Kernel Conjugate Gradient for Fast Kernel Machines

Nathan D. Ratliff, and J. Andrew Bagnell
Robotics Institute, Carnegie Mellon University,
Pittsburgh, PA. 15213 USA
{ndr,dbagnell}@ri.cmu.edu

Abstract

We propose a novel variant of the conjugate gradient algorithm, Kernel Conjugate Gradient (KCG), designed to speed up learning for kernel machines with differentiable loss functions. This approach leads to a better conditioned optimization problem during learning. We establish an upper bound on the number of iterations for KCG that indicates it should require less than the *square root* of the number of iterations that standard conjugate gradient requires. In practice, for various differentiable kernel learning problems, we find KCG consistently, and significantly, outperforms existing techniques. The algorithm is simple to implement, requires no more computation per iteration than standard approaches, and is well motivated by Reproducing Kernel Hilbert Space (RKHS) theory. We further show that data-structure techniques recently used to speed up kernel machine approaches are well matched to the algorithm by reducing the dominant costs of training: function evaluation and RKHS inner product computation.

1 Introduction

Kernel methods, in their various incarnations (e.g. Gaussian Processes (GPs), Support Vector Machines (SVMs), Kernel Logistic Regression (KLR)) have recently become a preferred approach to non-parametric machine learning and statistics. They enjoy this status because of their conceptual clarity, strong empirical performance, and theoretical foundations.

The primary drawback to kernel methods is their computational complexity. GPs require the inversion of an $n \times n$ (covariance/kernel) matrix, implying a running time of $O(n^3)$, where n is the size of the training set. SVMs require similar computation to solve the convex program, although intense research has gone into fast, specialized approximations [Schölkopf & Smola, 2002].

State-of-the-art approaches to kernel learning revolve largely around two techniques: iterative optimization algorithms, and learning by representing the solution with only a subset of the original data points. Our algorithm applies most directly to the former line of research, although we address the latter in the conclusions.

We propose a novel variant of the conjugate gradient algorithm, Kernel Conjugate Gradient (KCG), designed to speed up learning for kernel machines with differentiable loss functions (e.g. Gaussian Process mean inference, Kernel Logistic Regression). This algorithm is motivated by the understanding that all gradient-based methods rely, at least implicitly, on a particular metric or inner product [Schölkopf & Smola, 2002]. It is natural in kernel learning problems for the algorithm to inherit the metric on functions that the kernel provides. In Section 2.4 we show that such an approach can be interpreted as a Riemannian metric method similar to Amari's Natural Gradient [Amari & Nagaoka, 2000], although the kernel metric is much less expensive to compute.

In Section 5, we establish an upper bound on the number of iterations for KCG that indicates it should require fewer than the square root of the number of iterations that standard conjugate gradient requires. The algorithm is simple to implement, requires no more computation per iteration than standard approaches, and is well motivated by Reproducing Kernel Hilbert Space (RKHS) theory. In practice, for various differentiable kernel learning problems, we find KCG consistently, and significantly, outperforms existing techniques.

Recent research has demonstrated the benefits of space-partitioning data-structures to speed up certain kernel machines [Shen et al., 2006; Gray & Moore, 2001]. We show that these techniques work well with KCG as they reduce the dominant computational burdens in training: RKHS function evaluations and inner product computations.

2 Preliminaries

Below we briefly review the theory of kernel machines in terms of Reproducing Kernel Hilbert Spaces. We then describe the regularized risk functionals we are interested in optimizing during learning, and finish by reviewing the functional gradient, a generalization of the standard gradient to inner product spaces of functions.

2.1 Reproducing Kernel Hilbert Spaces

An RKHS of functions $\mathcal{H}_k$ is a complete inner product space, known as a Hilbert space, that arises from the completion of a set of basis functions $\mathcal{B}_k^{\mathcal{X}} = \{k(x, .) \mid x \in \mathcal{X}\}$, where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric, positive-definite *kernel* function, and $\mathcal{X}$ is a continuous domain sometimes known as an index set. A common kernel, and one used exclusively in our experiments, is the exponential Radial Basis Function (RBF) $k(x, x') = e^{\frac{-\|x-x'\|^2}{2\sigma^2}}$. The RKHS inner product between two functions $f = \sum_i \alpha_i k(x_i, .)$ and $g = \sum_j \beta_j k(x_j, .)$ is defined as

$$\langle f, g \rangle_{\mathcal{H}_k} = \sum_{i,j} \alpha_i \beta_j k(x_i, x_j). \tag{1}$$

Central to the idea of an RKHS is the *reproducing property* which follows directly from the above definition. It states that the basis functions $k(x, .) \in \mathcal{B}_k^{\mathcal{X}}$ are *representers of evaluation*. Formally, for all $f \in \mathcal{H}_k$ and $x \in \mathcal{X}$, $\langle f, k(x, .) \rangle_{\mathcal{H}_k} = f(x)$. When the basis functions are normalized, this means the evaluation of f at x is the scalar projection of f onto $k(x, .)$. Note that there exists a simple mapping $\phi : \mathcal{X} \to \mathcal{B}_k^{\mathcal{X}}$ between the domain $\mathcal{X}$ and the RKHS basis $\mathcal{B}_k^{\mathcal{X}}$ defined by the kernel as $\phi(x) = k(x, .)$. It follows that for any $x, x' \in \mathcal{X}$

$$\langle \phi(x), \phi(x') \rangle_{\mathcal{H}_k} = \langle k(x, .), k(x', .) \rangle_{\mathcal{H}_k} = k(x, x').$$

A complete overview of these concepts can be found in [Aronszajn, 1950].

A fundamental result first proven in [Kimeldorf & Wahba, 1971] and then generalized in [Schölkopf et al., 2001] is the *Representer Theorem*, which makes possible the direct minimization of a particular class of functionals. It states that given a subset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n \subset \mathcal{X} \times \mathbb{R}$ (i.e. the dataset), the minimizer of a functional with the form $F[f] = c((x_1, y_1, f(x_1)), \ldots, (x_n, y_n, f(x_n))) + g(\|f\|^2_{\mathcal{H}_k})$, where $c : \mathcal{X} \times \mathbb{R}^2 \to \mathbb{R}$ is arbitrary and $g : [0, \infty] \to \mathbb{R}$ is strictly monotonically increasing, must have the form $\tilde{f} = \sum_{x_i \in \mathcal{D}} \alpha_i k(x_i, .)$.

2.2 Regularized Risk Functionals

An important class of functionals, common in machine learning, for which the Representer Theorem holds is the regularized risk functional [Schölkopf & Smola, 2002]:

$$R[f] = \sum_{i=1}^n l(x_i, y_i, f(x_i)) + \frac{\lambda}{2} \langle f, f \rangle_{\mathcal{H}_k} \tag{2}$$

These functionals combine a data-dependent risk term, with a "prior" (or regularization) term that controls the complexity of the solution by penalizing the norm of the function f in the RKHS. Our work focuses on the case where l is differentiable in its third argument; many important kernel machines have this property. For instance, we can write Kernel Logistic Regression [Zhu & Hastie, 2001] in this form. Let $y \in \{-1, 1\}$. Then

$$R_{klr}[f] = \sum_{i=1}^n \log\left(1 + e^{y_i f(x_i)}\right) + \frac{\lambda}{2} \langle f, f \rangle_{\mathcal{H}_k} \tag{3}$$

Similarly, we may write the popular Gaussian Process Regression (for the mean function) and the Regularized Least Squares Classification algorithm in this form [1]:

$$R_{rls}[f] = \frac{1}{2} \sum_{i=1}^n (y_i - f(x_i))^2 + \frac{\lambda}{2} \langle f, f \rangle_{\mathcal{H}_k} \tag{4}$$

[1]These two differently motivated algorithms optimize the same functional.

Numerous other important examples occur in the literature [Schölkopf & Smola, 2002]. We focus on these, two of the best known kernel algorithms other than the non-differentiable Support Vector Machine, as they are particularly important tools in machine learning for classification and regression.

2.3 The Functional Gradient

For large-scale problems and problems for which direct solution is inapplicable, the most popular approach to optimizing these functionals is through the use of iterative, gradient-based techniques. Gradient-based algorithms, such as steepest descent, are traditionally defined with respect to the gradient arising from the Euclidean inner product between parameter vectors. There are many ways in which a given regularized risk functional can be parameterized, though, and each gives rise to a different parameter gradient. It is intuitively natural to follow the gradient defined uniquely by the RKHS inner product. We review some basic concepts behind the functional gradient below.

The functional gradient may be defined implicitly as the linear term of the change in a function due to a small perturbation ϵ in its input [Mason et al., 1999]

$$F[f + \epsilon g] = F[f] + \epsilon \langle \nabla F[f], g \rangle + O(\epsilon^2).$$

Following this definition using the RKHS inner product $\langle ., . \rangle_{\mathcal{H}_k}$ allows us to define the kernel gradient of a regularized risk functional [Schölkopf & Smola, 2002]. We use three basic formulas that can be easily derived from this definition:

1. *Gradient of the evaluation functional.* Define $F_x : \mathcal{H}_k \to \mathbb{R}$ as $F_x[f] = f(x) = \sum_i \alpha_i k(x_i, x)$. Then $\nabla_k F_x[f] = k(x, .)$.
2. *Gradient of the square RKHS norm.* Define $F_{\langle .,. \rangle} : \mathcal{H}_k \to \mathbb{R}$ as $F_{\langle .,. \rangle}[f] = \|f\|^2_{\mathcal{H}_k} = \langle f, f \rangle_{\mathcal{H}_k}$. Then $\nabla_k F_{\langle .,. \rangle}[f] = 2f$.
3. *Chain rule.* Let $g : \mathbb{R} \to \mathbb{R}$ be a differentiable function, and let $F : \mathcal{H}_k \to \mathbb{R}$ be an arbitrary differentiable functional. Then $\nabla_k g(F[f]) = g'(F[f]) \nabla_k F[f]$.

A straight forward application of the above formulas brings us to the following result. The kernel gradient of a regularized risk functional (Equation 2) is

$$\begin{aligned}
\nabla_k R[f] &= \sum_{i=1}^n \frac{\partial}{\partial z} l(x_i, y_i, z)|_{f(x_i)} \nabla_k F_{x_i}[f] + \lambda f \\
&= \sum_{i=1}^n \frac{\partial}{\partial z} l(x_i, y_i, z)|_{f(x_i)} k(x_i, .) + \\
&\quad \lambda \sum_{i=1}^n \alpha_i k(x_i, .) \\
&= \sum_{i=1}^n \left(\frac{\partial}{\partial z} l(x_i, y_i, z)|_{f(x_i)} + \lambda \alpha_i \right) k(x_i, .)
\end{aligned} \tag{5}$$

where we again use α_i to denote the parameters of the expansion of f in terms of the kernels $k(x_i, .)$.

Equation 5 allows us to easily find the kernel gradient of the functionals we described above and use in the remainder of this paper:

Kernel Logistic Regression (KLR):

$$\nabla_k R_{klr}[f] = \sum_{i=1}^{n} \left(\lambda \alpha_i + \frac{y_i}{1 + e^{-y_i f(x_i)}} \right) k(x_i, .) \qquad (6)$$

Regularized Least Squares (RLS), Gaussian Process:

$$\nabla_k R_{rls}[f] = \sum_{i=1}^{n} (f(x_i) - y_i + \lambda \alpha_i) k(x_i, .) \qquad (7)$$

2.4 The Kernel Gradient and Riemannian Metrics

Above we described the kernel gradient as a *function*; that is as a linear combination of basis functions. The kernel gradient as in Equation 5 demonstrates a property similar to the *Representer Theorem*: namely that $\nabla_k F[f] = \sum_{i=1}^{n} \gamma_i k(x_i, .)$ for appropriate $\gamma_i \in \mathbb{R}$. In other words, the kernel gradient of F is represented in the finite-dimensional subspace $\mathcal{S}_{\mathcal{D}} = \text{span}\{\mathcal{B}_k^{\mathcal{D}}\}$ of $\mathcal{H}_k$. A gradient descent type method through $\mathcal{S}_{\mathcal{D}}$ then amounts to modifying the coefficients α_i of the current function f by γ_i. That is, we can understand the kernel gradient as modifying parameters,

$$\tilde{f} \leftarrow f - \lambda \nabla_k F[f] \Leftrightarrow \tilde{\alpha}_i \leftarrow \alpha_i - \lambda \gamma_i,$$

just as a standard gradient descent algorithm would. The difference is that the coefficients γ_i for the kernel gradient are *not* the same as those of the parameter gradient. One can verify that they differ by $\nabla_\alpha F[f] = K\gamma$ where K is the kernel matrix and γ is the vector of coefficients.

We can derive this relation in another way. Starting from the parameter gradient, we define a Riemannian metric [Hassani, 1998] on the space of parameters. This defines our notion of size in the space of parameters. We then consider an alternate definition of the gradient, as the direction of steepest ascent for a "small" change in coefficients α:

$$\nabla F[\alpha] = \max_{\gamma} F[\alpha + \gamma] \; s.t. \; \|\gamma\| < \epsilon.$$

It can be shown that taking $\|\gamma\|_\alpha^2$ as $\gamma^T\gamma$ gives the vanilla parameter gradient $\nabla_\alpha F$, while defining the norm with respect to the RKHS inner product $\|\gamma\|_{\mathcal{H}_k}^2 = \sum_{i,j} \gamma_i \gamma_j k(x_i, x_j) = \gamma^T K \gamma$ gives the functional gradient coefficients $\nabla_k F = K^{-1} \nabla_\alpha F$. [Hassani, 1998]

This interpretation of the kernel gradient makes connections with other metric methods more clear. For instance, Amari [Amari & Nagaoka, 2000] considers the use of a metric derived from information geometry that leads to the "natural gradient". Such algorithms are applicable here as well since we can compute the metric for the probabilistic models given by Gaussian Processes or KLR. Unfortunately, computing the natural gradient in these cases is *very* expensive: for instance, in Gaussian Processes it is as expensive as inverting the kernel matrix, the very computational difficulty we are striving to avoid. By contrast, computing the kernel gradient is very cheap: cheaper in fact then the standard parameter gradient.[2]

[2]Furthermore, in practice we often find deriving the kernel gradient using the functional gradient rules easier than deriving the parameter gradient.

Algorithm 1 Kernel Conjugate Gradient

```
1:  procedure KCG($F : \mathcal{H}_k \to \mathbb{R}$, $f_0 \in \mathcal{H}_k$, $\epsilon > 0$)
2:      $i \leftarrow 0$
3:      $g_0 \leftarrow \nabla_k F[f_0] = \sum_{j=1}^{n} \gamma_j^{(0)} k(x_j, .)$
4:      $h_0 \leftarrow -g_0$
5:      while $\langle g_i, g_i \rangle_{\mathcal{H}_k} > \epsilon$ do
6:          $f_{i+1} \leftarrow f_i + \lambda_i h_i$ where $\lambda_i = \arg\min_\lambda F[f_i + \lambda h_i]$
7:          $g_{i+1} \leftarrow \nabla_k F[f_{i+1}] = \sum_{j=1}^{n} \gamma_j^{(i+1)} k(x_j, .)$
8:          $h_{i+1} \leftarrow -g_{i+1} + \eta_i h_i$ where $\eta_i = \frac{\langle g_{i+1} - g_i, g_{i+1} \rangle_{\mathcal{H}_k}}{\langle g_i, g_i \rangle_{\mathcal{H}_k}} = \frac{(\gamma^{(i+1)} - \gamma^{(i)})^T K \gamma^{(i+1)}}{\gamma^{(i)T} K \gamma^{(i)}}$
9:          $i \leftarrow i + 1$
10:     end while
11:     return $f_i$
12: end procedure
```

3 The Kernel Conjugate Gradient Algorithm

Both in theory and in practice it is understood that conjugate gradient (CG) methods outperform standard steepest descent procedures [Ashby et al., 1990]. These techniques have been used profusely throughout machine learning, in particular, for regularized risk minimization and kernel matrix inversion [Gibbs, 1997; Schölkopf & Smola, 2002].

In this section, we present an algorithm we term Kernel Conjugate Gradient (KCG) that takes advantage of conjugate direction search while utilizing the RKHS inner product $\langle f, g \rangle_{\mathcal{H}_k} = \alpha^T K \beta$. Algorithm 1 gives the general (nonlinear) KCG algorithm in Polak-Ribière form [Ashby et al., 1990]. In essence, Algorithm 1 comes directly from conjugate gradient by replacing all gradients by their functional equivalents and replacing Euclidean inner products with an RKHS inner product.

Note that the computational complexity per iteration of KCG is essentially identical to that of the more conventional Parameter Conjugate Gradient (PCG) algorithm. Intuitively, while the kernel inner product takes time $O(n^2)$ compared to the $O(n)$ vanilla inner product used by PCG, KCG is correspondingly more efficient in the gradient computation since $\nabla_\alpha F[f] = K\gamma$, where $\nabla_k F[f] = \sum_i \gamma_i k(x_i, .)$. It is possible in the case of RLS to step through an iteration of each algorithm and show that the number of operations is equivalent.

We emphasize that despite its somewhat involved derivation, the implementation of this algorithm is just a simple extension of PCG. The differences amount to only a change of inner product $\alpha^T\beta \to \sum_{i,j} \alpha_i \beta_j k(x_i, x_j) = \alpha^T K \beta$, and a different, though in some ways simpler, gradient computation. We also point out that the line optimization (step 6) can be solved in closed-form in the case of quadratic risk functionals (e.g. RLS). For starting point $f = \sum_i \alpha_i k(x_i, .)$ and search direction $h = \sum_i \gamma_i k(x_i, .)$ we have

$$\arg\min_\lambda F[f + \lambda h] = -\frac{\alpha^T K \gamma}{\gamma^T A \gamma}$$

where A is the Hessian of the quadratic functional when parameterized by α. Note that this formula differs from that derived under the parameter gradient ($-\alpha^T \gamma / \gamma^T A \gamma$) only in the numerator's inner product, as is a common theme throughout this algorithm. The theoretical and experimental results given below suggest that there is little reason why one should prefer PCG to KCG in most (differentiable) kernel algorithms.

4 Experimental Results - Kernel Conjugate Gradient

We bench-marked KCG against PCG for both classification and regression tasks. In all cases, KCG significantly out performed PCG.

Our first test was performed using KLR on the `USPS` dataset (with a training/test size of 7291/2007) for the common one-vs-all task of recognizing the digit 4. We used a length scale hyperparameter $\sigma = 5$ as was used in [Rifkin et al., 2003] for RLS classification, and a regularization constant $\lambda = 0$. Figure 1 summarizes the results in log scale.

Second, we used RLS for both regression and classification using the `Abalone` and `Soil` datasets in addition to the `USPS` dataset. The `Abalone` dataset [3] consisted of 3133 training examples and 1044 test examples in 8 attributes. Our experimental setup was equivalent to that in [Smola & Schölkopf, 2000]. The `Soil` dataset contained three-dimensional examples of soil pH levels in areas of Honduras partitioned into a training set of size 1709 and a test set of size 383. The latter dataset and corresponding hyperparameters ($\lambda = 0.1514$ and $\sigma = 0.296283$) were provided by [Gonzalez, 2005]. Again, the results are summarized in Figure 1.

For RLS, there exists a quadratic

$$p(\alpha) = \frac{1}{2}\alpha^T (K + \lambda I)\alpha - y^T \alpha$$

that can provide a lower bound to the regularized risk [Gibbs, 1997; Schölkopf & Smola, 2002]. As theory suggests (see below) the upper bound under KCG converges comparably with the lower bound, while the upper bound under PCG lags considerably behind. This implies faster convergence under a gap termination criterion [Schölkopf & Smola, 2002].

The right-most plots of Figure 1 contrast (in iterations, equivalent to multiples of wall-clock time) the speeds of PCG and KCG for both RLS and KLR. We plot of the number of iterations of each to reach the same level of performance in terms of loss on the data-set. Plots are terminated when convergence is achieved as measured with the gap termination criterion[Schölkopf & Smola, 2002], and hyper-parameters were chosen on a hold-out set. These figures confirm what analysis in the next section suggests: it takes more than the square of the amount of time to achieve the same level of performance using PCG as it does with KCG. Finally, in Table 2 we directly compare the number of iterations needed to achieve convergence on a number of data sets, all taken again from the UCI data repository, for both RLS and KLR. Averaged over the datasets, KCG is **54** times faster than the standard conjugate gradient approach.

[3]See UCI repository: `http://www.ics.uci.edu/~mlearn/MLRepository.html`

5 KCG Analysis

We derived the Kernel Conjugate Gradient algorithm from a normative point of view arguing that $\langle f, g \rangle_{\mathcal{H}_k}$ defined the natural notion of inner product in the RKHS and hence for the optimization procedure as well. The strong empirical performance of KCG noted in the previous section, while in some sense not surprising given we are using the "correct" inner product, deserves analysis. We examine here the linear case (as in RLS) where the analysis is more transparent, although presumably similar results hold near the optima of non-linear risk functionals.

We note a classic bound on the error reduction of CG (see [Luenberger, 2003]),

$$\|e_i\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i \|e_0\|_A,$$

where i is the iteration number, A is the Hessian of the quadratic form with condition number κ, $\|x\|_A = x^T A x$ is a norm on x, and $e_i = x_i - x^*$. Loosely speaking, this gives a running time complexity of $O(\sqrt{\kappa})$ for PCG.

We start by analyzing the effective condition number of KCG. As with essentially all variants of CG, the algorithm's dynamics can be described in terms of a preconditioning to the spectrum of the Hessian [Ashby et al., 1990]. It can be verified by inspection of the algorithm, that KCG is equivalent to an implicitly preconditioned conjugate gradient algorithm with preconditioner K[Ashby et al., 1990]. The following theorem relates the running time of PCG to KCG in light of the bound given above.

Theorem. *Let κ_{PCG} be the condition number of R_{rls} (Equation 4), and let κ_K be the condition number of the kernel matrix $K = [k(x_i, x_j)]_{i,j}$. Then the condition number κ_{KCG} resulting from preconditioning the RLS risk functional by K has the relation $\kappa_{PCG} = \kappa_K \kappa_{KCG}$.*

Proof. Let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$ be the eigenvalues of K. The condition number of K is then $\kappa_K = \sigma_1/\sigma_n$. The Hessian of R_{rls} is $A = K^T K + \lambda K$ and has eigenvalues $\sigma_i^2 + \lambda\sigma_i = \sigma_i(\sigma_i + \lambda)$, given in terms of the eigenvalues of K. This implies

$$\kappa_{PCG} = \frac{\sigma_1}{\sigma_n} \left(\frac{\sigma_1 + \lambda}{\sigma_n + \lambda} \right) = \kappa_K \frac{\sigma_1 + \lambda}{\sigma_n + \lambda}.$$

Since K is symmetric, positive-definite, the preconditioned Hessian becomes $K^{-1}A = K^{-1}(K^T K + \lambda K) = K + \lambda I$, with corresponding eigenvalues $\sigma_i + \lambda$. Thus, $\kappa_{PCG} = \kappa_K \kappa_{KCG}$. $\square$

The condition number κ_K of K is typically very large. In particular, as the regularization constant decreases, the asymptotic bound on the convergence of PCG approaches the square of the bound on KCG. Alternatively, as the regularization constant increases, κ_{KCG} approaches 1 implying an $O(1)$ convergence bound for KCG, while the convergence bound of PCG remains bounded below by $O(\kappa_K)$. We would thus expect a number of iterations for KCG that is dramatically less than that of PCG.

It is informative to note that the decrease in computational complexity from PCG to KCG ($O(\kappa^{1/2})$ to $O(\kappa^{1/4})$) is at least the amount we see from steepest descent($O(\kappa)$ to PCG $O(\kappa^{1/2})$) [Luenberger, 2003].

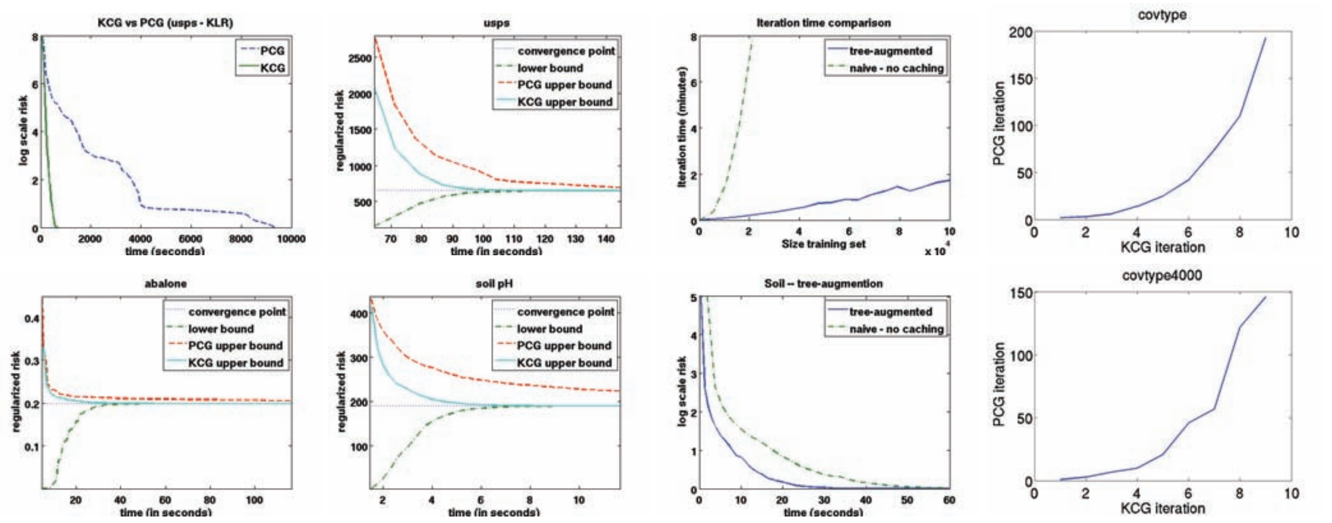

Figure 1: Upper left shows relative performances (in log scale) of KCG and PCG on the `USPS` data set optimizing KLR. The remaining three left-most plots show relative convergence under RLS; green and red lines depict PCG performance on the upper and lower bound gap-convergence quadratic forms, and the light blue line gives the (significantly tighter) performance of KCG on the upper bound. The third column shows the benefit of using KD-trees for a single run (bottom) and by training set size (top) using KCG. The right-most two plots show the equivalent number of PCG iterations required to achieve the performance per iteration of KCG on the `COVTYPE` data set from the UCI data repository. Top right and bottom right show the performances of RLS and KLR, respectively. An approximately quadratic relationship can be seen in both cases as the theory suggests.

6 Tree-Augmented Algorithms

For many stationary kernels, it is often the case that the majority of the basis functions in $\mathcal{B}_k^{\mathcal{D}}$ are nearly orthogonal. This often stems from a relationship between the degree of orthogonality of two basis functions $k(x,.), k(x',.) \in \mathcal{B}_k^{\mathcal{D}}$ and the Euclidean distance between x and x'. This is often the case when using the exponential RBF kernel given above, in which the orthogonality between two basis functions increases exponentially with the square Euclidean distance between the two points $\|x - x'\|^2$.

Previous work has shown how the evaluation of RKHS functions $f(x) = \sum_i \alpha_i k(x_i, x)$ can be made fast when this holds using N-body type algorithms [Gray & Moore, 2001]. Intuitively, the idea is to store the training data in a space-partitioning tree, such as a KD-tree [Moore, 1990] as was used in our experiments below, and recursively descend the tree during evaluation, pruning negligible contributions.

The maximum and minimum impact of each set of pruned points can be easily calculated resulting in upper and lower bounds on the evaluation error. We demonstrate how such data-structures and algorithms can be used to reduce the per iteration $O(n^2)$ computational cost of both KCG and PCG during learning as well as evaluation.

The inner loop computational bottleneck of KCG is in evaluating functions and calculating the kernel inner product. If we rewrite the RKHS inner product between $f = \sum_i \alpha_i k(x_i,.)$, $g = \sum_i \beta_i k(x_i,.)$ as $\langle f, g\rangle_{\mathcal{H}_k} = \sum_{i,j} \alpha_i \beta_j k(x_i, x_j) = \sum_i \alpha_i g(x_i)$, then reducing the computational complexity of RKHS function evaluations will simultaneously encompass both of these bottlenecks. Similarly, the iteration complexity of PCG is dominated by the computation of the parameter gradient. We can rewrite the parameter gradient as $\nabla_\alpha F[f] = [\nabla_k F[f](x_1), \nabla_k F[f](x_2), \ldots, \nabla_k F[f](x_n)]^T$ (see 2.4), reducing the complexity to that of finding $\nabla_k F[f] \in \mathcal{H}_k$ and evaluating it n times. As was the case with the KCG algorithm without trees, the tradeoff still balances out so that the per iteration complexity is essentially equivalent between KCG and PCG using tree-augmented function evaluation.

Noting $[f(x_1), \ldots, f(x_n)]^T = K\alpha$ suggests that the closed form quadratic line minimization for both the upper and lower bounds of RLS under either KCG or PCG can easily be augmented as well by expanding the Hessians $A_u = K^T K + \lambda K$ in the case of the upper bound and $A_l = K + \lambda I$ in the case of the lower bound. This was used in the tree-augmented RLS experiments described below.

7 Experimental Results - Tree-Augmented Algorithms

For these experiments, in addition to using the `Soil` dataset described in section 4, we performed large scale RLS regressions using a tree-augmented KCG on variable sized subsets of a `PugetSound` elevation map[4] using hyperparameters $\lambda = 0.1/n$ and $\sigma^2 = kN/(n\pi)$, where n is the size of the training set, and N is the size of the entire height map. In this case, we chose $N = 500 \times 500 = 250,000$, and $k = 15$. The largest resulting datasets were on the order of 100,000 points. It should be noted that the näive implementation in this case did not cache kernel evaluations in a kernel matrix as such matrices for datasets above $O(15,000)$ points proved

[4]`http://www.cc.gatech.edu/projects/large_models/`

RLS	cmc	covtype	glass	ionosphere	iris	page-block	pima	spam	wine
examples	1000	6000	150	300	120	5000	568	4000	128
PCG iters	1749	318	30	45	72	120	1825	2106	24
KCG iters	17	12	5	6	11	10	17	40	5
Speedup	102.9	26.5	6.0	7.5	6.5	12.0	107.4	52.6	4.8

KLR	cmc	covtype	glass	ionosphere	iris	page-block	pima	spam	wine
examples	1000	4000	150	300	120	-	568	2000	128
PCG iters	1490	232	35	177	74	-	682	11045	52
KCG iters	29	9	9	11	11	-	11	26	6
Speedup	51.4	25.8	3.9	16.1	6.7	-	62.0	424.8	8.7

Figure 2: Times (in iterations) to achieve convergence of RLS (top) and KLR (bottom) on a subset of UCI data-sets. KCG decreases the number of iterations on average by a factor of 54.

intractable for the machine on which the experiments were performed.

Figure 1 shows that tree-augmentation significantly outperforms the naïve algorithm. Extrapolating from the plot on the right, trees make possible accurate kernel learning on very large datasets without requiring explicit subset selection techniques.

8 Conclusions and Future Work

We have demonstrated that the novel gradient method, Kernel Conjugate Gradient, can dramatically improve learning speed for differentiable kernel machines. Furthermore, we have shown how this can be understood as a very efficient preconditioning that naturally derives from the inner product on functions defined by the kernel. In practice, for various differentiable kernel learning problems, we find KCG consistently, and significantly, outperforms existing techniques. We emphasize that the algorithm is simple to implement and requires no more computation per iteration than standard approaches.

Further, we demonstrated that space-partitioning data-structures, also developed by other authors [Shen et al., 2006], for optimizing Gaussian Processes extend naturally to other kernel methods. We find that this approach meshes well with the KCG algorithm, by significantly speeding up the inner loop computations of functions and inner products.

While conjugate gradient is a powerful optimization algorithm, there are other approaches, like Limited-Memory Quasi-Newton methods [Luenberger, 2003] that may also be derived in similar ways in terms of the kernel inner product. These algorithms have proved to be practical when using the Euclidean inner product; we expect they would also gain the benefits of preconditioning that KCG enjoys.

Finally, very large scale kernel applications, it seems, will invariably need to rely on sparse representation techniques that do not use kernels at all of the data points. Nearly all of these methods require the efficient solution of large kernel problems using an iterative approximation. It is natural to explore how Kernel Conjugate Gradient can speed up the expensive inner loop of these approximation procedures.

Acknowledgements

The authors gratefully acknowledge the partial support of this research by the DARPA Learning for Locomotion contract.

References

Amari, S., & Nagaoka, H. (2000). *Methods of information geometry*. Oxford University Press.

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*.

Ashby, S. F., Manteuffel, T. A., & Saylor, P. E. (1990). A taxonomy for conjugate gradient methods. *SIAM Journal on Numerical Analysis* (pp. 1542–1568).

Gibbs, M. N. (1997). *Bayesian gaussian processes for regression and classification*. Doctoral dissertation, University of Cambridge.

Gonzalez, J. P. (2005). Carnegie Mellon University, PhD. candidate.

Gray, A., & Moore, A. (2001). N-body problems in statistical learning. *Advances in Neural Information Processing Systems*. MIT Press.

Hassani, S. (1998). *Mathematical physics*. Springer.

Kimeldorf, G. S., & Wahba, G. (1971). Some results on Tchebycheffian spline functions. *J. Math. Anal. Applic.* (pp. 82–95).

Luenberger, D. G. (2003). *Linear and nonlinear programming, second edition*. Springer.

Mason, L., J.Baxter, Bartlett, P., & Frean, M. (1999). *Functional gradient techniques for combining hypotheses*. MIT Press.

Moore, A. (1990). *Efficient memory-based learning for robot control*. Doctoral dissertation, University of Cambridge.

Rifkin, Yeo, & Poggio (2003). Regularized least squares classification. *Advances in Learning Theory: Methods, Models and Applications*. IOS Press.

Schölkopf, B., Herbrich, R., & Smola, A. J. (2001). A generalized representer theorem. *Lecture Notes in Computer Science* (pp. 416–426). Springer-Verlag.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press.

Shen, Y., Ng, A., & Seeger, M. (2006). Fast gaussian process regression using kd-trees. In Y. Weiss, B. Schölkopf and J. Platt (Eds.), *Advances in neural information processing systems 18*, 1227–1234. Cambridge, MA: MIT Press.

Smola, A. J., & Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. *International Conference on Machine Learning*.

Zhu, J., & Hastie, T. (2001). Kernel logistic regression and the import vector machine. *Advances in Neural Information Processing Systems*.