
AxlePro: Momentum-Accelerated Batched Training of Kernel Machines

Yiming Zhang
Mathematics, UC San Diego

Parthe Pandit
C-MInDS, IIT Bombay

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 speed-up 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 modeling 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 Jacot et al. (2018), 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 Adlam et al. (2023); Arora et al. (2019); Bietti and Bach (2020); Han et al. (2022); Shankar et al. (2020) among others. Furthermore, recent work also shows that kernel machines can be re-engineered to learn task-specific features in a supervised manner Beaglehole et al. (2023); Radhakrishnan et al. (2022), which leads to further performance gains. Learning a kernel machine in a scalable manner

thus remains an important fundamental problem in machine learning.

Formally, kernel machines are functions of the form

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

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

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

$$\text{K_solve}(K, X, Y) : \text{Find } \alpha : K(X, X)\alpha = Y, \quad (1)$$

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 , (KM) with weights given by $\text{K_solve}(K, X, Y)$ is the minimum \mathbb{H} -norm solution to the least squares problem,

$$\min_{f \in \mathbb{H}} \mathcal{L}(f) := \frac{1}{2} \sum_{i=1}^n (f(x_i) - y_i)^2. \quad (2)$$

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 (1) subsumes the important case of kernel ridge regression. This somewhat easier case is equivalent to choosing a modified kernel $\tilde{K}(x, z) = K(x, z) + \lambda \cdot \mathbf{1}_{\{x=z\}}$, where $\lambda > 0$ is a tunable parameter of the ridge penalty, since our framework allows arbitrary kernels, even discontinuous ones.

1.1 Main contributions

¹ 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

Algorithm	precond ^p	momentum	batching	stable [‡]	order	complexity ^c	mem [*]
PCG Gardner et al. (2018)	✗	✓	✗	✗		$n^2 \cdot \sqrt{\kappa}$	n
EigenPro Ma and Belkin (2017)	✗	✗	✓	✓	1 st	$nm \cdot \kappa_m$	n
EigenPro 2 Ma and Belkin (2019)	✓	✗	✓	✓	1 st	$nm \cdot \kappa_m$	$\log^4 n$
ASkotch Rathore et al. (2024)	✗	✓	✓		2 nd	$(nm + m^2)\sqrt{\kappa_m}$	m
Alt.-proj. Wu et al. (2024)	✗	✗	✓		2 nd	$m^3 \cdot \kappa_m$	m^2
Algorithm 1 (ours)	✓	✓	✓	✓	1 st	$nm \cdot \sqrt{\kappa_m}$	$\log^4 n$

Table 1: Iterative algorithms for solving equation (1). m is the batch size. Condition number κ is the ratio of largest and smallest positive eigenvalues of $K(X, X)$, $\kappa_m, \tilde{\kappa}_m$ are modified versions of κ defined in equation (22), with $\kappa_m \leq \kappa$ depending on m in a data-dependent manner.

^pBy 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 $\lambda = 0^+$).

^cTotal complexities in big-O notation, product of per iteration complexity times number of iterations.

^{*}memory overheads due to preconditioning.

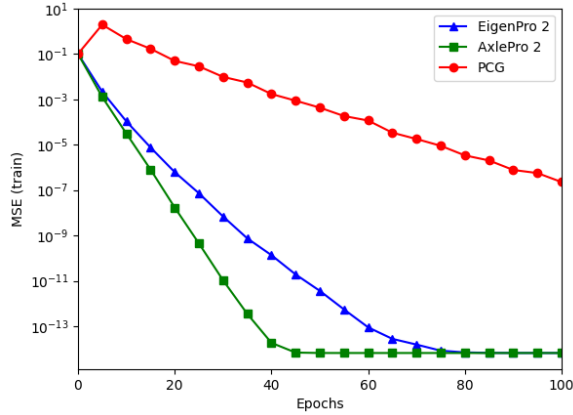


Figure 1: Comparison of iterative algorithms for the Laplace Kernel on CIFAR-10

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* [Abedsoltan et al. \(2023\)](#) (see Section 2).

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 [Meanti et al. \(2020\)](#); [Rudi et al. \(2017\)](#); [Yin et al. \(2021\)](#) for example. While these methods are the fastest in the asymptotic sense, they do not decrease training error monotonically, and

Algorithm 1 AxlePro 2

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

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 decompositions provide a way to train in a scalable manner [Carratino et al. \(2018\)](#); [Chen et al. \(2022\)](#); [Rahimi and Recht \(2007\)](#). See [Han et al. \(2022\)](#) and the references therein

Algorithm	precond ^p	momentum	batching	stable [‡]	order	flops ^c /batch	mem [*]
Cholesky Rudi et al. (2015)	✗	✓	✗	✗	2 nd	$p^2m + p^3$	p^2
FALKON Rudi et al. (2017)	✗	✗	✗	✓	2 nd	$pm + p^2$	p^2
EigenPro3 Abedsoltan et al. (2023)	✓	✗	✓	✓	1 st	$pm + p^2$	$\log^4 n$
EigenPro4 Abedsoltan et al. (2024a)	✓	✗	✓	✓	1 st	pm	$\log^4 n$
SketchySAGA Rathore et al. (2024)	✓	✓	✓		2 nd	$pm + p^2$	pm
Algorithm 2 (ours)	✓	✓	✓	✓	1 st	pm	$\log^4 n$

Table 2: Comparison between different iterative algorithms for solving (8). m is the batch size, typically $m \ll n$. Condition number κ is the ratio of largest to smallest positive eigenvalue, $\kappa_m, \tilde{\kappa}_m$ are modified versions of κ defined in equation (22), with $\tilde{\kappa}_m \leq \kappa_m \leq \kappa$.

^pBy fast preconditioning we mean the preconditioner must be calculated once and should be of size $o(n)$.

[‡]Stability to single precision computations.

^cnumber of flops/sample in big-O notation, product of per iteration complexity times number of iterations.

^{*}memory overheads due to preconditioning.

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 Chowdhury et al. (2018).

Primal space solvers such as SGD Camoriano et al. (2016); Shalev-Shwartz et al. (2007) and its variants can be very effective, and are universal, i.e., apply for any kernel. In fact Abedsoltan et al. (2023, 2024a); Ma and Belkin (2017, 2019) 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 . Our algorithm AxlePro falls in this category.

Second order methods with acceleration have also been considered recently in Rathore et al. (2024) and Wu et al. (2024) 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 (Gen-KM), defined formally in Section 2, has been an important way to train kernel machines on arbitrarily large datasets. This has become important since Abedsoltan et al. (2023) 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 given by (Gen-KM) are the preferred way to scale RKHS methods to large datasets with several new implementations. For example Abedsoltan et al. (2023, 2024a); Meanti et al. (2020); Rathore et al. (2024) 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 (KM) arise as the solutions to empirical risk minimization problems over an RKHS corresponding to K ,

$$\min_{f \in \mathbb{H}} \mathbb{L}(x_1, y_1, f(x_1), \dots, x_n, y_n, f(x_n)) + \lambda \cdot \mathbb{R}(\|f\|_{\mathbb{H}}) \quad (3)$$

due to the Representer Theorem Kimeldorf and Wahba (1970); Schölkopf et al. (2001). This theorem requires the mild condition that \mathbb{R} is an increasing function and $\lambda > 0$. The reproducing property of K , means we have,

$$\langle f, K(\cdot, x) \rangle_{\mathbb{H}} = f(x), \quad \forall f \in \mathbb{H}, \forall x \in \mathbb{X}, \quad (4)$$

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 $\mathbb{H} \rightarrow \mathbb{H}$, 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 (4).

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

$$Sf = (f(x_1), \dots, f(x_n)) \in \mathbb{R}^n. \quad (5)$$

One can show that the adjoint of S , defined in equa-

tion (5), denoted $S^* : \mathbb{R}^n \rightarrow \mathbb{H}$, acts as

$$S^* \alpha = \sum_{i=1}^n K(\cdot, x_i) \alpha_i \in \mathbb{H}, \quad \forall \alpha \in \mathbb{R}^n. \quad (6)$$

We define the empirical covariance operator,

$$\mathcal{K} := \frac{1}{n} S^* S = \frac{1}{|X|} \sum_{x \in X} K(x, \cdot) \otimes_{\mathbb{H}} K(x, \cdot). \quad (7)$$

Slicing. For an n -tuple $X = (x_1, x_2, \dots, x_n)$ of size n and an s -tuple of indices $J = (j_1, j_2, \dots, j_s)$, we denote by $X[J]$ the s -tuple $(x_{j_1}, x_{j_2}, \dots, x_{j_s})$, following `python` notation. For a matrix \mathbf{A} and vector \mathbf{v} , the quantities $\mathbf{A}[J]$ and $\mathbf{v}[J]$ are the submatrix and subvector of \mathbf{A} and \mathbf{v} respectively indexed by J . By \mathbf{H}_J we denote the $|J| \times n$ matrix formed by stacking the $|J|$ rows of identity I_n . By $S_J := \mathbf{H}_J S$ we denote the operator which maps $\mathbb{H} \rightarrow \mathbb{R}^{|J|}$ so that $S_J f = (f(x_{j_1}), f(x_{j_2}), \dots, f(x_{j_s}))$. Similarly, define $\mathcal{K}_J = \frac{1}{|J|} S_J^* S_J$.

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

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

$$f(x) = \sum_{i=1}^p \alpha_i K(x, z_i), \quad (\text{Gen-KM})$$

where $z_i \in \mathbb{R}^d$ are p landmarks or inducing points or model centers in the domain \mathbb{X} . These are better suited when training on very large datasets, i.e., when $n \geq 1$ billion samples [Meanti et al. \(2020\)](#). Such models have received renewed attention [Abedsoltan et al. \(2023, 2024a\)](#); [Rathore et al. \(2024\)](#). They can be trained via

$$\min_{f \in \mathbb{H}} \sum_{i=1}^n (f(x_i) - y_i)^2 \quad \text{subject to (Gen-KM)}, \quad (8)$$

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 [Ma and Belkin \(2017\)](#). We then derive AxlePro given in Algorithm 3, which extends EigenPro by applying a specific momentum acceleration inspired by the MaSS² algorithm from [Liu and Belkin \(2020\)](#). Finally we derive the faster version AxlePro 2 in Algorithm 1, which extends EigenPro 2 [Ma and Belkin \(2019\)](#). Similar to EigenPro 2 our algorithm AxlePro 2 applies an approximate preconditioner in AxlePro obtained via a Nyström approximation [Abedsoltan et al. \(2024b\)](#).

Consider the optimization problem,

$$f^* = \arg \min_{f \in \mathbb{H}} \mathbf{L}(f) = \frac{1}{2n} \|Sf - Y\|^2. \quad (9)$$

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

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

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

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

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

$$\tilde{\nabla} \mathbf{L}(f) = \frac{1}{m} S_B^* (S_B f - Y[B]) = \mathcal{K}_B (f - f^*), \quad (10)$$

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 (9). If we start by initializing $f_0 \in \text{range}(S^*)$, then one can show that the iterates f_t of gradient descent

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

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

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

$$f_{t+1} \leftarrow f_t - \eta_0 \tilde{\nabla}_f \mathbf{L}(f_t). \quad (11)$$

and converges exponentially as shown in [Ma et al. \(2018, Theorem 1\)](#).

²MaSS stands for Momentum-added Stochastic Solver, and it adjusts the momentum updates in the standard Nesterov momentum update of SGD to achieve the fast rate when interpolation is possible, as is typical with overparameterized models. See [Liu and Belkin \(2020\)](#) for a detailed discussion.

Proposition 1. *The following update equations*

$$\mathbf{v}_t \leftarrow K(X[B], X)\boldsymbol{\alpha}_t - Y[B] \quad (12a)$$

$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\alpha}_t - \eta_0 \mathbf{H}_B^\top \mathbf{v}_t. \quad (12b)$$

emulate one step of updates in equation (11) via the relation $\mathbf{f}_t = S^* \boldsymbol{\alpha}_t$.

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 Liu and Belkin (2020) yields

$$\mathbf{f}_{t+1} \leftarrow \mathbf{g}_t - \eta_1 \tilde{\nabla}_f \mathbf{L}(\mathbf{g}_t), \quad (13a)$$

$$\mathbf{g}_{t+1} \leftarrow (1 + \gamma) \mathbf{f}_{t+1} - \gamma \mathbf{f}_t + \eta_2 \tilde{\nabla}_f \mathbf{L}(\mathbf{g}_t), \quad (13b)$$

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 Liu and Belkin (2020, Theorem 2).

Now, suppose $\mathbf{f}_t, \mathbf{g}_t \in \text{range}(S^*)$, then \mathbf{f}_{t+1} and \mathbf{g}_{t+1} are still in $\text{range}(S^*)$, when $\tilde{\nabla}_f \mathbf{L}(\mathbf{f}_t) \in \text{range}(S_B^*)$.

Proposition 2. *The following update equations*

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

$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_1 \mathbf{H}_B^\top \mathbf{v}_t, \quad (14b)$$

$$\boldsymbol{\beta}_{t+1} \leftarrow (1 + \gamma) \boldsymbol{\alpha}_{t+1} - \gamma \boldsymbol{\alpha}_t + \eta_2 \mathbf{H}_B^\top \mathbf{v}_t, \quad (14c)$$

emulate one step of updates in equation (13) via the relation $\mathbf{f}_t := S^* \boldsymbol{\alpha}_t$ and $\mathbf{g}_t := S^* \boldsymbol{\beta}_t$.

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 Cutajar et al. (2016); Díaz et al. (2023). We use the following spectral preconditioner because it is the easiest to approximate elegantly, as shown in Abedsoltan et al. (2024b).

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

$$\mathcal{P} := \mathcal{I} - \sum_{i=1}^q \left(1 - \frac{\lambda_{q+1}}{\lambda_i}\right) \psi_i \otimes_{\mathbb{H}} \psi_i. \quad (15)$$

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 $\mathcal{P}\mathcal{K}$ is λ_{q+1}/n , which governs the convergence rate. Furthermore, their smallest eigenvalues are the same. 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

$$\mathbf{f}_{t+1} \leftarrow \mathbf{g}_t - \eta_1 \mathcal{P} \tilde{\nabla}_f \mathbf{L}(\mathbf{g}_t), \quad (16a)$$

$$\mathbf{g}_{t+1} \leftarrow (1 + \gamma) \mathbf{f}_{t+1} - \gamma \mathbf{f}_t + \eta_2 \mathcal{P} \tilde{\nabla}_f \mathbf{L}(\mathbf{g}_t). \quad (16b)$$

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

Proposition 3. *The following update equations*

$$\mathbf{v}_t \leftarrow K(X_m, X)\boldsymbol{\beta}_t - Y[B] \in \mathbb{R}^m \quad (17a)$$

$$\mathbf{w}_t \leftarrow \mathbf{F} \mathbf{F}^\top \mathbf{H}_B^\top \mathbf{v}_t \in \mathbb{R}^n \quad (17b)$$

$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_1 \mathbf{H}_B^\top \mathbf{v}_t + \eta_1 \mathbf{w}_t \quad (17c)$$

$$\boldsymbol{\beta}_{t+1} \leftarrow (1 + \gamma) \boldsymbol{\alpha}_{t+1} - \gamma \boldsymbol{\alpha}_t + \eta_2 \mathbf{H}_B^\top \mathbf{v}_t - \eta_2 \mathbf{w}_t \quad (17d)$$

emulate the updates in equation (16) via the relation $\mathbf{f}_t := S^* \boldsymbol{\alpha}_t$ and $\mathbf{g}_t := S^* \boldsymbol{\beta}_t$.

Remark 4. In Algorithm 3, lines (11-13) together implement equation (17c), whereas lines (14-16) together implement equation (17d). Note that here we have used the fact that $\mathbf{H}_B \mathbf{F} = \mathbf{F}[B] \in \mathbb{R}^{m \times q}$.

3.4 AxlePro 2: Approximating preconditioner via a Nyström extension

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

$$\mathcal{Q} := \mathcal{I} - \sum_{i=1}^q \left(1 - \frac{\delta_{q+1}}{\delta_i}\right) \phi_i \otimes \phi_i. \quad (18)$$

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

In the context of kernels the preconditioner in (18) is an effective approximation to (15), since it approximates the head of the spectrum well enough. See Abedsoltan et al. (2024b) for a thorough treatment on the approximation error.

Recall Definition 2 of a top- q eigensystem, and define $\mathbf{G} = \mathbf{D} \sqrt{\Delta^{-1}(I_q - \delta_{q+1} \Delta^{-1})} \in \mathbb{R}^{s \times q}$ where $(\mathbf{D}, \Delta, \delta_{q+1})$ is the top- q eigensystem of $K(X[J], X[J])$.

Algorithm 2 AxlePro 4

```

1: Input: positive definite kernel  $K$ , batch size  $m$ ,
   size of approximate preconditioner  $s$ , level of pre-
   conditioner  $q$ , learning rates  $\eta_1, \eta_2 > 0$ , damping
   factor  $\gamma \in (0, 1)$ , model centers  $Z = \{z_i\}_{i=1}^p$ .
2: Output:  $f : \mathcal{X} \rightarrow \mathbb{R}$  solving (9) approximately.
3: setup: Sample  $J \subset \{1, 2, \dots, n\}$  with  $|J| = s$ .
4:  $(\mathbf{D}, \Delta, \delta_{q+1}) \leftarrow$  top- $q$  eigensystem of  $K(X[J], X[J])$ 

5:  $\mathbf{G} \leftarrow \mathbf{D} \sqrt{\Delta^{-1}(I_q - \delta_{q+1}\Delta^{-1})} \in \mathbb{R}^{s \times q}$ 
6: Initialize  $\alpha = \beta = \mathbf{0}_p$ ,  $\mathbf{a} = \mathbf{b} = \mathbf{0}_n$ ,  $\mathbf{c} = \mathbf{d} = \mathbf{0}_s$ .
7: repeat
8:   Fetch batch of indices  $B \subset \{1, \dots, n\}$ ,  $|B| = m$ 
9:    $\mathbf{v} \leftarrow K(X[B], Z)\beta - Y[B] \in \mathbb{R}^m$ 
10:   $\mathbf{v} \leftarrow \mathbf{v} + K(X[B], X)\mathbf{b} + K(X[B], X[J])\mathbf{d}$ 
11:   $\mathbf{w} \leftarrow \mathbf{G}\mathbf{G}^\top K(X[J], X[B])\mathbf{v} \in \mathbb{R}^s$ 
12:   $\tilde{\alpha} \leftarrow \alpha$ 
13:   $\tilde{\mathbf{a}} \leftarrow \mathbf{a}$ 
14:   $\tilde{\mathbf{c}} \leftarrow \mathbf{c}$ 
15:   $\alpha \leftarrow \beta$ 
16:   $\beta \leftarrow (1 + \gamma)\alpha - \gamma\tilde{\alpha}$ 
17:   $\mathbf{c} \leftarrow \mathbf{d} + \eta_1\mathbf{w}$ 
18:   $\mathbf{d} \leftarrow (1 + \gamma)\mathbf{c} - \gamma\tilde{\mathbf{c}} - \eta_2\mathbf{w}$ 
19:   $\mathbf{a} \leftarrow \mathbf{b}$ 
20:   $\mathbf{a}[B] \leftarrow \mathbf{a}[B] - \eta_1\mathbf{v}$ 
21:   $\mathbf{b} \leftarrow (1 + \gamma)\mathbf{a} - \gamma\tilde{\mathbf{a}}$ 
22:   $\mathbf{b}[B] \leftarrow \mathbf{b}[B] + \eta_2\mathbf{v}$ 
23:  if projection condition holds then
24:     $\mathbf{A} \leftarrow K(Z, X)\mathbf{a} + K(Z, X[J])\mathbf{c} \in \mathbb{R}^p$ 
25:     $\alpha \leftarrow \alpha + \text{K.solve}(K, Z, \mathbf{A})$ 
26:     $\mathbf{C} \leftarrow K(Z, X)\mathbf{b} + K(Z, X[J])\mathbf{d} \in \mathbb{R}^n$ 
27:     $\beta \leftarrow \beta + \text{solve}(K, Z, \mathbf{C})$ 
28:    Reset  $\mathbf{a} = \mathbf{b} = \mathbf{0}_n$ ,  $\mathbf{c} = \mathbf{d} = \mathbf{0}_s$ 
29:  end if
30: until Stopping criterion is reached
31: return  $f(x) = \sum_{i=1}^n \alpha_i K(x, z_i)$ 

```

Proposition 4. *The following update equations*

$$\mathbf{v}_t \leftarrow K(X[B], X)\beta_t - Y[B] \in \mathbb{R}^m \quad (19a)$$

$$\mathbf{w}_t \leftarrow \mathbf{G}\mathbf{G}^\top K(X[J], X[B])\mathbf{v}_t \in \mathbb{R}^s \quad (19b)$$

$$\alpha_{t+1} \leftarrow \beta_t - \eta_1 \mathbf{H}_B^\top \mathbf{v}_t + \eta_1 \mathbf{H}_J^\top \mathbf{w}_t \quad (19c)$$

$$\beta_{t+1} \leftarrow (1 + \gamma)\alpha_{t+1} - \gamma\alpha_t + \eta_2 \mathbf{H}_B^\top \mathbf{v}_t - \eta_2 \mathbf{H}_J^\top \mathbf{w}_t \quad (19d)$$

emulate the updates (16) with \mathcal{P} replaced by \mathcal{Q} via the relation $f_t := S^* \alpha_t$ and $g_t := S^* \beta_t$.

Remark 5. In Algorithm 1, lines 12-14 together implement equation (19c), whereas lines 21-17 together implement equation (19d).

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 a per iteration complexity of $O(mp)$ on average. This idea is similar to the strategy in EigenPro 4 from [Abedsoltan et al. \(2024a\)](#).

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

$$f_{t+1} = g_t - \eta_1 \mathcal{Q} \tilde{\nabla} L(g_t) \quad (20a)$$

$$g_{t+1} = (1 + \gamma)f_{t+1} - \gamma f_t + \eta_2 \mathcal{Q} \tilde{\nabla} L(g_t) \quad (20b)$$

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

$$f_{kT} = \arg \min_{f \in \mathcal{M}} \|f - f_{kT-1}\|_{\mathbb{H}}^2 \quad (21a)$$

$$g_{kT} = \arg \min_{g \in \mathcal{M}} \|g - g_{kT-1}\|_{\mathbb{H}}^2 \quad (21b)$$

Proposition 5. *Algorithm 2 emulates equations (20) and (21) 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 $\text{span}(\{K(\cdot, x_i)\}_{i=1}^n)$, 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 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n > 0$ be the eigenvalues of $K(X, X)$. Let $\mathbf{e}_i = (e_{i1}, e_{i2}, \dots, e_{in}) \in \mathbb{R}^n$ be the eigenvector of $K(X, X)$ corresponding to eigenvalue λ_i . Define

$$L_1 := \max_{j \in [n]} K(x_j, x_j) - \sum_{i=1}^q e_{ij}^2 (\lambda_i - \lambda_{q+1}) \quad (22a)$$

$$L_m := L_1/m + (m - 1)\lambda_{q+1}/m \quad (22b)$$

$$\kappa_m := L_m/\lambda_n \quad (22c)$$

$$\tilde{\kappa}_m := n/m + (m - 1)/m \quad (22d)$$

Consider hyperparameters,

$$\eta_1^* = \frac{1}{L_m} \quad (23a)$$

$$\eta_2^* = \frac{\eta_1^*(m) \sqrt{\kappa_m \tilde{\kappa}_m}}{\sqrt{\kappa_m \tilde{\kappa}_m} + 1} \left(1 - \frac{1}{\tilde{\kappa}_m}\right) \quad (23b)$$

$$\gamma^* = \frac{\sqrt{\kappa_m \tilde{\kappa}_m} - 1}{\sqrt{\kappa_m \tilde{\kappa}_m} + 1}. \quad (23c)$$

Let $\varepsilon > 0$ and $\delta \in (0, 1)$ be parameters for controlling the approximation error of Nyström approximate preconditioner defined in equation (18).

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 $\{1, 2, \dots, n\}$, 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 (23). Then, with probability at least $1 - \delta$ over the randomness in X , we have the following upper bound for a constant C ,

$$\|f_t - f^*\|_{\mathbb{H}}^2 \leq C \cdot \exp\left(\frac{-t/(1+\varepsilon)}{\sqrt{\kappa_m \kappa_m}}\right). \quad (24)$$

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

$$f_{t+1} \leftarrow g_t - \eta_1 \mathcal{Q} \tilde{\nabla}_f \mathcal{L}(g_t), \quad (25a)$$

$$g_{t+1} \leftarrow (1+\gamma)f_{t+1} - \gamma f_t + \eta_2 \mathcal{Q} \tilde{\nabla}_f \mathcal{L}(g_t). \quad (25b)$$

Rewriting the above in terms of $\tilde{\nabla} \mathcal{L}(f_t) = \mathcal{K}_{B_t}(f_t - f^*)$ from (70), and subtracting f^* on both sides in the updates (25), we get

$$f_{t+1} - f^* \leftarrow g_t - f^* - \eta_1 \mathcal{Q} \mathcal{K}_{B_t}(g_t - f^*), \quad (26a)$$

$$g_{t+1} - f^* \leftarrow (1+\gamma)(f_{t+1} - f^*) - \gamma(f_t - f^*) + \eta_2 \mathcal{Q} \mathcal{K}_{B_t}(g_t - f^*). \quad (26b)$$

To proceed, we need the following technical lemma.

Lemma 7. Consider the data dependent kernel

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

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

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

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

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

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

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

Proof. (a) Note that by definition

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

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

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

Defining $\hat{f}_t := T^{*-1}(f_t - f^*)$ and $\hat{g}_t := T^{*-1}(g_t - f^*)$, and using the fact that $\mathcal{Q} = T^*T$, we can write (26) as

$$\hat{f}_{t+1} \leftarrow \hat{g}_t - \eta_1 TK_{B_t} T^* \hat{g}_t, \quad (27a)$$

$$\hat{g}_{t+1} \leftarrow (1+\gamma)\hat{f}_{t+1} - \gamma\hat{f}_t + \eta_2 TK_{B_t} T^* \hat{g}_t. \quad (27b)$$

Note that in equation (27), we have that $\mathbb{E}_{B_t} TK_{B_t} T^* = TKT^* = \frac{1}{n}R^*R$. Hence the above iteration is (13) specialized to solve

$$\min_{f \in \hat{\mathbb{H}}_q} \frac{1}{2} \|Rf - Y\|^2. \quad (28)$$

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

Specializing the convergence result of Liu and Belkin (2020, Thm. 2) for the case of square loss and search space $\hat{\mathbb{H}}_q$, we can conclude that for hyperparameters in equations (23), updates (27) converge as

$$\|\hat{f}_t\|_{\hat{\mathbb{H}}_q}^2 \leq \exp\left(-t/\sqrt{\tilde{c}_m c_m}\right) \|T^{*-1}f^*\|_{\hat{\mathbb{H}}_q}^2 \quad (29)$$

where

$$\tilde{c}_m = \frac{1}{m} \cdot \mathbb{E} \left\| \hat{K}_q(\cdot, x) \right\|_{(TK_q T^*)^{-1}} + \frac{m-1}{m} \quad (30a)$$

$$c_m = d_m / \lambda_n(R^*R) \quad (30b)$$

$$d_m = d_1/m + (m-1)\lambda_1(R^*R)/m \quad (30c)$$

$$d_1 = \max_{\hat{f} \in \text{range}(R^*)} \left\| \hat{f} \right\|_{\hat{\mathbb{H}}_q}^2 \quad (30d)$$

We also have the following result.

Proposition 8 (Abdolsoltan et al. (2024b, 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

$$\lambda_1(\mathcal{R}) \leq (1+\varepsilon)^2 \lambda_1(\mathcal{PK}) = n(1+\varepsilon)^2 \lambda_{q+1}, \quad (31a)$$

$$\lambda_n(R^*R) \geq (1+\varepsilon)^2 \lambda_n(\mathcal{PK}) = n \frac{1}{(1+\varepsilon)^2} \lambda_n. \quad (31b)$$

Due to the above proposition, we have that $c_m \leq (1+\varepsilon)^4 \kappa_m$. In the Appendix, we show that $d_1 = L_1$

whereby $d_m = L_m$ and we also show that $\tilde{c}_m = \tilde{\kappa}_m$, which leads to the bound

$$\|f_t - f^*\|_{\hat{\mathbb{H}}_q}^2 \leq \exp\left(\frac{-t/(1+\varepsilon)^2}{\sqrt{\tilde{\kappa}_m \kappa_m}}\right) \|T^{*-1} f^*\|_{\hat{\mathbb{H}}_q}^2. \quad (32)$$

Eq. (24) follows due to equivalence of \mathbb{H} and $\hat{\mathbb{H}}_q$. \square

5 NUMERICAL EXPERIMENTS

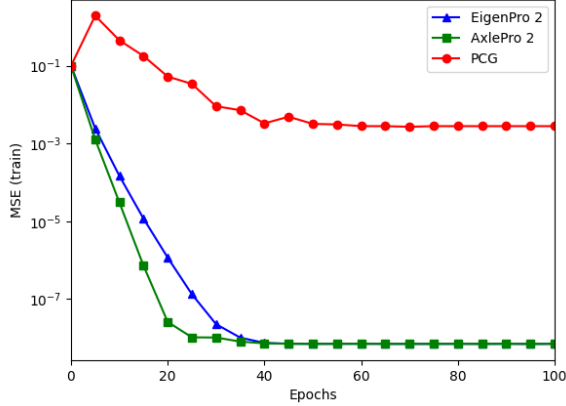


Figure 2: **Numerical stability to low precision.** Convergence comparison on CIFAR10 dataset with Laplacian kernel over single precision arithmetic.

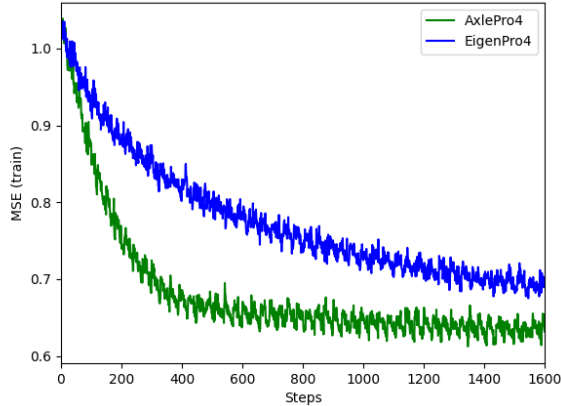


Figure 3: Progress of Algorithm 2 v/s EigenPro 4 from [Abedsoltan et al. \(2024a\)](#) with respect to number of steps on randomly generated dataset ($n = 6000, p = 3000$).

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 (1) and show the mean squared error (MSE) over the training dataset.

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%	99.12%
$d = 784$	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.

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 [Shankar et al. \(2020\)](#), which is the state-of-the-art kernel for this dataset. Experiment details are in Appendix D.

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 [Ma and Belkin \(2019\)](#) 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 according to the datasets.

We follow the standard PCG algorithm used in [Gardner et al. \(2018\)](#), which computes the precondition matrix based on the pivoted Cholesky decomposition [Bach \(2013\)](#); [Harbrecht et al. \(2012\)](#). And we set the rank of approximation to be 300.

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. However going beyond square loss remains unresolved even for the EigenPro family of algorithms. The main challenge in doing so remains managing the complexity of approximating the preconditioner at each iteration.

Acknowledgments

Y.Z is supported by the TILOS institute under grant NSF CCF-2112665. P.P. is supported by the DST INSPIRE faculty fellowship and Schmidt Sciences.

References

- Amirhesam Abedsoltan, Mikhail Belkin, and Parthe Pandit. Toward large kernel models. *arXiv preprint arXiv:2302.02605*, 2023.
- Amirhesam Abedsoltan, Siyuan Ma, Parthe Pandit, and Mikhail Belkin. Fast training of large kernel models with delayed projections. *arXiv preprint arXiv:2411.16658*, 2024a.
- 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)*, 2024b.
- 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.
- 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.
- Francis Bach. Sharp analysis of low-rank kernel matrix approximations. In *Conference on learning theory*, pages 185–209. PMLR, 2013.
- Daniel Beaglehole, Adityanarayanan Radhakrishnan, Parthe Pandit, and Mikhail Belkin. Mechanism of feature learning in convolutional neural networks. *arXiv preprint arXiv:2309.00570*, 2023.
- Alberto Bietti and Francis Bach. Deep equals shallow for relu networks in kernel regimes. *arXiv preprint arXiv:2009.14397*, 2020.
- 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.
- Luigi Carratino, Alessandro Rudi, and Lorenzo Rosasco. Learning with sgd and random features. *Advances in Neural Information Processing Systems*, 31, 2018.
- 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.
- 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. URL <https://proceedings.mlr.press/v80/chowdhury18a.html>.
- Kurt Cutajar, Michael Osborne, John Cunningham, and Maurizio Filippone. Preconditioning kernel matrices. In *International conference on machine learning*, pages 2529–2538. PMLR, 2016.
- 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.
- 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.
- 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.
- 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. ISSN 0168-9274. doi: <https://doi.org/10.1016/j.apnum.2011.10.001>. URL <https://www.sciencedirect.com/science/>

- [article/pii/S0168927411001814](#). Third Chilean Workshop on Numerical Analysis of Partial Differential Equations (WONAPDE 2010).
- Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.
- 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.
- Chaoyue Liu and Mikhail Belkin. Accelerating sgd with momentum for over-parameterized learning. In *International Conference on Learning Representations*, 2020.
- 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.
- Siyuan Ma and Mikhail Belkin. Kernel machines that adapt to gpus for effective large batch training. *Proceedings of Machine Learning and Systems*, 1:360–373, 2019.
- 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.
- 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.
- 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.
- Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.
- 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.
- Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. Less is more: Nyström computational regularization. *Advances in neural information processing systems*, 28, 2015.
- Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. *Advances in neural information processing systems*, 30, 2017.
- Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.
- 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.
- 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.
- Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. 28(3):1139–1147, 17–19 Jun 2013. URL <https://proceedings.mlr.press/v28/sutskever13.html>.
- 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.
- 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. URL <https://proceedings.mlr.press/v139/yin21a.html>.

Checklist

- For all models and algorithms presented, check if you include:
 - A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
 - An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
 - (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]
- For any theoretical claim, check if you include:

- (a) Statements of the full set of assumptions of all theoretical results. [Yes]
 - (b) Complete proofs of all theoretical results. [Yes]
 - (c) Clear explanations of any assumptions. [Yes]
3. For all figures and tables that present empirical results, check if you include:
- (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
 - (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
 - (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
 - (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
- (a) Citations of the creator If your work uses existing assets. [Yes]
 - (b) The license information of the assets, if applicable. [Not Applicable]
 - (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
 - (d) Information about consent from data providers/curators. [Not Applicable]
 - (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
- (a) The full text of instructions given to participants and screenshots. [Not Applicable]
 - (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
 - (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Appendices

A Notation

 Table 4: **Notation**

n	number of samples
$x_i \in \mathcal{X}, y_i \in \mathbb{R}$	input, target
$Y \in \mathbb{R}^n$	vector of targets $(y_i) \in \mathbb{R}^n$
$X = (x_1, x_2, \dots, x_n)$	tuple of inputs
$K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$	positive definite kernel with \mathbb{H} as its RKHS.
$S : \mathbb{H} \rightarrow \mathbb{R}^n$	Sampling operator, defined as $S(f) = f(X) = (f(x_i)) \in \mathbb{R}^n$
$S^* : \mathbb{R}^n \rightarrow \mathbb{H}$	Adjoint of sampling operator, defined as $S^* \alpha = \sum_{i=1}^n K(\cdot, x) \alpha_i$
\mathcal{K}	empirical covariance operator using n samples, defined as $\frac{1}{n} S S^* : \mathbb{H} \rightarrow \mathbb{H}$
$K(X, X)$	kernel matrix. Also equals $S S^*$
B	mini-batch of indices, subset of $\{1, 2, \dots, n\}$
$m = B $	batch size
$X[B] \in \mathcal{X}^m$	minibatch of size m
$\mathbf{v}[B] \in \mathbb{R}^m$	subvector of $\mathbf{v} \in \mathbb{R}^n$ corresponding to indices B
\mathbf{H}_B	selector matrix, rows of I_n corresponding to B , so that $\mathbf{v}[B] = \mathbf{H}_B \mathbf{v}$.
q	level of preconditioner
$(\mathbf{E}, \Lambda, \lambda_{q+1})$	top- q eigensystem of $K(X, X)$
$\mathcal{P} : \mathbb{H} \rightarrow \mathbb{H}$	preconditioner in \mathbb{H} defined as $\mathcal{I} - \sum_{i=1}^q \left(1 - \frac{\lambda_{q+1}}{\lambda_i}\right) \psi_i \otimes_{\mathbb{H}} \psi_i$ where $\psi_i = S^* \mathbf{e}_i / \sqrt{\lambda_i} \in \mathbb{H}$ are top- q eigenfunction of \mathcal{K} with eigenvalues $\frac{1}{n} \lambda_i$
\mathbf{F}	rescaled eigenvectors defined as $\mathbf{E} \sqrt{I_q - \lambda_{q+1} \Lambda^{-1}}$
$\mathbf{F}[B]$	rows of \mathbf{F} corresponding to the mini-batch B
J	subset of indices $\{1, 2, \dots, n\}$ to obtain Nyström approximation of \mathcal{P}
$s = J $	size of Nyström subset
$X[J]$	Nyström subsample of size s
$(\mathbf{D}, \Delta, \delta_{q+1})$	top- q eigensystem of $\frac{1}{n} K(X[J], X[J])$
$\mathcal{Q} : \mathbb{H} \rightarrow \mathbb{H}$	Nyström approximation of \mathcal{P} defined as $\mathcal{I} - \sum_{i=1}^q \left(1 - \frac{\delta_{q+1}}{\delta_i}\right) \phi_i \otimes_{\mathbb{H}} \phi_i$ where $(\delta_i/s, \phi_i)$ is an eigen-pair of $\frac{1}{s} S_J^* S_J$
\mathbf{G}	rescaled eigenvectors defined as $\mathbf{D} \sqrt{(I_q - \delta_{q+1} \Delta^{-1}) \Delta^{-1}}$

B EigenPro algorithms

Lemma 9. Suppose Algorithm 3 is run for t iterations with hyperparameters given by equation (23), then we have,

$$\|f_t - f^*\|_{\mathbb{H}}^2 \leq \exp\left(-t/\sqrt{\tilde{\kappa}_m \kappa_m}\right) \|f^*\|_{\mathbb{H}}^2. \quad (33)$$

Proof. This is an immediate corollary of Liu and Belkin (2020, Theorem 2) and the discussion that ensues therein on the convergence rate of MaSS. Preconditioning only changes the largest eigenvalue from λ_1 to λ_{q+1} . This can be via the following reduction commonly applied to analyze spectral preconditioning: (16) is actually (13) with the following modified kernel,

$$\tilde{K}(x, z) := K(x, z) - K(x, X) \mathbf{E} \mathbf{Q} \mathbf{E}^\top K(X, z), \quad (34)$$

where $\mathbf{Q} = \Lambda^{-1}(I_q - \lambda_{q+1} \Lambda^{-1})$, where $(\mathbf{E}, \Lambda, \lambda_{q+1})$ is the top- q eigensystem of $K(X, X)$. An interested reader can find this reduction argument in Ma and Belkin (2017, 2019). The largest eigenvalue of $\tilde{K}(X, X)$ can be verified to be λ_{q+1} .

Algorithm 3 AxlePro

```

1: Input: positive definite kernel  $K$ , batch size  $m$ , learning rates  $\eta_1, \eta_2 > 0$ , damp factor  $\gamma \in (0, 1)$ .
2: Output:  $f : \mathcal{X} \rightarrow \mathbb{R}$  solving (9) approximately.
3: setup:  $\alpha \leftarrow \mathbf{0}_n$ , and  $\beta \leftarrow \mathbf{0}_n$ .
4:  $(\mathbf{E}, \Lambda, \lambda_{q+1}) \leftarrow$  top- $q$  eigensystem of  $K(X, X)$ 
5:  $\mathbf{F} \leftarrow \mathbf{E} \sqrt{I_q - \lambda_{q+1} \Lambda^{-1}} \in \mathbb{R}^{n \times q}$ 
6: repeat
7:   Fetch batch of indices  $B \subset \{1, \dots, n\}$ ,  $|B| = m$ 
8:    $\mathbf{v} \leftarrow K(X[B], X)\beta - Y[B] \in \mathbb{R}^m$ 
9:    $\mathbf{w} \leftarrow \mathbf{F}\mathbf{F}[B]^\top \mathbf{v} \in \mathbb{R}^n$ 
10:   $\tilde{\alpha} \leftarrow \alpha$  {copy state for momentum}
11:   $\alpha \leftarrow \beta$ 
12:   $\alpha[B]^- \leftarrow \eta_1 \mathbf{v}$  {gradient step-1}
13:   $\alpha^+ \leftarrow \eta_1 \mathbf{w}$  {correction-1}
14:   $\beta \leftarrow \alpha + \gamma(\alpha - \tilde{\alpha})$  {momentum}
15:   $\beta[B]^+ \leftarrow \eta_2 \mathbf{v}$  {gradient step-2}
16:   $\beta^- \leftarrow \eta_2 \mathbf{w}$  {correction-2}
17: until Stopping criterion is reached
18: return  $f(x) = \sum_{i=1}^n \alpha_i K(x, x_i)$ 

```

Algorithm	FLOPS/batch	memory
EigenPro 2	$nm + sm + 2sq + m + s$	$n + sq$
AxlePro 2	$nm + sm + 2sq + \mathbf{2m} + \mathbf{2s} + \mathbf{n}$	$\mathbf{2n} + sq$
EigenPro 3	$2pm + ps + ms + 2sq + m + p$	$p + sq$
AxlePro 3	$2pm + ps + ms + 2sq + \mathbf{2m} + \mathbf{2p}$	$\mathbf{2p} + sq$

Table 5: The bolded quantities indicate overheads due to preconditioning. Here $s = |J|$. We have omitted the lower-order terms of n and $2n$ in the per iteration time since they are dominated by mn . Memory requirement and per iteration complexity of EigenPro and AxlePro. The algorithm AxlePro is based on applying momentum acceleration to EigenPro [Ma and Belkin \(2017\)](#), whereas AxlePro2 is based on applying momentum acceleration to EigenPro 2 [Ma and Belkin \(2019\)](#). The overhead of storing the previous parameter is n as seen the last column.

The other key difference is the quantities L_1 and $\tilde{\kappa}_m$ defined in [Liu and Belkin \(2020\)](#). Using elementary properties of our Hilbert space \mathbb{H} , we can easily show that our L_1 and $\tilde{\kappa}_m$ are upper bounds to these quantities in [Liu and Belkin \(2020\)](#). \square

C Numerical Experiments (continued)

See Table 6, Table 7, Table 8, Table 9, Table 10 for experiment comparisons.

D Details on Numerical Experiments

Hardware. Experiments were run on the SDSC Expanse GPU cluster with 32GB RAM, with 1 NVIDIA V100 SMX2 with 32GB VRAM, and 1 Xeon Gold 6248 CPU.

Datasets and Kernels. Our experiments were conducted on the following datasets: Cifar-10($n = 50,000, d = 3072$), Stellar Classification($n = 95,000, d = 13$), and EMNIST Digits($n = 240,000, d = 784$). We test the performance for both Gaussian kernel and Laplacian kernel. And for Cifar-10 dataset, we also compare the performance with Myrtle-5 kernel [Shankar et al. \(2020\)](#), which is the state-of-the-art kernel for this dataset.

1. **EigenPro.** We select the precondition level based on the GPU memory so that the batch size at the linear rate regime critical point m_1^* fully exploits GPU memory. This step involves computing eigenpairs of the kernel matrix and we use Nyström approximation with $20k$ subsamples. The optimal learning rate can also be computed using the computed eigenpairs.

Algorithm 4 AxlePro 3

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:** $f : \mathcal{X} \rightarrow \mathbb{R}$ solving (9) approximately.
 3: **setup:** Sample $J \subset \{1, 2, \dots, n\}$ with $|J| = s$.
 4: $(\mathbf{D}, \Delta, \delta_{q+1}) \leftarrow$ top- q eigensystem of $K(X[J], X[J])$
 5: $\mathbf{G} \leftarrow \mathbf{D} \sqrt{\Delta^{-1}(I_q - \delta_{q+1} \Delta^{-1})} \in \mathbb{R}^{s \times q}$
 6: Initialize $\alpha = \beta = \mathbf{0}_p$.
 7: **repeat**
 8: Fetch batch of indices $B \subset \{1, \dots, n\}$, $|B| = m$
 9: $\mathbf{v} \leftarrow K(X[B], Z)\beta + K(X[B], X)\mathbf{b} + K(X[B], X[J])\mathbf{d} - Y[B] \in \mathbb{R}^m$ {gradient}
 10: $\mathbf{w} \leftarrow \mathbf{G}\mathbf{G}^\top K(X[J], X[B])\mathbf{v} \in \mathbb{R}^s$ {correction}
 11: $\tilde{\alpha} \leftarrow \alpha$
 12: $\tilde{\mathbf{c}} \leftarrow \mathbf{c}$
 13: $\alpha \leftarrow \beta$
 14: $\beta \leftarrow (1 + \gamma)\alpha - \gamma\tilde{\alpha}$ {momentum-1}
 15: $\mathbf{A} \leftarrow K(Z, X)\mathbf{a} + K(Z, X[J])\mathbf{c} \in \mathbb{R}^p$
 16: $\alpha \leftarrow \alpha + \text{K_solve}(K, Z, \mathbf{A})$
 17: $\mathbf{C} \leftarrow K(Z, X)\mathbf{b} + K(Z, X[J])\mathbf{d} \in \mathbb{R}^n$
 18: $\beta \leftarrow \beta + \text{solve}(K, Z, \mathbf{C})$
 19: **until** Stopping criterion is reached
 20: **return** $f(x) = \sum_{i=1}^n \alpha_i K(x, z_i)$

Algorithm 5 EigenPro 1 Ma and Belkin (2017)

Input: kernel K , batch size m , learning rate $\eta_0 > 0$.
Output: $f : \mathcal{X} \rightarrow \mathbb{R}$ that solves (9) approximately.
setup: $\alpha \leftarrow \mathbf{0}_n$.
 $(\mathbf{E}_q, \Lambda, \lambda_{q+1}) \leftarrow$ top- q eigensystem of $K(X, X)$
 $\mathbf{F}_q \leftarrow \mathbf{E}_q \sqrt{I_q - \lambda_{q+1} \Lambda^{-1}}$
repeat
 Fetch a batch of indices $B \subset \{1, 2, \dots, n\}$
 $\mathbf{v} \leftarrow K(X[B], X)\alpha - Y[B] \in \mathbb{R}^m$
 $\mathbf{w} \leftarrow \mathbf{F}_q \mathbf{F}_q[B]^\top \mathbf{v} \in \mathbb{R}^n$
 $\alpha[B] \leftarrow \alpha[B] - \eta_0 \mathbf{v}$ {gradient step}
 $\alpha \leftarrow \alpha + \eta_0 \mathbf{w}$ {correction}
until Stopping criterion is reached
return $f_t(x) = \sum_{i=1}^n \alpha_i K(x, x_i)$

2. AxlePro. For the precondition level, we follow the same way as in Eigenpro. The only hyperparameter needs to be tuned is the smallest eigenvalue of the kernel matrix due to the Nyström approximation and we choose this factor according to the datasets.

3. FALKON. According to the GPU memory, we set the number of inducing points to be $20k$ with the uniform sampling. We do not use any regularization. We comment here that even in our experiments the MSE does not decrease significantly due to the subsampling, the classification accuracy is indeed increasing during training.

4. PCG. We choose the rank to be 100 for the pivoted Cholesky decomposition.

5. GPYTORCH. We use instances of the class `IndependentMultitaskGPMModel` in order to deal with multiclass classification problems. The optimizer was set to be ‘Adam’ with learning rate 0.05. As in the case of Falkon, while it is hard to notice significant decrease in the logarithmic scale plot of MSE, the classification accuracy is improved during optimization.

Unless otherwise specified, we do not tune λ_n for AxlePro and use the following setup for our experiments.

For Cifar-10 dataset, we scale the data by a factor of 0.05 as our bandwidth selection. With Gaussian kernel, we

Algorithm 6 EigenPro 2 [Ma and Belkin \(2019\)](#)

Input: kernel K , batch size m , indices $J = \{j_\ell\}_{\ell=1}^s \subset \{1, 2, \dots, n\}$, learning rate $\eta_0 > 0$.
Output: $f : \mathcal{X} \rightarrow \mathbb{R}$ that solves (9) approximately.
setup: $\alpha \leftarrow \mathbf{0}_n$.
 $(D, \Delta, \delta_{q+1}) \leftarrow$ top- q eigensystem of $K(X[J], X[J])$
 $G \leftarrow D\sqrt{\Delta^{-1}(I_q - \delta_{q+1}\Delta^{-1})}$
repeat
 Fetch a batch of indices $B \subset \{1, 2, \dots, n\}$
 $v \leftarrow K(X[B], X)\alpha - Y[B] \in \mathbb{R}^m$
 $w \leftarrow GG^\top K(X[J], X[B])v \in \mathbb{R}^s$
 $\alpha[B] \leftarrow \alpha[B] - \eta_0 v$ {gradient step}
 $\alpha[J] \leftarrow \alpha[J] + \eta_0 w$ {correction}
until Stopping criterion is reached
return $f_t(x) = \sum_{i=1}^n \alpha_i K(x, x_i)$

Algorithm 7 EigenPro 3 [Abedsoltan et al. \(2023\)](#)

Input: kernel K , batch size m , centers $Z = \{z_i\}$, indices $J = \{j_\ell\}_{\ell=1}^s \subset \{1, 2, \dots, n\}$, learning rate $\eta_0 > 0$.
Output: $f : \mathcal{X} \rightarrow \mathbb{R}$ that solves (9) approximately.
setup: $\alpha \leftarrow \mathbf{0}_p$.
 $(D, \Delta, \delta_{q+1}) \leftarrow$ top- q eigensystem of $K(X[J], X[J])$
 $G \leftarrow D\sqrt{\Delta^{-1}(I_q - \delta_{q+1}\Delta^{-1})}$
repeat
 Fetch a batch of indices $B \subset \{1, 2, \dots, n\}$
 $v \leftarrow K(X[B], Z)\alpha - Y[B] \in \mathbb{R}^m$ {gradient}
 $w \leftarrow GG^\top K(X[J], X[B])v \in \mathbb{R}^s$ {correction}
 $h \leftarrow K(Z, X[B])v - K(Z, X[J])w$
 $\theta \leftarrow \text{K_solve}(K, Z, h)$ {projection}
 $\alpha \leftarrow \alpha - \eta_0 \cdot \theta$
until Stopping criterion is reached
return $f_t(x) = \sum_{i=1}^n \alpha_i K(x, z_i)$

set $(m, q, s) = (2000, 500, 20, 000)$, while for Laplacian kernel, we use $q = 300$ instead. For the Myrtle5 kernel experiment, we follow the original paper [Shankar et al. \(2020\)](#) to use ZCA preprocessing (without Leave-One-Outtilting and ZCA augmentation) and store the kernel matrix before performing regression. For this kernel, we set $(m, q, s) = (2000, 300, 10, 000)$.

For Star Classification dataset, we use $(m, q, s) = (2000, 150, 20, 000)$ for Gaussian kernel (bandwidth=2) and $(m, q, s) = (2000, 600, 20, 000)$ for Laplacian kernel (bandwidth=4). We preprocess the data with mean subtraction and standard deviation normalization.

For EMNIST Digits dataset, we preprocess the data by mean subtraction and scale by a factor of 0.001. We set $(m, q, s) = (700, 600, 20, 000)$ for Gaussian kernel (bandwidth=1) and $(m, q, s) = (800, 600, 20, 000)$ for Laplacian kernel (bandwidth=1).

E Exact preconditioner

Calculating the exact preconditioner requires finding the top- q eigensystem of $K(X, X)$ which can cost n^2q FLOPs. Instead, we can use a Nyström extension. Let $J = \{j_1, j_2, \dots, j_s\} \subseteq \{1, 2, \dots, n\}$. Next, let $(\tilde{E}, \Lambda, \lambda_{q+1})$ be the top- q eigensystem of $K(X[J], X[J])$, where the columns of $\tilde{E} \in \mathbb{R}^{s \times q}$ are the eigenvectors of $K(X[J], X[J])$. Finally, obtain approximate eigenvectors $E \in \mathbb{R}^{n \times q}$ of $K(X, X)$ as

$$E \leftarrow K(X, X[J])\tilde{E}$$

Algorithm 8 EigenPro 4 [Abedsoltan et al. \(2024a\)](#)

Input: kernel K , batch size m , centers $Z = \{z_i\}$, indices $J = \{j_\ell\}_{\ell=1}^s \subset \{1, 2, \dots, n\}$, learning rate $\eta_0 > 0$.
Output: $f : \mathcal{X} \rightarrow \mathbb{R}$ that solves (9) approximately.
setup: $\alpha \leftarrow \mathbf{0}_p, \mathbf{a} \leftarrow \mathbf{0}_n, \mathbf{c} \leftarrow \mathbf{0}_{|J|}, \mathbf{h} \leftarrow \mathbf{0}_0$
 $(\mathbf{D}, \Delta, \delta_{q+1}) \leftarrow \text{top-}q \text{ eigensystem of } K(X[J], X[J])$
 $\mathbf{G} \leftarrow \mathbf{D} \sqrt{\Delta^{-1}} (\mathbf{I}_q - \delta_{q+1} \Delta^{-1})$
repeat
 Fetch a batch of indices $B \subset \{1, 2, \dots, n\}$
 $\mathbf{v} \leftarrow K(X[B], Z)\alpha + K(X[B], X)\mathbf{a} + K(X[B], X[J])\mathbf{c} - Y[B] \in \mathbb{R}^m$ {gradient}
 $\mathbf{w} \leftarrow \mathbf{G}\mathbf{G}^\top K(X[J], X[B])\mathbf{v} \in \mathbb{R}^s$ {correction}
 $\mathbf{a}[B] \leftarrow -\eta_0 \mathbf{v}$ {update temporary weights-1}
 $\mathbf{c} \leftarrow \mathbf{c} + \eta_0 \mathbf{w}$ {update temporary weights-2}
 $\mathbf{h} \leftarrow \mathbf{h} - \eta_0 (K(Z, X[B])\mathbf{v} + K(Z, X[J])\mathbf{w})$ {accumulate gradients}
 if projection condition holds **then**
 $\boldsymbol{\theta} \leftarrow \text{K_solve}(K, Z, \mathbf{h})$ {delayed projection}
 $\alpha \leftarrow \alpha - \eta_0 \cdot \boldsymbol{\theta}$
 Reset $\mathbf{a} \leftarrow \mathbf{0}_n, \mathbf{c} \leftarrow \mathbf{0}_s, \mathbf{h} \leftarrow \mathbf{0}_p$,
 end if
until Stopping criterion is reached
return $f_t(x) = \sum_{i=1}^n \alpha_i K(x, z_i)$

Due to the matrix multiplication, the vectors \mathbf{E} obtained as such are not orthonormal. Hence we run a thin QR decomposition to refine \mathbf{E} as,

$$\mathbf{E}_{\text{ref}} \leftarrow \text{thinQR}(\mathbf{E}).$$

This only has a total complexity $s^2q + qsn + nq^2$, where s^2q is the cost of getting eigenvectors $\tilde{\mathbf{E}}$, and sqn is the cost of Nyström extension, and nq^2 is the cost of thin QR decomposition. Since $q \leq s \leq n$, the term nsq dominates.

F Details on convergence analysis

Proof of Lemma 7(d). We consider the norms restricted to the subspace $\text{span}(\{K(\cdot, x_i)\}_{i=1}^n)$ since we only care about $\|f\|_{\mathbb{H}}$ and $\|\hat{f}\|_{\mathbb{H}_q}$ for $f = \sum_{i=1}^n \alpha_i K(\cdot, x_i)$.

Now, suppose $f = \hat{f} = \sum_{i=1}^n \alpha_i \hat{K}_q(\cdot, x_i)$.

Claim: $\text{span}(\{K(\cdot, x_i)\}_{i=1}^n) = \text{span}(\{\hat{K}_q(\cdot, x_i)\}_{i=1}^n)$.

W.L.O.G, we assume $J = (1, 2, \dots, s)$. Denote $\mathbf{v}_i := \mathbf{G}\mathbf{G}^\top K(X[J], x_i) \in \mathbb{R}^s$

$$\begin{aligned}
 \hat{K}_q(\cdot, x_i) &= K(\cdot, x_i) - K(\cdot, X[J])\mathbf{G}\mathbf{G}^\top K(X[J], x_i) \\
 &= K(\cdot, X)(e_i - [\mathbf{v}_i \ \mathbf{0}_{n-s}]^T)
 \end{aligned}$$

So $\sum_{i=1}^n \alpha_i \hat{K}_q(\cdot, x_i) = \hat{K}_q(\cdot, X)\alpha = K(\cdot, X)A\alpha$, where $A := \mathbf{I}_n - \begin{pmatrix} \mathbf{G}\mathbf{G}^\top K(J, X) \\ \mathbf{0}_{(n-s) \times n} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_s - \mathbf{G}\mathbf{G}^\top K(J, J) & -\mathbf{G}\mathbf{G}^\top K(J, J^C) \\ \mathbf{0}_{(n-s) \times s} & \mathbf{I}_{(n-s) \times (n-s)} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_s - \mathbf{D}(\mathbf{I}_q - \delta_{q+1}\Delta^{-1})\mathbf{D}^\top & -\mathbf{G}\mathbf{G}^\top K(J, J^C) \\ \mathbf{0}_{(n-s) \times s} & \mathbf{I}_{(n-s) \times (n-s)} \end{pmatrix}$. Since A is an upper triangular block matrix, we can easily get $\frac{\delta_{q+1}}{\delta_1} = \min(\frac{\delta_{q+1}}{\delta_1}, 1) \leq \lambda(A) \leq \max(\frac{\delta_{q+1}}{\delta_q}, 1) = 1$. This also implies A is invertible and hence our Claim holds.

As a consequence,

$$\begin{aligned}
 \|f\|_{\mathbb{H}}^2 &= (A\alpha)^T K(X, X)(A\alpha) \\
 &\leq \lambda_{\max}(K(X, X)) \|A\alpha\|_2^2 \\
 &\leq \lambda_{\max}(K(X, X)) \lambda_{\max}^2(A) \|\alpha\|_2^2 \\
 &\leq \lambda_{\max}(K(X, X)) \|\alpha\|_2^2.
 \end{aligned}$$

Similarly, $\|f\|_{\mathbb{H}}^2 \geq \lambda_{\min}(K(X, X)) \lambda_{\min}^2(A) \|\alpha\|_2^2 \geq \lambda_{\min}(K(X, X))^{\frac{\delta_q+1}{\delta_1^2}} \|\alpha\|_2^2$

On the other hand,

$$\|\hat{f}\|_{\mathbb{H}_q}^2 = \alpha^T \hat{K}_q(X, X) \alpha \leq \lambda_{\max}(\hat{K}_q(X, X)) \|\alpha\|_2^2.$$

Similarly, $\|\hat{f}\|_{\mathbb{H}_q}^2 \geq \lambda_{\min}(\hat{K}_q(X, X)) \|\alpha\|_2^2$.

So by proposition 8, with high probability, there exist constants a, a' depending (polynomially) on the eigenvalues of $K(X, X)$ such that $a \|f\|_{\mathbb{H}} \leq \|\hat{f}\|_{\mathbb{H}_q} \leq a' \|f\|_{\mathbb{H}}$. \square

F.1 Condition numbers for analysis of MaSS

Let λ_1 and λ_n be the largest and smallest non-zero eigenvalues of \mathcal{K} .

Let a mini-batch of size m be $\{(\tilde{x}_i, \tilde{y}_i)\}_{i=1}^m$. Defining mini-batch covariance operator as follows

$$\tilde{\mathcal{K}}^{(m)} := \frac{1}{m} \sum_{i=1}^m K(\tilde{x}_i, \cdot) \otimes K(\tilde{x}_i, \cdot) \quad (35)$$

$$\tilde{\mathcal{K}}_{\mathcal{P}}^{(m)} := \frac{1}{m} \sum_{i=1}^m k_{\mathcal{P}}(\tilde{x}_i, \cdot) \otimes k_{\mathcal{P}}(\tilde{x}_i, \cdot) \quad (36)$$

where $\tilde{\mathcal{K}}_{\mathcal{P}}$ is the covariance operator after preconditioning. The kernel function $k_{\mathcal{P}}$ can be defined in the same way as in Lemma 7 with the exact preconditioner.

F.2 Identifying quantities defined by Liu and Belkin (2020)

Suppose we run MaSS to solve $\min_{f \in \mathbb{H}} \frac{1}{n} \sum_{i=1}^n (S_{\{i\}} f - Y_i)^2$, with Hessian $H = \mathcal{K}$. Note that $S_{\{i\}} f = \langle K(x_i, \cdot), f \rangle_{\mathbb{H}}$. The authors define the quantities L, μ to be the largest and smallest eigenvalues of the Hessian $H : \mathbb{H} \mapsto \mathbb{H}$, and $\kappa = L/\mu$ to be the condition number. In our case, $L = \lambda_1/n$, $\mu = \lambda_n/n$ and $\kappa = \frac{\lambda_1}{\lambda_n}$.

They also define the quantity L_1 to be the smallest number such that

$$\mathbb{E}[\|K(\tilde{x}, \cdot)\|_{\mathbb{H}}^2 K(\tilde{x}, \cdot) \otimes_{\mathbb{H}} K(\tilde{x}, \cdot)] \lesssim L_1 H \quad (37)$$

where \mathbb{E} is over the random variable \tilde{x} from the empirical distribution. Note that in our case $\|K(\tilde{x}, \cdot)\|_{\mathbb{H}}^2 = K(\tilde{x}, \tilde{x})$ since,

$$\|K(x, \cdot)\|_{\mathbb{H}}^2 = \langle K(x, \cdot), K(x, \cdot) \rangle_{\mathbb{H}} = K(x, x) \leq \beta := \max_i K(x_i, x_i) \quad (38)$$

Then we have $L_1 \leq \beta$, since

$$\mathbb{E} \left[\|K(\tilde{x}, \cdot)\|^2 K(\tilde{x}, \cdot) \otimes K(\tilde{x}, \cdot) \right] \preceq \beta \mathbb{E} [K(\tilde{x}, \cdot) \otimes K(\tilde{x}, \cdot)] = \beta \mathcal{K}$$

Thus

$$L_m := \frac{L_1}{m} + \frac{(m-1)L}{m} \leq \frac{\beta + (m-1)\frac{\lambda_1}{n}}{m} \quad (39)$$

Defining $\tilde{\kappa}$ as the smallest positive real number such that

$$\mathbb{E} \left[\|K(\tilde{x}, \cdot)\|_{\mathcal{K}^{-1}}^2 K(\tilde{x}, \cdot) \otimes K(\tilde{x}, \cdot) \right] \preceq \tilde{\kappa} \mathcal{K} \quad (40)$$

Deriving $\|K(x_i, \cdot)\|_{\mathcal{K}^{-1}}^2$. Recall that $\psi_i = S^* \mathbf{e}_i / \sqrt{\lambda_i}$. Also note that $\mathcal{K}^{-1} = \sum_{i=1}^n \frac{n}{\lambda_i} \psi_i \otimes \psi_i$.

$$\begin{aligned} \|K(x_i, \cdot)\|_{\mathcal{K}^{-1}}^2 &= \langle K(x_i, \cdot), \mathcal{K}^{-1} K(x_i, \cdot) \rangle_{\mathbb{H}} \\ &= \left\langle K(x_i, \cdot), \left(\sum_{j=1}^n \frac{n}{\lambda_j} \psi_j \otimes \psi_j \right) K(x_i, \cdot) \right\rangle_{\mathbb{H}} \\ &= \sum_{j=1}^n \frac{n}{\lambda_j} \langle \psi_j, K(x_i, \cdot) \rangle_{\mathbb{H}}^2 = \sum_{j=1}^n \frac{n}{\lambda_j} \psi_j^2(x_i) \stackrel{(a)}{=} \sum_{j=1}^n \frac{1}{\lambda_j} n \lambda_j e_{ji}^2 \end{aligned} \quad (41a)$$

$$= n \sum_{j=1}^n e_{ji}^2 = n \|\mathbf{e}_j\|^2 = n \quad (41b)$$

where (a) follows from the fact that $\psi_j(x_i) = S_{\{i\}} \psi_j = S_{\{i\}} S^* \mathbf{e}_j / \sqrt{\lambda_j} = \mathbf{H}_{\{i\}} S S^* \mathbf{e}_j / \sqrt{\lambda_j} = \mathbf{H}_{\{i\}} \mathbf{e}_j \sqrt{\lambda_j}$.

Deriving $\tilde{\kappa}$

$$\mathbb{E} \left[\|K(\tilde{x}, \cdot)\|_{\mathcal{K}^{-1}}^2 K(\tilde{x}, \cdot) \otimes K(\tilde{x}, \cdot) \right] \stackrel{(a)}{=} n \mathbb{E} [K(\tilde{x}, \cdot) \otimes K(\tilde{x}, \cdot)] = n \mathbb{E} [\tilde{\mathcal{K}}^{(1)}] = n \mathcal{K} \stackrel{(b)}{\implies} \tilde{\kappa} = n$$

where (a) follows from equation (41b) and (b) from definition of $\tilde{\kappa}$ in equation (40). This also implies $\tilde{c}_m = \tilde{\kappa}_m$ in lemma 7 since (b) holds for any kernel function.

F.3 Formulae for hyperparameters

$$L_1 = \beta, L = \frac{\lambda_1}{n}, \tilde{\kappa} = n$$

$$L_m = \frac{L_1}{m} + \frac{(m-1)L}{m} = \frac{\beta + (m-1)\frac{\lambda_1}{n}}{m} \quad (42a)$$

$$\kappa_m = \frac{n L_m}{\lambda_n} \quad (42b)$$

$$\tilde{\kappa}_m = \frac{\tilde{\kappa}}{m} + \frac{m-1}{m} = 1 + \frac{n-1}{m} \quad (42c)$$

$$\eta_1(m) = \frac{1}{L_m} = \frac{m}{\beta + (m-1)\frac{\lambda_1}{n}} \quad (42d)$$

$$\eta_2(m) = \eta_1 \frac{\sqrt{\kappa_m \tilde{\kappa}_m}}{1 + \sqrt{\kappa_m \tilde{\kappa}_m}} \left(1 - \frac{1}{\tilde{\kappa}_m} \right) \quad (42e)$$

$$\gamma(m) = \frac{\sqrt{\kappa_m \tilde{\kappa}_m} - 1}{\sqrt{\kappa_m \tilde{\kappa}_m} + 1} \quad (42f)$$

F.4 Condition numbers after preconditioning

Note that after preconditioning we are operating in Hilbert space $\mathbb{H}_{\mathcal{P}}$. Also, the largest eigenvalue of $\mathcal{K}_{\mathcal{P}}$ is λ_{q+1} i.e., $L = \lambda_{q+1}$

Similar to equation (37), defining L_1 as the smallest positive number such that

$$\mathbb{E} \left[\|k_{\mathcal{P}}(\tilde{x}, \cdot)\|^2 k_{\mathcal{P}}(\tilde{x}, \cdot) \otimes k_{\mathcal{P}}(\tilde{x}, \cdot) \right] \preceq L_1 \mathcal{K}_{\mathcal{P}} \quad (43)$$

Deriving $\|k_{\mathcal{P}}(\mathbf{x}_i, \cdot)\|^2$

$$\begin{aligned}
 \|k_{\mathcal{P}}(\mathbf{x}_i, \cdot)\|^2 &= \langle k_{\mathcal{P}}(\mathbf{x}_i, \cdot), k_{\mathcal{P}}(\mathbf{x}_i, \cdot) \rangle_{\mathbb{H}_{\mathcal{P}}} \\
 &= k_{\mathcal{P}}(\mathbf{x}_i, \mathbf{x}_i) \\
 &\stackrel{(a)}{=} K(\mathbf{x}_i, \mathbf{x}_i) - \sum_{j=1}^q \left(1 - \frac{\lambda_{q+1}}{\lambda_j}\right) \lambda_j e_{ji}^2 \\
 &= K(\mathbf{x}_i, \mathbf{x}_i) - \sum_{j=1}^q (\lambda_j - \lambda_{q+1}) e_{ji}^2
 \end{aligned} \tag{44}$$

where (a) is from the definition of $k_{\mathcal{P}}$.

Define $\beta_{\mathcal{P}}$ as the maximum norm of $k_{\mathcal{P}}(\mathbf{x}_i, \cdot)$ among the samples

$$\beta_{\mathcal{P}} := \max_i \|k_{\mathcal{P}}(\mathbf{x}_i, \cdot)\|^2 = \max_i \left\{ K(\mathbf{x}_i, \mathbf{x}_i) - n \sum_{j=1}^q (\lambda_j - \lambda_{q+1}) e_{ji}^2 \right\} \tag{45}$$

Deriving L_1

$$\mathbb{E} \left[\|k_{\mathcal{P}}(\tilde{\mathbf{x}}, \cdot)\|^2 k_{\mathcal{P}}(\tilde{\mathbf{x}}, \cdot) \otimes k_{\mathcal{P}}(\tilde{\mathbf{x}}, \cdot) \right] \preceq \beta_{\mathcal{P}} \mathbb{E} [k_{\mathcal{P}}(\tilde{\mathbf{x}}, \cdot) \otimes k_{\mathcal{P}}(\tilde{\mathbf{x}}, \cdot)] = \beta_{\mathcal{P}} \mathbb{E} [\tilde{\mathcal{K}}_{\mathcal{P}}^{(1)}] = \beta_{\mathcal{P}} \mathcal{K}_{\mathcal{P}}$$

which implies $L_1 \leq \beta_{\mathcal{P}}$ due to the definition of L_1 in equation (43).

Deriving L_m

$$L_m := \frac{L_1}{m} + \frac{(m-1)L}{m} \leq \frac{\beta_{\mathcal{P}} + (m-1)\lambda_{q+1}}{m} \tag{46}$$

Defining $\tilde{\kappa}$ as the smallest positive real number such that

$$\mathbb{E} \left[\|k_{\mathcal{P}}(\tilde{\mathbf{x}}, \cdot)\|_{\mathcal{K}_{\mathcal{P}}^{-1}}^2 k_{\mathcal{P}}(\tilde{\mathbf{x}}, \cdot) \otimes k_{\mathcal{P}}(\tilde{\mathbf{x}}, \cdot) \right] \preceq \tilde{\kappa} \mathcal{K}_{\mathcal{P}} \tag{47}$$

Deriving $\|k_{\mathcal{P}}(\mathbf{x}_i, \cdot)\|_{\mathcal{K}_{\mathcal{P}}^{-1}}^2$

$$\begin{aligned}
 \|k_{\mathcal{P}}(\mathbf{x}_i, \cdot)\|_{\mathcal{K}_{\mathcal{P}}^{-1}}^2 &= \langle k_{\mathcal{P}}(\mathbf{x}_i, \cdot), \mathcal{K}_{\mathcal{P}}^{-1} k_{\mathcal{P}}(\mathbf{x}_i, \cdot) \rangle_{\mathbb{H}_{\mathcal{P}}} \\
 &\stackrel{(a)}{=} \left\langle k_{\mathcal{P}}(\mathbf{x}_i, \cdot), \left(\sum_{j=1}^q \frac{1}{\lambda_{q+1}} \psi'_j \otimes \psi'_j + \sum_{j=q+1}^n \frac{1}{\lambda_j} \psi'_j \otimes \psi'_j \right) k_{\mathcal{P}}(\mathbf{x}_i, \cdot) \right\rangle_{\mathbb{H}_{\mathcal{P}}} \\
 &= \sum_{j=1}^q \frac{1}{\lambda_{q+1}} \langle \psi'_j, k_{\mathcal{P}}(\mathbf{x}_i, \cdot) \rangle_{\mathbb{H}_{\mathcal{P}}}^2 + \sum_{j=q+1}^n \frac{1}{\lambda_j} \langle \psi'_j, k_{\mathcal{P}}(\mathbf{x}_i, \cdot) \rangle_{\mathbb{H}_{\mathcal{P}}}^2 \\
 &\stackrel{(b)}{=} n \left\{ \sum_{j=1}^q e_{ji}^2 + \sum_{j=q+1}^n e_{ji}^2 \right\} \\
 &\stackrel{(c)}{=} n
 \end{aligned} \tag{48}$$

where (a) is from the eigendecomposition of $\mathcal{K}_{\mathcal{P}}$, (b) follows from using eigenfunction evaluation with ψ' in $\mathbb{H}_{\mathcal{P}}$ and (c) follows from the fact that $\mathbf{E} = [\mathbf{e}_1 \ \mathbf{e}_2 \ \dots \ \mathbf{e}_n]$ is an orthonormal matrix which implies $\mathbf{E}^\top \mathbf{E} = \mathbf{I} = \mathbf{E} \mathbf{E}^\top$.

Deriving $\tilde{\kappa}$

$$\mathbb{E} \left[\|k_{\mathcal{P}}(\tilde{x}, \cdot)\|_{\mathcal{K}_{\mathcal{P}}}^2 k_{\mathcal{P}}(\tilde{x}, \cdot) \otimes k_{\mathcal{P}}(\tilde{x}, \cdot) \right] \quad (49)$$

$$\stackrel{(a)}{=} n \mathbb{E} [k_{\mathcal{P}}(\tilde{x}, \cdot) \otimes k_{\mathcal{P}}(\tilde{x}, \cdot)] \quad (50)$$

$$= n \mathbb{E} \left[\widetilde{\mathcal{K}}_{\mathcal{P}}^{(1)} \right] \quad (51)$$

$$\stackrel{(b)}{\implies} \tilde{\kappa} = n \quad (52)$$

where (a) follows from equation (48) and (b) from definition of $\tilde{\kappa}$ in equation (47).

F.5 MaSS parameters after preconditioning

Using (46), η_1 defined in (23a) can be written as

$$\eta_1^*(m) = \frac{m}{\beta_{\mathcal{P}} + (m-1)\lambda_{q+1}}$$

Using (52), (42), we can write η_2, γ defined in (23b), (23c) as

$$\begin{aligned} \eta_2^*(m) &= \frac{\eta_1^*(m) \sqrt{(\beta_{\mathcal{P}} + (m-1)\lambda_{q+1})(n+m-1)}}{\sqrt{(\beta_{\mathcal{P}} + (m-1)\lambda_{q+1})(n+m-1)} + m\sqrt{\lambda_n}} \\ \gamma^*(m) &= \frac{\sqrt{(\beta_{\mathcal{P}} + (m-1)\lambda_{q+1})(n+m-1)} - m\sqrt{\lambda_n}}{\sqrt{(\beta_{\mathcal{P}} + (m-1)\lambda_{q+1})(n+m-1)} + m\sqrt{\lambda_n}} \end{aligned}$$

Since the kernel matrix is positive definite we see that $\lambda_{q+1} \geq \beta/n$ and using (52) we get regime critical points m_1^*, m_2^* defined in [Liu and Belkin \(2020\)](#) as

$$\begin{aligned} m_1^* &= \min \left(\frac{\beta_{\mathcal{P}}}{\lambda_{q+1}}, n \right) = \frac{\beta_{\mathcal{P}}}{\lambda_{q+1}} \\ m_2^* &= \max \left(\frac{\beta_{\mathcal{P}}}{\lambda_{q+1}}, n \right) = n \end{aligned}$$

Note that there is no saturation regime for kernel methods

If optimal mini-batch size for linear regime $m_* := m_1^*$ is used, then optimal MaSS parameters are

$$\begin{aligned} \eta_1^* &= \frac{m_*^2}{\beta_{\mathcal{P}}(2m_* - 1)} \\ \eta_2^* &= \frac{m_*^2 \sqrt{n + m_* - 1}}{\beta_{\mathcal{P}}(2m_* - 1) \sqrt{n + m_* - 1} + m_* \sqrt{m_* \lambda_n} \beta_{\mathcal{P}}(2m_* - 1)} \\ \gamma^* &= \frac{\sqrt{\beta_{\mathcal{P}}(2m_* - 1)} \sqrt{n + m_* - 1} - m_* \sqrt{m_* \lambda_n}}{\sqrt{\beta_{\mathcal{P}}(2m_* - 1)} \sqrt{n + m_* - 1} + m_* \sqrt{m_* \lambda_n}} \end{aligned}$$

Informal: Assume large natural image dataset with gaussian/laplacian kernel. Due to the eigenvalue decay, m_* is reasonably large and $\beta_{\mathcal{P}}$ is usually very close to 1. The MaSS parameters can be approximated as follows

$$\begin{aligned} \eta_1^* &\approx \frac{m_*}{2} \\ \eta_2^* &\approx \frac{m_* \sqrt{n + m_* - 1}}{2 \sqrt{n + m_* - 1} + m_* \sqrt{2 \lambda_n}} \\ \gamma^* &\approx \frac{\sqrt{2} \sqrt{n + m_* - 1} - m_* \sqrt{\lambda_n}}{\sqrt{2} \sqrt{n + m_* - 1} + m_* \sqrt{\lambda_n}} \end{aligned}$$

Proposition 10. Suppose $m \leq \frac{n}{2}$, we have

$$m\sqrt{\tilde{\kappa}_m \kappa_m} \leq n\sqrt{\kappa}$$

Proof. According to the definitions of $\tilde{\kappa}_m$ and κ_m , the statement is equivalent to

$$m^2 \left(\frac{n}{m} + \frac{m-1}{m} \right) \left(\frac{L_1 + (m-1)\lambda_{q+1}}{\lambda_n m} \right) \leq n^2 \frac{\lambda_1}{\lambda_n},$$

i.e.

$$(n+m-1)(L_1 + (m-1)\lambda_{q+1}) \leq n^2 \lambda_1.$$

Since $L_1 = \max_i K(x_i, x_i) \leq \lambda_1$, we have

$$\begin{aligned} (n+m-1)(L_1 + (m-1)\lambda_{q+1}) &\leq 2n(\lambda_1 + (m-1)\lambda_1) \\ &\leq 2mn\lambda_1 \\ &\leq n^2 \lambda_1. \end{aligned}$$

□

G Acceleration of EigenPro 4

Accelerated iteration of EigenPro 3 in RKHS:

$$\begin{aligned} f_{t+1} &\leftarrow \text{proj}_{\mathcal{Z}}(g_t - \eta_1 \mathcal{P}_s \tilde{\nabla}_f \mathbf{L}(g_t)) \\ g_{t+1} &\leftarrow \text{proj}_{\mathcal{Z}}((1+\gamma)f_{t+1} - \gamma f_t + \eta_2 \mathcal{P}_s \tilde{\nabla}_f \mathbf{L}(g_t)) \end{aligned}$$

Suppose you project after every T iterations.

Consider the AxlePro2 iterations

$$f_{t+1} = g_t - \eta \mathcal{P}_s \tilde{\nabla}_f \mathbf{L}(g_t) \tag{53}$$

$$g_{t+1} = (1+\gamma)f_{t+1} - \gamma f_t + \eta_2 \mathcal{P}_s \tilde{\nabla}_f \mathbf{L}(g_t) \tag{54}$$

for $t = 1, 2, \dots, T$ and $f_T = \text{proj}_{\mathcal{Z}}(f_T)$ which requires $f_T(Z)$ and $g_T(Z)$ which will be computed iteratively.

Suppose B_t is the batch at step t , define $C_t := C_{t-1} \cup B_t$ and $C_0 = \emptyset$ and let $f_t = S_Z^* \alpha_t^Z + S_{C_t}^* \alpha_t^X + S_J^* \alpha_t^J$ and $g_t = S_Z^* \beta_t^Z + S_{C_t}^* \beta_t^X + S_J^* \beta_t^J$.

Then $\tilde{\nabla}_f \mathbf{L}(g_t) \in \text{range}(S_{B_{t+1}}^*)$ given by

$$\begin{aligned} \tilde{\nabla}_f \mathbf{L}(g_t) &= S_{B_{t+1}}^* \mathbf{v}_t \\ \mathbf{v}_t &:= g_t(X[B_{t+1}]) - y_t \\ g_t(X[B_{t+1}]) &= K(X[B_{t+1}], Z) \beta_t^Z + K(X[B_{t+1}], X[C_t]) \beta_t^X + K(X[B_{t+1}], X[J]) \beta_t^J \\ \mathbf{w}_t &:= \mathbf{G} \mathbf{G}^\top K(X[J], X[B_{t+1}]) \mathbf{v}_t \\ \mathcal{P}_s \tilde{\nabla}_f \mathbf{L}(g_t) &= S_{B_{t+1}}^* \mathbf{v}_t - S_J^* \mathbf{w}_t \end{aligned}$$

$$\begin{aligned} S_Z^* \alpha_{t+1}^Z + S_{C_{t+1}}^* \alpha_{t+1}^X + S_J^* \alpha_{t+1}^J &\leftarrow S_Z^* \beta_t^Z + S_{C_t}^* \beta_t^X + S_J^* \beta_t^J - \eta_1 (S_{B_{t+1}}^* \mathbf{v}_t - S_J^* \mathbf{w}_t) \\ S_Z^* \beta_{t+1}^Z + S_{C_{t+1}}^* \beta_{t+1}^X + S_J^* \beta_{t+1}^J &\leftarrow (1+\gamma)(S_Z^* \alpha_{t+1}^Z + S_{C_t}^* \alpha_{t+1}^X + S_J^* \alpha_{t+1}^J) - \gamma(S_Z^* \alpha_t^Z + S_{C_t}^* \alpha_t^X + S_J^* \alpha_t^J) \\ &\quad + \eta_2 (S_{B_{t+1}}^* \mathbf{v}_t - S_J^* \mathbf{w}_t) \end{aligned}$$

This gives us

$$\alpha_{t+1}^Z \leftarrow \beta_t^Z \quad (55a)$$

$$\beta_{t+1}^Z \leftarrow (1 + \gamma)\alpha_{t+1}^Z - \gamma\alpha_t^Z \quad (55b)$$

$$\alpha_{t+1}^J \leftarrow \beta_t^J + \eta_1 \mathbf{w}_t \quad (55c)$$

$$\beta_{t+1}^J \leftarrow (1 + \gamma)\alpha_{t+1}^J - \gamma\alpha_t^J - \eta_2 \mathbf{w}_t \quad (55d)$$

$$\alpha_{t+1}^X[C_t] \leftarrow \beta_t^X[C_t] \quad (55e)$$

$$\alpha_{t+1}^X[B_{t+1}] \leftarrow -\eta_1 \mathbf{v}_t \quad (55f)$$

$$\beta_{t+1}^X[C_t] \leftarrow (1 + \gamma)\alpha_{t+1}^X[C_t] - \gamma\alpha_t^X[C_t] \quad (55g)$$

$$\beta_{t+1}^X[B_{t+1}] \leftarrow (1 + \gamma)\alpha_{t+1}^X[B_{t+1}] + \eta_2 \mathbf{v}_t = (\eta_2 - (1 + \gamma)\eta_1)\mathbf{v}_t \quad (55h)$$

Claim 1. If we run the extrapolation iteration $a_{t+1} = (1 + \gamma)a_t - \gamma a_{t-1}$ initialized at $a_1 \neq a_0$, then

$$a_t = a_0 + \frac{1 - \gamma^t}{1 - \gamma}(a_1 - a_0). \quad (56)$$

Proof. We will prove this by induction. Observe that the $t = 1$ case hold trivially. Assume that the formula is true for $t \leq \tau$ for some $\tau \geq 1$. If we prove that the formula holds for $t = \tau + 1$, we are done.

$$a_{\tau+1} = (1 + \gamma)a_\tau - \gamma a_{\tau-1} = (1 + \gamma) \left(a_0 + \frac{1 - \gamma^\tau}{1 - \gamma}(a_1 - a_0) \right) - \gamma \left(a_0 + \frac{1 - \gamma^{\tau-1}}{1 - \gamma}(a_1 - a_0) \right) \quad (57)$$

$$= a_0 + \frac{a_1 - a_0}{1 - \gamma} ((1 + \gamma)(1 - \gamma^\tau) - \gamma(1 - \gamma^{\tau-1})) \quad (58)$$

$$= a_0 + \frac{a_1 - a_0}{1 - \gamma} (1 + \gamma - \gamma^\tau - \gamma^{\tau+1} - \gamma + \gamma^\tau) = a_0 + \frac{1 - \gamma^{\tau+1}}{1 - \gamma}(a_1 - a_0). \quad (59)$$

which proves the claim. \square

Let t_B be the time step when batch B is chosen for the first time, i.e., $B_t = B$.

$$\beta_t^X[B] = c_t \mathbf{v}_{t_B} \quad \text{where} \quad c_t = \begin{cases} 0 & t < t_B \\ c_1 & t = t_B \\ (c_1 + \frac{1 - \gamma^{(t - t_B + 1)}}{1 - \gamma} c_2) & t > t_B \end{cases} \quad (60)$$

where $c_1 = -((1 + \gamma)\eta_1 - \eta_2)$, and $c_2 = \eta_2 - \eta_1$

Finally, after T iterations, we need to project f_T and g_T back onto $\text{span}(S_Z^*)$. To that end, we must calculate $g_T(Z)$. To that end, consider

$$f_T(Z) = K(Z, Z)\alpha_T^Z + K(Z, X[C_T])\alpha_T^X + K(Z, X[J])\alpha_T^J \quad (61)$$

$$g_T(Z) = K(Z, Z)\beta_T^Z + K(Z, X[C_T])\beta_T^X + K(Z, X[J])\beta_T^J \quad (62)$$

We can compute

$$f_{T+1} = \text{proj}_Z(f_T) \quad (63)$$

$$g_{T+1} = \text{proj}_Z(g_T) \quad (64)$$

by

$$f_{T+1} = S_Z^* K(Z, Z)^{-1} f_T(Z) \quad (65)$$

$$g_{T+1} = S_Z^* K(Z, Z)^{-1} g_T(Z) \quad (66)$$

which can be implemented as

$$\alpha_{T+1}^Z = K(Z, Z)^{-1} f_T(Z) = \alpha_T^Z + K(Z, Z)^{-1} (K(Z, X[C_T])\alpha_T^X + K(Z, X[J])\alpha_T^J) \quad (67)$$

$$\beta_{T+1}^Z = K(Z, Z)^{-1} g_T(Z) = \beta_T^Z + K(Z, Z)^{-1} (K(Z, X[C_T])\beta_T^X + K(Z, X[J])\beta_T^J) \quad (68)$$

$$\mathbf{v}_t \leftarrow K(X[B], Z)\boldsymbol{\beta}_t - Y[B] \in \mathbb{R}^m \quad (69a)$$

$$\mathbf{w}_t \leftarrow K(Z, X[B])\mathbf{v}_t - K(Z, X[J])\mathbf{G}\mathbf{G}^\top K(X[J], X[B])\mathbf{v}_t \in \mathbb{R}^{|Z|} \quad (69b)$$

$$\mathbf{u}_t \leftarrow \text{solve } K(Z, Z)\mathbf{u}_t = \mathbf{w}_t \quad (69c)$$

$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_1 \mathbf{u}_t \quad (69d)$$

$$\boldsymbol{\beta}_{t+1} \leftarrow (1+\gamma)\boldsymbol{\alpha}_{t+1} - \gamma\boldsymbol{\alpha}_t + \eta_2 \mathbf{u}_t \quad (69e)$$

emulate the updates Acceleration algorithm of EigenPro 3.

H ADDITIONAL PROOFS

Proof of Proposition 1. For $f_t = S^* \boldsymbol{\alpha}_t$, a stochastic gradient with respect to the mini-batch $(X[B], Y[B])$ is given by

$$\tilde{\nabla}_f \mathcal{L}(f_t) = \frac{1}{|B|} S_B^* (S_B f_t - Y[B]) \quad (70a)$$

$$= \frac{1}{|B|} S_B^* (S_B S^* \boldsymbol{\alpha}_t - Y[B]) = S_B^* \mathbf{v}_t \quad (70b)$$

Observe that $S_B = \mathbf{H}_B S$, whereby $S_B^* = S^* \mathbf{H}_B^\top$. The claim follows immediately. \square

Proof of Proposition 3. We start by showing that

$$\mathcal{P} S_B^* = S_B^* - S^* \mathbf{F} \mathbf{F}^\top \mathbf{H}_B^\top. \quad (71)$$

For a vector $\mathbf{u} \in \mathbb{R}^m$, observe that

$$\mathcal{P} S_B^* \mathbf{u} = S_B^* \mathbf{u} - \sum_{i=1}^q \left(1 - \frac{\lambda_{q+1}}{\lambda_i}\right) \psi_i \otimes_{\mathbb{H}} \psi_i S_B^* \mathbf{u} \quad (72)$$

Now the term $\psi_i \otimes_{\mathbb{H}} \psi_i S_B^* \mathbf{u}$ simplifies as

$$\begin{aligned} \psi_i \langle \psi_i, S_B^* \mathbf{u} \rangle_{\mathbb{H}} &= \frac{1}{\lambda_i} S^* \mathbf{e}_i \langle S^* \mathbf{e}_i, S_B^* \mathbf{u} \rangle_{\mathbb{H}} \\ &= \frac{1}{\lambda_i} S^* \mathbf{e}_i \langle \mathbf{e}_i, S S^* \mathbf{H}_B^\top \mathbf{u} \rangle_{\mathbb{R}^n} \\ &= \frac{1}{\lambda_i} S^* \mathbf{e}_i \mathbf{e}_i^\top K(X, X) \mathbf{H}_B^\top \mathbf{u} \\ &= S^* \mathbf{e}_i \mathbf{e}_i^\top \mathbf{H}_B^\top \mathbf{u} \end{aligned}$$

where we have used the definition of adjoint and the fact that $S_B^* = (\mathbf{H}_B S)^* = S^* \mathbf{H}_B^\top$. Summing the q terms, and observing that $\mathbf{F} = \sum_{i=1}^q (1 - \frac{\lambda_{q+1}}{\lambda_i}) \mathbf{e}_i \mathbf{e}_i^\top$, we have $\mathcal{P} S_B^* \mathbf{u} = S_B^* \mathbf{u} - S^* \mathbf{F} \mathbf{F}^\top \mathbf{H}_B^\top \mathbf{u}$ for all $\mathbf{u} \in \mathbb{R}^m$. Since \mathbf{u} is arbitrary, this proves the claim in equation (71).

Next, observe that $\mathcal{P} \tilde{\nabla}_f \mathcal{L}(g_t) = \mathcal{P} S_B^* (S_B S^* \boldsymbol{\beta}_t - Y[B]) = \mathcal{P} S_B^* \mathbf{v}_t$, and rewriting the updates in equation (27a) using the relation $f_t = S^* \boldsymbol{\alpha}_t$ and $g_t = S^* \boldsymbol{\beta}_t$, we get

$$\begin{aligned} S^* \boldsymbol{\alpha}_{t+1} &\leftarrow S^* \boldsymbol{\beta}_t - \eta_1 \mathcal{P} S_B^* \mathbf{v}_t \\ &= S^* \boldsymbol{\beta}_t - \eta_1 S_B^* \mathbf{v}_t + \eta_1 S^* \mathbf{F} \mathbf{F}^\top \mathbf{H}_B^\top \mathbf{v}_t \\ &= S^* (\boldsymbol{\beta}_t - \eta_1 \mathbf{H}_B^\top \mathbf{v}_t + \eta_1 \mathbf{w}_t) \end{aligned}$$

which is indeed equation (17c). A similar calculation can also show equation (17d) emulates equation (27b). \square

Proof of Proposition 4. We start by showing that

$$\mathcal{Q} S_B^* = S_B^* - S_J^* \mathbf{G} \mathbf{G}^\top K(X[J], X[B]). \quad (73)$$

For a vector $\mathbf{u} \in \mathbb{R}^m$, observe that

$$\mathcal{Q}S_B^* \mathbf{u} = S_B^* \mathbf{u} - \sum_{i=1}^q \left(1 - \frac{\delta_{q+1}}{\delta_i}\right) \phi_i \otimes_{\mathbb{H}} \phi_i S_B^* \mathbf{u} \quad (74)$$

Now the term $\phi_i \otimes_{\mathbb{H}} \phi_i S_B^* \mathbf{u}$ simplifies as

$$\begin{aligned} \phi_i \langle \phi_i, S_B^* \mathbf{u} \rangle_{\mathbb{H}} &= \frac{1}{\delta_i} S_J^* \mathbf{d}_i \langle S_J^* \mathbf{d}_i, S_B^* \mathbf{u} \rangle_{\mathbb{H}} \\ &= \frac{1}{\delta_i} S_J^* \mathbf{d}_i \langle \mathbf{d}_i, S_J S_I^* \mathbf{u} \rangle_{\mathbb{R}^m} \\ &= \frac{1}{\delta_i} S_J^* \mathbf{d}_i \mathbf{d}_i^\top K(X[J], X[B]) \mathbf{u}. \end{aligned}$$

Summing the q terms, and observing that $\mathbf{G} = \sum_{i=1}^q \frac{1}{\delta_i} (1 - \frac{\delta_{q+1}}{\delta_i}) \mathbf{d}_i \mathbf{d}_i^\top$ we have $\mathcal{Q}S_B^* \mathbf{u} = S_B^* \mathbf{u} - S_J^* \mathbf{G} \mathbf{G}^\top K(X[J], X[B]) \mathbf{u}$ for all $\mathbf{u} \in \mathbb{R}^m$. Since \mathbf{u} is arbitrary, this proves the claim in equation (73). The rest of the proof proceeds similar to the proof of Proposition 3. \square

Table 6: Laplacian Kernel

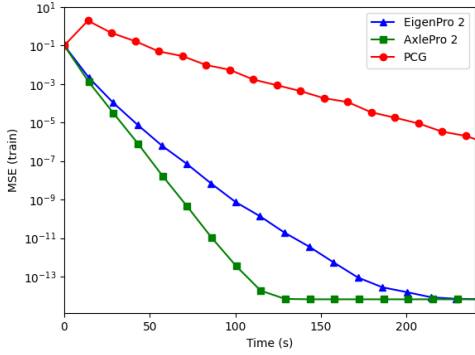
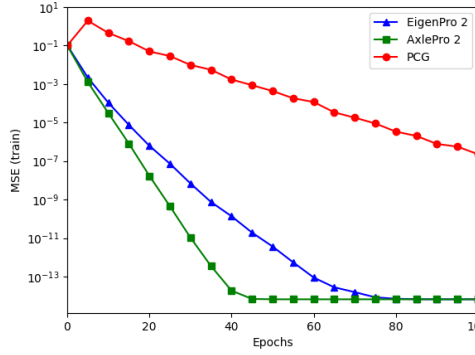
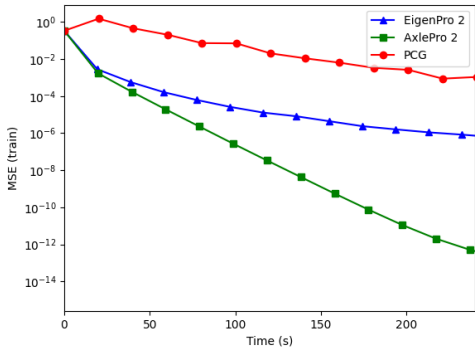
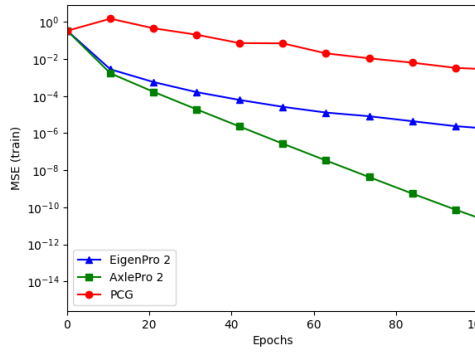
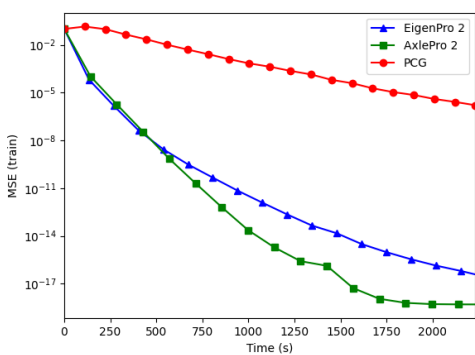
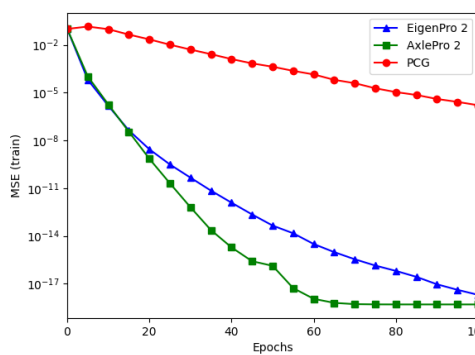
Dataset	MSE v/s Time	MSE v/s Epochs
CIFAR10		
Stellar Classification		
EMNIST Digits		

Table 7: Gaussian Kernel

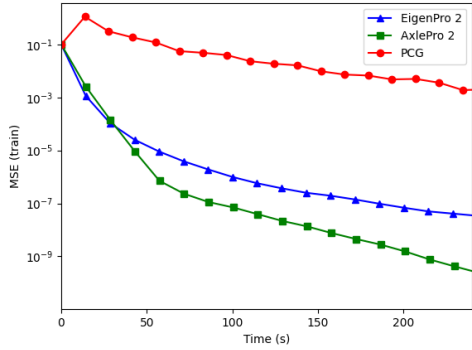
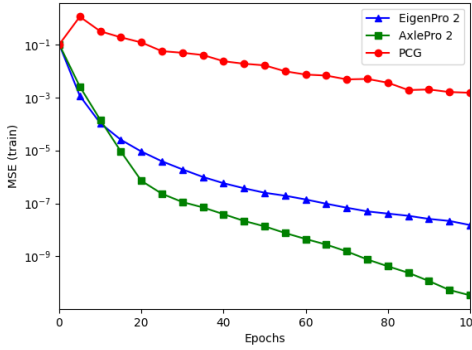
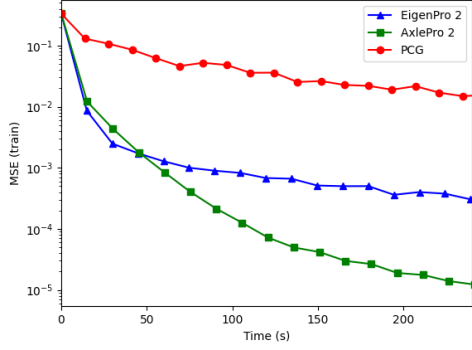
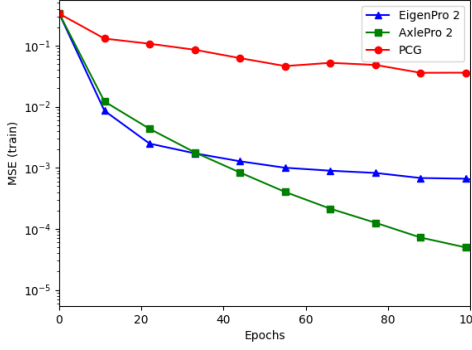
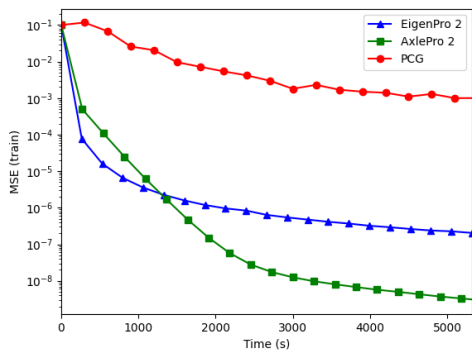
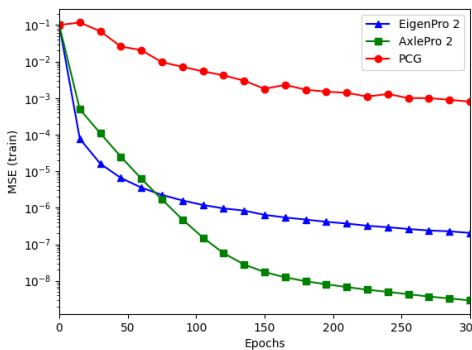
Dataset	MSE v/s Time	MSE v/s Epochs
CIFAR10		
Stellar Classification		
EMNIST Digits		

Table 8: Laplacian Kernel with single precision

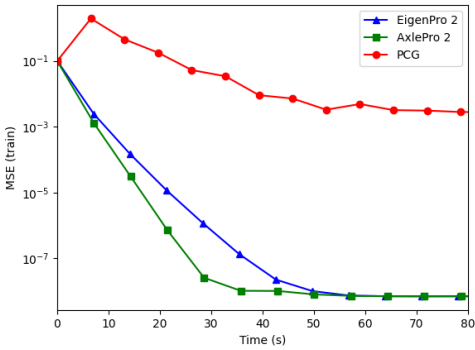
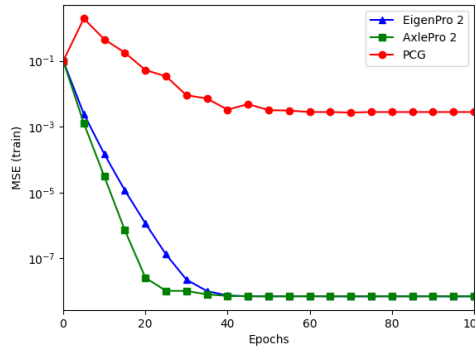
Dataset	MSE v/s Time	MSE v/s Epochs
CIFAR10		

Table 9: Myrtle5 Kernel with stored kernel matrix

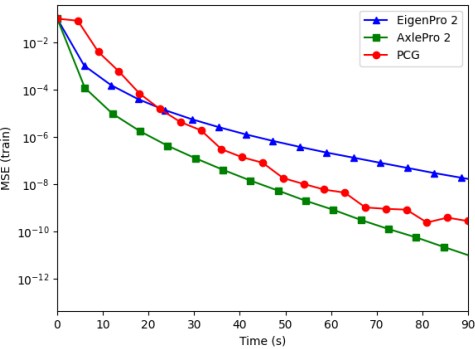
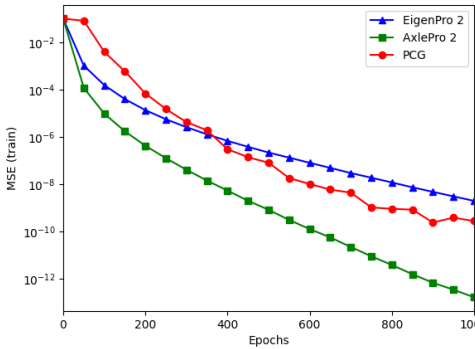
Dataset	MSE v/s Time	MSE v/s Epochs
CIFAR10		

Table 10: Acceleration by Nyström approximation

Dataset	MSE v/s Time	MSE v/s Epochs
$n = 5k,$ $s = 800$		
