm{O}(r^2)$ cost substituting every appearance of these expressions[2] in equations (8) and (9) with their current value, and operating in the appropriate order.

5 Experiments

5.1 Toy 1D problem

The proposed model approximates the behavior of a GP with stationary kernel using a linear combination r sines and cosines, with the frequencies given by the sampling points. In this section we investigate the resulting model in a 1D toy problem.

Since the model contains only a finite number of samples from the spectrum of the kernel, the approximation becomes poorer as we move away from the sampling points. How far we can go while still retaining a good approximation to the original model depends on the number of samples. In Figure 1.a we train a 40 samples approximation with a noiseless sinc. With the hyperparameters obtained, we plot the posterior of the 40 samples PSDS-GP, a 70 samples PSDS-GP and the full GP. We can see a similar behavior for the three models in

[2]For this to be true, we must keep an SVD decomposition of $\boldsymbol{\Phi}_E^T \boldsymbol{\Phi}_E$ constantly updated (so we do not need to compute the costly inversion in eqs. (8) and (9)). Since all updates are rank-one, and the first matrix has also rank one, this can be done at no additional cost.

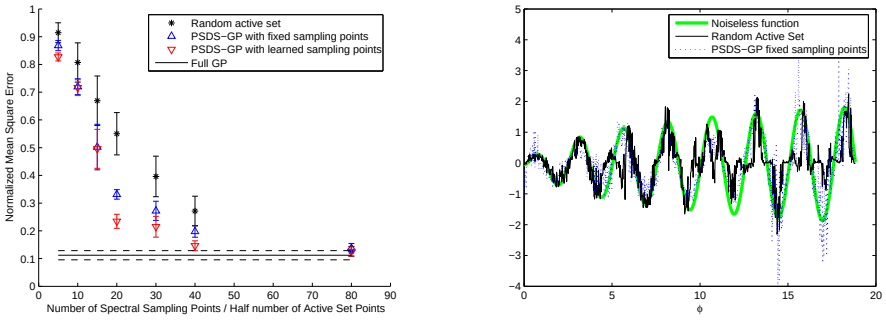

Figure 2: On the left, Normalized Mean Square Error for different approximations are compared. On the right, PSDS-GP with fixed sampling points and random active set selection as a function ϕ.

the left part of the plot, but as we move away from the training data, the 40 samples PSDS-GP mean shows a periodic behavior and cannot be used to make predictions. The 70 samples PSDS-GP is more accurate and can be used to make predictions on the whole plot, but if we moved furher to the right, it would also be no longer valid. The conclusion is that the number of sampling points does not only influence the quality of the aproximation but also the extent of the validity of the model. One std deviation above and below is also plotted and seems good enough for the three models. (Notice that the predicted variance remains approximately constant, oscillations in the plot are due to oscillations in the mean).

Figure 1.b shows the effect of learning the locations of the sampling points. First, a model with fixed sampling points is partially trained (training stops before convergence), and it is plotted with a dashed line. The model is poor, in part because the length scale has not been properly determined. Then the sampling points locations are learned minimizing the negative log marginal likelihood. Learning the sampling points modifies the effective kernel, and includes the effect of learning the lengthscale parameters. The result, plotted with a continuos line is much more accurate. In Figure 1.c we see the expected expansion on the distribution on the sampling points. Notice that the expansion is not exactly uniform. Learning the locations is even better that using the original sampling points with the appropiate lengthscale.

5.2 2D spiral problem

To assess the performance of PSDS-GP in a low dimensional problem, we use a nonlinear dataset (taken from [12]) generated with $x_1(\phi) = \frac{1}{2}\sqrt{\phi}\cos\phi + N(0,\sigma^2)$, $x_2(\phi) = \frac{1}{2}\sqrt{\phi}\sin\phi + N(0,\sigma^2)$ and $f(\phi) = \ln(1+\phi)\sin\left(\frac{5}{2}\phi\right) + N(0,\sigma^2)$. We select $\sigma = 0.1$, and $\phi \in [0, 6\pi]$. 1000 points are used for training/testing.

We compare random active set selection, PSDS-GP with fixed sampling points, PSDS-GP learning sampling points and full GP. The more conservative $2r$ dimension assumption from Section 2.1 holds for the comparison. In the left plot of Figure 2 we show mean squared errors and one std deviation above and below, averaged from ten generations of the dataset. We can see that PSDS-GP with fixed sampling frequencies is better than simple random selection of the active set, especially for small sizes. The error variance is also smaller. Both methods can be applied online, but with 20 spectral samples we achieve higher accuracy than using a 60 points active set. With 80 spectral samples, the kernel approximation is quite accurate and we nearly reach the results of a full GP. Learning the locations further improves accuracy.

5.3 Kin-40k and Pumadyn-32nm datasets

Finally, we run PSDS-GP on the same regression tasks as previous approximations [3] and [9]. We follow precisely their preprocessing and testing methods. For each problem, we use ten random test/train splits and average errors are reported. Datasets are *kin-40k* (10000 training samples, 30000 testing) and *pumadyn-32nm* (7168 training, 1024 testing), both artificially created using a robot arm simulator. They are highly non-linear and low noise.

For both datasets, we compare the method "smo-bart" from [3] (which is the best performing of the three compared methods), the pseudo-inputs method (SPGP) of [9], and PSDS-GP. We first learn the hyperparameters minimizing the negative log-marginal likelihood, leaving sampling points fixed. The results of this process are plotted as upward facing triangles. Then we try to further improve the marginal likelihood by allowing sampling points to move, maintaining the hyperparameters fixed. These final results are plotted as downward facing triangles.

Learning the sampling points is quite beneficial in *kin-40k*. If we compare PSDS-GP results (Figure3, downward facing triangles) with SPGP learning both hyperparameters and pseudo-inputs (blue circles), we see we achieve significantly better performance, even with the conservative comparison. Since our method also learns hyperparameters and sampling points, this would be the fair comparison. If we allow SPGP to fix the hyperparameters to that obtained from a full GP (red squares), performances are similar. All three methods from [3] are outperformed.

In the *pumadyn-32nm* problem, we run into local minima problems when learning the hyperparameters. This problem is also reported in [3]. When they tried learning the hyperparameters using random initialization, sometimes the method failed (leaving them with the same error a linear regressin would), and some other times convergence was achieved. This happens because only 4 of the inputs are useful to solve this problem, and they must be singled out. This can be achieved running a full GP on a 1024 point subset of the training data. We need to do this and initialize the value of the θ_d corresponding to meaningless input dimensions close to zero to escape local minima. The other hyperparameters and meaningful θ_d are still initialized as above, not from

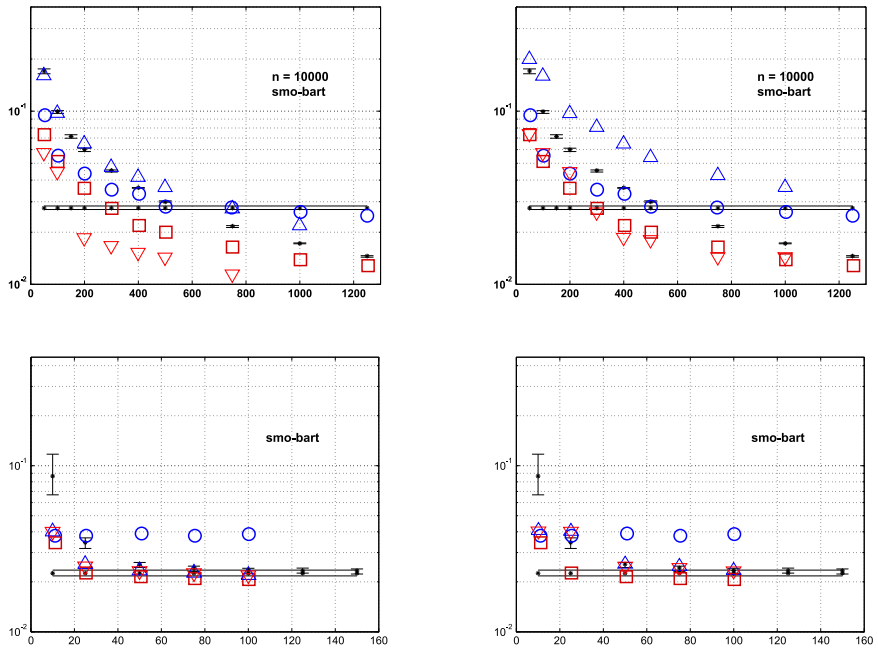

Figure 3: Plots show mean square test error as a function of active/pseudo sample size. On the left, PSDS-GP errors are plotted as a function of the number of spectral samples; on the right, they are plotted as a function of twice the number of samples (the most conservative comparison). Top row plots correspond to *kin-40k*, bottom row to *pumadyn-32nm*. We have added upward facing triangles (PSDS-GP with fixed sampling points) and downward facing triangles (letting PSDS-GP learn sampling points) to the plots from [9] (circles show SPGP with both hyperparameter and pseudo-input learning from random initialization; squares, for *kin-40k*, show SPGP with hyperparameters obtained from a full GP and fixed, whereas for *pumadyn-32nm* they are only initialized from the full GP) and from [3] (only best method, "smo-bart", is shown, using barred stars). The horizontal lines are a full GP trained on a subset from the data.

the full GP. Note that this is not a problem with the approximation to the evidence, since with this initialization the achieved evidence is much higher, but a problem with the optimization process. Learning the sampling points in the the *pumadyn-32nm* problem has nearly no effect (probably due to the low effective dimension). Results are similar to the other sparse approximations. Since for this problem some lengthscales are favorably initialized to escape local minima, SPGP with random initialization (blue circles) is not a fair comparison.

However, still four lengthscales and both σ_0 and σ_n are being learnt from random initialization. In SPGP with red squares all hyperparameters are initialized from full GP.

6 Conclusions and future work

We have proposed a sparse GP approximation that is similar to frequency dual of the state-of-the-art pseudo-inputs method SPGP (the SPGP has an additional white noise process in the prior). The performance of the proposed approximation in the experiments is better than SPGP when both methods need to learn the hyperparameters. If they are known, similar performances are achieved. When fixing the sampling points our methods decreases its performance for higher dimensional problems, but we obtain an online sparse GP method. PSDS-GP is, therefore, mainly useful for low dimensional problems.

Future lines of this work will include its application to other kernel methods, and to evaluate the performance when hyperparameters are updated online (this, together with a forgetting factor for the reduced covariance matrix would lead to an adaptive algorithm, instead of just an online procedure). This adaptiv version could prove to be useful on low dimensional tasks (i.e. typical signal processing applications).

References

[1] L. Csató and M. Opper. Sparse online Gaussian processes. *Neural Computation*, 14(3):641–669, 2002.

[2] C. E. Rasmussen and C. K.I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, 2006.

[3] M. Seeger, C. K. I. Williams, and N. D. Lawrence. Fast forward selection to speed up sparse gaussian process regression. In C. M. Bishop and B. J. Frey, editors, *AISTATS-9*, 2003.

[4] A. J. Smola and P. Bartlett. Sparse greedy gaussian process regression. In *Advances in Neural Information Processing Systems 13*. MIT Press, 2000.

[5] V. Tresp. A bayesian committee machine. *Neural Computation*, 12:2719–2741, 2000.

[6] C. K. I. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*. MIT Press, 2000.

[7] L. Csató. Sparse online gaussian processes. *Neural Computation*, 14:641–668, 2002.

[8] Joaquín Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *J. Mach. Learn. Res.*, 6:1939–1959, 2005.

[9] E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1259–1266. MIT Press, 2006.

[10] J. M. Hammersley. *Ann. New York Acad. Sci.*, 86:844–874, 1960.

[11] W. J. Morokoff and R. E. Caflisch. Quasi-Monte Carlo integration. *J. Comp. Phys.*, 122:218–230, 1995.

[12] A. Pozdnoukhov and S. Bengio. Semi-supervised kernel methods for regression estimation. In *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing*, volume 5, pages 577–580, 2006.