Approximation of Gaussian Process Regression Models after Training

Thorsten Suttorp and Christian Igel

Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany
{thorsten.suttorp, christian.igel}@neuroinformatik.rub.de

Abstract. The evaluation of a standard Gaussian process regression model takes time linear in the number of training data points. In this paper, the models are approximated in the feature space after training. It is empirically shown that the time required for evaluation can be drastically reduced without considerable loss in performance.

1 Introduction

The training time of a standard Gaussian process model is polynomial and its execution time (i.e., the time required to evaluate the model for a given input after training) is linear in the number of training samples. This scaling behavior limits the applicability of standard Gaussian processes for large scale machine learning problems. In this paper, we assume that training is done offline and training time is not (yet) a problem, but execution time is crucial. This is a rather special situation, but a scenario we are indeed facing in practice, for example in the domain of real-time driver assistance systems or biometric applications.

Most papers on generating sparse Gaussian process models directly address complexity reduction during training. In most instances this is realized by incorporating approximations in the (Bayesian) derivation of Gaussian processes, e.g., [1, 2, 3]. A good overview of approximation methods for Gaussian processes is given in [4]. Here we consider a two-stage process. The idea is to first generate an as good as possible solution by our favorite (large-scale) training algorithm and to reduce the complexity of the result by a general, robust approximation algorithm [5, 6]. While in general more time consuming, this process may have the potential to generate better solutions than combining learning and approximation in one step because the learning algorithm can make the best possible use of the available information before the solution is carefully approximated. Anyway, this conceptual clear separation between training and approximation has the practical advantage that complexity reduction can be performed independent of the way the initial model has been obtained (e.g., using freely available software: `shark-project.sourceforge.net`).

2 Gaussian Process Regression

In the following, we derive Gaussian regression models from the viewpoint of regularization networks. Our goal is to learn a real-valued function $f : \mathcal{X} \to \mathbb{R}$ based on sample data $S = \{(x_1, t_1), \dots, x_\ell, t_\ell)\} \in (\mathcal{X} \times \mathbb{R})^\ell$, where $\mathcal{X}$ denotes the input space. We consider linear regression in some feature space $\mathcal{F}$ endowed with a dot product

$$f(x) = \langle \phi(x), w \rangle \ .$$

The feature map $\phi : \mathcal{X} \to \mathcal{F}$ maps input elements to a reproducing kernel Hilbert space $\mathcal{F}$ such that $\langle \phi(x), \phi(z) \rangle = k(x, z)$ for a positive definite kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The vector $w \in \mathcal{F}$ is chosen such that the regularized empirical risk

$$R(w, S, \lambda) = \sum_{i=1}^{\ell} (t_i - f(x_i))^2 + \lambda \|f\|_{\mathcal{F}}^2$$

is minimized. This is achieved by expressing w as $w = \sum_{i=1}^{\ell} \alpha_i \phi(x_i)$ and computing $\alpha = (K + \lambda I)^{-1} t$, where $t = (t_1, \ldots, t_\ell)^{\mathrm{T}}$, $K \in \mathbb{R}^{\ell \times \ell}$ is the kernel matrix with $[K]_{ij} = k(x_i, x_j)$, and $\alpha \in \mathbb{R}^{\ell}$.

These regression models are called regularization networks [7, 8] and can also be derived in a Bayesian framework as special Gaussian processes. A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution. Here, these random variables are the outputs $f(x_i)$ given the inputs x_i. We assume that the observations of the outputs are subject to i.i.d. Gaussian noise with zero mean and variance λ $\big($i.e., $t_i = f(x_i) + \epsilon_i$ and $\epsilon_i \sim \mathcal{N}(0, \lambda)\big)$. If we set the covariance matrix of the Gaussian process to the kernel matrix and the mean to zero, then $f(x)$ is the maximum a posteriori estimate of an observation given an input x. For a detailed introduction to Gaussian processes we refer to recent textbooks [9, 10].

3 Resilient Approximation of Gaussian Process Models

For many practical applications the calculation of $f(x) = \sum_{i=1}^{\ell} \alpha_i k(x_i, x)$ obtained by Gaussian process regression is computationally too expensive. This expression is well known from classification with support vector machines, and a number of authors address its approximation [11, 12, 6]. Almost all work in this area is based on an idea presented in [11], where it is suggested to approximate $f = \sum_{i=1}^{\ell} \alpha_i k(x_i, \cdot)$ by a function $f' = \sum_{i=1}^{L} \beta_i k(z_i, \cdot)$ with $z_i \in \mathcal{X}$, $\beta_i \in \mathbb{R}$, and $L \ll \ell$ such that the distance function $\rho^2 := \|f - f'\|_{\mathcal{F}}^2$ is minimized.

The reduced set $\{z_1, \ldots, z_L\}$ of *approximating vectors* (AVs) can be constructed by iteratively adding single vectors to a given solution. The following algorithm implements this idea. The different steps are discussed below.

Algorithm: Approximation of Gaussian Process Models

```
1 initialize $f_1 := f$
2 for $i = 1, \ldots, L$ do
3     determine $z_i$ by minimizing $\rho^2(z_i) = \|f_i - \beta_i \phi(z_i)\|_{\mathcal{F}}^2$
4     calculate optimal $\beta_1, \ldots, \beta_i$
5     $f_{i+1} \leftarrow f - \sum_{j=1}^{i} \beta_j \phi(z_j)$
6 adjust $z_1, \ldots, z_L$ and $\beta_1, \ldots, \beta_L$ by global gradient descent
7 determine optimal offset $b'$
```

In [5] the authors note that for the case that the approximation consists of a single term (i.e., $f' = \beta \phi(z)$) it is not necessary to directly minimize ρ^2, but it is instead possible to minimize the distance between f and its orthogonal projection onto $\mathrm{span}(\phi(z))$, which is given by $\left\| \frac{\langle f, \phi(z) \rangle}{\langle \phi(z), \phi(z) \rangle} \phi(z) - f \right\|_{\mathcal{F}}^2 = \|f\|_{\mathcal{F}}^2 - \frac{\langle f, \phi(z) \rangle^2}{\langle \phi(z), \phi(z) \rangle}$. We obtain the distance measure

$$E(z) := -\frac{\langle f, \phi(z) \rangle^2}{\langle \phi(z), \phi(z) \rangle} = -\frac{f^2(z)}{k(z,z)} \;,$$

which can be minimized using gradient methods. Therefore, the derivatives of $E(z)$ with respect to all components of a single vector z are needed:

$$\partial_z \frac{f^2(z)}{k(z,z)} = \frac{1}{k^2(z,z)} \cdot \Big(\partial_z f^2(z) \cdot k(z,z) - f^2(z) \cdot \partial_z k(z,z) \Big) \;.$$

The parameter L is usually chosen as large as possible given the time constraints of the application. In general, the accuracies on the training or (even better) on an external test set achieved by the original and approximated model can guide the choice of L.

Choosing Initial Vectors. For applying the iterative technique described above starting points for the optimization of $E(z)$ have to be selected. We suggest to use α-*proportional selection,* which has its origin in stochastic universal sampling known from evolutionary algorithms [13]. A weighted roulette wheel containing all inputs x_i is simulated. The slots are sized according to the corresponding $|\alpha_i|$, and L equally spaced markers are placed along the outside of the wheel. The wheel is spun once, and the slots that are hit by the markers define the L initial vectors. This heuristic is based on the idea that vectors that received a big $|\alpha_i|$ during Gaussian process regression are more important than those with small ones.

Determining the Optimal Coefficients. In each iteration of the approximation algorithm the coefficients $\beta = (\beta_1, \ldots, \beta_L)$ have to be determined anew. The optimal coefficients for approximating $f = \sum_{i=1}^{\ell} \alpha_i k(x_i, \cdot)$ by $f' = \sum_{i=1}^{L} \beta_i k(z_i, \cdot)$ for linear independent $\phi(z_1), \ldots, \phi(z_L)$ can be computed as $\beta = (K^z)^{-1} K^{zx} \alpha$, where $K_{ij}^z := k(z_i, z_j)$ and $K_{ij}^{zx} := k(z_i, x_j)$, see [12].

Global Gradient Descent. For further minimizing the distance function ρ^2 gradient descent can be applied to all parameters of the sparse solution f' [5]. The derivatives of ρ^2 with respect to all components of an AV are given by

$$\partial_{z_i} \rho^2 = -2 \sum_{j=1}^{\ell} \alpha_j \beta_i \partial_{z_i} K_{ij}^{zx} + 2 \sum_{j=1}^{L} \beta_j \beta_i \partial_{z_i} K_{ij}^{zz}$$

and the derivatives with respect to the coefficients β_i by

$$\partial_{\beta_i} \rho^2 = -2 \sum_{j=1}^{\ell} \alpha_j K_{ij}^{zx} + 2 \sum_{j=1}^{L} \beta_j K_{ij}^{zz} \;.$$

Computing the Optimal Offset. In our experiments we found that the approximation f' could be improved when an additional offset was incorporated. Therefore, we consider the differences of f and f': $b'(x) = \sum_{i=1}^{\ell} \alpha_i k(x_i, x) - \sum_{i=1}^{L} \beta_i k(z_i, x)$ for all training inputs and compute their mean

$$b' = \frac{1}{\ell} \sum_{i=1}^{\ell} b'(x_i) \ .$$

The resulting regression function that is used throughout this study is given by $\tilde{f}(x) = f'(x) + b'$.

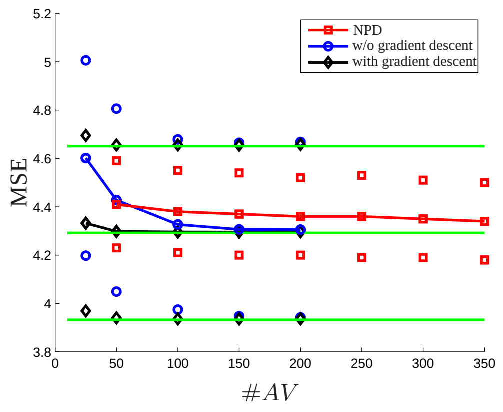
Resilient Minimization of the Distance Function. Approximating the regression function of a Gaussian process is basically the same as approximating the decision function of a support vector machine. In [6] we employed the efficient Rprop (resilient backpropagation) algorithm [14, 15] for the resilient approximation of SVMs. Because we found this algorithm to provide reliably good results, we propose to apply it also to the problem of Gaussian process approximation.

The Rprop algorithm is an efficient and robust first order gradient method that considers only the signs of the partial derivatives of the error function E to be optimized and not their amount. The iRprop$^+$ algorithm used in this study implements weight backtracking. It partially retracts "unfavorable" previous steps. Whether a parameter change was "unfavorable" is decided based on the evolution of the partial derivatives and the overall error [15]. The standard parameters give reliably good results, and the initial step sizes used in the algorithm can be chosen small, so that no parameter-tuning is required when applying resilient backpropagation.

4 Experiments

Setup. We applied the proposed algorithm to the Abalone data set, where the ages of abalone are predicted from easily obtainable physical measurements. Second, we considered the Boston data set. Here, housing values in suburbs of Boston are predicted. Both data sets are available from the UCI machine learning repository [16] and are frequently used real word data sets for regression.

We adopted the experimental setup from [17]. For each data set we performed the following preprocessing: We transformed each continuous feature to zero mean and unit variance. The gender encoding (male/female/infant) of the abalone was mapped to $\{(1,0,0),(0,1,0),(0,0,1)\}$. The Boston (Abalone) data set was randomly partitioned into 100 (10) splits with 481 (3000) examples for training and 25 (1177) examples for testing. In the experiments we used Gaussian kernels $k(x,z) = \exp(-\gamma \|x-z\|^2)$, and for every partition we trained a Gaussian process with well performing hyperparameters ($\gamma = 10^{-1.5}, \lambda = 10^{-1.5}$ and $\gamma = 10^{-1.2}, \lambda = 10^{-2}$ for Abalone and Boston respectively). The resulting Gaussian process models were approximated for different predefined numbers of AVs using *resilient approximation* (with and without final gradient descent) as described in Section 3, and in each case the mean squared error (MSE) on the test data was observed.



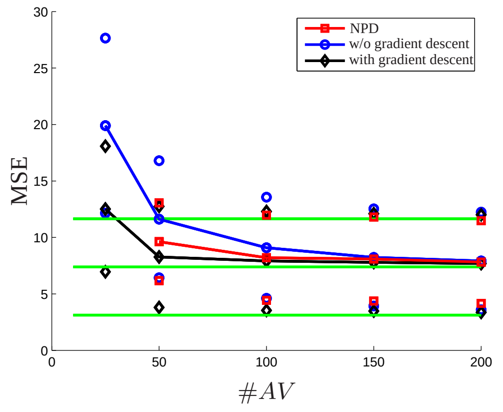


Fig. 1: Results for the approximation of Gaussian process regression models. Averages of the mean squared errors with the standard deviations are depicted. Top plot: results for Abalone. Bottom plot: results for Boston.

Results. The results of the approximation trials are depicted in Fig. 1. In each case, the number of AVs is plotted against the MSE on the test data. In addition to the mean the standard deviation is depicted. The light gray horizontal lines give the MSE and the corresponding standard deviation of our original Gaussian process model.

Resilient approximation (with and without final gradient descent) led to regression functions that achieved the performance of the original Gaussian process at a fraction of its computational costs.

The performance of our approximation algorithm was compared to the results presented in [17], which are obtained by applying nonlinear pseudodiscriminants (NPDs). Because of the different quality of the approaches and probably different kernel widths, resilient approximation and NPD are not directly comparable. Nevertheless, the results provide an indication of the quality of our approach. The experiments gave a clear ranking of the algorithms and showed that resilient approximation with final gradient descent performed best.

5 Conclusions

The approximation of Gaussian process models can be used to obtain a regression function for real-world applications requiring fast decisions. For both real world data sets that were considered in this study the performance of the approximation achieved the performance of the original Gaussian process model at a fraction of its computational costs. The final gradient descent improved the quality of the approximations. Adding an offset parameter in the sparse model improved the performance.

References

[1] A. J. Smola and P. L. Bartlett. Sparse greedy Gaussian process regression. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13, pages 619–625. MIT Press, 2001.

[2] M. Seeger, C. K. I. Williams, and N. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In C. M. Bishop and B. J. Frey, editors, *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2003.

[3] E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2006.

[4] J. Quiñonero-Candela, C. E. Rasmussen, and C. K. I. Williams. Approximation methods for Gaussian process regression. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*, pages 203–223. MIT Press, 2007.

[5] B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. J. Smola. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, 1999.

[6] T. Suttorp and C. Igel. Resilient simplification of kernel classifiers. In J. Marques de Sá et al., editors, *Proceedings of the 17th International Conference on Artificial Neural Networks (ICANN 2007)*, volume 4668 of *LNCS*, pages 139–148. Springer-Verlag, 2007.

[7] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[8] T. Poggio and S. Smale. The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society (AMS)*, 50(5):537–544, 2003.

[9] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[10] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Springer-Verlag, 2006.

[11] C. J. C. Burges. Simplified support vector decision rules. In *Proceedings of the 13th International Conference on Machine Learning (ICML 1996)*, pages 71–77, 1996.

[12] B. Schölkopf, P. Knirsch, A. J. Smola, and C. J. C. Burges. Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature space. In P. Levi, R.-J. Ahlers, F. May, and M. Schanz, editors, *DAGM-Symposium*, pages 124–132. Springer-Verlag, 1998.

[13] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, 1987.

[14] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons – From backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces*, 16(5):265–278, 1994.

[15] C. Igel and M. Hüsken. Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*, 50(C):105–123, 2003.

[16] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[17] E. Andelić, M. Schafföner, M. Katz, S. E. Krüger, and A. Wendemuth. Kernel least-squares models using updates of the pseudoinverse. *Neural Computation*, 18:2928–2935, 2006.