Scalable Bounding of Predictive Uncertainty in Regression Problems with SLAC

Arno Blaas[1,2], Adam D. Cobb[1], Jan-Peter Calliess[2], and Stephen J. Roberts[1,2]

[1]Machine Learning Research Group, [2]Oxford-Man Institute, University of Oxford, Oxford, United Kingdom

Abstract. We propose SLAC, a sparse approximation to a Lipschitz constant estimator that can be utilised to obtain uncertainty bounds around predictions of a regression method. As we demonstrate in a series of experiments on real-world and synthetic data, this approach can yield fast and robust predictive uncertainty bounds that are as reliable as those of Gaussian Processes or Bayesian Neural Networks, while reducing computational effort markedly.

Keywords: Predictive Uncertainty Bounds · Regression · Lipschitz Interpolation

1 Introduction and Background

Machine learning methods are typically utilised in regression tasks where little is known a priori and one prefers 'black-box' approaches that are endowed with the capacity to flexibly learn rich function classes. However, when regression methods are employed in decision making, quantifying the uncertainty around predictions can often be key. While many such methods exist, uncertainty bounds often rest on assumptions that are hard to establish *a priori*, necessitating either manual or optimisation based tuning approaches to yield sufficient black-box learning capabilities. Unfortunately, this renders the bounds less interpretable and typically, the computational effort intractable for many applications.

We consider two widely used methods that offer uncertainty quantifications, Gaussian Processes (GPs) [1] and Bayesian Neural Networks (BNNs) [2]. In regression, both approaches compute Gaussian posterior input-output relationships that allow the usage of subjective confidence intervals around predictions for a given input as uncertainty bounds. As with all Bayesian methods, the problem is that these subjective bounds, as well as the pertaining predictions, are contingent on a priori choices such as the nature of the probability space and the prior distribution. In GPs, these are encoded in the kernel and mean functions (as well as their hyper-parameters). In BNNs, the prior is encoded by the architecture of the network, as well as by the hyperparameters, particularly the dropout rate.

To facilitate black-box learning and to avoid hand-tuning, approaches for model selection involve optimising these a priori choices to maximise some data-dependent criterion function such as the marginal log-likelihood. Unfortunately,

this pragmatic approach has undesirable side-effects. Firstly, it can greatly inflate the computational effort required for training and make these approaches too slow to use to derive uncertainty bounds in online, or quickly changing, settings where frequent model re-training is necessary. Secondly, since also the optimiser (and its initialisations) becomes part of the training algorithm, the uncertainty bounds become dependent on the optimiser's solution (with its performance often dependent on initialisations). This makes the interpretation of the uncertainty bounds questionable and their accuracy has to be assessed empirically on a case by case basis.

While we might argue that the tension between the desire to have reliable uncertainty bounds and fast, reliable black-box learning can never be fully reconciled, in this paper we present our early work on SLAC (**S**parse **L**azily **A**dapted Lipschitz-**C**onstant) uncertainty bounds as an approach to ameliorate this tension. Conceived from black-box learners whose training is magnitudes faster and simpler than for existing methods, SLAC bounds demonstrably serve their purpose in practice: on a range of benchmark comparisons against GPs and BNNs, they compare favourably against their competitors both in terms of computational time and the reliability of their uncertainty bounds. Code for the experiments described in this paper can be found online [1].

2 Task and Approach

In supervised Machine Learning applications, we typically desire to learn from a set of training points $\mathcal{D}_n := \{(s_i, f(s_i))|i = 1, \dots, n\} \subset (\mathcal{X}, \mathcal{Y})$, generated by some unknown target function f in order to predict its values for new test points $\mathcal{T}_q := \{(t_i|i = 1, \dots, q\} \subset \mathcal{X}$. In many real-world applications it is necessary that these predictions can reliably quantify uncertainty. For standard regression problems, where $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}$, this typically translates to finding two functions $l(\cdot) \leqslant u(\cdot)$ such that $\forall x \in \mathcal{X} : \quad Pr\left(l(x) \leqslant f(x) \leqslant u(x)|\mathcal{D}_n\right) = 1 - \delta$, with $\delta \in [0, 1)$. For safety critical applications $\delta \approx 0$ is the most important case as in these applications we ultimately wish to bound the uncertainty with high probability and thereby (almost) replace it with certainty. We thus use the phrase *uncertainty bounds* (or simply *bounds*) for $l(\cdot)$ and $u(\cdot)$ to indicate $\delta \approx 0$.

2.1 Definition of our Model

Our model is originally inspired by the deterministic bounds (i.e. $\delta = 0$) that can be inferred if f is known to be Lipschitz-continuous with best Lipschitz-constant L^* with respect to some metrics $\mathfrak{d} : \mathcal{X}^2 \to \mathbb{R}_{\geqslant 0}$, $\mathfrak{d}_{\mathcal{Y}} : \mathcal{Y}^2 \to \mathbb{R}_{\geqslant 0}$, which is defined as L^* being the smallest value for which it holds that $\forall x, x' \in \mathcal{X} : \mathfrak{d}_{\mathcal{Y}}(f(x), f(x')) \leqslant L^* \mathfrak{d}(x, x')$. In this case, Sukharev [3] has shown that with

[1] https://github.com/arblox/SLAC

bounds defined as

$$\begin{aligned} \mathfrak{u}_{L*}(x;\mathcal{D}_n) &:= \min_{s_i \in \mathcal{X}_n} (f_i + L^* \mathfrak{d}(s_i, x)) \\ \mathfrak{l}_{L*}(x;\mathcal{D}_n) &:= \max_{s_i \in \mathcal{X}_n} (f_i - L^* \mathfrak{d}(s_i, x)), \end{aligned} \tag{1}$$

it holds for all $x \in \mathcal{X}$ that

$$\mathfrak{l}_{L*}(x;\mathcal{D}_n) \leqslant f(x) \leqslant \mathfrak{u}_{L*}(x;\mathcal{D}_n). \tag{2}$$

Unfortunately, in most regression problems L^* is unknown. Still, in light of this strong guarantee when L^* is known, we stick to the prior assumption that our target function f is Lipschitz-continuous to define our model. It is worth noting that this Lipschitz-continuity assumption is actually implicitly shared by many other models, including many popular neural network based models [4]. However, as opposed to such models, we are only interested in the quality of uncertainty bounds, and thus do not assume one potential function value to be more likely than another as long as they both lie inside the uncertainty bounds of our model. That is, we assume a uniform distribution between the prediction bounds of our model. With bounds of the form in Eqns. (1), this yields in the following model for inference at $x \in \mathcal{X}$:

$$f(x; L_n, \mathcal{D}_n) \sim U(\mathfrak{l}_{L_n}(x;\mathcal{D}_n), \mathfrak{u}_{L_n}(x;\mathcal{D}_n)), \tag{3}$$

where L_n acts as a hyperparameter to be learned from the data.

2.2 Uncertainty Bounds with Lazily Adapted Constants

The canonical approach to learn L_n for our model is to lazily adapt it, i.e. to set it to the smallest value that is compatible with the hypothesis that f is L_n-Lipschitz-continuous (hence LAC, Lazily Adapted Constant). This approach, which comes from Lipschitz Interpolation literature [5,6], yields

$$L_n^{\mathrm{LAC}} := \max_{s_i, s_j \in \mathcal{X}_n} \frac{\mathfrak{d}_{\mathcal{Y}}(f_i, f_j)}{\mathfrak{d}(s_i, s_j)}, \tag{4}$$

as resulting estimator of L^*, i.e. the maximum slope observable in the available training data. It can be easily shown that $\mathfrak{l}_{L_n^{\mathrm{LAC}}}(x;\mathcal{D}_n) \leqslant \mathfrak{u}_{L_n^{\mathrm{LAC}}}(x;\mathcal{D}_n)$ for all $x \in \mathcal{X}$, hence the resulting model is well-defined.

2.3 Computational Complexity and Sparse Approximation

Naturally, the computational complexity for computing L_n^{LAC} from scratch is $\mathcal{O}(n^2)$ [5]. While this natural quadratic scalability is better than the natural cubic computational cost of GPs [1], it is still prohibitive for truly large data sets. This is why we introduce a simple sparse approximation which is inspired by Strongin's estimator for the one-dimensional case [6]: after ordering all training

points such that $s_1 < s_2 < \ldots < s_n$ it can be seen from elementary analysis that $L_n^{\mathrm{LAC}} = \max_{2 \leqslant i \leqslant n} \frac{\mathfrak{d}_{\mathcal{Y}}(f_i, f_{i-1})}{\mathfrak{d}(s_i, s_{i-1})}$, which only requires $\mathcal{O}(n)$ operations. For higher dimensions, we similarly apply the idea of only inspecting the slopes between ordered points. Randomly drawing J permutations $\pi_j : \{1, \ldots, n\} \to \{1, \ldots, n\}$, we evaluate $L_n^{\mathrm{LAC}}(j) := \max_{2 \leqslant i \leqslant n} \frac{\mathfrak{d}_{\mathcal{Y}}(f_{\pi_j(i)}, f_{\pi_j(i-1)})}{\mathfrak{d}(s_{\pi_j(i)}, s_{\pi_j(i-1)})}$ and set the sparse lazily adapted constant (SLAC) to be

$$L_n^{\mathrm{SLAC}} := \max_j L_n^{\mathrm{LAC}}(j) = \max_j \max_{2 \leqslant i \leqslant n} \frac{\mathfrak{d}_{\mathcal{Y}}(f_{\pi_j(i)}, f_{\pi_j(i-1)})}{\mathfrak{d}(s_{\pi_j(i)}, s_{\pi_j(i-1)})}. \tag{5}$$

Obtaining L_n^{SLAC} is $\mathcal{O}(nJ)$, where in our experiments (Section 3) we found that the bounds for $J = 100$ are comparably conservative to those of L_n^{LAC}, which massively speeds up training for larger data sets and made L_n^{SLAC} by far the fastest method overall.

By definition, it is clear that $L_n^{\mathrm{SLAC}} \leqslant L_n^{\mathrm{LAC}}$. A problem that naturally arises with this sparse approximation for values of $L_n^{\mathrm{SLAC}} < L_n^{\mathrm{LAC}}$ is that in these cases the model might not be not well-defined everywhere. In such cases it can happen that $\mathfrak{l}_{L_n^{\mathrm{SLAC}}}(x; \mathcal{D}_n) > \mathfrak{u}_{L_n^{\mathrm{SLAC}}}(x; \mathcal{D}_n)$ for some regions in $\mathcal{X}$. Therefore we define $\tilde{\mathcal{D}}_n \subset \mathcal{D}_n$ to be the subset of training points for which $\mathfrak{l}_{L_n^{\mathrm{SLAC}}}(x; \tilde{\mathcal{D}}_n) \leqslant \mathfrak{u}_{L_n^{\mathrm{SLAC}}}(x; \tilde{\mathcal{D}}_n)$ for all $x \in \mathcal{X}$. This makes the resulting sparse model:

$$f(x; L_n^{\mathrm{SLAC}}, \mathcal{D}_n) \sim U(\mathfrak{l}_{L_n^{\mathrm{SLAC}}}(x; \tilde{\mathcal{D}}_n), \mathfrak{u}_{L_n^{\mathrm{SLAC}}}(x; \tilde{\mathcal{D}}_n)). \tag{6}$$

3 Experiments

3.1 Data Sets

Real-World Data. We tested the performance of the SLAC uncertainty bounds on 8 multidimensional real-world regression data sets of between 300 and 45000 data points, that are publicly available in the UCI machine learning repository. Each data set was randomly split into training and testing set accounting for 80% and 20% of the data respectively and centred and normalised using the training set mean and standard deviation.

Synthetic Data. Additionally, we demonstrated the robustness of the performance of the uncertainty bounds of our model, by pressure-testing it on a set of 100 synthetic functions on the interval $[-1, 1]$ with Lipschitz-constants ranging between 14.7 and 129.8. These were generated by randomising the weights of a 2-hidden-layer neural network with 2886 units in the first hidden layer and 577 units in the second hidden layer. Each function was converted into a data set of 1000 data points on an equally spaced grid over $[-1, 1]$, from which we cut out test sets of 200 data points corresponding to the intervals $[-0.94, -0.86]$, $[-0.60, -0.52]$,$[-0.42, -0.34]$,$[-0.12, -0.04]$, and $[0.20, 0.28]$.

Table 1: Characteristics of data and comparison of training times across UCI data sets. We denote with * values for which sparse GPs were used. SLAC was consistently by far the fastest method on all data sets and its advantage over LAC is most evident on the biggest data sets. Earlier stopping of optimisation during BNN training was found to result in less consistent bounds across data sets.

	Size of Data		Average Training Time in Seconds			
Data Set	Data Points	Dimensions	**GP**	**BNN**	**LAC**	**SLAC**
Boston Housing	504	13	23 (4)	724 (195)	1	**0.4 (0)**
Concrete Strength	1,030	8	30 (7)	1.488 (430)	4	**1 (0)**
Energy Efficiency	768	8	16 (3)	2,733 (1,100)	2	**1 (0)**
Kin8nm	8,192	8	177* (41)	23,879 (5,499)	242	**11 (0)**
Naval Propulsion	11,934	16	159* (117)	28,049 (6,978)	536	**22 (1)**
Power Plant	9,568	4	248* (27)	23,395 (7,390)	332	**14 (0)**
Protein Structure	45,730	9	1,295* (134)	59,689 (11,180)	14,158	**43 (3)**
Yacht Dynamics	308	6	47 (3)	982 (388)	0.4	**0.3 (0)**

3.2 Experimental Setup

Our Model. We used Euclidean distance as metric for $\mathcal{X}$ and $\mathcal{Y}$. For SLAC, J was set to 100 and training was done 10 times repeatedly on each training set to analyse its sensitivity to the random seed used for drawing the permutations π_j.

Baselines. For the GPs, we assumed Gaussian noise. We used the automatic relevance determination (ARD) versions of the squared exponential (SE), the Matern32 and the Matern52 covariance functions. Training was done by marginal likelihood maximisation as implemented in the GPflow package. For data sets with more than 2000 training points, we resorted to sparse approximations through variational inference as proposed in [7] using 250 inducing points. For the real-world data sets, we performed 10 random initialisations for each training process to inspect the stability of the prediction bounds with respect to optimisation initialisation. For the BNNs, we used three different architectures with ReLu activation functions, labelled *low*, *medium* and *high*, with respectively 2, 3 and 4 hidden layers of $1,024$ units each. We used MC dropout [2] to approximate the posterior. As the model precision τ is vital for calibrating the uncertainty bounds with MC dropout, we selected it together with the lengthscale l by maximising the log-likelihood employing the sample efficient technique of Bayesian optimisation implemented in GPyOpt. Furthermore, we set dropout to 0.1 after experimenting empirically over the training data. For both baselines, we analysed both the three standard deviation bounds ($\delta = 0.003$) and four standard deviation bounds ($\delta = 0.00006$).

Table 2: Comparison of uncertainty bounds across UCI data sets. For GPs and BNNs the uncertainty bounds are for $\delta = 0.003$. With more than 99% of test points in bounds for most data sets, SLAC bounds are about as conservative as the three standard deviation bounds of GPs and BNNs. Average distance between bounds for SLAC is larger on many data sets, but overall still comparable. On the Concrete Strength, Power Plant, and Protein Structure data set, we suspect noise in the data causes LAC bounds to be unacceptably wide. However, SLAC seems to be more robust to noise by not evaluating every possible slope.

	% Test Points in Bounds				Avg. Distance of Bounds			
Data Set	**GP**	**BNN**	**LAC**	**SLAC**	**GP**	**BNN**	**LAC**	**SLAC**
Boston Housing	97.1 (0.0)	98.0 (0.5)	100	100 (0.0)	1.5 (0.0)	2.3 (1.1)	5.7	5.2 (0.4)
Concrete Strength	96.6 (33.6)	99.5 (0.2)	97.6	98.2 (0.8)	1.6 (0.6)	2.5 (1.5)	20.4	15.3 (3.4)
Energy Efficiency	98.7 (0.5)	100 (0.0)	100	99.5 (1.2)	0.2 (69.1)	4.5 (1.9)	2.5	2.5 (0.1)
Kin8nm	99.5 (0.2)	100 (0.1)	99.9	99.6 (0.2)	2.2 (1.1)	2.5 (3.5)	4.7	3.6 (0.5)
Naval Propulsion	100 (0.0)	100 (0.0)	100	99.4 (0.4)	5.9 (69.1)	5.1 (1.1)	1.7	1.3 (0.3)
Power Plant	99.4 (0.1)	100 (0.1)	100	99.4 (0.6)	1.7 (1.1)	8.2 (2.4)	14.6	4.5 (3.5)
Protein Structure	100 (0.1)	95.8 (0.8)	99.8	99.5 (0.3)	4.8 (0.1)	2.6 (0.2)	60.2	8.4 (2.0)
Yacht Dynamics	98.4 (33.0)	100 (0.0)	98.4	97.7 (0.8)	0.1 (5583.7)	2.2 (0.1)	2.3	2.3 (0.1)

Evaluation Metrics. In order to assess the quality of the uncertainty bounds, we apply two criteria. Firstly, we assess whether the bounds are conservative enough. To evaluate this we computed the ratio of test points inside the bounds, $\frac{|\{t_j \in T_q | l(t_j) \leqslant f(t_j) \leqslant u(t_j)\}|}{|T_q|}$ and compare this to $1 - \delta$. For the synthetic data, we additionally analysed the number of functions that are entirely inside the uncertainty bounds of a model over the test set intervals. Secondly, we assess how tight the bounds are. Whilst from a risk analysis perspective, conservativeness is most important, it is obvious that useful uncertainty bounds should be as close together as possible while achieving the desired conservativeness. We thus computed the average distance of the bounds on the test set, $\frac{\sum_{t_j \in T_q} |u(t_j) - l(t_j)|}{|T_q|}$, and prefer smaller average distance conservative bounds.

4 Results and Discussion

The results of our experiments are shown in Tables 1 - 3. For brevity, we only show the three standard deviation bounds ($\delta = 0.003$) for GPs and BNNs. The four standard deviation bounds ($\delta = 0.00006$), omitted here, had identical training times and were 33% wider, which resulted in $99\% - 100\%$ of test points being inside these bounds for all data sets. In summary, we found across both types of data that training our sparse model to calculate the SLAC bounds was magnitudes faster than training sophisticated models like GPs or BNNs (Table 1). The resulting SLAC bounds were comparable in performance to the three and four standard deviation bounds of GPs and BNNs, especially in terms of conservativeness, which seemed to be a reliable feature of SLAC bounds (Tables 2, 3). Their average distance was roughly the same as for BNNs and usually

Table 3: Comparison of uncertainty bounds across 100 random functions. For GPs and BNNs the bounds are for $\delta = 0.003$. In the one-dimensional case, LAC = SLAC as the training inputs can be ordered by value. Again, training for SLAC is by far the fastest and yields bounds which are comparable to the three standard deviation bounds of GPs and BNNs in terms of conservativeness and tightness.

	Training Time	**Bound Performance**		
Method	Avg. Time in s	% Test Points in Bounds	Avg. Distance of Bounds	Functions in Bounds
GP^{SE}	29.5	98.6 (2.9)	1.96 (0.27)	73
$GP^{Matern32}$	28.6	90.3 (7.3)	0.30 (0.12)	13
$GP^{Matern52}$	29.2	97.5 (4.0)	0.50 (0.20)	56
$L^{LAC/SLAC}$	**0.03**	99.8 (0.8)	1.84 (0.92)	92
BNN^{low}	4.6	98.5 (5.2)	8.98 (8.17)	88
BNN^{medium}	11.6	99.7 (1.5)	5.80 (6.59)	93
BNN^{high}	17.8	100.0 (0.0)	18.98 (0.00)	93

about twice as large as for GPs. However, noisy data caused this distance to be unnecessarily large. Nevertheless, SLAC was able to cope with such data better than LAC due to its sparsity (Table 2), as not every slope in the training data was taken into account.

Overall this is still early work and there are a number of topics which we plan to investigate in the future. In particular, we desire to investigate the effect of J more rigourously. It seems beneficial to find theoretical guarantees for how big it needs to be for a given desired conservativeness and also to analyse its choice to make the SLAC bounds more robust to noise in the data. However, our results already provide empirical evidence that SLAC bounds with $J = 100$ can be a fast, stable and reliable alternative to three to four standard deviation bounds of GPs and BNNs.

References

1. Rasmussen, C.E.: Gaussian Processes for Machine Learning (2006)
2. Gal, Y., Ghahramani, Z.: Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In: International conference on machine learning. pp. 1050–1059 (2016)
3. Sukharev, A.G.: Optimal search of a root of a function that satisfies a Lipschitz condition. In: Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki. (1976)
4. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014)
5. Calliess, J.P.: Lazily adapted constant kinky inference for nonparametric regression and model-reference adaptive control. ArXiv preprint arXiv:1701.00178 (2016)

6. Strongin, R.G.: On the convergence of an algorithm for finding a global extremum. Engineering in Cybernetics (1973)
7. Titsias, M.: Variational learning of inducing variables in sparse Gaussian processes. In: Artificial Intelligence and Statistics. pp. 567–574 (2009)