Momentum Accelerated Training of Kernel Machines

Anonymous Author
Anonymous Institution

Abstract

In this paper we derive a novel iterative algorithm for learning kernel machines. Our algorithm, AxlePro, extends the EigenPro family of algorithms via momentum-based acceleration. AxlePro can be applied to train kernel machines with arbitrary positive semidefinite kernels. We provide a convergence guarantee for the algorithm and demonstrate the speedup of AxlePro over competing algorithms via numerical experiments. Furthermore, we also derive a version of AxlePro to train large kernel models over arbitrarily large datasets.

1 INTRODUCTION

Deep Neural Networks have become the dominant paradigm of modelling and data analysis for large scale problems in machine learning today. These model architectures have shown remarkable progress in our ability to extract information from complex datasets, breaking unprecedented benchmarks every so often. Certain architectures of these networks, such as wide networks [18], have been shown to behave like the more classical Kernel machines. This has sparked renewed interest into the capabilities of kernel methods and their application towards solving large scale problems in machine learning.

Newer families of kernels, so-called *neural kernels*, have also shown remarkable performance on contemporary machine learning problems, see [4, 5, 8, 16, 32] among others. Furthermore, recent work also shows that kernel machines can be re-engineered to learn task-specific features in a supervised manner [7, 25], which leads to further performance gains. Learning a kernel machine in a scalable manner thus remains an important fundamental problem in machine learning.

Preliminary work. Under review by AISTATS 2025. Do not distribute.

Formally, kernel machines are functions of the form

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i). \tag{1}$$

where x_i are training inputs, $\alpha_i \in \mathbb{R}$ are trainable weights, and *kernel* $K : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is a positive definite function.

We consider the training of (1) by solving the $n \times n$ positive definite linear system of equations,

$$\texttt{K\_solve}(K, X, Y): \quad \text{Find } \boldsymbol{\alpha} \ : \ K(X, X)\boldsymbol{\alpha} = Y, \tag{2}$$

where $K(X, X) = (K(x_i, x_j))$ is a $n \times n$ positive definite matrix of pairwise evaluations of the kernel K on training inputs, and $Y \in \mathbb{R}^n$ is the vector of all n targets. If $\mathbb{H}$ is the RKHS corresponding to K, (1) with weights given by $\texttt{K\_solve}(K, X, Y)$ is the minimum $\mathbb{H}$-norm solution to the least squares problem,

$$\min_{f \in \mathbb{H}} \mathsf{L}(f) := \tfrac{1}{2} \sum_{i=1}^{n} (f(x_i) - y_i)^2. \tag{3}$$

Remark 1. While there are many variations, such as different loss functions and regularizers, we leave aside these generalizations for future work. However we note that solving (2) subsumes the important case of kernel ridge regression. This somewhat easier case is equivalent to choosing a modified kernel $\widetilde{K}(x, z) = K(x, z) + \lambda \cdot \mathbf{1}_{\{x=z\}}$, where $\lambda > 0$ is a tunable parameter of the ridge penalty, since our framework allows arbitrary kernels, even discontinuous ones.

1.1 Main contributions

We derive a new class of algorithms, Algorithms (1-3), called AxlePro. These algorithms are derived to emulate momentum-accelerated preconditioned stochastic gradient descent in the RKHS, as detailed in Section 3. Our theoretical result Theorem 6 provides a guarantee on the exponential convergence of Algorithm 1. Our numerical experiments demonstrate that AxlePro is fast, numerically stable, and scalable. We provide a `PyTorch` implementation. We also derive a version of AxlePro in Section 3.5, given in Algorithm 2, that can train so-called *general kernel models* [1] (see Section 2).

Algorithm	fast precond.^{p}	momentum	batching	stable‡	order	complexity^c	mem.$^⋆$
PCG [15]	✗	✓	✗	✗		$n^2 \cdot \sqrt{\kappa}$	n
EigenPro [22]	✗	✗	✓	✓	1st	$nm \cdot \kappa_m$	n
EigenPro 2 [23]	✓	✗	✓	✓	1st	$nm \cdot \kappa_m$	$\log^4 n$
ASkotch [27]	✗	✓	✓		2nd	$(nm + m^2)\sqrt{\kappa_m}$	m
Alt.-proj. [33]	✗	✗	✓		2nd	$m^3 \cdot \kappa_m$	m^2
Algorithm 1 (ours)	✓	✓	✓	✓	1st	$nm \cdot \sqrt{\kappa_m}$	$\log^4 n$

Table 1: Iterative algorithms for solving equation (2). m is the batch size. Condition number κ is the ratio of largest and smallest positive eigenvalues of $K(X, X)$, $\kappa_m, \widetilde{\kappa}_m$ are modified versions of κ defined in equation (24), with $\kappa_m \leq \kappa$ depending on m in a data-dependent manner.
^{p}By fast preconditioning we mean the preconditioner is calculated once and should be of size $o(n)$.
‡Stability to single precision computations, and support for ridgeless regression (KRR with $\lambda = 0^+$).
^{c}Total complexities in big-Oh notation, product of per iteration complexity times number of iterations.
$^⋆$memory overheads due to preconditioning.

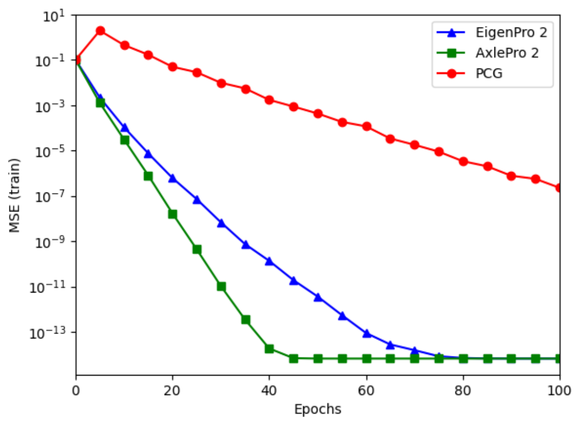

Figure 1: Comparison of iterative algorithms for the Laplace Kernel on CIFAR-10

Algorithm 1 AxlePro 2

```
1: Input: positive definite kernel $K$, batch size $m$, size of approximate preconditioner $s$, learning rates $\eta_1, \eta_2 > 0$, damping factor $\gamma \in (0, 1)$.
2: Output: $f : \mathcal{X} \to \mathbb{R}$ solving (11) approximately.
3: setup: Sample $J \subset \{1, 2, \ldots, n\}$ with $|J| = s$.
4: $(\boldsymbol{D}, \Delta, \delta_{q+1}) \leftarrow$ top-$q$ eigensystem of $K(X[J], X[J])$
5: $\boldsymbol{G} \leftarrow \boldsymbol{D}\sqrt{\Delta^{-1}(I_q - \delta_{q+1}\Delta^{-1})} \in \mathbb{R}^{s \times q}$
6: $\boldsymbol{\alpha} \leftarrow \mathbf{0}_n$, and $\boldsymbol{\beta} \leftarrow \mathbf{0}_n$.
7: repeat
8:     Fetch batch of indices $B \subset \{1, .., n\}$, $|B| = m$
9:     $\boldsymbol{v} \leftarrow K(X[B], X)\boldsymbol{\beta} - Y[B] \in \mathbb{R}^m$
10:    $\boldsymbol{w} \leftarrow \boldsymbol{G}\boldsymbol{G}^\top K(X[J], X[B])\boldsymbol{v} \in \mathbb{R}^s$
11:    $\widetilde{\boldsymbol{\alpha}} \leftarrow \boldsymbol{\alpha}$
12:    $\boldsymbol{\alpha} \leftarrow \boldsymbol{\beta}$
13:    $\boldsymbol{\alpha}[B] \leftarrow \boldsymbol{\alpha}[B] - \eta_1 \boldsymbol{v}$
14:    $\boldsymbol{\alpha}[J] \leftarrow \boldsymbol{\alpha}[J] + \eta_1 \boldsymbol{w}$
15:    $\boldsymbol{\beta} \leftarrow (1 + \gamma)\boldsymbol{\alpha} - \gamma\widetilde{\boldsymbol{\alpha}}$
16:    $\boldsymbol{\beta}[B] \leftarrow \boldsymbol{\beta}[B] + \eta_2 \boldsymbol{v}$
17:    $\boldsymbol{\beta}[J] \leftarrow \boldsymbol{\beta}[J] - \eta_2 \boldsymbol{w}$
18: until Stopping criterion is reached
19: return $f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i)$
```

1.2 Related work

The problem of scalable training algorithms for kernel models has been studied extensively in the past. Approaches for scalable training include: Linear system solvers for symmetric positive definite matrices. Examples of such solvers include conjugate gradient (CG) and preconditioned version PCG. See [24,29,34] for example. While these methods are the fastest in the asymptotic sense, they do not decrease training error monotonically, and require a large number of epochs. Hence, even though they can be implemented in a matrix-free manner, the total cost of computations is very large for machine learning applications. Furthermore CG based methods are also sensitive to quantization noise and hence are not stable to single-precision computations.

Randomized methods such as those based on random fourier features, sketching, or low-rank decompostions provide a way to train in a scalable manner [10,11,26]. See [16] and the references therein for a more recent overview on sketching and random features. However this approach is not universal, i.e., requires special structure in the kernel. Sketching based solutions have also been studied in this context [12].

Primal space solvers such as SGD [9,31] and its variants can be very effective, and are universal, i.e., apply for any kernel. In fact [1,2,22,23] are state of the art in solving kernel machines in a fast and scalable manner. These methods derive an iteration in the RKHS, but emulate these computations exactly in $\mathbb{R}^n$.

Algorithm	fast precond.[p]	momentum	batching	stable[‡]	order	FLOPS[c]/batch	mem[⋆]
Cholesky [28]	✗	✓	✗	✗	2nd	$p^2m + p^3$	p^2
FALKON [24, 29]	✗	✗	✗	✓	2nd	$pm + p^2$	p^2
EigenPro 3 [1]	✓	✗	✓	✓	1st	$pm + p^2$	$\log^4 n$
EigenPro 4 [2]	✓	✗	✓	✓	1st	pm	$\log^4 n$
SketchySAGA [27]	✓	✓	✓		2nd	$pm + p^2$	pm
Algorithm 2 (ours)	✓	✓	✓	✓	1st	pm	$\log^4 n$

Table 2: Comparison between different iterative algorithms for solving (10). m is the batch size, typically $m \ll n$. Condition number κ is the ratio of largest to smallest positive eigenvalue, $\kappa_m, \widetilde{\kappa}_m$ are modified versions of κ defined in equation (24), with $\widetilde{\kappa}_m \leq \kappa_m \leq \kappa$.
[p]By fast preconditioning we mean the preconditioner must be calculated once and should be of size $o(n)$.
[‡]Stability to single precision computations.
[c]number of flops/sample in big-Oh notation, product of per iteration complexity times number of iterations.
[⋆]memory overheads due to preconditioning.

Our algorithm Algorithm 1 falls in this category.

Second order methods with acceleration have also been considered recently in [27] and [33] which inherently apply a stochastic Newton-type iteration in the RKHS. However, these methods involve recomputing a preconditioner at each step, which may become prohibitive. An in-depth comparison of the numerical performance with these methods on real-world datasets is deferred to future work.

General kernel models of the form equation (9) has been an important way to train kernel machines on arbitrarily large datasets. This has become important since [1] showed that kernel machines need to scale the model size independent of the dataset size to improve performance, similar to what is done in practice for neural network models. Models equation (9) are the preferred way to scale RKHS methods to large datasets with several new implementations. For example [1, 2, 24, 27] consider this problem. Our Algorithm 2 is along this direction, with a comparison given in table 2.

2 PRELIMINARIES

We consider the standard setting of supervised learning where we are provided a training dataset with n samples $\{(x_i, y_i)\}_{i=1}^n$ and want to obtain a map $x \mapsto f(x)$, where we $x_i \in \mathbb{X}$, an abstract space, and $y_i \in \mathbb{R}$.

Kernel machines such as equation (1) arise as the solutions to empirical risk minimization problems over an RKHS corresponding to K,

$$\min_{f \in \mathbb{H}} \sum_{i=1}^{n} \mathsf{L}_i(x_i, y_i, f(x_i)) + \lambda \mathsf{R}\left(\|f\|_{\mathbb{H}}\right). \tag{4}$$

due to the Representer Theorem [19, 30]. This theorem requires the mild condition that R is an increasing function and $\lambda > 0$. The reproducing property of K, means we have,

$$\langle f, K(\cdot, x)\rangle_{\mathbb{H}} = f(x), \qquad \forall f \in \mathbb{H}, \forall x \in \mathbb{X}, \tag{5}$$

where $K(\cdot, x) \in \mathbb{H}$ is the *Reisz representer* for the evaluation functional corresponding to the point x.

Notation. A detailed table of notations is in the appendix. For tuples $A = (a_i) \in \mathbb{X}^k$ and $U = (u_i) \in \mathbb{X}^\ell$, $K(A, U) \in \mathbb{R}^{k \times \ell}$ denotes the kernel matrix with entries $K(a_i, u_j)$. For any $f \in \mathbb{H}$, by $f \otimes_{\mathbb{H}} f$, we denote the rank-1 self-adjoint operator $\mathbb{H} \to \mathbb{H}$, which acts as $(f \otimes_{\mathbb{H}} f)(g) = f \langle f, g\rangle_{\mathbb{H}}$. Here, if $f = K(x, \cdot)$, then we have $(K(x, \cdot) \otimes_{\mathbb{H}} K(x, \cdot))(g) = K(\cdot, x) g(x)$, by equation (5).

Definition 1 (Sampling operator)**.** For an n-tuple of inputs $X = (x_i) \in \mathbb{X}^n$, denote by $S : \mathbb{H} \to \mathbb{R}^n$ the evaluation operator which evaluates any function $f \in \mathbb{H}$, i.e.,

$$Sf = (f(x_1), \ldots, f(x_n)) \in \mathbb{R}^n. \tag{6}$$

One can show that the adjoint of S, defined in equation (6), denoted $S^* : \mathbb{R}^n \to \mathbb{H}$, acts as

$$S^* \boldsymbol{\alpha} = \sum_{i=1}^{n} K(\cdot, x_i)\alpha_i \in \mathbb{H}, \quad \forall \boldsymbol{\alpha} \in \mathbb{R}^n. \tag{7}$$

We define the empirical covariance operator,

$$\mathcal{K} := \tfrac{1}{n} S^* S = \frac{1}{|X|} \sum_{x \in X} K(x, \cdot) \otimes_{\mathbb{H}} K(x, \cdot). \tag{8}$$

Slicing. For an n-tuple $X = (x_1, x_2, \ldots, x_n)$ of size n and an s-tuple of indices $J = (j_1, j_2, \ldots, j_s)$, we denote by $X[J]$ the s-tuple $(x_{j_1}, x_{j_2}, \ldots, x_{j_s})$, following `python` notation. For a matrix $\boldsymbol{A}$ and vector $\boldsymbol{v}$, the quantities $\boldsymbol{A}[J]$ and $\boldsymbol{v}[J]$ are the submatrix and subvector of $\boldsymbol{A}$ and $\boldsymbol{v}$ respectively indexed by J. By

$\boldsymbol{H}_J$ we denote the $|J| \times n$ matrix formed by stacking the $|J|$ rows of identity I_n. By $S_J := \boldsymbol{H}_J S$ we denote the operator which maps $\mathbb{H} \to \mathbb{R}^{|J|}$ so that $S_J f = (f(x_{j_1}), f(x_{j_2}), \ldots, f(x_{j_s}))$. Similarly, define $\mathcal{K}_J = \frac{1}{|J|} S_J^* S_J$.

Definition 2 (top-q eigsystem)**.** Given a symmetric positive definite matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, for $i = 1, 2, \ldots, q+1$, let $\boldsymbol{e}_i, \lambda_i$ be the top-q+1 eigenpairs of $\boldsymbol{A}$, i.e., $\boldsymbol{A}\boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$, with descending eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{q+1}$ 0. Let $\boldsymbol{E} := [\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_q] \in \mathbb{R}^{n\times q}$, and $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_q) \in \mathbb{R}^{q\times q}$ be a diagonal matrix. We call the tuple $(\boldsymbol{E}, \Lambda, \lambda_{q+1})$ the top-q eigsystem of $\boldsymbol{A}$.

Spectral duality. Observe that $\mathcal{K} = \frac{1}{n} S^* S$, and $SS^* = K(X, X)$ is the kernel matrix. Let $\boldsymbol{e}$ be an eigenvector of $K(X, X)$ with eigenvalue λ with unit norm, i.e. $\|\boldsymbol{e}\| = 1$ and $K(X, X)\boldsymbol{e} = \lambda \boldsymbol{e}$. Then, one can show using standard arguments about singular vectors of finite rank operators that, $\psi := S^*\boldsymbol{e}/\sqrt{\lambda}$ is an eigenfunction of $\mathcal{K}$ with eigenvalue λ/n. Note that $\|\psi\|_{\mathbb{H}}^2 = \langle\psi, \psi\rangle_{\mathbb{H}} = \frac{1}{\lambda}\langle S^*\boldsymbol{e}, S^*\boldsymbol{e}\rangle_{\mathbb{H}} = \frac{1}{\lambda}\langle SS^*\boldsymbol{e}, \boldsymbol{e}\rangle = \langle \boldsymbol{e}, \boldsymbol{e}\rangle = 1$. The normalization $1/\sqrt{\lambda}$ used to define ψ ensures that $\|\psi\|_{\mathbb{H}} = 1$ which is necessary for how ψ gets used in the next section.

General kernel models. Consider models

$$f(x) = \sum_{i=1}^{p} \alpha_i K(x, z_i) \tag{9}$$

here $z_i \in \mathbb{R}^d$ are p *landmarks* or *inducing points* or *model centers* in the domain $\mathbb{X}$ are better suited when training on very large datasets, i.e., when $n \geq 1$ billion samples [24]. Such models have received renewed attention [1, 2, 27]. They can be trained via

$$\min_{f\in\mathbb{H}} \sum_{i=1}^{n} (f(x_i) - y_i)^2 \quad \text{subject to (9)}, \tag{10}$$

where $\mathbb{H}$ is the reproducing kernel hilbert space (RKHS) corresponding to the kernel K.

3 ALGORITHM DERIVATION

All proofs in this section are in the Appendix.

Overview. We start by describing the derivation of EigenPro, which emulates a preconditioned SGD iteration in the RKHS [22]. We then derive AxlePro given in Algorithm 3, which extends EigenProby applying a specific momentum acceleration inspired by the MaSS algorithm from [20]. Finally we derive the faster version AxlePro 2 in Algorithm 1, which extends EigenPro 2 [23]. Similar to EigenPro 2 our algorithm AxlePro 2 applies an approximate preconditioner in AxlePro obtained via a Nyström approximation [3].

Consider the optimization problem,

$$f^* = \arg\min_{f\in\mathbb{H}} \mathsf{L}(f) = \tfrac{1}{2n}\|Sf - Y\|^2. \tag{11}$$

The minimum-norm solution to the above problem is equation (1) with $\boldsymbol{\alpha}^* = \texttt{solve}(K, X, Y)$ from equation (2).

For $f \in \mathcal{H}$, the gradient and Hessian of L at f are

$$\nabla_f \mathsf{L}(f) = \tfrac{1}{n} S^*(Sf - Y) \text{ and } \nabla_f^2 \mathsf{L}(f) = \tfrac{1}{n} S^* S = \mathcal{K}.$$

Note this is the Fréchet derivative, and belongs to the RKHS $\nabla_f \mathsf{L} \in \mathbb{H}$, and the Hessian is an operator on the RKHS $\nabla_f^2 \mathsf{L} : \mathbb{H} \to \mathbb{H}$.

Note that for a minibatch $B \subset \{1, 2, \ldots, n\}$ of size $|B| = m$, the stochastic gradient is

$$\widetilde{\nabla}\mathsf{L}(f) = \tfrac{1}{m} S_B^*(S_B f - Y[B]) = \mathcal{K}_B(f - f^*), \tag{12}$$

where we denote by $\mathcal{K}_B := \frac{1}{|B|} S_B^* S_B$.

3.1 EigenPro: Preconditioned SGD in RKHS

Our goal is to solve equation (11). If we start by initializing $f_0 \in \operatorname{range}(S^*)$, then one can show that the iterates f_t of gradient descent

$$f_{t+1} = f_t - \eta\nabla\mathsf{L}(f_t) = f_t - \tfrac{\eta}{n} S^*(Sf_t - Y)$$

also lie in $\operatorname{range}(S^*)$. Hence we can write $f_t = S^*\boldsymbol{\alpha}_t$, by assuming $f_0 \in \operatorname{range}(S^*)$.

Next, SGD in $\mathbb{H}$, with learning rate η_0, can be written as

$$f_{t+1} \leftarrow f_t - \eta_0 \widetilde{\nabla}_f \mathsf{L}(f_t). \tag{13}$$

and converges exponentially as shown in [21, Theorem 1].

Proposition 1. *The following update equations*

$$\boldsymbol{v}_t \leftarrow K(X[B], X)\boldsymbol{\alpha}_t - Y[B] \tag{14a}$$

$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\alpha}_t - \eta_0 \boldsymbol{H}_B^\top \boldsymbol{v}_t. \tag{14b}$$

emulate one step of updates in equation (13) *via the relation* $f_t = S^*\boldsymbol{\alpha}_t$*.*

Remark 2. To avoid carrying around $\frac{1}{|B|}$ in all expressions, we henceforth absorb $\frac{1}{|B|}$ in the learning rate η, which anyway has a non-trivial dependence on batch size $|B|$ for the optimal setting.

3.2 MaSS in RKHS

The MaSS updates in $\mathbb{H}$, following the update rule from [20] yields

$$f_{t+1} \leftarrow g_t - \eta_1 \widetilde{\nabla}_f \mathsf{L}(g_t), \tag{15a}$$

$$g_{t+1} \leftarrow (1+\gamma) f_{t+1} - \gamma f_t + \eta_2 \widetilde{\nabla}_f \mathsf{L}(g_t), \tag{15b}$$

where $\eta_1, \eta_2 > 0$ are learning rates and $\gamma \in (0, 1)$ is a damping factor.

Remark 3. Note that for $\gamma = 0$, this is equivalent to SGD with learning rate $\eta_0 = \eta_1 - \eta_2$.

This iteration converges exponentially fast as shown in [20, Theorem 2].

Now, suppose $f_t, g_t \in \text{range}(S^*)$, then f_{t+1} and g_{t+1} are still in range(S^*), when $\widetilde{\nabla}_f \mathsf{L}(f_t) \in \text{range}(S_B^*)$.

Proposition 2. *The following update equations*

$$\boldsymbol{v}_t \leftarrow K(X[B], X)\boldsymbol{\beta}_t - Y[B] \tag{16a}$$
$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_1 \boldsymbol{H}_B^\top \boldsymbol{v}_t, \tag{16b}$$
$$\boldsymbol{\beta}_{t+1} \leftarrow (1+\gamma)\boldsymbol{\alpha}_{t+1} - \gamma\boldsymbol{\alpha}_t + \eta_2 \boldsymbol{H}_B^\top \boldsymbol{v}_t, \tag{16c}$$

emulate one step of updates in equation (15) *via the relation* $f_t := S^*\boldsymbol{\alpha}_t$ *and* $g_t := S^*\boldsymbol{\beta}_t$.

The proof is similar to that of Proposition 1.

3.3 AxlePro: Preconditioned MaSS in RKHS

A key challenge in the convergence rate of iterative methods is that the condition numbers of kernel matrices $K(X, X)$ are very high. Consequently, the iterative algorithms take a large number of iterations to converge, typically κ or $\sqrt{\kappa}$ where κ is the condition number of $K(X, X)$. Several strategies for preconditioning have been studied in the literature [13, 14]. We use the following spectral preconditioner because it is easiest to approximate elegantly, as shown in [3].

Recall that ψ_i are eigenvectors of $\mathcal{K}$. The spectral preconditioner $\mathcal{P} : \mathbb{H} \to \mathbb{H}$ is given by,

$$\mathcal{P} := \mathcal{I} - \sum_{i=1}^{q} \left(1 - \tfrac{\lambda_{q+1}}{\lambda_i}\right) \psi_i \otimes_{\mathbb{H}} \psi_i. \tag{17}$$

Note that $\mathcal{P}f = f - \sum_{i=1}^{q} \left(1 - \frac{\lambda_{q+1}}{\lambda_i}\right) \langle f, \psi_i\rangle_{\mathbb{H}} \psi_i$. This choice of preconditioner is motivated by the fact that the largest eigenvalue of $\mathcal{K}$ is λ_1/n, whereas the largest eigenvalue of $\mathcal{PK}$ is λ_{q+1}/n, which governs the convergence rate. Furthermore, their smallest eigenvalues are the same, λ_n. Note that here we have used the fact that $\|\psi_i\|_{\mathbb{H}} = 1$.

Suppose we precondition the gradients before making any updates. Then the preconditioned MaSS update equations in $\mathbb{H}$ become

$$f_{t+1} \leftarrow g_t - \eta_1 \mathcal{P}\widetilde{\nabla}_f \mathsf{L}(g_t), \tag{18a}$$
$$g_{t+1} \leftarrow (1+\gamma) f_{t+1} - \gamma f_t + \eta_2 \mathcal{P}\widetilde{\nabla}_f \mathsf{L}(g_t). \tag{18b}$$

Define $\boldsymbol{F} = \boldsymbol{E}\sqrt{I_q - \lambda_{q+1}\Lambda^{-1}} \in \mathbb{R}^{n\times q}$ where $(\boldsymbol{E}, \Lambda, \lambda_{q+1})$ is the top-q eigensystem of $K(X, X)$.

Proposition 3. *The following update equations*

$$\boldsymbol{v}_t \leftarrow K(X_m, X)\boldsymbol{\beta}_t - Y[B] \in \mathbb{R}^m \tag{19a}$$
$$\boldsymbol{w}_t \leftarrow \boldsymbol{F}\boldsymbol{F}^\top \boldsymbol{H}_B^\top \boldsymbol{v}_t \in \mathbb{R}^n \tag{19b}$$
$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_1 \boldsymbol{H}_B^\top \boldsymbol{v}_t + \eta_1 \boldsymbol{w}_t \tag{19c}$$
$$\boldsymbol{\beta}_{t+1} \leftarrow (1+\gamma)\boldsymbol{\alpha}_{t+1} - \gamma\boldsymbol{\alpha}_t + \eta_2 \boldsymbol{H}_B^\top \boldsymbol{v}_t - \eta_2 \boldsymbol{w}_t \tag{19d}$$

emulate the updates in equation (18) *via the relation* $f_t := S^*\boldsymbol{\alpha}_t$ *and* $g_t := S^*\boldsymbol{\beta}_t$.

Remark 4. In Algorithm 3, lines (11-13) together implement equation (19c), whereas lines (14-16) together implement equation (19d). Note that here we have used the fact that $\boldsymbol{H}_B\boldsymbol{F} = \boldsymbol{F}[B] \in \mathbb{R}^{m\times q}$.

3.4 AxlePro 2: Approximating preconditioner via a Nyström extension

Suppose J is a subset of s distinct indices from $\{1, 2, \ldots, n\}$. Define the approximate preconditioner

$$\mathcal{Q} := \mathcal{I} - \sum_{i=1}^{q} \left(1 - \tfrac{\delta_{q+1}}{\delta_i}\right) \phi_i \otimes \phi_i. \tag{20}$$

where $(\phi_i, \frac{\delta_i}{s})$ are eigenpairs of $\mathcal{K}_J$, and $(\boldsymbol{d}_i, \delta_i)$ are eigenpairs of $K(X[J], X[J])$, with $\|\phi_i\|_{\mathbb{H}} = 1$. Note that we still have the relation $\phi_i = \frac{S_J^*\boldsymbol{d}_i}{\sqrt{\delta_i}}$.

Define $\boldsymbol{G} = \boldsymbol{D}\sqrt{\Delta^{-1}(I_q - \delta_{q+1}\Delta^{-1})} \in \mathbb{R}^{s\times q}$ where $(\boldsymbol{D}, \Delta, \delta_{q+1})$ is the top-q eigensystem of $K(X[J], X[J])$.

Proposition 4. *The following update equations*

$$\boldsymbol{v}_t \leftarrow K(X[B], X)\boldsymbol{\beta}_t - Y[B] \in \mathbb{R}^m \tag{21a}$$
$$\boldsymbol{w}_t \leftarrow \boldsymbol{G}\boldsymbol{G}^\top K(X[J], X[B])\boldsymbol{v}_t \in \mathbb{R}^s \tag{21b}$$
$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_1 \boldsymbol{H}_B{}^\top \boldsymbol{v}_t + \eta_1 \boldsymbol{H}_J{}^\top \boldsymbol{w}_t \tag{21c}$$
$$\boldsymbol{\beta}_{t+1} \leftarrow (1+\gamma)\boldsymbol{\alpha}_{t+1} - \gamma\boldsymbol{\alpha}_t + \eta_2 \boldsymbol{H}_B^\top \boldsymbol{v}_t - \eta_2 \boldsymbol{H}_J^\top \boldsymbol{w}_t \tag{21d}$$

emulate the updates (18) *with* $\mathcal{P}$ *replaced by* $\mathcal{Q}$ *via the relation* $f_t := S^*\boldsymbol{\alpha}_t$ *and* $g_t := S^*\boldsymbol{\beta}_t$.

Remark 5. In Algorithm 1, lines 12-14 together implement equation (21c), whereas lines 21-17 together implement equation (21d).

3.5 AxlePro 4: Training general kernel models

Since the model is constrained to be in a subspace $\mathcal{M} = \{K(\cdot, z_i)\}_{i=1}^{p} \subset \mathbb{H}$, we must apply projected PSGD with momentum acceleration.

However, an approximate projection costs $O(p^2)$ for a model with p centers, whereas the PSGD step with momentum costs $O(mp)$ for a batch of size m. To amortize the cost of projection, we delay the projection step until after $T = \frac{p}{m}$ steps. This leads to the per iteration goes to become $O(mp)$ on average.

Algorithm 2 AxlePro 4

```
1: Input: positive definite kernel $K$, batch size $m$, size of approximate preconditioner $s$, learning rates $\eta_1, \eta_2 > 0$, damping factor $\gamma \in (0,1)$, model centers $Z = \{z_i\}_{i=1}^p$.
2: Output: $f : \mathcal{X} \to \mathbb{R}$ solving (11) approximately.
3: setup: Sample $J \subset \{1, 2, \dots, n\}$ with $|J| = s$.
4: $(\boldsymbol{D}, \Delta, \delta_{q+1}) \leftarrow$ top-$q$ eigensystem of $K(X[J], X[J])$
5: $\boldsymbol{G} \leftarrow \boldsymbol{D}\sqrt{\Delta^{-1}(I_q - \delta_{q+1}\Delta^{-1})} \in \mathbb{R}^{s\times q}$
6: Initialize $\boldsymbol{\alpha} = \boldsymbol{\beta} = \mathbf{0}_p$, $\boldsymbol{a} = \boldsymbol{b} = \mathbf{0}_n$, $\boldsymbol{c} = \boldsymbol{d} = \mathbf{0}_s$.
7: repeat
8:     Fetch batch of indices $B \subset \{1,..,n\}$, $|B| = m$
9:     $\boldsymbol{v} \leftarrow K(X[B], Z)\boldsymbol{\beta} - Y[B] \in \mathbb{R}^m$
10:    $\boldsymbol{v} \leftarrow \boldsymbol{v} + K(X[B], X)\boldsymbol{b} + K(X[B], X[J])\boldsymbol{d}$
11:    $\boldsymbol{w} \leftarrow \boldsymbol{G}\boldsymbol{G}^\top K(X[J], X[B])\boldsymbol{v} \in \mathbb{R}^s$
12:    $\widetilde{\boldsymbol{\alpha}} \leftarrow \boldsymbol{\alpha}$
13:    $\widetilde{\boldsymbol{a}} \leftarrow \boldsymbol{a}$
14:    $\widetilde{\boldsymbol{c}} \leftarrow \boldsymbol{c}$
15:    $\boldsymbol{\alpha} \leftarrow \boldsymbol{\beta}$
16:    $\boldsymbol{\beta} \leftarrow (1+\gamma)\boldsymbol{\alpha} - \gamma\widetilde{\boldsymbol{\alpha}}$
17:    $\boldsymbol{c} \leftarrow \boldsymbol{d} + \eta_1\boldsymbol{w}$
18:    $\boldsymbol{d} \leftarrow (1+\gamma)\boldsymbol{c} - \gamma\widetilde{\boldsymbol{c}} - \eta_2\boldsymbol{w}$
19:    $\boldsymbol{a} \leftarrow \boldsymbol{b}$
20:    $\boldsymbol{a}[B] \leftarrow \boldsymbol{a}[B] - \eta_1\boldsymbol{v}$
21:    $\boldsymbol{b} \leftarrow (1+\gamma)\boldsymbol{a} - \gamma\widetilde{\boldsymbol{a}}$
22:    $\boldsymbol{b}[B] \leftarrow \boldsymbol{b}[B] + \eta_2\boldsymbol{v}$
23:    if projection condition holds then
24:        $A \leftarrow K(Z, X)\boldsymbol{a} + K(Z, X[J])\boldsymbol{c} \in \mathbb{R}^p$
25:        $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \texttt{solve}(K, Z, A)$
26:        $C \leftarrow K(Z, X)\boldsymbol{b} + K(Z, X[J])\boldsymbol{d} \in \mathbb{R}^n$
27:        $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \texttt{solve}(K, Z, C)$
28:        Reset $\boldsymbol{a} = \boldsymbol{b} = \mathbf{0}_n$, $\boldsymbol{c} = \boldsymbol{d} = \mathbf{0}_s$
29:    end if
30: until Stopping criterion is reached
31: return $f(x) = \sum_{i=1}^n \alpha_i K(x, z_i)$
```

This leads to the following update equations. For $t = kT + 1, kT + 2, \dots, (k+1)T - 1$

$$f_{t+1} = g_t - \eta_1 \mathcal{Q}\widetilde{\nabla}\mathsf{L}(g_t) \tag{22a}$$

$$g_{t+1} = (1+\gamma)f_{t+1} - \gamma f_t + \eta_2 \mathcal{Q}\widetilde{\nabla}\mathsf{L}(g_t) \tag{22b}$$

whereas for $t = kT$, we perform the step

$$f_{kT} = \arg\min_{f\in\mathcal{M}} \|f - f_{kT-1}\|_{\mathbb{H}}^2 \tag{23a}$$

$$g_{kT} = \arg\min_{f\in\mathcal{M}} \|g - g_{kT-1}\|_{\mathbb{H}}^2 \tag{23b}$$

Proposition 5. *Algorithm 2 emulates equations* (22) *and* (23) *with* $f_0 = g_0 = \mathbf{0}_{\mathbb{H}}$.

The proof of this proposition has been provided in the Appendix for completeness. Note however, that f_t and g_t for $t \neq kT$ also have components that are not in $\mathcal{M}$. However they are in $\mathsf{span}(\{K(\cdot, x_i)\}_{i=1}^n)$, and hence can be tracked through the updates.

4 CONVERGENCE ANALYSIS

We provide a convergence for Algorithm 1. Proofs of intermediate results are in the Appendix.

We need to define a few quantities to describe the convergence properties of Algorithm 1. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$ be the eigenvalues of $K(X, X)$. Let $\boldsymbol{e}_i = (e_{i1}, e_{i2}, \dots, e_{in}) \in \mathbb{R}^n$ be the eigenvector of $K(X, X)$ corresponding to eigenvalue λ_i. Define

$$L_1 := \max_{j\in[n]} K(x_j, x_j) - \sum_{i=1}^q e_{ij}^2(\lambda_i - \lambda_{q+1}) \tag{24a}$$

$$L_m := L_1/m + (m-1)\lambda_{q+1}/m \tag{24b}$$

$$\kappa_m := L_m/\lambda_n \tag{24c}$$

$$\widetilde{\kappa}_m := n/m + (m-1)/m \tag{24d}$$

Consider hyperparameters,

$$\eta_1^* = \frac{1}{L_m} \tag{25a}$$

$$\eta_2^* = \frac{\eta_1^*(m)\sqrt{\kappa_m\widetilde{\kappa}_m}}{\sqrt{\kappa_m\widetilde{\kappa}_m} + 1}\left(1 - \frac{1}{\widetilde{\kappa}_m}\right) \tag{25b}$$

$$\gamma^* = \frac{\sqrt{\kappa_m\widetilde{\kappa}_m} - 1}{\sqrt{\kappa_m\widetilde{\kappa}_m} + 1} \tag{25c}$$

Let $\varepsilon > 0$ and $\delta \in (0,1)$ be parameters for controlling the approximation error of Nyström approximate preconditioner defined in equation (20).

Theorem 6. *Fix $\varepsilon > 0$ and $\delta \in (0,1)$. Suppose X consists of i.i.d. samples from an arbitrary distribution on $\mathbb{X}$, and J consists of distinct indices chosen from of $\{1, 2, \dots, n\}$, independent of X with*

$$|J| \geq c \cdot \frac{\log^4(1+n)}{\varepsilon^4}\log\frac{4}{\delta}.$$

for some universal constant c. Now, suppose Algorithm 1 is run for t iterations with hyperparameters given by equation (25)*. Then, with probability at least $1-\delta$ over the randomness in X, we have the following upper bound for a constant C,*

$$\|f_t - f^*\|_{\mathbb{H}}^2 \leq C \cdot \exp\left(\frac{-t/(1+\varepsilon)^2}{\sqrt{\widetilde{\kappa}_m\kappa_m}}\right). \tag{26}$$

Proof of Theorem 6. Note that by Proposition 4, Algorithm 1 is emulating the updates

$$f_{t+1} \leftarrow g_t - \eta_1 \mathcal{Q}\widetilde{\nabla}_f\mathsf{L}(g_t), \tag{27a}$$

$$g_{t+1} \leftarrow (1+\gamma)f_{t+1} - \gamma f_t + \eta_2 \mathcal{Q}\widetilde{\nabla}_f\mathsf{L}(g_t). \tag{27b}$$

Rewriting the above in terms of $\widetilde{\nabla}\mathsf{L}(f_t) = \mathcal{K}_{B_t}(f_t - f^*)$ from (82), and subtracting f^* on both sides in the updates (27), we get

$$\begin{aligned} f_{t+1} - f^* &\leftarrow g_t - f^* - \eta_1 \mathcal{Q}\mathcal{K}_{B_t}(g_t - f^*), \quad (28a)\\ g_{t+1} - f^* &\leftarrow (1+\gamma)(f_{t+1} - f^*) - \gamma(f_t - f^*)\\ &\quad + \eta_2 \mathcal{Q}\mathcal{K}_{B_t}(g_t - f^*). \quad (28b)\end{aligned}$$

To proceed, we need the following technical lemma.

Lemma 7. *Consider the data dependent kernel*

$$\widehat{K}_q(x,z) = K(x,z) - K(x, X[J])\boldsymbol{G}\boldsymbol{G}^\top K(X[J], z)$$

and $\widehat{\mathbb{H}}_q$ be the RKHS associated with $\widehat{K}_q$. Define the operator $T : \mathbb{H} \to \widehat{\mathbb{H}}_q$ such that $T(K(\cdot,x)) = \widehat{K}_q(\cdot,x)$.

(a) $T^*f = f_{\mathbb{H}}$, where $f_{\mathbb{H}} \in \mathbb{H}$ is pointwise the same function as $f \in \widehat{\mathbb{H}}_q$.

(b) $T^*T = \mathcal{Q}$

(c) $R = ST^*$ is the sampling operator $\widehat{\mathbb{H}}_q \to \mathbb{R}^n$ such that $R\widehat{f} = \widehat{f}(X) \in \mathbb{R}^n$ for all $\widehat{f} \in \widehat{\mathbb{H}}_q$.

(d) $\widehat{\mathbb{H}}_q$ is equivalent to $\mathbb{H}$, i.e., there exist constants a, a' such that $a\|f\|_{\mathbb{H}} \le \left\|\widehat{f}\right\|_{\widehat{\mathbb{H}}_q} \le a'\|f\|_{\mathbb{H}}$.

(e) $T\mathcal{K}T^* = \frac{1}{n}R^*R$ is the empirical covariance operator of the kernel $\widehat{K}_q$, and $R^*R = \widehat{K}_q(X,X)$.

Proof. (a) Note that by definition

$$\left\langle TK(\cdot,x), \widehat{K}_q(\cdot,z)\right\rangle_{\widehat{\mathbb{H}}_q} = (TK(\cdot,x))(z) = \widehat{K}_q(z,x)$$
$$\left\langle K(\cdot,x), T^*\widehat{K}_q(\cdot,z)\right\rangle_{\mathbb{H}} = \left(T^*\widehat{K}_q(\cdot,z)\right)(x)$$

which proves the claim in (a). Parts (b), (c), (e) follow immediately. Proof of (d) is in the Appendix. ☐

Defining $\widehat{f}_t := T^{*-1}(f_t - f^*)$ and $\widehat{g}_t := T^{*-1}(g_t - f^*)$, and using the fact that $\mathcal{Q} = T^*T$, we can write (28) as

$$\widehat{f}_{t+1} \leftarrow \widehat{g}_t - \eta_1 T\mathcal{K}_{B_t}T^*\widehat{g}_t, \quad (29a)$$
$$\widehat{g}_{t+1} \leftarrow (1+\gamma)\widehat{f}_{t+1} - \gamma\widehat{f}_t + \eta_2 T\mathcal{K}_{B_t}T^*\widehat{g}_t. \quad (29b)$$

Note that in equation (29), we have that $\mathbb{E}_{B_t} T\mathcal{K}_{B_t}T^* = T\mathcal{K}T^* = \frac{1}{n}R^*R$. Hence the above iteration is (15) specialized to solve

$$\min_{f\in\widehat{\mathbb{H}}_q} \tfrac{1}{2}\|Rf - Y\|^2. \quad (30)$$

however in this case the stochastic gradients will be $\frac{1}{|B_t|}R^*_{B_t}R_{B_t} = T\mathcal{K}_{B_t}T^*$, and expected Hessian $T\mathcal{K}T^*$.

Specializing the convergence result of [20, Thm. 2] for the case of square loss and search space $\widehat{\mathbb{H}}_q$, we can conclude that for hyperparameters in equations (25), updates (29) converge as

$$\left\|\widehat{f}_t\right\|^2_{\widehat{\mathbb{H}}_q} \le \exp\left(-t/\sqrt{\widetilde{c}_m c_m}\right)\left\|T^{*-1}f^*\right\|^2_{\widehat{\mathbb{H}}_q} \quad (31)$$

where

$$\widetilde{c}_m = \frac{1}{m}\cdot\mathbb{E}\left\|\widehat{K}_q(\cdot,x)\right\|_{(T\mathcal{K}_qT^*)^{-1}} + \frac{m-1}{m} \quad (32a)$$
$$c_m = d_m/\lambda_n(R^*R) \quad (32b)$$
$$d_m = d_1/m + (m-1)\lambda_1(R^*R)/m \quad (32c)$$
$$d_1 = \max_{\widehat{f}\in\mathsf{range}(R^*)}\left\|\widehat{f}\right\|^2_{\widehat{\mathbb{H}}_q} \quad (32d)$$

We also have the following result.

Proposition 8 ([3, Thm. 2])**.** *Let X and J satisfy assumptions stated in Theorem 6, then with probability at least $1-\delta$ over the randomness in X and J, the following inequalities hold*

$$\lambda_1(\mathcal{R}) \le (1+\varepsilon)^2\lambda_1(\mathcal{P}\mathcal{K}) = n(1+\varepsilon)^2\lambda_{q+1}, \quad (33a)$$
$$\lambda_n(R^*R) \ge (1+\varepsilon)^2\lambda_n(\mathcal{P}\mathcal{K}) = n\frac{1}{(1+\varepsilon)^2}\lambda_n. \quad (33b)$$

Due to the above proposition, we have that $c_m \le (1+\varepsilon)^4\kappa_m$. In the Appendix, we show that $d_1 = L_1$ whereby $d_m = L_m$ and we also show that $\widetilde{c}_m = \widetilde{\kappa}_m$, which leads to the bound

$$\|f_t - f^*\|^2_{\widehat{\mathbb{H}}_q} \le \exp\left(\frac{-t/(1+\varepsilon)^2}{\sqrt{\widetilde{\kappa}_m\kappa_m}}\right)\left\|T^{*-1}f^*\right\|^2_{\widehat{\mathbb{H}}_q}. \quad (34)$$

Eq. (26) follows due to equivalence of $\mathbb{H}$ and $\widehat{\mathbb{H}}_q$. ☐

5 NUMERICAL EXPERIMENTS

Due to space limitations, only a few key experiments are presented in the main paper while other variants are provided in the Appendix.

All experiments are solving equation (2) and show the mean squared error (MSE) over the training dataset. Note that all plots use a log-scale on the Y-axis. Experiments were run on a machine with 32GB RAM, 1 NVIDIA V100 SMX2 with 32GB VRAM, and 1 Xeon Gold 6248 CPU.

Our experiments were conducted on the following datasets: CIFAR-10, Stellar Classification, and EMNIST Digits. We test the performance for both Gaussian kernel and Laplacian kernel. For CIFAR-10 dataset, we also compare the performance with Myrtle-5 kernel [32], which is the state-of-the-art kernel for this dataset. Experiment details are in Appendix D.

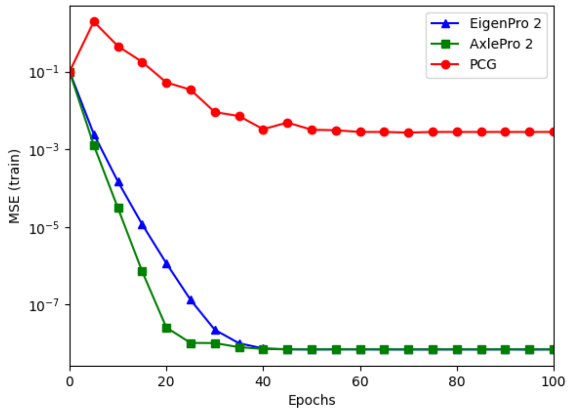

Figure 2: **Numerical stability to low precision.** Convergence comparison on CIFAR10 dataset with Laplacian kernel over single precision arithmetic.

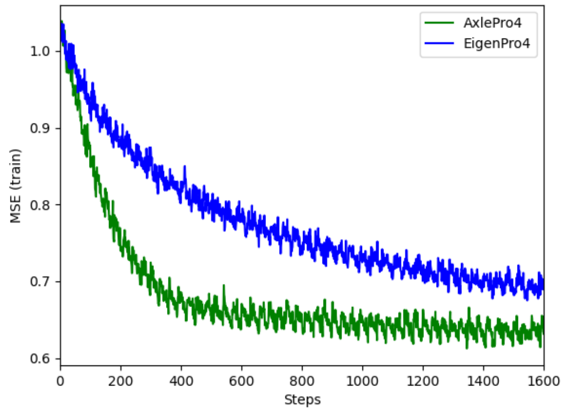

Figure 3: Progress of Algorithm 2 v/s EigenPro 4 from [2] with respect to number of steps on randomly generated dataset ($n = 6000, p = 3000$).

The batch size is set to be the same for AxlePro and EigenPro. This batch size is optimized to minimize total computation time by maximizing GPU utilization, see [23] for a detailed discussion on this topic.

For AxlePro, the only hyperparameter needs to be tuned is the smallest eigenvalue of the kernel matrix due to the Nyström approximation and we chose this factor from according to the datasets.

We follow the standard PCGalgorithm used in [15], which computes the precondition matrix based on the pivoted Cholesky decomposition [6,17]. And we set the rank of approximation to be 300.

Dataset	Algorithm	Gaussian	Laplacian
CIFAR10	AxlePro	48.35%	**58.84%**
$n = 50,000$	EigenPro	**49.38%**	**58.84%**
$d = 3072$	CG	28.35%	54.11%
	PCG	45.32%	**58.84%**
	Falkon	**53.37%**	56.47%
	GPyTorch	10.00%	-
Stellar	AxlePro	**95.92%**	94.84%
Classification	EigenPro	95.42%	94.84%
$n = 95,000$	CG	61.78%	94.52%
$d = 13$	PCG	19.68%	94.84%
	Falkon	95.80%	**94.90%**
	GPyTorch	92.10%	-
EMNIST	AxlePro	**99.40%**	**99.12%**
digits	EigenPro	**99.40%**	**99.12%**
n=240,000	CG	95.78%	99.15%
$d = 784$	PCG	98.56%	**99.12%**
	Falkon	98.94%	98.45%
	GPyTorch	87.31%	-

Table 3: Performance comparison in terms of test accuracy between different methods for training kernel models with Gaussian and Laplacian kernels

6 DISCUSSION

In this paper we presented a derivation for an algorithm AxlePro, for training kernel models using a momentum accelerated version of preconditioned SGD in the RKHS. We provided a convergence guarantee in Theorem 6, and compared the performance with other algorithms.

While AxlePro emulates accelerated PSGD with approximate preconditioning, other iterative updates such as ADAM and its variants cannot be emulated as elegantly via finite dimensional updates. This is primarily because we use a spectral preconditioner among other options available for preconditioning. Deriving other emulated algorithms would lead us to the optimal performance for training kernel methods. Extending this work beyond the square loss would also be useful.

References

[1] Amirhesam Abedsoltan, Mikhail Belkin, and Parthe Pandit. Toward large kernel models. *arXiv preprint arXiv:2302.02605*, 2023.

[2] Amirhesam Abedsoltan, Siyuan Ma, Parthe Pandit, and Mikhail Belkin. Eigenpro4. github.com/EigenPro/EigenPro, 2024.

[3] Amirhesam Abedsoltan, Parthe Pandit, Luis Rademacher, and Mikhail Belkin. On the nystrom approximation for preconditioning in kernel machines. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2024.

[4] Ben Adlam, Jaehoon Lee, Shreyas Padhy, Zachary Nado, and Jasper Snoek. Kernel regression with infinite-width neural networks on millions of examples. *arXiv preprint arXiv:2303.05420*, 2023.

[5] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *Advances in neural information processing systems*, 32, 2019.

[6] Francis Bach. Sharp analysis of low-rank kernel matrix approximations. In *Conference on learning theory*, pages 185–209. PMLR, 2013.

[7] Daniel Beaglehole, Adityanarayanan Radhakrishnan, Parthe Pandit, and Mikhail Belkin. Mechanism of feature learning in convolutional neural networks. *arXiv preprint arXiv:2309.00570*, 2023.

[8] Alberto Bietti and Francis Bach. Deep equals shallow for relu networks in kernel regimes. *arXiv preprint arXiv:2009.14397*, 2020.

[9] Raffaello Camoriano, Tomás Angles, Alessandro Rudi, and Lorenzo Rosasco. Nytro: When subsampling meets early stopping. In *Artificial Intelligence and Statistics*, pages 1403–1411. PMLR, 2016.

[10] Luigi Carratino, Alessandro Rudi, and Lorenzo Rosasco. Learning with sgd and random features. *Advances in Neural Information Processing Systems*, 31, 2018.

[11] Yifan Chen, Ethan N Epperly, Joel A Tropp, and Robert J Webber. Randomly pivoted cholesky: Practical approximation of a kernel matrix with few entry evaluations. *arXiv preprint arXiv:2207.06503*, 2022.

[12] Agniva Chowdhury, Jiasen Yang, and Petros Drineas. An iterative, sketching-based framework for ridge regression. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 989–998. PMLR, 10–15 Jul 2018.

[13] Kurt Cutajar, Michael Osborne, John Cunningham, and Maurizio Filippone. Preconditioning kernel matrices. In *International conference on machine learning*, pages 2529–2538. PMLR, 2016.

[14] Mateo Díaz, Ethan N Epperly, Zachary Frangella, Joel A Tropp, and Robert J Webber. Robust, randomized preconditioning for kernel ridge regression. *arXiv preprint arXiv:2304.12465*, 2023.

[15] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.

[16] Insu Han, Amir Zandieh, Jaehoon Lee, Roman Novak, Lechao Xiao, and Amin Karbasi. Fast neural kernel embeddings for general activations. *Advances in neural information processing systems*, 35:35657–35671, 2022.

[17] Helmut Harbrecht, Michael Peters, and Reinhold Schneider. On the low-rank approximation by the pivoted cholesky decomposition. *Applied Numerical Mathematics*, 62(4):428–440, 2012. Third Chilean Workshop on Numerical Analysis of Partial Differential Equations (WONAPDE 2010).

[18] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[19] George S Kimeldorf and Grace Wahba. A correspondence between bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, 41(2):495–502, 1970.

[20] Chaoyue Liu and Mikhail Belkin. Accelerating sgd with momentum for over-parameterized learning. In *International Conference on Learning Representations*, 2020.

[21] Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International Conference on Machine Learning*, pages 3325–3334. PMLR, 2018.

[22] Siyuan Ma and Mikhail Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. *Advances in neural information processing systems*, 30, 2017.

[23] Siyuan Ma and Mikhail Belkin. Kernel machines that adapt to gpus for effective large batch train-

ing. *Proceedings of Machine Learning and Systems*, 1:360–373, 2019.

[24] Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. Kernel methods through the roof: handling billions of points efficiently. *Advances in Neural Information Processing Systems*, 33:14410–14422, 2020.

[25] Adityanarayanan Radhakrishnan, Daniel Beaglehole, Parthe Pandit, and Mikhail Belkin. Feature learning in neural networks and kernel machines that recursively learn features. *arXiv preprint arXiv:2212.13881*, 2022.

[26] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[27] Pratik Rathore, Zachary Frangella, and Madeleine Udell. Have askotch: Fast methods for large-scale, memory-constrained kernel ridge regression. *arXiv preprint arXiv:2407.10070*, 2024.

[28] Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. Less is more: Nyström computational regularization. *Advances in neural information processing systems*, 28, 2015.

[29] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. *Advances in neural information processing systems*, 30, 2017.

[30] Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

[31] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, pages 807–814, 2007.

[32] Vaishaal Shankar, Alex Fang, Wenshuo Guo, Sara Fridovich-Keil, Jonathan Ragan-Kelley, Ludwig Schmidt, and Benjamin Recht. Neural kernels without tangents. In *International conference on machine learning*, pages 8614–8623. PMLR, 2020.

[33] Kaiwen Wu, Jonathan Wenger, Haydn T Jones, Geoff Pleiss, and Jacob Gardner. Large-scale gaussian processes via alternating projection. In *International Conference on Artificial Intelligence and Statistics*, pages 2620–2628. PMLR, 2024.

[34] Rong Yin, Weiping Wang, and Dan Meng. Distributed nyström kernel learning with communications. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12019–12028. PMLR, 18–24 Jul 2021.