Dependent Gaussian Process Models for MIMO Nonlinear Dynamical Systems using PSO

Gang Cao and Edmund M-K Lai
School of Engineering and Advanced Technology
Massey University
Auckland, New Zealand
Email: {g.cao, elai}@massey.ac.nz

Abstract—**We explore the use of particle swarm optimization (PSO) to learn the hyperparameters for Dependent Gaussian processes (DGPs) which producing a convolution function to establish dependencies between outputs. We employ this convolution function to directly compute the predictive outputs without calculating covariance matrix. This differentiates our method from other hyperparameters learning evolutionary algorithms. We show experimental results of proposed approaches for single and multiple outputs.**

Keywords-dependent Gaussian processes; particle swarm optimization; nonlinear dynamic systems

I. Introduction

Gaussian process (GP) is a probabilistic, non-parametric method that has been used to model various non-linear dynamic processes [1]. It has the advantage that the variance of the outputs can naturally be obtained so that the performance can easily be assessed. While other parametric methods such as artificial neural networks and fuzzy models approximate a system through a set of selected basis functions, GPs model the real relationships between measured data [2]. The key in obtaining good GP models lies in the estimation of the hyperparameters governing the GPs.

So far, GP has mostly been applied to model multiple input single output (MISO) systems [3]. For multiple input multiple output (MIMO) systems, one of the important issues is to capture the auto-covariance and the cross-covariance between outputs. Although there are some known positive definite auto-covariance functions, it is difficult to find cross-covariance functions that result in positive definite covariance matrices. In [3], dependent Gaussian processes (DGPs) have been proposed as an alternative formulation of the covariance function. It treats GPs as white noise sources convolved with smoothing kernels and therefore can also be viewed as a kind of convolved GPs [4]. The hyperparameters of the DGPs can be learnt using a conjugate gradient method with the maximum likelihood (ML) or the maximum a posteriori (MAP) criterion. However, the performance of conjugate gradient highly depends on the initial values, especially in high dimensional objective functions with multiple local minima. The computational burden also increases significantly with the dimension of the problem.

In this paper, we explore the use of particle swarm optimization (PSO) to learn the hyperparameters for DGPs. There have been some attempts to use evolutionary methods for hyperparameters learning problem [2, 5]. However, they have not been applied to models involving DPGs before.

The rest of this paper is organized as follows. In Section II, the concept of DPG is briefly reviewed. How PSO is used for learning the hyperparameters is described in detail in Section III. Section IV presents the results for specific examples. Finally, Section V concludes the paper with indications of some future research directions.

II. Dependent Gaussian Processes

This brief review of DPG mostly follows that in [4].

The output $y(t)$ of a linear time-invariant system with an impulse response $h(t)$ and input $x(t)$ can be obtained by linear convolution:

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(t-\tau)\, x(\tau)\, d\tau \qquad (1)$$

If the input is Gaussian white noise and the system is stable, then the output is Gaussian. A system with M inputs and N dependent outputs can be modelled by a set of $M \times N$ impulse responses as shown in Fig.1. All the outputs are obtained from Gaussian white noises convolved with the relevant smoothing kernels. The *n-th* output is given by (2).

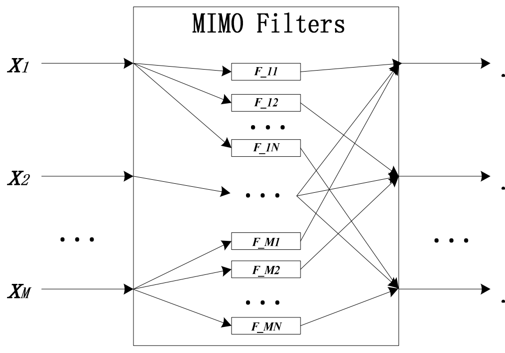


Figure 1. Multiple dependent outputs obtained by Gaussian white noise convolved with smoothing kernels.

$$y_n(t) = \sum_{m=1}^{M} \int_{-\infty}^{\infty} h_{mn}(t-\tau) x_m(\tau) d\tau \qquad (2)$$

Dependencies exist between the output processes because they are obtained from a common input dataset. In this way, the problem of estimating positive definite covariance functions has been transformed into one of estimating impulse responses.

The impulse responses can have various different forms as long as the filter is stable. In this work, Gaussian responses of the form

$$h_{mn}(\mathbf{s}) = \mathbf{v}_{mn} \exp\left(-\frac{1}{2} (\mathbf{s} - \boldsymbol{\mu}_{mn})^T \mathbf{A}_{mn} (\mathbf{s} - \boldsymbol{\mu}_{mn}) \right) \qquad (3)$$

will be used. Here $\mathbf{v}_{mn} \in \mathbb{R}$, $\mathbf{s}, \boldsymbol{\mu}_{mn} \in \mathbb{R}^d$, d is the dimension of the problem and $\boldsymbol{A}_{mn}$ is a $d \times d$ positive definite matrix. So the functions $cov_{ij}^{\mathrm{Y}}(\mathbf{s})$ that define the auto-covariance $(i = j)$ and cross-covariance $(i \neq j)$ between outputs i and j, for a given separation s between arbitrary inputs can be expressed as

$$cov_{ij}^{\mathrm{Y}}(\boldsymbol{s}) = K \exp\left(-\frac{1}{2} (\boldsymbol{s} - \Delta\boldsymbol{\mu})^{\mathrm{T}} \sum (\boldsymbol{s} - \Delta\boldsymbol{\mu}) \right) \qquad (4)$$

where $\sum = \mathbf{A}_{mi}(\mathbf{A}_{mi} + \mathbf{A}_{mj})^{-1}\mathbf{A}_{mj}$, $\Delta\boldsymbol{\mu} = \boldsymbol{\mu}_{mi} - \boldsymbol{\mu}_{mj}$and

$$K = \frac{(2\pi)^{\frac{d}{2}} v_{mi} v_{mj}}{\sqrt{|\mathbf{A}_{mi} + \mathbf{A}_{mj}|}}$$

So a covariance matrix from these functions can be obtained as

$$\boldsymbol{C}_{ij} = \begin{bmatrix} cov_{ij}^{\mathrm{Y}}(x_{i1} - x_{j1}) & \cdots & cov_{ij}^{\mathrm{Y}}(x_{i1} - x_{jn_j}) \\ \vdots & \ddots & \vdots \\ cov_{ij}^{\mathrm{Y}}(x_{in_i} - x_{j1}) & \cdots & cov_{ij}^{\mathrm{Y}}(x_{in_i} - x_{jn_j}) \end{bmatrix} \qquad (5)$$

The log-likelihood is given by

$$\mathbf{L} = -\frac{1}{2}\log|\boldsymbol{C}| - \frac{1}{2}\mathbf{y}^{\mathrm{T}}\boldsymbol{C}^{-1}\mathbf{y} - \frac{\mathrm{W}}{2}\log 2\pi \qquad (6)$$

where n_i denotes the number of observation in *i-th* dataset, $\mathrm{W} = \sum_{i=1}^{N} n_i$ denotes the total number of observations and $\mathbf{y} = [(y_{11} \cdots y_{1n_1}), \cdots, (y_{i1} \cdots y_{in_i}), \cdots, (y_{N1} \cdots y_{Nn_N})]^{\mathrm{T}}$.

Similar to single output GP models, the predictive distribution at a test point $\mathbf{x}_*$ on the *i-th* output is Gaussian with mean $m(\mathbf{x}_*)$ and variance $v(\mathbf{x}_*)$ given by

$$m(\mathbf{x}_*) = \mathbf{k}^{\mathrm{T}}\boldsymbol{C}^{-1}\mathbf{y} \qquad (7)$$

$$v(\mathbf{x}_*) = \kappa\text{-}\mathbf{k}^{\mathbf{T}}\boldsymbol{C}^{-1}\mathbf{k} \qquad (8)$$

where

$$\kappa = cov_{ii}^{\mathrm{Y}}(0) = v_i^2 + \omega_i^2 + \sigma_i^2 \qquad (9)$$

$$\mathbf{k}^{\mathrm{T}} = \left[\mathbf{k}_1^{\mathrm{T}}, \cdots \mathbf{k}_i^{\mathrm{T}}, \cdots \mathbf{k}_N^{\mathrm{T}} \right] \qquad (10)$$

$$\mathbf{k}_i^{\mathrm{T}} = \left[cov_{ij}^{\mathrm{Y}}(\mathbf{x}_* - x_{j1}), \cdots, cov_{ij}^{\mathrm{Y}}(\mathbf{x}_* - x_{jn_j}) \right] \qquad (11)$$

III. Particle Swarm Optimization

Many optimization problems involve objective functions which consist of multiple local minima. Gradient descent methods are not effective in such cases as the algorithm can get stuck in a local minimum. One effective approach for solving such optimization problems is known as particle swarm optimization (PSO). It was first introduced in 1995 [6]. PSO solves an optimization problem by simulating the social behaviour of organisms. It consists of many particles where each individual one represents a solution to the problem. The particles can move around in the search space and optimize its position according to its own as well as the whole population's experience.

Assuming the search space is D-dimensional and the *i-th* particle of the swarm is denoted by $x_i = \{x_{i1}, \cdots, x_{iD}\}$. In the *t-th* epoch, the position and velocity of every particle would be respectively updated by

$$x_i^{t+1} = x_i^t + v_i^t \qquad (12)$$

$$v_i^{t+1} = \omega_i^t v_i^t + c_1 r_1 (pbest_i^t - x_i^t) + c_2 r_2 (gbest^t - x_i^t) \qquad (13)$$

where ω is the inertia factor, c_1 and c_2 are the acceleration coefficients, r_1 and r_2 are two random numbers which are uniformly distributed in the range $[0,1]$. Also, in (13), $pbest_i = \{pbest_{i1}, \cdots, pbest_{iD}\}$, and $gbest$ denote the best indivdual position and the best global position respectively. The best individual position in next epoch is updated by

$$pbest_i^{t+1} = \begin{cases} x_i^{t+1}, & \text{if } f(x_i^{t+1}) < f(pbest_i^t) \\ pbest_i^t, & \text{otherwise} \end{cases} \qquad (14)$$

where $\boldsymbol{f}(\cdot)$ denotes the fitness function. The best global position in next epoch is updated by

$$gbest^{t+1} = argmin\left(f(gbest^t), f(pbest_i^{t+1}) \right) \qquad (15)$$

The Inertia factor ω plays an important role in PSO. In practice, a larger ω is used at the beginning of the search to achieve better global search ability. A smaller value of ω is used towards the end of the optimization to provide better local search ability. The inertia factor in each epoch is obtained by

$$\omega^t = \omega_{end} + (\omega_{start} - \omega_{end}) \exp\left(-k \times (\frac{t}{T_{max}})^2 \right) \qquad (16)$$

where ω_{start} and ω_{end} denote the pre-determined start and end values of the inertia factor respectively. k is a control factor controlling the shape of function, and T_{max} is the maximum epoch number.

IV. Hyperparameter Learning Using PSO

Based on the DGP introduced in Section II, a set of hyperparameters $\boldsymbol{\theta} = \{\boldsymbol{v}_{mn}, \boldsymbol{\mu}_{mn}, \boldsymbol{A}_{mn}\}$ can be defined for the Gaussian filters. For real-world problems, a noise term, $\boldsymbol{\eta}_n(t) \sim \mathcal{N}(0, \sigma^2)$, is usually added to the right hand side of (2) to account for measurement errors and other noises. However, this noise term will not be included as part of the hyperparameters that are to be optimized. In order to reduce the number of hyperparameters, we further assume that the impulse responses h_{mn} are the same for each output. Therefore, the hyperparameters become $\boldsymbol{\theta} = \{\boldsymbol{v}_n, \boldsymbol{\mu}_n, \boldsymbol{A}_n\}$, for $n = 1, \dots, \mathrm{N}$ where N is the total number of outputs. We shall also assume that $\boldsymbol{A}_n = \exp(\boldsymbol{\varphi}_n)$. The training dataset $\mathcal{D} = \{\mathcal{D}_1 \cdots \mathcal{D}_\mathrm{N}\}$, where $\mathcal{D}_\mathrm{i} = \{\mathbf{x}_{i,} y_i\}_{i=1}^{n_i}$ is composed of training data for each output.

To learn these hyperparameters using PSO, the particle and fitness function need to be defined. In our PSO algorithm, each specific set of hyperparameters is treated as a particle. The search range of a particle plays an important role in the performance. However, there is no systematic method to determine this range; it is entirely based on trial and error. In our implementation, a particle is initialized based on following Gaussian distributions:

$$\boldsymbol{v}_n \sim N(1, 0.5^2) \tag{17}$$

$$\boldsymbol{\mu}_n \sim N(1, 0.5^2) \tag{18}$$

$$\boldsymbol{\varphi}_n \sim N(3.5, 1^2) \tag{19}$$

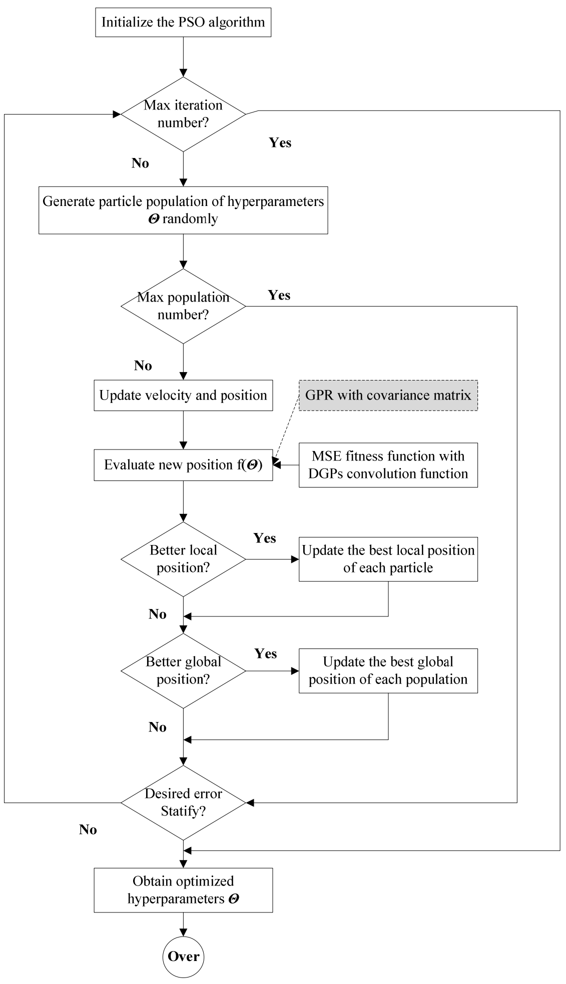
Therefore, the searching range of each individual hyperparameter is in the same range as its distribution.

The mean squared error (MSE)

$$MSE = \frac{1}{L}\sum_{i=1}^{L}|e_i|^2 \tag{20}$$

will be used as the fitness function, where L is the number of data samples and e is instantaneous error between the actual and the predictive outputs. The predictive outputs are computed from (2) without the need for the covariance matrix. This differentiates our method from other hyperparameters learning evolutionary algorithms where the covariance matrix needs to be computed in order to assess the performance of an individual particle in each epoch.

Fig.2 shows the flow chart of hyperparameters learning using PSO. Firstly, PSO algorithm is initialized. Then in each epoch, several particles are generated randomly. The position and velocity of each particle is updated according to (12) and (13) respectively. The fitness of the updated particles is evaluated by (20) to find the particle with the best fitness. The algorithm terminates when it reaches the maximum number of epochs or the desired error. The optimal set of hyperparameters will be used to compute the covariance matrix.



Initialize the PSO algorithm
Max iteration number?
Yes
No
Generate particle population of hyperparameters $\boldsymbol{\Theta}$ randomly
Max population number?
Yes
No
Update velocity and position
GPR with covariance matrix
Evaluate new position f($\boldsymbol{\Theta}$)
MSE fitness function with DGPs convolution function
Better local position?
Yes
Update the best local position of each particle
No
Better global position?
Yes
Update the best global position of each population
No
Desired error Statify?
No
Obtain optimized hyperparameters $\boldsymbol{\Theta}$
Over

Figure 2. Flow chart of hyperparameters learning using PSO. The dashed line and grey area show other evalution approach different with our method.

V. Experiments and Results

The method outlined above is tested using two examples. The first one is a single output nonlinear dynamical system while method the second one is a multiple output application for the prediction of the mechanical properties of steel.

The PSO parameters used in both examples are the same: particle population is 50; maximum iteration number T_{max} is 2000; inertia factor $\omega_{start}$and ω_{end} are 0.4 and 0.9 respectively; two acceleration coefficients are both 1.49445; control factor k is 10; the desired error of fitness function is 1e-9.

A. Nonlinear Dynamic System Modeling

The single-output nonlinear dynamic system is defined as follows:

$$\begin{aligned} y(k) &= 0.893y(k-1) + 0.037y^2(k-1) - 0.05y(k-2) \\ &\quad -0.05u(k-1)y(k-1) + 0.157u(k-1) \end{aligned} \tag{21}$$

where u is the input signal of system and y is the output signal of system. For this system, we define $\boldsymbol{x} = [u(k-1), y(k-1), y(k-2)]$ and $y = y(k)$. Our task here is modeling a multiple inputs and single output nonlinear dynamic system using Dependent Gaussian processes. Separate training and test data are generated by simulation assuming $\boldsymbol{u} \sim \mathcal{U}(-2, 4)$. The training data consist of 34 samples and the test data have 20 samples. To evaluate the performance of particles, we simply added white noise with variance $\sigma^2 = 1$ to the output convolution of (2) and compute the corresponding MSE by (20).

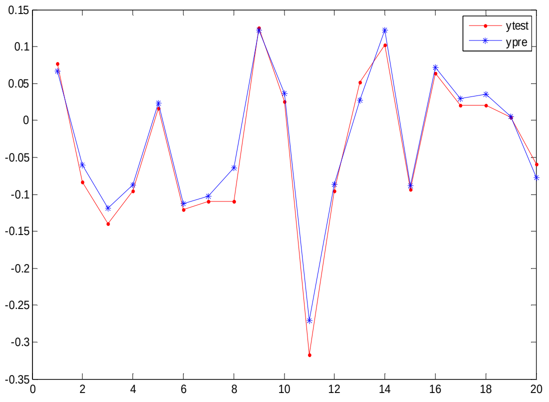


Figure 3. Comparison between predictive outputs and test outputs

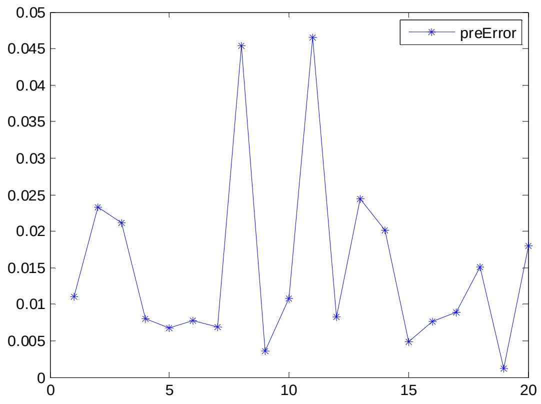


Figure 4. Comparison between predictive outputs and test outputs

In this experiment, the optimized hyperparameters obtained are: $\boldsymbol{\theta}_{\text{optimal}}$ = {1.1574, -0.1042, 0.3308}.

Based on these hyperparameters, the covariance matrix $\boldsymbol{C}$ is computed by (4), and the predicted outputs is computed by (7). Fig.3 shows the difference between 20 predicted outputs and the corresponding test outputs. Fig.4 shows the absolute error between predicted and test outputs. All 20 predictive absolute errors are less than 0.05. It shows that the model error is reasonably small.

B. Steel mechanical properties prediction

The second example involves real data obtained from industrial steel production. The data consist of the measured values of chemical elements (C, Si, Mn, P, S, V), process parameters (slab thickness and final temperature) and mechanical properties (yield strength σ_s, tensile strength σ_b and elongation δ_5). The goal is to relate these mechanical properties to the 6 chemical elements and the 2 process parameters. The inputs are therefore given by x = $[Ct_C, Ct_{si}, Ct_{mn}, Ct_p, Ct_s, Ct_v, Tk_{slab}, T_{final}]$, where Ct_C denotes C element content for example, Tk_{slab} denotes slab thickness and T_{final} denotes the final temperature. The three outputs are $y = [\,\sigma_s\,, \sigma_b, \delta_5]$. Therefore, the hyperparameters are $\boldsymbol{\theta} = [v_1, v_2, v_3, \mu_1, \mu_2, \mu_3, \varphi_1, \varphi_2, \varphi_3]$.

The whole dataset consists of 84 samples collected from a steel-making factory. The range of the inputs and outputs is quite different for each parameter. We normalize all training and test data to a range of [0.1, 0.9]. The preprocessed dataset is divided into four parts – 3 training datasets $D_{train} = \{D_1, D_2, D_3\}$, one for each output, and a test dataset $D_{test} = \{x_j, y_j\}_{j=1}^{20}$. Similar to the first experiment, we simply added white noise with variance $\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = 1$ to the right hand side of (2) and compute the corresponding MSE by (20).

It turns out that we are unable to compare the predictive outputs to the test outputs because the resultant covariance matrix is ill conditioned. The optimization was repeated several times with different PSO parameters. However, the results are similar. One possible reason is that the matrix A in (3) may not be positive definite during the optimization. This matrix plays an important role because not only the covariance matrix depends on it, it also influences hyperparameter learning in the PSO algorithm. A second possible reason is that we have assumed impulse responses h_{mn} for same output are the same. This restriction may produce a poorly optimized set of hyperparameters, which in turn produces a poor covariance matrix.

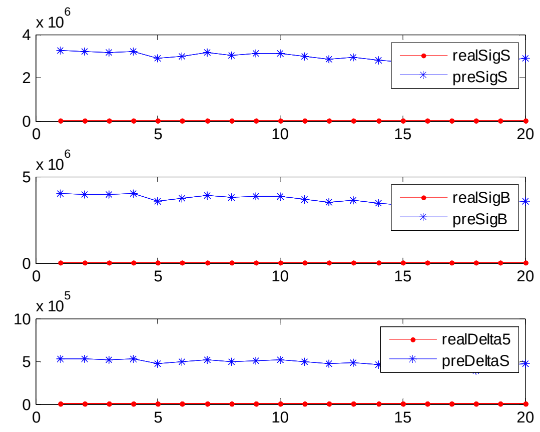


Figure 5. The reusluts of mechanical proporties prediction using DGPs with a PSO learning method

VI. Conclusions

We explored the use of PSO to learn the hyperparameters for DGPs. Compared with other evolutionary learning approach for hyperparameters, the performance of each generation of hyperparameters can be directly evaluated because the outputs can be obtained by Gaussian white noise convolved with smoothing kernels in DGPs. Our experiments show that our approach works well for a single-output problem, but is unable to obtain a proper model for a multiple output one. We have suggested possible reasons for that. Therefore, as future work, we will focus on solving this problem. After that, our method for learning hyperparameters will be compared with others. Finally, we also will improve our hyperparameters learning method so as to improve its accuracy and stability.

References

[1] G. Gregor and G. Lightbody, "Gaussian process approach for modelling of nonlinear systems," Engineering Applications of Artificial Intelligence, vol. 22, no. 4–5, pp. 522–533, Jun. 2009.

[2] D. Petelin, B. Filipic, and J. Kocijan, "Optimization of Gaussian process models with evolutionary algorithms," Adaptive and Natural Computing Algorithms,vol.6593, pp. 420–429, 2011.

[3] P. Boyle and M. Frean, "Dependent gaussian processes," Advances in neural information processing systems, Vol. 17, pp. 217–224, 2005.

[4] M. A. Alvarez and N. D. Lawrence, "Computationally Efficient Convolved Multiple Output Gaussian Processes.," Journal of Machine Learning Research, vol. 12, no. 5, pp. 1459–1500, 2011.

[5] F. Zhu, C. Xu, and G. Dui, "Particle swarm Hybridize with Gaussian Process Regression for displacement prediction," IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), pp. 522–525, 2010.

[6] J. Kennedy and R. Eberhart, "Particle swarm optimization," IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948, 1995