

Abstract

The computational and storage complexity of kernel machines presents the primary barrier to their scaling to large, modern, datasets. A common way to tackle the scalability issue is to use the conjugate gradient algorithm, which relieves the constraints on both storage (the kernel matrix need not be stored) and computation (both stochastic gradients and parallelization can be used). Even so, conjugate gradient is not without its own issues: the conditioning of kernel matrices is often such that conjugate gradients will have poor convergence in practice. Preconditioning is a common approach to alleviating this issue. Here we propose preconditioned conjugate gradients for kernel machines, and develop a broad range of preconditioners particularly useful for kernel matrices. We describe a scalable approach to both solving kernel machines and learning their hyperparameters. We show this approach is exact in the limit of iterations and outperforms state-of-the-art approximations for a given computational budget.

1. Introduction

Kernel machines, in enabling flexible feature space representations of data, comprise a broad and important class of tools throughout machine learning and statistics; prominent examples include support vector machines (Schölkopf & Smola, 2001) and Gaussian processes (GPs) (Rasmussen & Williams, 2006). At the core of most kernel machines is the need to solve linear systems involving the Gram matrix $K = \{k(x_i, x_j \mid \theta)\}_{i,j=1,...,n}$, where the kernel function $k$, parameterized by $\theta$, implicitly specifies the feature space representation of data points $x_i$. Because $K$ grows with the number of data points $n$, a fundamental computational bottleneck exists: storing $K$ is $\mathcal{O}(n^2)$, and solving a linear system with $K$ is $\mathcal{O}(n^3)$. As the need for large-scale kernel machines grows, much work has been directed towards this scaling issue.

Standard approaches to kernel machines involve a factorization (typically Cholesky) of $K$, which is efficient and exact but maintains the quadratic storage and cubic runtime costs. This cost is particularly acute when adapting (or learning) hyperparameters $\theta$ of the kernel function, as $K$ must then be factorized afresh for each $\theta$. To alleviate this burden, numerous works have turned to approximate methods (Candela & Rasmussen, 2005; Snelson & Ghahramani, 2007; Rahimi & Recht, 2008) or methods that exploit structure in the kernel (Gilboa et al., 2015). Approximate methods can achieve attractive scaling, often through the use of low-rank approximations to $K$, but they can incur a potentially severe loss of accuracy. An alternative to factorization is found in the conjugate gradient method (CG), which is used to directly solve linear systems via a sequence of matrix-vector products. Any kernel structure can then be exploited to enable fast multiplications, driving similarly attractive runtime improvements, and eliminating the storage burden (neither $K$ nor its factors need be represented in memory). Unfortunately, in the absence of special structure that accelerates multiplications, CG performs no better than $\mathcal{O}(n^3)$ in the worst case, and in practice finite numerical precision often results in a degradation of run-
time performance compared to factorization approaches.

Throughout optimization, the typical approach to the slow convergence of CG is to apply preconditioners to improve the geometry of the linear system being solved (Golub & Van Loan, 1996). While preconditioning has been explored in domains such as spatial statistics (Chen, 2005; Stein et al., 2012; Ashby & Falgout, 1996), the application of preconditioning to kernel matrices in machine learning has received little attention. Here we design and study preconditioned conjugate gradient methods (PCG) for use in kernel machines, and provide a full exploration of the use of approximations of $K$ as preconditioners.

Our contributions are as follows. (i) Extending the work in (Davies, 2014), we apply a broad range of kernel matrix approximations as preconditioners. Interestingly, this step allows us to exploit the important developments of approximate kernel machines to accelerate the exact computation that PCG offers. (ii) As a motivating example used throughout the paper, we analyze and provide a general framework to both learn kernel parameters and make predictions in GPs. (iii) We extend stochastic gradient learning for GPs (Filippone & Engler, 2015; Anitescu et al., 2012) to allow any likelihood that factorizes over the data points by developing an unbiased estimate of the gradient of the approximate log-marginal likelihood. We demonstrate this contribution in making the first use of PCG for GP classification. (iv) We evaluate datasets over a range of problem size and dimensionality. Because PCG is exact in the limit of iterations (unlike approximate techniques), we demonstrate a tradeoff between accuracy and computational effort that improves beyond state-of-the-art approximation and factorization approaches.

In all, we show that PCG, with a thoughtful choice of preconditioner, is a competitive strategy which is possibly even superior than existing approximation and CG-based techniques for solving general kernel machines.

2. Motivating example – Gaussian Processes

Gaussian processes (GPs) are the fundamental building block of many probabilistic kernel machines that can be applied in a large variety of modeling scenarios (Rasmussen & Williams, 2006). Throughout the paper, we will denote by $X = \{x_1, \ldots, x_n\}$ a set of $n$ input vectors and use $y = \{y_1, \ldots, y_n\}^\top$ for the corresponding labels. GPs are formally defined as collections of random variables characterized by the property that any finite number of them is jointly Gaussian distributed. The specification of a kernel function determines the covariance structure of such random variables

$$
cov(f(x), f(x')) = k(x, x' | \theta).
$$

In this work we focus in particular on the popular Radial Basis Function (RBF) kernel

$$
k(x_i, x_j | \theta) = \sigma^2 \exp\left[\frac{-1}{2} \sum_{r=1}^{d} \frac{(x_i - x_j)^2}{l^2_r}\right],
$$

where $\theta$ represents the collection of the kernel parameters $\sigma^2$ and $l^2_r$. Defining $f_i = f(x_i)$ and $f = (f_1, \ldots, f_n)^\top$, and assuming a zero mean GP, we have

$$
f \sim \mathcal{N}(f | 0, K),
$$

where $K$ is the $n \times n$ Gram matrix with elements $K_{ij} = k(x_i, x_j | \theta)$. Note that the kernel above and many popular kernels in machine learning give rise to dense kernel matrices. Observations are then modeled through a transformation $h$ of a set of GP-distributed latent variables, specifying the model

$$
y_i \sim p(y_i | h(f_i)), \quad f \sim \mathcal{N}(f | 0, K).
$$

2.1. The need for preconditioning

The success of nonparametric models based on kernels hinges on the adaptation of kernel parameters $\theta$. The motivation for preconditioning begins with an inspection of the log-marginal likelihood of GP models with prior $\mathcal{N}(f | 0, K)$. In Gaussian processes with a Gaussian likelihood $y_i \sim \mathcal{N}(y_i | f_i, \lambda)$, we have analytic forms for

$$
\log[p(y | \theta, X)] = -\frac{1}{2} \log(|K_x|) - \frac{1}{2} y^\top K_x^{-1} y + \text{const},
$$

and its derivatives with respect to kernel parameters $\theta_i$,

$$
g_i = -\frac{1}{2} \text{Tr}\left(K_x^{-1} \frac{\partial K_x}{\partial \theta_i}\right) + \frac{1}{2} y^\top K_x^{-1} \frac{\partial K_x}{\partial \theta_i} K_x^{-1} y.
$$

where $K_x = K + \lambda I$. The traditional approach involves factorizing the kernel matrix $K_x = LL^\top$ using the Cholesky algorithm (Golub & Van Loan, 1996) which costs $O(n^3)$ operations. After that, all other operations cost $O(n^2)$ except for the trace term in the calculation of $g_i$ which once again requires $O(n^3)$ operations. Similar computations are required for computing mean and variance predictions for test data (Rasmussen & Williams, 2006). Note that the solution of a linear system is required for computing the variance at every test point.

This approach is not viable for large $n$ and, consequently, many approaches have been proposed to approximate these computations, thus leading to approximate optimal values for $\theta$ and approximate predictions. Here we investigate the
possibility of avoiding approximations altogether, by arguing that for parameter optimization it is sufficient to obtain an unbiased estimate of the gradient $g_i$. In particular, when such an estimate is available, it is possible to employ stochastic gradient optimization that has strong theoretical guarantees (Robbins & Monro, 1951). In the case of GPs, the problematic terms in eq. 2 are the solution of the linear system $K_y^{-1}y$ and the trace term. In this work we make use of a stochastic linear algebra result that allows for an approximation of the trace term,

$$\text{Tr} \left( K_y^{-1} \frac{\partial K_y}{\partial \theta_i} \right) \approx \frac{1}{N_r} \sum_{i=1}^{N_r} r^{(i)^T} K_y^{-1} \frac{\partial K_y}{\partial \theta_i} r^{(i)},$$

where the $N_r$ vectors $r^{(i)}$ have components drawn from $\{-1,1\}$ with probability $1/2$. Verifying that this is an unbiased estimate of the trace term is straightforward considering that $E \left( r^{(i)^T} r^{(i)} \right) = I$ (Gibbs, 1997).

This result shows that all it takes to calculate stochastic gradients is the ability to efficiently solve linear systems. Linear systems can be iteratively solved using conjugate gradient (CG) (Golub & Van Loan, 1996). The advantage of this formulation is that we can attempt to optimize kernel parameters using stochastic gradient optimization without having to store $K_y$ and, given that the most expensive operation is now multiplying the kernel matrix by vectors, only $O(n^2)$ computations are required. However, it is well known that the convergence of the CG algorithm depends on the condition number $\kappa(K_y)$ (ratio of largest to smallest eigenvalues), so the suitability of this approach may also be curtailed if $K_y$ is badly conditioned. To this end, a well-known approach for improving the conditioning of a matrix, which in turn accelerates convergence, is preconditioning. This necessitates the introduction of a preconditioning matrix, $P$, which should be chosen in such a way that $P^{-1} K_y$ approximates the identity matrix, $I$. Intuitively, this can be obtained by setting $P = K_y$; however, given that in Preconditioned CG (PCG) we are required to solve linear systems involving $P$, this choice would be no easier than solving the original system. Thus we must choose $P$ which approximates $K_y$ as closely as possible, but which can also be easily inverted. The PCG algorithm is shown in Algorithm 1.

### Algorithm 1 The Preconditioned CG Algorithm, adapted from (Golub & Van Loan, 1996)

**Require:** data $X$, vector $v$, convergence threshold $\epsilon$, initial vector $x_0$, maximum no. of iterations $T$

$$r_0 = v - K_y x_0; \quad z_0 = P^{-1} r_0; \quad p_0 = z_0$$

**for** $i = 0 : T$ **do**

$$\alpha_i = \frac{r_i^T z_i}{r_i^T K_y z_i}$$

$$x_{i+1} = x_i + \alpha_i p_i$$

$$r_{i+1} = r_i + \alpha_i K_y p_i$$

**if** $\|r_{i+1}\| < \epsilon$ **then**

return $x = x_{i+1}$

**end if**

$$z_{i+1} = P^{-1} r_{i+1}$$

$$\beta_i = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$$

$$p_{i+1} = p_{i+1} + \beta_i p_i$$

**end for**

### 2.2. Non-Gaussian Likelihoods

When the likelihood $p(y_i \mid f_i)$ is not Gaussian, it is no longer possible to analytically integrate out latent variables. Instead, techniques such as Gaussian approximations (see, e.g., (Kuss & Rasmussen, 2005; Nickisch & Rasmussen, 2008) and methods attempting to characterize the full posterior $p(f, \theta \mid y)$ (Murray et al., 2010; Filippone et al., 2013) may be required. Among the various schemes to recover tractability in the case of models with a non-Gaussian likelihood, we choose the Laplace approximation, as we can formulate it in a way that only requires the solution of linear systems. The GP models we consider assume that the likelihood factorizes across all data points $p(y_i \mid f) = \prod_{i=1}^n p(y_i \mid f_i)$. The use of CG for computing the Laplace approximation has been proposed elsewhere (Flaxman et al., 2015), but we make the first use of preconditioning and stochastic gradient estimation within the Laplace approximation to compute stochastic gradients for non-conjugate models.

Defining $W = -\nabla_f \nabla_f \log[p(y \mid f)]$ (a diagonal matrix), carrying out the Laplace approximation algorithm, computing its derivatives wrt $\theta$, and making predictions, all possess the same computational bottleneck: the solution of linear systems involving the matrix $B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}}$ (Rasmussen & Williams, 2006). For a given $\theta$, each iteration of the Laplace approximation algorithm requires solving one linear system involving $B$ and two matrix-vector multiplications involving $K$; the linear system involving $B$ can be solved using CG or PCG. The Laplace approximation yields the mode $f$ of the posterior over latent variables and offers an approximate log-marginal likelihood in the form:

$$\log[p(y \mid \theta, X)] = -\frac{1}{2} \log |B| - \frac{1}{2} f^T K^{-1} f + \log[p(y \mid \hat{f})]$$

which poses the same computational challenges as the regression case. Once again, we therefore seek an alternative way to learn kernel parameters by stochastic gradient optimization based on computing unbiased estimates of the gradient of the approximate log-marginal likelihood. This is complicated further by the inclusion of an additional "implicit" term accounting for the change in the solution given by the Laplace approximation for a change in $\theta$. The full derivation of the gradient is rather lengthy and is deferred to the supplementary material. Nonetheless, it is worth not-
3. Preconditioning Kernel Matrices

Here we consider choices for kernel preconditioners, and for the sake of clarity we focus on preconditioners for \( K_y \). Unless stated otherwise, we shall consider standard left preconditioning, whereby the original problem of solving \( K_y z = \nu \) is transformed by applying a preconditioner, \( P \), to both sides of this equation. This formulation may thus be expressed as \( P^{-1} K_y z = P^{-1} \nu \).

3.1. Nyström type approximations

The Nyström method was originally proposed to approximate the eigendecomposition of kernel matrices (Williams & Seeger, 2000); as a result, it offers a way to obtain a low rank approximation of \( K \). This method selects a subset of \( m \ll n \) data (inducing points) collected in the set \( U \) which are intended for approximating the spectrum of \( K \). The resulting approximation is \( \hat{K} = K_{UX} K_{UU}^{-1} K_{UX} \) where \( K_{UU} \) denotes the evaluation of the kernel function over the inducing points, and \( K_{UX} \) denotes the evaluation of the kernel function between the input points and the inducing points. The resulting preconditioner \( P = K_{UX} K_{UU}^{-1} K_{UX} + \lambda I \) can be inverted using the matrix inversion lemma

\[
P^{-1} = \lambda^{-1} \left[ I - K_{UX} \left( K_{UU} + K_{UX} K_{UX} \right)^{-1} K_{UX} \right] \nu,
\]

which has \( O(m^3) \) complexity.

3.1.1. Fully and Partially Independent Training Conditional

The use of a subset of data for approximating a GP kernel has also been utilized in the fully and partially independent training conditional approaches (FITC and PITC, respectively) for approximating GP regression (Candela & Ras- mussen, 2005). In the former case, the prior covariance of the approximation can be written as follows:

\[
P = K_{UX} K_{UU}^{-1} K_{UX} + \text{diag} \left( K - K_{UX} K_{UU}^{-1} K_{UX} \right) + \lambda I.
\]

As the name implies, this formulation enforces that the latent variables associated with \( U \) are taken to be completely conditionally independent. On the other hand, the PITC method extends on this approach by enforcing that although inducing points assigned to a designated block are conditionally dependent on each other, there is no dependence between points placed in different blocks:

\[
P = K_{UX} K_{UU}^{-1} K_{UX} + \text{bldiag} \left( K - K_{UX} K_{UU}^{-1} K_{UX} \right) + \lambda I.
\]

For the FITC preconditioner, the diagonal resulting from the training conditional can be added to the diagonal noise matrix, and the inversion lemma can be invoked as for the Nyström case. Meanwhile, for the PITC preconditioner, the noise diagonal can be added to the block diagonal matrix, which can then be inverted block-by-block. Once again, matrix inversion can then be carried out as before, where the inverted block diagonal matrix takes the place of \( \lambda I \) in the original formulation.

3.2. Approximate factorization of kernel matrices

This group of preconditioners relies on approximations to \( K \) that factorize as \( \hat{K} = \Phi \Phi^\top \). We shall consider different ways of determining \( \Phi \) such that \( P \) can be inverted at a lower cost than the original kernel matrix \( K \). Once again, this enables us to employ the matrix inversion lemma, and express the linear system:

\[
P^{-1} \nu = \left( \Phi \Phi^\top + \lambda I \right)^{-1} \nu = \lambda^{-1} \left[ I - \Phi \left( I + \Phi^\top \Phi \right)^{-1} \Phi^\top \right] \nu.
\]

We now review a few methods to approximate the kernel matrix \( K \) in the form \( \Phi \Phi^\top \).

3.2.1. SPECTRAL APPROXIMATION

The spectral approach uses random Fourier features for deriving a sparse approximation of a GP (Rahimi & Recht, 2008). This approach for GPs was introduced in (Lázaro-Gredilla et al., 2010), and relies on the assumption that stationary kernel functions can be represented as the Fourier transform of non-negative measures. As such, the elements of \( K \) can be approximated as follows:

\[
\hat{K}_{ij} = \frac{\sigma^2}{m} \sum_{r=1}^{m} \cos \left( 2\pi s_r^\top (x_i - x_j) \right).
\]

In the equation above, the vectors \( s_r \) denote the spectral points (or frequencies) which in the case of the RBF kernel can be sampled from \( \mathcal{N} \left( 0, \frac{1}{\Lambda^2} \right) \), where \( \Lambda = \left[ 1/\ell_1^2, \ldots, 1/\ell_n^2 \right] \). To the best of our knowledge, this is the first time such an approximation has been considered for the purpose of preconditioning kernel matrices.

3.2.2. PARTIAL SVD

Another factorization approach that we consider in this work is the partial singular value decomposition (SVD) method (Golub & Van Loan, 1996). The SVD method factorizes the original kernel matrix \( K \) into \( \Lambda A \Lambda^\top \), where \( A \) is a unitary matrix and \( \Lambda \) is a diagonal matrix of singular values. Here, we shall consider a variation of this technique called randomized truncated SVD (Halko et al., 2011), which constructs an approximate low rank SVD factorization of \( K \) using random sampling to accelerate computations.
3.2.3. Structured Kernel Interpolation (SKI)

Some recent work on approximating GPs has exploited the fast computation of Kronecker matrix-vector multiplications when inputs are located on a Cartesian grid (Gilboa et al., 2015). Unfortunately, not all datasets meet this requirement, thus limiting the widespread application of Kronecker inference. To this end, SKI (Wilson & Nickisch, 2015) is an approximation technique which exploits the benefits of the Kronecker product without imposing any requirements on the structure of the training data. In particular, a grid of inducing points, \( U \), is constructed, and the covariance between the training data and \( U \) is then represented as \( K_{XU} = WK_{UU} \). In this formulation, \( W \) denotes a sparse interpolation matrix for assigning weights to the elements of \( K_{UU} \). In this manner, a preconditioner exploiting Kronecker structure can be constructed as \( P = WK_{UU}W^\top + \lambda I \). If we consider \( V = W/\sqrt{\lambda} \), we can rewrite the (inverse) preconditioner as \( P^{-1} = \lambda^{-1}(V K_{UU}V^\top + I)^{-1} \). Since this can no longer be solved directly, we solve this (inner-loop) linear system using the CG algorithm (all within one iteration of the outer-loop PCG). For badly conditioned systems, although the computational requirements on the structure of the training data. In particular, we evaluate the convergence in solving \( K_yz = y \) using iterative methods, where \( y \) denotes the labels of the designated dataset, and \( K_y \) is constructed using different configurations of kernel parameters.

With this experiment, we aim to assess the quality of different preconditioners based on how many matrix-vector products they require, which, for most approaches, corresponds to the number of iterations taken by PCG to converge. The convergence threshold is set to \( \epsilon^2 = n \cdot 10^{-10} \) so as to roughly accept an average error of \( 10^{-5} \) on each element of the solution.

For every variation, we set the parameters of the preconditioners so as to have a complexity lower than \( \mathcal{O}(n^2) \) cost associated with matrix-vector products; by doing so, we can assume that the latter computations are the dominant cost for large \( n \). In particular, for Nyström-type methods, we set \( m = \sqrt{n} \) inducing points, so that when we invert the preconditioner using the matrix inversion lemma, the cost is in \( \mathcal{O}(m^3) = \mathcal{O}(n^{3/2}) \). Similarly, for the Spectral preconditioner, we set \( m = \sqrt{n} \) random features. For the SKI preconditioner, we take an equal number of elements on the grid for each dimension; under this assumption, Kronecker products have \( \mathcal{O}(dn^{d+1}) \) cost (Gilboa et al., 2015), and we set the size of the grid so that the complexity of applying the preconditioner matches \( \mathcal{O}(n^{3/2}) \), so as to be consistent with the other preconditioners. For the Regularized approach, each iteration needed to apply the precondi-
Preconditioning Kernel Matrices

<table>
<thead>
<tr>
<th>Concrete dataset</th>
<th>Power plant dataset</th>
<th>Protein dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Concrete dataset" /></td>
<td><img src="image" alt="Power plant dataset" /></td>
<td><img src="image" alt="Protein dataset" /></td>
</tr>
</tbody>
</table>

Figure 1. Comparison of preconditioners for different settings of kernel parameters. The lengthscales $l$ and the noise variance $\lambda$ are shown on the x and y axes respectively. The top figure indicates the number of iterations required to solve the corresponding linear system using CG, whilst the bottom part of the figure shows the rate of improvement (negative - blue) or degradation (positive - red) achieved by using PCG to solve the same linear system. Parameters and results are reported in $\log_{10}$. Symbols added to facilitate reading in B/W print.

We focus on an isotropic RBF variant of the kernel in eq. 1, fixing the marginal variance $\sigma^2$ to one. We vary the lengthscales $l$ and the noise variance $\lambda$ in $\log_{10}$ scale. The top part of fig. 1 shows the number of iterations that the standard CG algorithm takes, where we have capped the number of iterations to 100,000.

The bottom part of the figure reports the improvement offered by various preconditioners measured as

$$\log_{10}\left(\frac{\# \text{ PCG iterations}}{\# \text{ CG iterations}}\right).$$

It is worth noting that when both CG and PCG fail to converge within the upper bound, the improvement will be marked as 0, i.e. neither a gain or a loss within the given bound. The results plotted in fig. 1 indicate that the low-rank preconditioners (PITC, FITC and Nyström) achieve significant reductions in the number of iterations for each dataset, and all approaches work best when the lengthscales are longer, characterising smoother processes. In contrast, preconditioning seems to be less effective when the lengthscale is shorter, corresponding to a kernel matrix that is more sparse. However, for cases yielding positive results, the improvement is often in the range of an order of magnitude, which can be substantial when a large number of iterations is required by the CG algorithm.

The results also confirm that, as alluded to in the previous section, Block Jacobi preconditioning is generally a poor preconditioner, particularly when the corresponding kernel matrix is dense. The only minor improvements were observed when CG itself converges quickly, in which case preconditioning serves very little purpose either way.

The regularization approach with flexible conjugate gradient does not appear to be effective in any case either, particularly due to the substantial amount of iterations required for solving an inner system at every iteration of the PCG algorithm. This implies that introducing additional small jitter to the diagonal does not necessarily make the system much easier to solve, whilst adding an overly large offset would negatively impact convergence of the outer algorithm. One could assume that tuning the value of this...
parameter could result in slightly better results; however, preliminary experiments in this regard yielded only minor improvements.

The results for SKI preconditioning are similarly discouraging at face value. When the matrix $K_y$ is very badly conditioned, an excessive number of inner iterations are required for every iteration of outer PCG. This greatly increases the duration of solving such systems, and as a result, this method was not included in the comparison for the Protein dataset, where it was evident that preconditioning the matrix in this manner would not yield satisfactory improvements. Notwithstanding that these experiments depict a negative view of SKI preconditioning, it must be said that we assumed a fairly simplistic interpolation procedure in our experiments, where each data point was mapped to nearest grid location. The size of the constructed grid is also hindered considerably by the constraint imposed by our upper bound on complexity. Conversely, more sophisticated interpolation strategies or even grid formulation procedures could possibly speed up the convergence of CG for the inner systems. In line with this thought, however, one could argue that the preconditioner would no longer be straightforward to construct, which goes against our innate preference towards easily derived preconditioners.

5. Impact of preconditioning on GP learning

One of the primary objectives of this work is to reformulate GP regression and classification in such a way that preconditioning can be effectively exploited. In section 2, we demonstrated how preconditioning can indeed be applied to GP regression problems, and also proposed a novel way of rewriting GP classification in terms of solving linear systems (where preconditioning can thus be employed). We can now evaluate how the proposed preconditioned GP techniques compare to other state of the art methods.

To this end, in this section, we empirically report on the generalization ability of GPs as a function of the time taken to optimize parameters $\theta$ and compute predictions. In particular, for each of the methods featured in our comparison, we iteratively run the optimization of kernel parameters for a few iterations and predict on unseen data, and assess how prediction accuracy varies over time for different methods.

The analysis provided in this report is inspired by (Chalupka et al., 2013), although we do not propose an approximate method to learn GP kernel parameters. Instead, we put forward a means of accelerating the optimization of kernel parameters without any approximation\(^2\). Given the predictive mean and variance for the $N_{\text{test}}$ test points, say $m_{s_i}$ and $s^2_{s_i}$, we report two error measures, namely the Root Mean Square Error, $\text{RMSE} = \sqrt{\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} (m_{s_i} - y_{s_i})^2}$, along with the negative log-likelihood on the test data, $- \sum_{i=1}^{N_{\text{test}}} \log[p(y_{s_i} | m_{s_i}, s^2_{s_i})]$, where $y_{s_i}$ denotes the label of the $i$th of $N_{\text{test}}$ data points. For classification, instead of the RMSE we report the error rate of the classifier.

We can make use of stochastic gradients for GP models to optimize kernel parameters using off-the-shelf stochastic gradient optimization algorithms. In order to reduce the number of parameters to tune, we employ ADAGRAD (Duchi et al., 2011) – an optimization algorithm having a single step-size parameter. For the purpose of this experiment, we do not attempt to optimize this parameter, since this would require additional computations. Nonetheless, our experience with training GP models indicates that the choice of this parameter is not critical: we set the step-size to one.

Fig. 2 shows the two error measures over time for a selection of approaches. In the figure, PCG and CG refer to stochastic gradient optimization of kernel parameters using ADAGRAD, where linear systems are solved with PCG and CG, respectively. In view of the results obtained in our comparison of preconditioners, we decide to proceed with the Nyström preconditioning method. Furthermore, we construct the preconditioner with $n = 4\sqrt{n}$ points randomly selected from the input data at each iteration, such that the overall complexity of the PCG method matches plain CG. For these methods, stochastic estimates of trace terms are carried out using $N_r = 4$ random vectors. The baseline CHOL method refers to the optimization of kernel parameters using the L-BFGS algorithm, where the exact log-marginal likelihood and its gradient are calculated using the full Cholesky decomposition of $K_y$ or $B$.

Alongside these approaches for optimizing kernel parameters without approximation, we also evaluate the performance of approximate GP methods. For this experiment, we chose to compare against approximations found in the software package GPstuff (Vanhatalo et al., 2013), namely the fully and partial independent training conditional approaches (FITC, PITC), and the sparse variational GP (VAR) (Titsias, 2009). In order to match the computational cost of CG/PCG, which is in $O(n^2)$, we set the number of inducing points for the approximate methods to be $n^{2/3}$.

All methods are initialized from the same set of kernel parameters, and the curves are averaged over 5 folds (3 for the Protein and EEG datasets). For the sake of integrity, we ran each method in the comparison individually on a workstation with Intel Xeon E5-2630 CPU having 16 cores and 128GB RAM. We also ensured that all methods reported

\(^2\)The one proviso to this statement is that, for non-Gaussian likelihood, stochastic gradients target the approximate log-marginal likelihood obtained by the Laplace approximation.
Preconditioning Kernel Matrices

Figure 2. RMSE and negative log of the likelihood on $\sqrt{n}$ held out test data over time. GP models employ the ARD kernel in eq. 1. GP classification: Spam dataset ($n = 4601, d = 57$) and EEG dataset ($n = 14979, d = 14$). GP regression: Power Plant dataset ($n = 9568, d = 4$) and Protein dataset ($n = 45730, d = 9$). Curves are averaged over multiple repetitions.

6. Discussion and Conclusions

Careful attention to numerical properties is essential in scaling machine learning to large and realistic datasets. Here we have introduced the use of preconditioning to the implementation of kernel machines, specifically, prediction and learning of kernel parameters for GPs. Our novel scheme permits the use of any likelihood that factorizes over the data points, allowing us to tackle both regression and classification. We have shown robust performance improvements, in both accuracy and computational cost, over a host of state-of-the-art approximation methods for kernel machines. Notably, our method is exact in the limit of iterations, unlike approximate alternatives. We have also shown that the use of PCG is competitive with exact Cholesky decomposition in modestly sized datasets, when the Cholesky factors can be feasibly computed. When data and thus the kernel matrix grow large enough, Cholesky factorization becomes unfeasible, leaving PCG as the optimal choice.

One of the key features of a PCG implementation is that it does not require storage of any $O(n^2)$ objects. We plan to extend our implementation to compute the elements of $K$ on the fly in one case, and in another case store $K$ in a distributed fashion (e.g. in TensorFlow/Spark). Furthermore, while we have focused on solving linear systems, we can also use preconditioning for other iterative algorithms involving the $K$ matrix, e.g., those to solve $\log(K)v$ and $K^{1/2}v$ (Chen et al., 2011), as is often useful in estimating marginal likelihoods for probabilistic kernel models like GPs.
Acknowledgements

KC and MF are grateful to Pietro Michiardi and Daniele Venzano for assisting the completion of this work by providing computational resources for running the experiments. JPC acknowledges support from the Sloan Foundation, The Simons Foundation (SCGB#325171 and SCGB#325233), and The Grossman Center at Columbia University.

References


Preconditioning Kernel Matrices


