AxlePro: Accelerating Batched Training of Kernel Machines with Momentum

Anonymous Authors[1]

Abstract

Kernel methods are a classical family of nonlinear predictive models in machine learning which have regained popularity over the past few years. We provide a novel algorithm for learning kernel models. Our algorithm, AxlePro, is derived to emulate a function space iteration based on momentum-accelerated preconditioned stochastic gradient descent, and applies to arbitrary positive semidefinite kernel functions. We also provide a limited memory version LM-AxlePro via approximate preconditioning, which makes the implementation fast, scalable, batch-friendly, and memory-efficient. Both these emulated algorithms come equipped with convergence guarantees in the function space. We demonstrate their faster speed in comparison to the state of the art via experiments.

1. Introduction

Deep Neural Networks have become the dominant paradigm of modelling and data analysis for large scale problems in machine learning today. These model architectures have shown remarkable progress in our ability to extract information from complex datasets, breaking unprecedented benchmarks every so often. Certain architectures of these networks, such as wide networks 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 kernel functions, so-called *neural kernels*, have also shown remarkable performance on contemporary machine learning problems, see Shankar et al. (2020); Bietti & Bach (2020); Arora et al. (2019); Adlam et al. (2023); Han et al. (2022) among others. Furthermore, recent work also shows that kernel machines can be re-engineered to learn task-specific features in a supervised manner Radhakrishnan et al. (2022); Beaglehole et al. (2023), which leads to further performance gains.

Training a kernel machine involves solving,

$$\mathbf{K}\boldsymbol{\alpha} = Y, \tag{1}$$

a linear system of equations, where $\mathbf{K} = (K(x_i, x_j))$ is a $n \times n$ positive definite matrix of pairwise evaluations of a kernel function K on training inputs, and Y is the vector of all n targets. While there are many variations, such as different loss functions and regularizers, we leave aside generalizations for future work.

1.1. Main contribution

We propose two new algorithms, called AxlePro and its limited memory version LM-AxlePro. These algorithms are derived to emulate infinite dimensional iterations in the RKHS, based on momentum accelerated preconditioned SGD. Hence they come equipped with function space guarantees on convergence. Our numerical experiments show that LM-AxlePro is the fastest scalable algorithm for training of kernel machines. Along with empirical verification for acceleration using LM-AxlePro, we also provide a `PyTorch` implementation that can take advantage of GPUs.

1.2. Properties of AxlePro

Fast early stage training. A peculiar property of machine learning problems is that training may require very few number of epochs. In fact, state-of-the-art large language models are often trained with less than a single epoch. AxlePro updates the model several times in a single epoch (once per mini-batch), similar to SGD or EigenPro Ma & Belkin (2019). On the other hand, packages such as FALKON or GPyTorch, which are based on PCG, only update the model once over an entire epoch, which slows them down drastically in early stages of training.

Fast late stage training. While EigenPro is fast in early stage of training, it suffers from slower late stage training. AxlePro is faster than EigenProin late stage training due to an appropriately chosen momentum-based acceleration.

Stable. Both AxlePro and LM-AxlePro are stable to single precision arithmetic. This is often not the case with

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Algorithm name	Derivation	cost per step	steps to convergence
GPyTorch Gardner et al. (2018)	Preconditioned conjugate gradient (PCG)	n^2	$\sqrt{\kappa}$
EigenPro Ma & Belkin (2017)	Preconditioned SGD (PSGD)	$nm + nq$	κ_m
EigenPro2.0 Ma & Belkin (2019)	PSGD+approx. preconditioning	nm	$\kappa_m(1+\varepsilon)$
AxlePro (ours, Proposition 5)	PSGD+momentum	$nm + nq$	$\sqrt{\widetilde{\kappa}_m \kappa_m}$
LM-AxlePro (ours, Proposition 6)	PSGD+momentum+approx. preconditioning	nm	$\sqrt{\widetilde{\kappa}_m \kappa_m}(1+\varepsilon)$

Table 1. Comparison between different iterative algorithms. m is the batch size, typically $m \ll n$. Condition number κ is the ratio of largest to smallest positive eigenvalue, $\kappa_m, \widetilde{\kappa}_m$ are modified versions of κ defined in equation (23), with $\widetilde{\kappa}_m \leq \kappa_m \leq \kappa$. Here q is the level of the preconditioner and ε is the error due to approximate preconditioning.

Algorithm 1 AxlePro

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

Algorithm 2 LM-AxlePro

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

CG based solvers which are more sensitive to numerical roundoff errors.

Memory-efficient. LM-AxlePro requires storage of $2n$ floats where n is the number of training samples, this is only slightly worse than the n float storage requirement of EigenPro2.0, and doesn't raise practical challenges. There is room some improvement in this regard which we leave for future work. LM-AxlePro can use large batch sizes so that GPUs are fully utilized to minimize training time, similar to EigenPro2.0.

1.3. Related work

The problem of scalable training algorithms for kernel models has been studied extensively in the past with several lines of attack. Approaches for scalable training include:

1. **Linear system solvers** for symmetric positive definite matrices. Examples of such solvers include conjugate gradient (CG) and preconditioned version PCG. See Rudi et al. (2017); Meanti et al. (2020); Yin et al. (2021) for example. While these methods are the fastest in the asymptotic sense, they do not decrease training error monotonically, and require a large number of epochs. Hence, even though they can be implemented in a matrix-free manner, the total cost of computations is very large for machine learning applications.

2. **Randomized methods** such as those based on random fourier features, sketching, or low-rank decompostions provide a way to train in a scalable manner Rahimi & Recht (2007); Carratino et al. (2018); Chen et al. (2022). See Han et al. (2022) and the references therein for a more recent overview on sketching and random features. However this approach is not universal, i.e., requires special structure in the kernel function. Sketching based solutions have also been studied in this context Chowd-

Algorithm 3 EigenPro2.0 Ma & Belkin (2019)

Input: kernel K, batch size m, indices $J = \{j_\ell\}_{\ell=1}^{s} \subset \{1, 2, \ldots, n\}$, learning rate $\eta_0 > 0$.
Output: $f : \mathcal{X} \to \mathbb{R}$ that solves (5) approximately.
setup: $\boldsymbol{\alpha} \leftarrow \mathbf{0}_n$.
$(\boldsymbol{D}, \Delta, \delta_{q+1}) \leftarrow$ top-q eigensystem of $K(X[J], X[J])$
$\boldsymbol{G} \leftarrow \boldsymbol{D}\sqrt{\Delta^{-1}(I_q - \delta_{q+1}\Delta^{-1})}$
repeat
  Fetch a batch of indices $B \subset \{1, 2, .., n\}$
  $\boldsymbol{v} \leftarrow K(X[B], X)\boldsymbol{\alpha} - Y[B] \in \mathbb{R}^m$
  $\boldsymbol{w} \leftarrow \boldsymbol{G}\boldsymbol{G}^\top K(X[J], X[B])\boldsymbol{v} \in \mathbb{R}^s$
  $\boldsymbol{\alpha}[B] - \leftarrow \eta_0 \boldsymbol{v}$ {gradient step}
  $\boldsymbol{\alpha}[J] + \leftarrow \eta_0 \boldsymbol{w}$ {correction}
until Stopping criterion is reached
return $f_t(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i)$

hury et al. (2018).

3. **Primal space solvers** such as SGD Shalev-Shwartz et al. (2007); Camoriano et al. (2016) and its variants can be very effective. In fact Ma & Belkin (2017; 2019) are state of the art in solving kernel machines in a fast and scalable manner for any kernel functions. These methods derive an iteration in the RKHS, but emulate these computations exactly in the euclidean space.

2. Problem formulation

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 assume $x_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$. Kernel machines are functions of the form

$$f(x) = \sum_{i=1}^{n} K(x, x_i)\alpha_i. \tag{2}$$

where $\alpha_i \in \mathbb{R}$ are weights and $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite kernel function. Such models arise as the solutions to empirical risk minimization problems over the Reproducing Kernel Hilbert Spaces (RKHS) corresponding to K, thanks to the Representer Theorem Kimeldorf & Wahba (1970); Schölkopf et al. (2001). The reproducing property of K, means we have,

$$\langle f, K(\cdot, x)\rangle_{\mathcal{H}} = f(x), \qquad \forall f \in \mathcal{H}, x \in \mathcal{X}, \tag{3}$$

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

Notation. For tuples $A = (a_i) \in \mathcal{X}^k$ and $U = (u_i) \in \mathcal{X}^\ell$, $K(A, U) \in \mathbb{R}^{k \times \ell}$ denotes the kernel matrix with entries $K(a_i, u_j)$. For any $f \in \mathcal{H}$, by $f \otimes_{\mathcal{H}} f$, we denote the rank-1 self-adjoint operator $\mathcal{H} \to \mathcal{H}$, which acts as $(f \otimes_{\mathcal{H}} f)(g) = f\langle f, g\rangle_{\mathcal{H}}$. Here, if $f = K(x, \cdot)$, then we have $(K(x, \cdot) \otimes_{\mathcal{H}} K(x, \cdot))(g) = K(\cdot, x)g(x)$, by equation (3). A more detailed table of notations is provided in the appendix.

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

$$Sf = (f(x_1), \ldots, f(x_n)) \in \mathbb{R}^n. \tag{4}$$

In this paper we are interested in finding,

$$f^* = \arg\min_{f \in \mathcal{H}} \mathsf{L}(f) = \tfrac{1}{2n}\left\|Sf - Y\right\|^2. \tag{5}$$

The minimum-norm solution to the above problem is equation (2) with $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ that solves equation (1).

Finding a solution $\boldsymbol{\alpha}$ to equation (1) is computationally challenging since inverting a matrix can take $O(n^3)$ steps. Fortunately, for the purposes of machine learning, it suffices to solve (1) approximately. Several such approximate solvers exist which can provide solutions in $O(n^2)$ steps. Such solvers involve iterative methods such as conjugate gradient descent, stochastic gradient descent, and variants.

Definition 2 (Adjoint of evaluation operator)**.** The adjoint of S, defined in equation (4), denoted $S^* : \mathbb{R}^n \to \mathcal{H}$, is the extension operator which one can show acts as

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

Define the empirical covariance operator,

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

Definition 3 (Slicing)**.** For an n-tuple $X = (x_1, x_2, \ldots, x_n)$ of size n and an s-tuple of indices $J = (j_1, j_2, \ldots, j_s)$, we denote by $X[J]$ the s-tuple $(x_{j_1}, x_{j_2}, \ldots, x_{j_s})$, following `numpy` notation. For a matrix $\boldsymbol{A}$ and vector $\boldsymbol{v}$, the quantities $\boldsymbol{A}[J]$ and $\boldsymbol{v}[J]$ are the submatrix and subvector of $\boldsymbol{A}$ and $\boldsymbol{v}$ respectively indexed by J. By $\boldsymbol{H}_J$ we denote the $|J| \times n$ matrix formed by stacking the $|J|$ rows of identity I_n. By S_J we denote the operator $\boldsymbol{H}_J S$ which maps $\mathcal{H} \to \mathbb{R}^{|J|}$ so that $S_J f = (f(x_{j_1}), f(x_{j_2}), \ldots, f(x_{j_s}))$. Similarly, define $\mathcal{K}_J = \frac{1}{|J|}S_J^* S_J$.

Definition 4 (top-q eigsystem)**.** Given a symmetric positive definite matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, for $i = 1, 2, \ldots, q+1$, let $\boldsymbol{e}_i, \lambda_i$ be the top-$q+1$ eigenpairs of $\boldsymbol{A}$, i.e., $\boldsymbol{A}\boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$, with descending eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{q+1} 0$. Let $\boldsymbol{E} := [\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_q] \in \mathbb{R}^{n \times q}$, and $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_q) \in \mathbb{R}^{q \times q}$ be a diagonal matrix. We call the tuple $(\boldsymbol{E}, \Lambda, \lambda_{q+1})$ the top-q eigsystem of $\boldsymbol{A}$.

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

3. AxlePro: derivation of the algorithm

We first start by deriving AxlePro given in algorithm 1, and then proceed to derive the limited memory version LM-AxlePro in algorithm 2. AxlePro is derived by emulating PSGD in the RKHS. LM-AxlePro approximates the preconditioner in AxlePro by applying a Nyström approximation for the precondioner Abedsoltan et al. (2024).

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

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

Note this is the Fréchet derivative, and belongs to the RKHS $\nabla_f \mathsf{L} \in \mathcal{H}$, and the Hessian is an operator on the RKHS $\nabla_f^2 \mathsf{L} : \mathcal{H} \to \mathcal{H}$. We are now ready to derive the algorithm 2.

3.1. SGD in the primal

Our goal is to solve equation (5). If we start by initializing $f_0 \in \text{range}(S^*)$, then one can show that the iterates f_t of gradient descent

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

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

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

$$f_{t+1} \leftarrow f_t - \eta_0 \widetilde{\nabla}_f \mathsf{L}(f_t). \tag{8}$$

which converges exponentially as shown in Ma et al. (2018, Theorem 1).

Proposition 1. *The following update equations*

$$\boldsymbol{v}_t \leftarrow K(X[B], X)\boldsymbol{\alpha}_t - Y[B] \tag{9a}$$
$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\alpha}_t - \eta_0 \boldsymbol{H}_B^\top \boldsymbol{v}_t. \tag{9b}$$

emulate one step of updates in equation (8) *via the relation* $f_t = S^* \boldsymbol{\alpha}_t$.

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

$$\widetilde{\nabla}_f \mathsf{L}(f_t) = \frac{1}{|B|} S_B^*(S_B f_t - Y[B]) \tag{10a}$$
$$= \frac{1}{|B|} S_B^*(S_B S^* \boldsymbol{\alpha}_t - Y[B]) = S_B^* \boldsymbol{v}_t \tag{10b}$$

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

Remark 1. 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: Acceleration via momentum

The MaSS updates in $\mathcal{H}$, following the update rule from Liu & Belkin (2020) yields

$$f_{t+1} \leftarrow g_t - \eta_1 \widetilde{\nabla}_f \mathsf{L}(g_t), \tag{11a}$$
$$g_{t+1} \leftarrow (1+\gamma) f_{t+1} - \gamma f_t + \eta_2 \widetilde{\nabla}_f \mathsf{L}(g_t), \tag{11b}$$

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

Remark 2. 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 & Belkin (2020, Theorem 2).

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

Proposition 2. *The following update equations*

$$\boldsymbol{v}_t \leftarrow K(X[B], X)\boldsymbol{\beta}_t - Y[B] \tag{12a}$$
$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_1 \boldsymbol{H}_B^\top \boldsymbol{v}_t, \tag{12b}$$
$$\boldsymbol{\beta}_{t+1} \leftarrow (1+\gamma)\boldsymbol{\alpha}_{t+1} - \gamma \boldsymbol{\alpha}_t + \eta_2 \boldsymbol{H}_B^\top \boldsymbol{v}_t, \tag{12c}$$

emulate one step of updates in equation (11) *via the relation* $f_t := S^* \boldsymbol{\alpha}_t$ *and* $g_t := S^* \boldsymbol{\beta}_t$.

The proof is similar to that of Proposition 1.

3.3. Acceleration via spectral preconditioning

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

Define the spectral preconditioner $\mathcal{P} : \mathcal{H} \to \mathcal{H}$ given by,

$$\mathcal{P} := \mathcal{I} - \sum_{i=1}^{q} \left(1 - \tfrac{\lambda_{q+1}}{\lambda_i}\right) \psi_i \otimes_{\mathcal{H}} \psi_i, \tag{13}$$

$$\mathcal{P} f = f - \sum_{i=1}^{q} \left(1 - \tfrac{\lambda_{q+1}}{\lambda_i}\right) \langle f, \psi_i \rangle_{\mathcal{H}} \psi_i. \tag{14}$$

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, λ_n. Note that here we have used the fact that $\|\psi_i\| = 1$.

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

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

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

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

Proposition 3. *The following update equations*

$$\boldsymbol{v}_t \leftarrow K(X_m, X)\boldsymbol{\beta}_t - Y[B] \in \mathbb{R}^m \tag{16a}$$

$$\boldsymbol{w}_t \leftarrow \boldsymbol{F}\boldsymbol{F}^\top \boldsymbol{H}_B^\top \boldsymbol{v}_t \in \mathbb{R}^n \tag{16b}$$

$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_1 \boldsymbol{H}_B^\top \boldsymbol{v}_t + \eta_1 \boldsymbol{w}_t \tag{16c}$$

$$\boldsymbol{\beta}_{t+1} \leftarrow (1+\gamma)\boldsymbol{\alpha}_{t+1} - \gamma \boldsymbol{\alpha}_t + \eta_2 \boldsymbol{H}_B^\top \boldsymbol{v}_t - \eta_2 \boldsymbol{w}_t \tag{16d}$$

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

Remark 3. In Algorithm 1, lines (11-13) together implement equation (16c), whereas lines (14-16) together implement equation (16d). Note that here we have used the fact that $\boldsymbol{H}_B \boldsymbol{F} = \boldsymbol{F}[B] \in \mathbb{R}^{m \times q}$.

Proof. We start by showing that

$$\mathcal{P} S_B^* = S_B^* - S^* \boldsymbol{F}\boldsymbol{F}^\top \boldsymbol{H}_B^\top. \tag{17}$$

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

$$\mathcal{P} S_B^* \boldsymbol{u} = S_B^* \boldsymbol{u} - \sum_{i=1}^{q} \left(1 - \frac{\lambda_{q+1}}{\lambda_i}\right) \psi_i \otimes_{\mathcal{H}} \psi_i S_B^* \boldsymbol{u} \tag{18}$$

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

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

where we have used the defintion of adjoint and the fact that $S_B^* = (\boldsymbol{H}_B S)^* = S^* \boldsymbol{H}_B^\top$. Summing the q terms, and observing that $\boldsymbol{F} = \sum_{i=1}^{q} (1 - \frac{\lambda_{q+1}}{\lambda_i}) \boldsymbol{e}_i \boldsymbol{e}_i^\top$, we have $\mathcal{P} S_B^* \boldsymbol{u} = S_B^* \boldsymbol{u} - S^* \boldsymbol{F}\boldsymbol{F}^\top \boldsymbol{H}_B^\top \boldsymbol{u}$ for all $\boldsymbol{u} \in \mathbb{R}^m$. Since $\boldsymbol{u}$ is arbitrary, this proves the claim in equation (17).

Next, observe that $\mathcal{P} \widetilde{\nabla}_f \mathsf{L}(g_t) = \mathcal{P} S_B^* (S_B S^* \boldsymbol{\beta}_t - Y[B]) = \mathcal{P} S_B^* \boldsymbol{v}_t$, and rewriting the updates in equation (15a) using the relation $f_t = S^* \boldsymbol{\alpha}_t$ and $g_t = S^* \boldsymbol{\beta}_t$, we get

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

which is indeed equation (16c). A similar calculation can also show equation (16d) emulates equation (15b). □

3.4. Applying a Nyström approximate preconditioner

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

$$\mathcal{Q} := \mathcal{I} - \sum_{i=1}^{q} \left(1 - \tfrac{\delta_{q+1}}{\delta_i}\right) \phi_i \otimes \phi_i. \tag{19}$$

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

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

Proposition 4. *The following update equations*

$$\boldsymbol{v}_t \leftarrow K(X[B], X)\boldsymbol{\beta}_t - Y[B] \in \mathbb{R}^m \tag{20a}$$

$$\boldsymbol{w}_t \leftarrow \boldsymbol{G}\boldsymbol{G}^\top K(X[J], X[B]) \boldsymbol{v}_t \in \mathbb{R}^s \tag{20b}$$

$$\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_1 \boldsymbol{H}_B{}^\top \boldsymbol{v}_t + \eta_1 \boldsymbol{H}_J{}^\top \boldsymbol{w}_t \tag{20c}$$

$$\boldsymbol{\beta}_{t+1} \leftarrow (1+\gamma)\boldsymbol{\alpha}_{t+1} - \gamma \boldsymbol{\alpha}_t + \eta_2 \boldsymbol{H}_B^\top \boldsymbol{v}_t - \eta_2 \boldsymbol{H}_J^\top \boldsymbol{w}_t \tag{20d}$$

emulate the updates (15) *with* $\mathcal{P}$ *replaced by* $\mathcal{Q}$ *via the relation* $f_t := S^* \boldsymbol{\alpha}_t$ *and* $g_t := S^* \boldsymbol{\beta}_t$.

Remark 4. In Algorithm 2, lines (12-14) together implement equation (20c), whereas lines (15-17) together implement equation (20d).

Proof. We start by showing that

$$\mathcal{Q} S_B^* = S_B^* - S_J^* \boldsymbol{G}\boldsymbol{G}^\top K(X[J], X[B]). \tag{21}$$

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

$$\mathcal{Q} S_B^* \boldsymbol{u} = S_B^* \boldsymbol{u} - \sum_{i=1}^{q} \left(1 - \frac{\delta_{q+1}}{\delta_i}\right) \phi_i \otimes_{\mathcal{H}} \phi_i S_B^* \boldsymbol{u} \tag{22}$$

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

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

Summing the q terms, and observing that $\boldsymbol{G} = \sum_{i=1}^{q} \frac{1}{\delta_i}(1 - \frac{\delta_{q+1}}{\delta_i}) \boldsymbol{d}_i \boldsymbol{d}_i^\top$ we have $\mathcal{Q} S_B^* \boldsymbol{u} = S_B^* \boldsymbol{u} - S_J^* \boldsymbol{G}\boldsymbol{G}^\top K(X[J], X[B]) \boldsymbol{u}$ for all $\boldsymbol{u} \in \mathbb{R}^m$. Since $\boldsymbol{u}$ is arbitrary, this proves the claim in equation (21). The rest of the proof proceeds similar to the proof of Proposition 3. □

4. Numerical Experiments

Due to space limitations, only a few key experiments are presented in the main paper while other variants are provided in the Appendix. All experiments are solving equation (1) under different choices of dataset, (X, Y), choice of kernel K. Table 1 shows that AxlePro never compromises on the performance. Hence all our experiments show training loss (MSE), and demonstrate that AxlePro is the fastest algorithm to solve equation (1). Note that all plots use a log-scale on the Y-axis.

Hardware. Experiments were run on a machine with 32GB RAM, 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, Stellar Classification, and EMNIST Digits. 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. Experiment details can be found in Appendix C.

Batch size. Only AxlePro and EigenPro are batched algorithms, whereas FALKON, PCG, and GPYTORCH do not require a batch size. 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 & Belkin (2019) for a deeper discussion.

Comparison with other iterative methods. We compare AxlePro with other scalable solvers for training kernel machines.

1. EigenPro. We select batch size and precondition level as in Ma & Belkin (2019). 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.

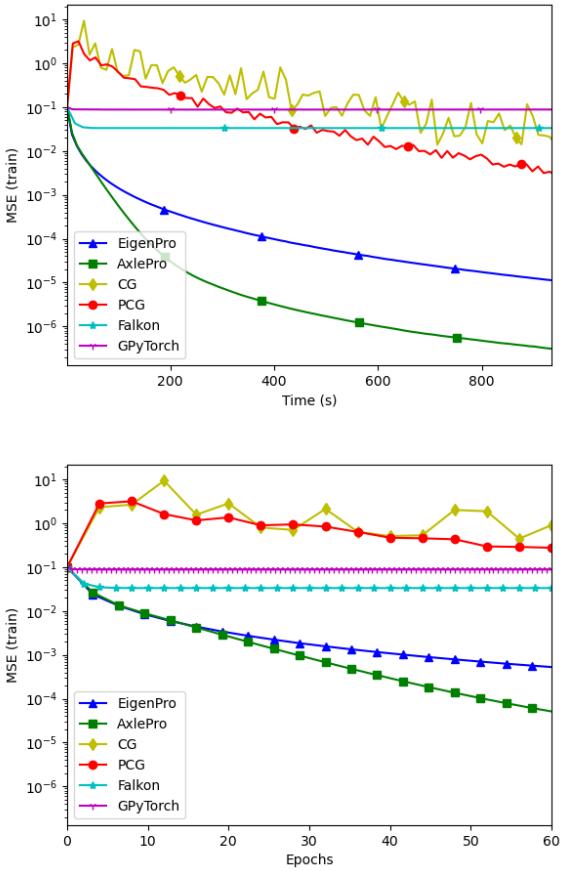

Figure 1. **Matrix free behavior is fast.** Convergence comparison on CIFAR10 dataset with Gaussian kernel. Our AxlePro method achieves the fastest convergence rate compared with other methods in terms of both time and number of epochs.

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 chose this factor from according to the datasets.

3. FALKON. According to the GPU memory, we set the number of inducing points to be $20k$ with 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 `IndependentMultitaskGPModel` in order to deal with multiclass classification problems. The optimizer is set to be 'Adam' with learning rate 0.05. As in the case of

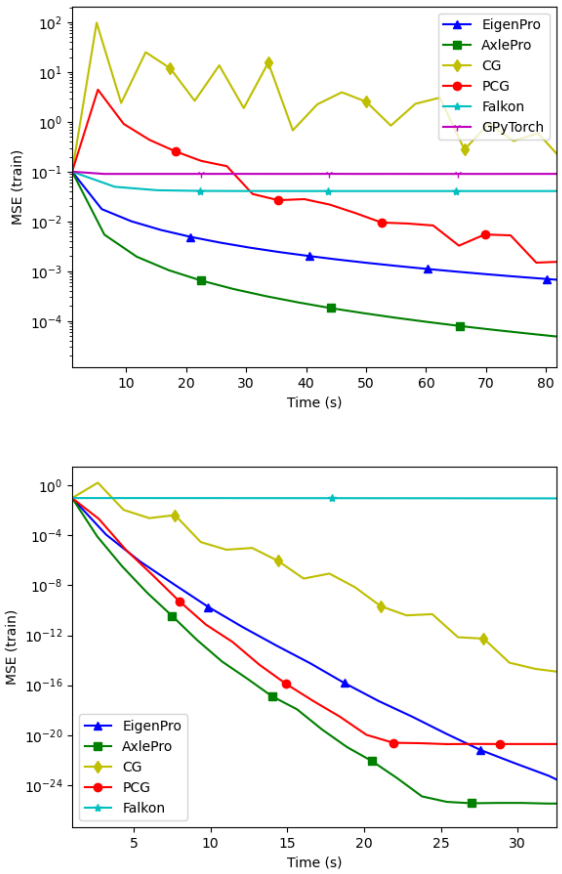

Figure 2. **AxlePro is faster even when kernel matrices are stored.** Convergence comparison on CIFAR10 dataset with saved kernel matrix and Gaussian kernel (top) and Laplacian kernel (bottom)

Falkon, while it is hard to notice significant decrease in the logarithmic scale plot of MSE, the classification accuracy is improved during optimization.

For different datasets, we choose the corresponding bandwidth for a reasonable magnitude for kernel evaluations.

5. Convergence analysis

We need to define a few quantities to describe the convergence properties of algorithm 1. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$ be the eigenvalues of $K(X, X)$.

$$L_1 := \max_{x \in X} K(x, x) \tag{23a}$$

$$L_m := L_1/m + (m-1)\lambda_{q+1}/m \tag{23b}$$

$$\kappa_m := L_m/\lambda_n \tag{23c}$$

$$\widetilde{\kappa}_m := n/m + (m-1)/m \tag{23d}$$

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

Table 2. Performance comparison in terms of test accuracy between different methods for training kernel models with Gaussian and Laplacian kernels

Consider parameters, inherited from Liu & Belkin (2020),

$$\eta_1^* = \frac{1}{L_m} \qquad \gamma^* = \frac{\sqrt{\kappa_m \widetilde{\kappa}_m} - 1}{\sqrt{\kappa_m \widetilde{\kappa}_m} + 1} \tag{24a}$$

$$\eta_2^* = \frac{\eta_1^*(m)\sqrt{\kappa_m \widetilde{\kappa}_m}}{\sqrt{\kappa_m \widetilde{\kappa}_m} + 1} \left(1 - \frac{1}{\widetilde{\kappa}_m}\right) \tag{24b}$$

Proposition 5. *Suppose algorithm 1—AxlePro—is run for t iterations with hyperparameters given by equation* (24)*, then we have,*

$$\|f_t - f^*\|_{\mathcal{H}}^2 \leq \exp\left(-t/\sqrt{\widetilde{\kappa}_m \kappa_m}\right) \|f^*\|_{\mathcal{H}}^2 . \tag{25}$$

Proof. This is an immediate corollary of Liu & 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 via the following reduction commonly applied to analyze spectral preconditioning: (15) is actually (11) with the following modified kernel,

$$\widetilde{K}(x, z) := K(x, z) - K(x, X)\boldsymbol{E}\boldsymbol{Q}\boldsymbol{E}^\top K(X, z), \tag{26}$$

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

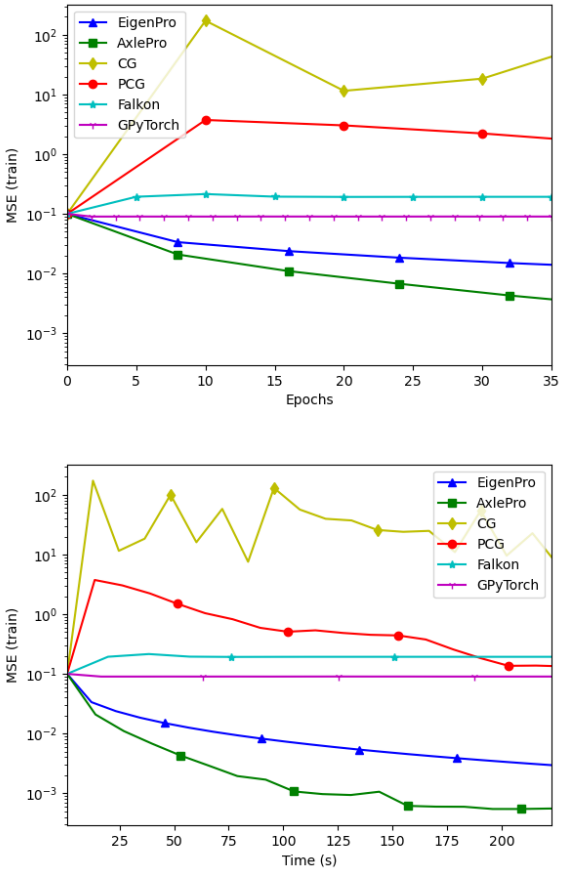

Figure 3. **Numerical stability to low precision.** Convergence comparison on CIFAR10 dataset with Gaussian kernel over single precision arithmetic (32 bit floats).

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

Proposition 6. *Let $\varepsilon > 0$ and $\delta \in (0,1)$ be parameters for controlling the approximation error of Nyström approximate preconditioner defined in equation* (19)*. Suppose X consists of i.i.d. samples from any distribution on $\mathcal{X}$, and J consists of s distinct indices chosen from of $\{1,2,\ldots,n\}$, independent of X such that*

$$s = |J| \gtrsim \frac{\log^4(1+n)}{\varepsilon^4} \log \frac{4}{\delta}.$$

Now, suppose algorithm 2—LM-AxlePro—is run for t iterations with hyperparameters given by equation (24)*. Then, the following inequality holds,*

$$\|f_t - f^*\|_{\mathcal{H}}^2 \leq \exp\left(\frac{-t/(1+\varepsilon)^4}{\sqrt{\widetilde{\kappa}_m \kappa_m}}\right) \|f^*\|_{\mathcal{H}}^2, \qquad (27)$$

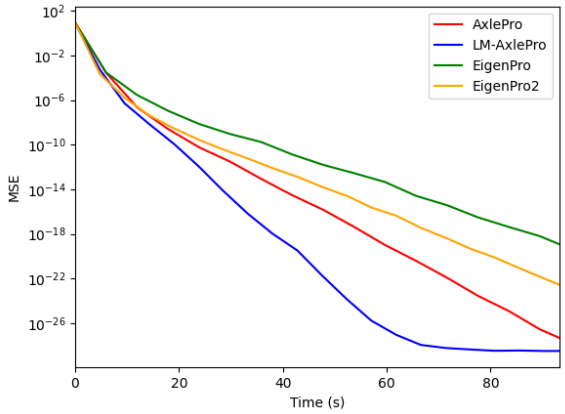

Figure 4. **Low memory footprint can provide speed-ups for large problem sizes.** Comparing algorithm 2 LM-AxlePro , algorithm 1 AxlePro, algorithm 3 EigenPro2, and algorithm 4 EigenPro1. Furthermore, the setup time for LM-AxlePro is much lower than AxlePro.

with probability at least $1-\delta$ (over the randomness in X, J).

Proof. This is a direct corollary of Abedsoltan et al. (2024, Theorem 2), which says the effect of approximating $\mathcal{P}$ with $\mathcal{Q}$ only alters the resulting condition numbers by a factor of $(1+\varepsilon)^4$. Thus, in proposition 5, $\widetilde{\kappa}_m$ gets replaced by $(1+\varepsilon)^4\widetilde{\kappa}_m$ and κ_m gets replaced by $(1+\varepsilon)^4\kappa_m$. □

Analysis of computational complexity Table 3 compares the per iteration complexity of AxlePro and LM-AxlePro with EigenPro. The setup times in the original EigenPro and AxlePro can be n^2q, since it involves obtaining the top-q eigenvectors of $K(X,X)$. In practice, this can be well approximated via (i) a subsampling, then (ii) Nyström extension, followed by (iii) a re-normalization using a thin-QR factorization, which is discussed in detail in Appendix A.

6. Discussion

In this paper we presented an algorithm AxlePro, for training kernel models using a momentum accelerated version of preconditioned SGD in the primal space. We (i) provided convergence guarantees, (ii) analyzed computational costs, (iii) gave an implementation, (iv) that AxlePro is the fastest among competing training algorithms without compromising on performance.

While AxlePro is able to emulate MaSS with approximate preconditioning in the RKHS, other iterative updates such as ADAM and its variants cannot be emulated as elegantly via finite dimensional updates. This is primarily because we

Algorithm	setup	per-iteration time	memory
EigenPro	nsq	$nm + \mathbf{nq} + \mathbf{mq}$	$n + nq$
EigenPro2.0	sq^2	$nm + \mathbf{sm} + \mathbf{2sq}$	$n + sq$
AxlePro	nsq	$nm + \mathbf{nq} + \mathbf{mq}$	$2n + nq$
LM-AxlePro	sq^2	$nm + \mathbf{sm} + \mathbf{2sq}$	$2n + sq$

Table 3. 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 & Belkin (2017), whereas LM-AxlePro is based on applying momentum acceleration to EigenPro2.0 Ma & Belkin (2019). The overhead of storing the previous parameter is n as seen the last column.

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.

References

Abedsoltan, A., Belkin, M., Pandit, P., and Rademacher, L. On the nystrom approximation for preconditioning in kernel machines. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2024.

Adlam, B., Lee, J., Padhy, S., Nado, Z., and Snoek, J. Kernel regression with infinite-width neural networks on millions of examples. *arXiv preprint arXiv:2303.05420*, 2023.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. R., and Wang, R. On exact computation with an infinitely wide neural net. *Advances in neural information processing systems*, 32, 2019.

Beaglehole, D., Radhakrishnan, A., Pandit, P., and Belkin, M. Mechanism of feature learning in convolutional neural networks. *arXiv preprint arXiv:2309.00570*, 2023.

Bietti, A. and Bach, F. Deep equals shallow for relu networks in kernel regimes. *arXiv preprint arXiv:2009.14397*, 2020.

Camoriano, R., Angles, T., Rudi, A., and Rosasco, L. Nytro: When subsampling meets early stopping. In *Artificial Intelligence and Statistics*, pp. 1403–1411. PMLR, 2016.

Carratino, L., Rudi, A., and Rosasco, L. Learning with sgd and random features. *Advances in Neural Information Processing Systems*, 31, 2018.

Chen, Y., Epperly, E. N., Tropp, J. A., and Webber, R. J. Randomly pivoted cholesky: Practical approximation of a kernel matrix with few entry evaluations. *arXiv preprint arXiv:2207.06503*, 2022.

Chowdhury, A., Yang, J., and Drineas, P. An iterative, sketching-based framework for ridge regression. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 989–998. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/chowdhury18a.html.

Cutajar, K., Osborne, M., Cunningham, J., and Filippone, M. Preconditioning kernel matrices. In *International conference on machine learning*, pp. 2529–2538. PMLR, 2016.

Díaz, M., Epperly, E. N., Frangella, Z., Tropp, J. A., and Webber, R. J. Robust, randomized preconditioning for kernel ridge regression. *arXiv preprint arXiv:2304.12465*, 2023.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.

Han, I., Zandieh, A., Lee, J., Novak, R., Xiao, L., and Karbasi, A. Fast neural kernel embeddings for general activations. *Advances in neural information processing systems*, 35:35657–35671, 2022.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

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

Liu, C. and Belkin, M. Accelerating sgd with momentum for over-parameterized learning. In *International Conference on Learning Representations*, 2020.

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

Ma, S. and Belkin, M. Kernel machines that adapt to gpus for effective large batch training. *Proceedings of Machine Learning and Systems*, 1:360–373, 2019.

Ma, S., Bassily, R., and Belkin, M. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International Conference on Machine Learning*, pp. 3325–3334. PMLR, 2018.

Meanti, G., Carratino, L., Rosasco, L., and Rudi, A. Kernel methods through the roof: handling billions of points efficiently. *Advances in Neural Information Processing Systems*, 33:14410–14422, 2020.

Radhakrishnan, A., Beaglehole, D., Pandit, P., and Belkin, M. Feature learning in neural networks and kernel machines that recursively learn features. *arXiv preprint arXiv:2212.13881*, 2022.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

Rudi, A., Carratino, L., and Rosasco, L. Falkon: An optimal large scale kernel method. *Advances in neural information processing systems*, 30, 2017.

Schölkopf, B., Herbrich, R., and Smola, A. J. A generalized representer theorem. In *International conference on computational learning theory*, pp. 416–426. Springer, 2001.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, pp. 807–814, 2007.

Shankar, V., Fang, A., Guo, W., Fridovich-Keil, S., Ragan-Kelley, J., Schmidt, L., and Recht, B. Neural kernels without tangents. In *International conference on machine learning*, pp. 8614–8623. PMLR, 2020.

Yin, R., Wang, W., and Meng, D. Distributed nyström kernel learning with communications. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12019–12028. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/yin21a.html.

Algorithm 4 EigenPro1.0 (SGD + preconditioning)

Input: kernel K, batch size m, learning rate $\eta_0 > 0$.
Output: $f : \mathcal{X} \to \mathbb{R}$ that solves (5) approximately.
setup: $\boldsymbol{\alpha} \leftarrow \mathbf{0}_n$.
$(\boldsymbol{E}_q, \Lambda, \lambda_{q+1}) \leftarrow$ top-q eigensystem of $K(X, X)$
$\boldsymbol{F}_q \leftarrow \boldsymbol{E}_q \sqrt{I_q - \lambda_{q+1}\Lambda^{-1}}$
repeat
  Fetch a batch of indices $B \subset \{1, 2, \ldots, n\}$
  $\boldsymbol{v} \leftarrow K(X[B], X)\boldsymbol{\alpha} - Y[B] \in \mathbb{R}^m$
  $\boldsymbol{w} \leftarrow \boldsymbol{F}_q \boldsymbol{F}_q[B]^\top \boldsymbol{v} \in \mathbb{R}^n$
  $\boldsymbol{\alpha}[B] - \leftarrow \eta_0 \boldsymbol{v}$ {SGD step}
  $\boldsymbol{\alpha} + \leftarrow \eta_0 \boldsymbol{w}$ {correction}
until Stopping criterion is reached
return $f_t(x) = \sum_{i=1}^n \alpha_i K(x, x_i)$

Appendices

A. Exact preconditioner

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

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

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

$$\boldsymbol{E}, \_ \leftarrow \texttt{thinQR}(\boldsymbol{E}).$$

This only has a total complexity $s^2 q + qsn + nq^2$, where $s^2 q$ is the cost of getting eigenvectors $\widetilde{\boldsymbol{E}}$, 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.

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

B. Details on convergence analysis

B.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 $\{(\widetilde{x}_i, \widetilde{y}_i)\}_{i=1}^m$. Defining mini-batch covariance operator as follows

$$\widetilde{\mathcal{K}}^{(m)} := \frac{1}{m}\sum_{i=1}^m K(\widetilde{x}_i, \cdot) \otimes K(\widetilde{x}_i, \cdot) \tag{28}$$

$$\widetilde{\mathcal{K}}_{\mathcal{P}}^{(m)} := \frac{1}{m}\sum_{i=1}^m k_{\mathcal{P}}(\widetilde{x}_i, \cdot) \otimes k_{\mathcal{P}}(\widetilde{x}_i, \cdot) \tag{29}$$

where $\widetilde{\mathcal{K}}_{\mathcal{P}}$ is the covariance operator after preconditioning.

B.2. Identifying quantities defined by Liu & Belkin (2020)

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

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

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

$$\|K(x,\cdot)\|_{\mathcal{H}}^2 = \langle K(x,\cdot), K(x,\cdot)\rangle_{\mathcal{H}} = K(x,x) \leq \beta := \max_i K(x_i,x_i) \tag{30}$$

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

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

Thus

$$L_m := \frac{L_1}{m} + \frac{(m-1)L}{m} \leq \frac{\beta + (m-1)\frac{\lambda_1}{n}}{m} \tag{31}$$

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

$$\mathbb{E}\left[\|K(\widetilde{x},\cdot)\|_{\mathcal{K}^{-1}}^2 K(\widetilde{x},\cdot) \otimes K(\widetilde{x},\cdot)\right] \preceq \widetilde{\kappa}\, \mathcal{K} \tag{32}$$

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

$$\begin{aligned}
\|K(x_i,\cdot)\|_{\mathcal{K}^{-1}}^2 &= \left\langle K(x_i,\cdot),\, \mathcal{K}^{-1}K(x_i,\cdot)\right\rangle_{\mathcal{H}} \\
&= \left\langle K(x_i,\cdot), \left(\sum_{j=1}^n \frac{n}{\lambda_j}\psi_j \otimes \psi_j\right) K(x_i,\cdot)\right\rangle_{\mathcal{H}} \\
&= \sum_{j=1}^n \frac{n}{\lambda_j}\left\langle \psi_j, K(x_i,\cdot)\right\rangle_{\mathcal{H}}^2 \;=\; \sum_{j=1}^n \frac{n}{\lambda_j}\psi_j^2(x_i) \overset{(a)}{=} \sum_{j=1}^n \frac{1}{\lambda_j} n\lambda_j e_{ji}^2 \qquad (33a) \\
&= n\sum_{j=1}^n e_{ji}^2 = n\,\|\boldsymbol{e}_j\|^2 = n \qquad (33b)
\end{aligned}$$

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

$$\begin{aligned}
\mathbb{E}\left[\|K(\widetilde{x},\cdot)\|_{\mathcal{K}^{-1}}^2 K(\widetilde{x},\cdot) \otimes K(\widetilde{x},\cdot)\right] &\overset{(a)}{=} n\,\mathbb{E}\left[K(\widetilde{x},\cdot) \otimes K(\widetilde{x},\cdot)\right] \\
&= n\,\mathbb{E}\left[\widetilde{\mathcal{K}}^{(1)}\right] \\
&= n\,\mathcal{K} \\
\overset{(b)}{\Longrightarrow}\; \widetilde{\kappa} &= n
\end{aligned}$$

where (a) follows from equation (33b) and (b) from definition of $\widetilde{\kappa}$ in equation (32)

B.3. Formulae for hyperparameters

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

$$L_m = \frac{L_1}{m} + \frac{(m-1)L}{m} = \frac{\beta + (m-1)\frac{\lambda_1}{n}}{m} \tag{34}$$

$$\kappa_m = \frac{nL_m}{\lambda_n} \tag{35}$$

$$\widetilde{\kappa}_m = \frac{\widetilde{\kappa}}{m} + \frac{m-1}{m} = 1 + \frac{n-1}{m} \tag{36}$$

$$\eta_1(m) = \frac{1}{L_m} = \frac{m}{\beta + (m-1)\frac{\lambda_1}{n}} \tag{37}$$

$$\eta_2(m) = \eta_1 \frac{\sqrt{\kappa_m \widetilde{\kappa}_m}}{1 + \sqrt{\kappa_m \widetilde{\kappa}_m}} \left(1 - \frac{1}{\widetilde{\kappa}_m}\right) \tag{38}$$

$$\gamma(m) = \frac{\sqrt{\kappa_m \widetilde{\kappa}_m} - 1}{\sqrt{\kappa_m \widetilde{\kappa}_m} + 1} \tag{39}$$

B.4. Condition numbers after preconditioning

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

Similar to **??**, defining L_1 as the smallest positive number such that

$$\mathbb{E}\left[\|k_{\mathcal{P}}(\widetilde{x}, \cdot)\|^2 k_{\mathcal{P}}(\widetilde{x}, \cdot) \otimes k_{\mathcal{P}}(\widetilde{x}, \cdot)\right] \preceq L_1 \mathcal{K}_{\mathcal{P}} \tag{40}$$

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

$$\begin{aligned}
\|k_{\mathcal{P}}(\boldsymbol{x}_i, \cdot)\|^2 &= \langle k_{\mathcal{P}}(\boldsymbol{x}_i, \cdot), k_{\mathcal{P}}(\boldsymbol{x}_i, \cdot)\rangle_{\mathcal{H}_{\mathcal{P}}} \\
&= k_{\mathcal{P}}(\boldsymbol{x}_i, \boldsymbol{x}_i) \\
&\overset{(a)}{=} K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{j=1}^{q} \left(1 - \frac{\lambda_{q+1}}{\lambda_j}\right) \lambda_j e_{ji}^2 \\
&= K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{j=1}^{q} (\lambda_j - \lambda_{q+1}) e_{ji}^2
\end{aligned} \tag{41}$$

where (a) is from (**??**) and (**??**)
Define $\beta_{\mathcal{P}}$ as the maximum norm of $k_{\mathcal{P}}(\boldsymbol{x}_i, \cdot)$ among the samples

$$\beta_{\mathcal{P}} := \max_i ||k_{\mathcal{P}}(\boldsymbol{x}_i, \cdot)||^2 \tag{42}$$

$$= \max_i \left\{ K(\boldsymbol{x}_i, \boldsymbol{x}_i) - n \sum_{j=1}^{q} (\lambda_j - \lambda_{q+1}) e_{ji}^2 \right\} \tag{43}$$

Deriving L_1

$$\begin{aligned}
&\mathbb{E}\left[\|k_{\mathcal{P}}(\widetilde{x}, \cdot)\|^2 k_{\mathcal{P}}(\widetilde{x}, \cdot) \otimes k_{\mathcal{P}}(\widetilde{x}, \cdot)\right] \\
&\preceq \beta_{\mathcal{P}}\, \mathbb{E}\left[k_{\mathcal{P}}(\widetilde{x}, \cdot) \otimes k_{\mathcal{P}}(\widetilde{x}, \cdot)\right] \\
&= \beta_{\mathcal{P}}\, \mathbb{E}\left[\widetilde{\mathcal{K}}_{\mathcal{P}}^{(1)}\right] \\
&= \beta_{\mathcal{P}}\, \mathcal{K}_{\mathcal{P}} \\
\overset{(a)}{\implies} L_1 &\leq \beta_{\mathcal{P}}
\end{aligned}$$

where (a) follows from the definition of L_1 in equation (40)
Deriving L_m

$$L_m := \frac{L_1}{m} + \frac{(m-1)L}{m} \leq \frac{\beta_{\mathcal{P}} + (m-1)\lambda_{q+1}}{m} \tag{44}$$

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

$$\mathbb{E}\left[\|k_{\mathcal{P}}(\widetilde{x},\cdot)\|^2_{\mathcal{K}_{\mathcal{P}}^{-1}}\, k_{\mathcal{P}}(\widetilde{x},\cdot)\otimes k_{\mathcal{P}}(\widetilde{x},\cdot)\right] \preceq \widetilde{\kappa}\,\mathcal{K}_{\mathcal{P}} \tag{45}$$

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

$$\begin{aligned}
&\|k_{\mathcal{P}}(\boldsymbol{x}_i,\cdot)\|^2_{\mathcal{K}_{\mathcal{P}}^{-1}} = \left\langle k_{\mathcal{P}}(\boldsymbol{x}_i,\cdot),\, \mathcal{K}_{\mathcal{P}}^{-1}k_{\mathcal{P}}(\boldsymbol{x}_i,\cdot)\right\rangle_{\mathcal{H}_{\mathcal{P}}}\\
&\overset{(a)}{=} \left\langle k_{\mathcal{P}}(\boldsymbol{x}_i,\cdot), \left(\sum_{j=1}^{q}\frac{1}{\lambda_{q+1}}\psi'_j\otimes\psi'_j + \sum_{j=q+1}^{n}\frac{1}{\lambda_j}\psi'_j\otimes\psi'_j\right)k_{\mathcal{P}}(\boldsymbol{x}_i,\cdot)\right\rangle_{\mathcal{H}_{\mathcal{P}}}\\
&= \sum_{j=1}^{q}\frac{1}{\lambda_{q+1}}\left\langle\psi'_j, k_{\mathcal{P}}(\boldsymbol{x}_i,\cdot)\right\rangle^2_{\mathcal{H}_{\mathcal{P}}} + \sum_{j=q+1}^{n}\frac{1}{\lambda_j}\left\langle\psi'_j, k_{\mathcal{P}}(\boldsymbol{x}_i,\cdot)\right\rangle^2_{\mathcal{H}_{\mathcal{P}}}\\
&\overset{(b)}{=} n\left\{\sum_{j=1}^{q}e_{ji}^2 + \sum_{j=q+1}^{n}e_{ji}^2\right\}\\
&\overset{(c)}{=} n
\end{aligned} \tag{46}$$

where (a) is from (**??**), (b) follows from using **??** with ψ' in $\mathcal{H}_{\mathcal{P}}$ and (c) follows from the fact that $\boldsymbol{E} = [\boldsymbol{e}_1\ \boldsymbol{e}_2\ \ldots\ \boldsymbol{e}_n]$ is an orthonormal matrix which implies $\boldsymbol{E}^\top\boldsymbol{E} = \boldsymbol{I} = \boldsymbol{E}\boldsymbol{E}^\top$
Deriving $\widetilde{\kappa}$

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

$$\overset{(a)}{=} n\,\mathbb{E}\left[k_{\mathcal{P}}(\widetilde{x},\cdot)\otimes k_{\mathcal{P}}(\widetilde{x},\cdot)\right] \tag{48}$$

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

$$= n\,\mathcal{K}_{\mathcal{P}} \tag{49}$$

$$\overset{(b)}{\implies} \widetilde{\kappa} = n \tag{50}$$

where (a) follows from equation (46) and (b) from definition of $\widetilde{\kappa}$ in equation (45)

B.5. MaSS parameters after preconditioning

Using (44), η_1 defined in (24a) can be written as

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

Using (50), (**??**), (**??**) we can write η_2, γ defined in (24b), (**??**) as

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

$$\gamma^*(m) = \frac{\sqrt{(\beta_{\mathcal{P}} + (m-1)\lambda_{q+1})\,(n+m-1)} - m\sqrt{\lambda_n}}{\sqrt{(\beta_{\mathcal{P}} + (m-1)\lambda_{q+1})\,(n+m-1)} + m\sqrt{\lambda_n}}$$

Since the kernel matrix is positive definite we see that $\lambda_{q+1} \geq \beta/n$ and using (50) we get regime critical points m_1^*, m_2^* defined in (**??**), (**??**) as

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

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

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

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

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

Proposition 7. *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

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

□

C. 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.

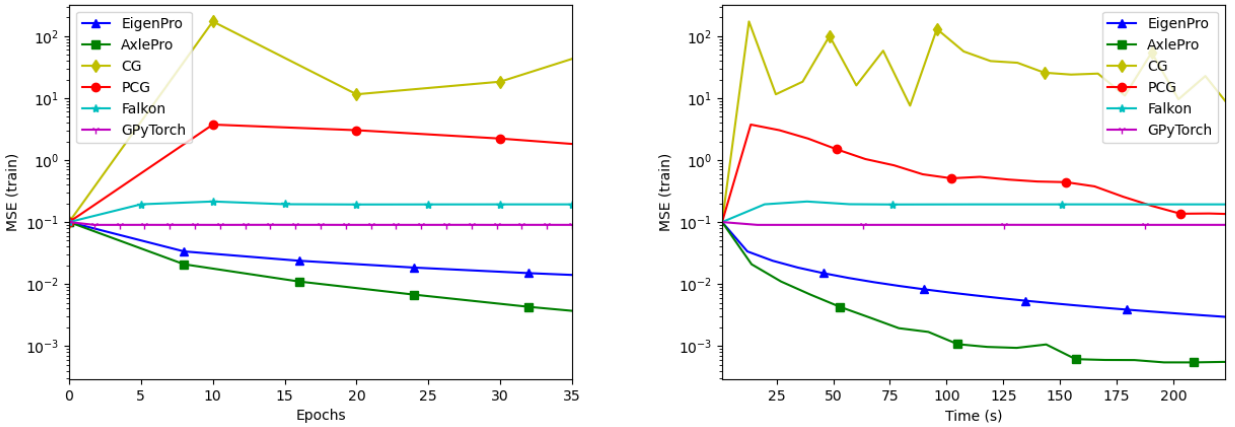

Figure 5. **Numerical stability to low precision.** Convergence comparison on CIFAR10 dataset with Gaussian kernel over single precision arithmetic (32 bit floats).

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.

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 `IndependentMultitaskGPModel` 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 tune λ_n by a factor of 0.5 for (LM-)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 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).

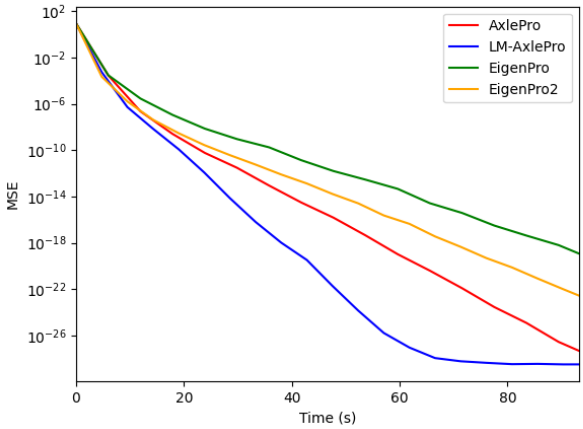

Figure 6. **Low memory footprint can provide speed-ups for large problem sizes.** Comparing algorithm 2 LM-AxlePro , algorithm 1 AxlePro, algorithm 3 EigenPro2, and algorithm 4 EigenPro1 (n=20,000, m=100, s=600, q=200 for all algorithms). Furthermore, the setup time for LM-AxlePro is much lower than AxlePro.

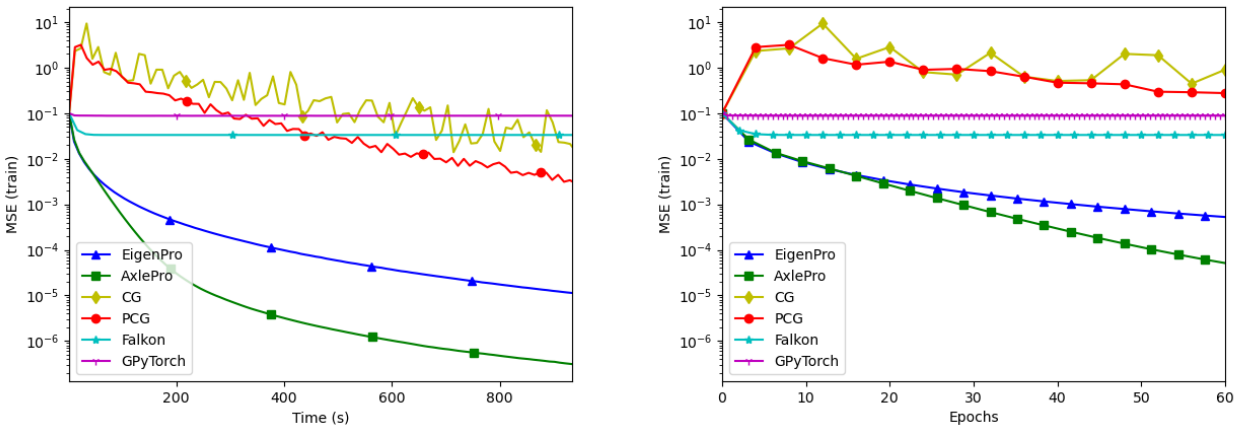

Figure 7. Convergence Comparison on CIFAR10 dataset with Gaussian kernel.

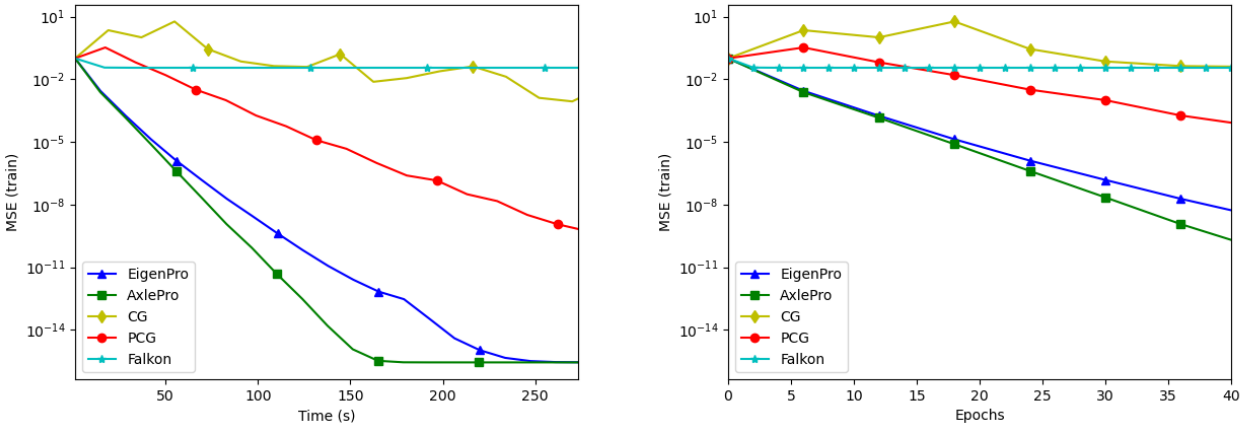

Figure 8. Convergence Comparison on CIFAR10 dataset with Laplacian kernel.

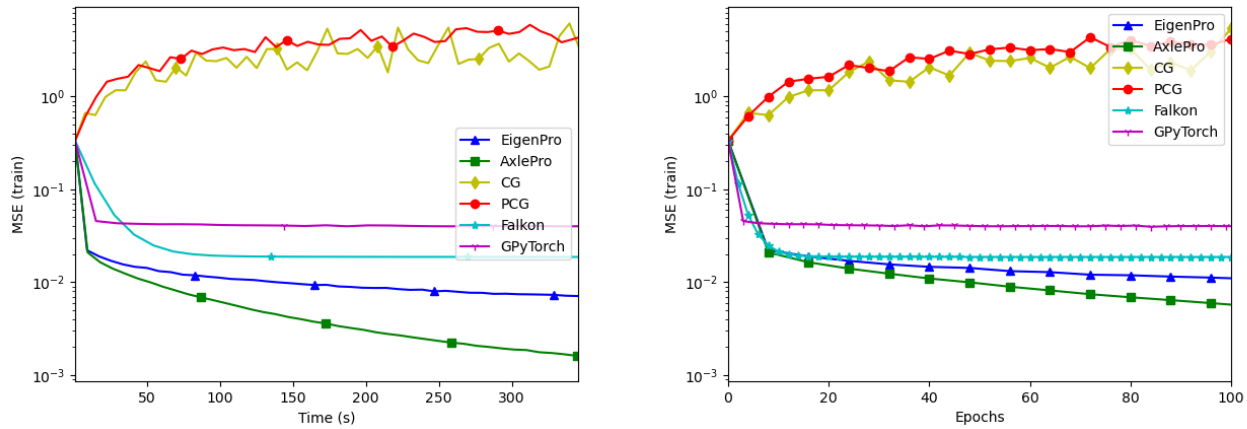

Figure 9. Convergence Comparison on Stellar Classification dataset with Gaussian kernel.

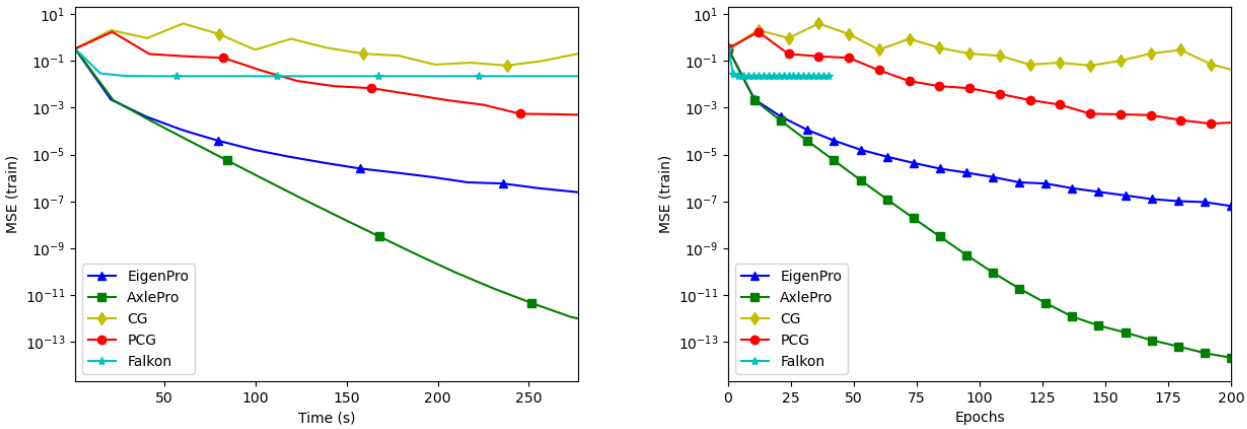

Figure 10. Convergence Comparison on Stellar Classification dataset with Laplacian kernel.

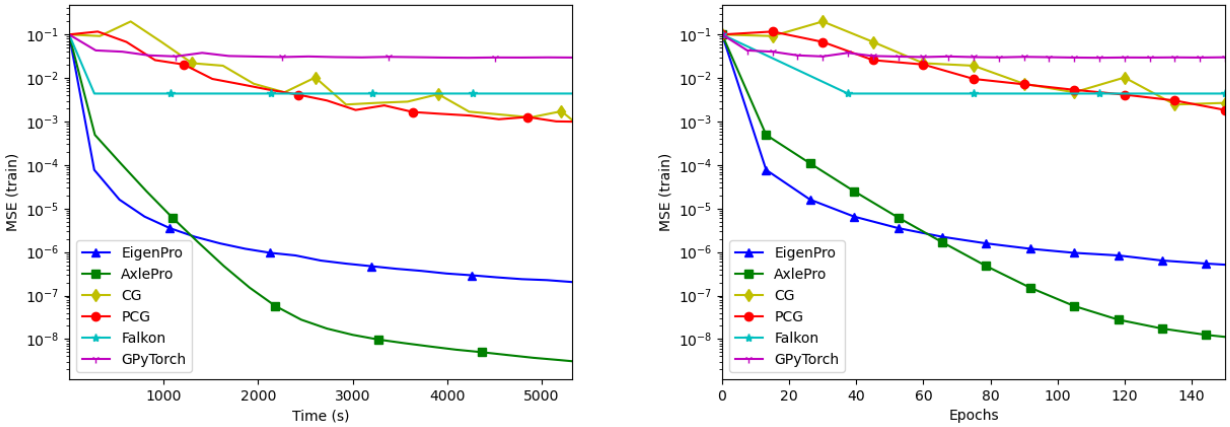

Figure 11. Convergence Comparison on EMNIST Digits dataset with Gaussian kernel.

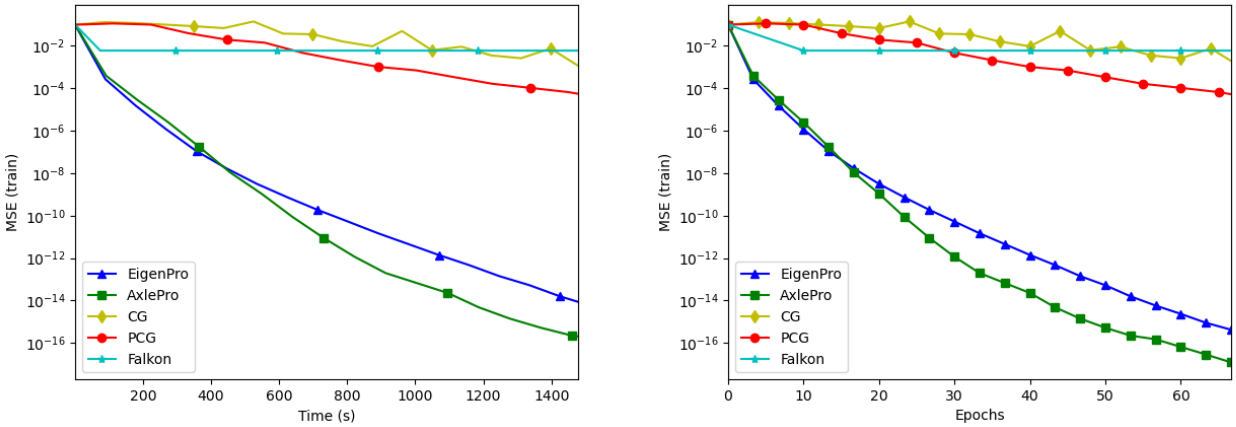

Figure 12. Convergence Comparison on EMNIST Digits dataset with Laplacian kernel.

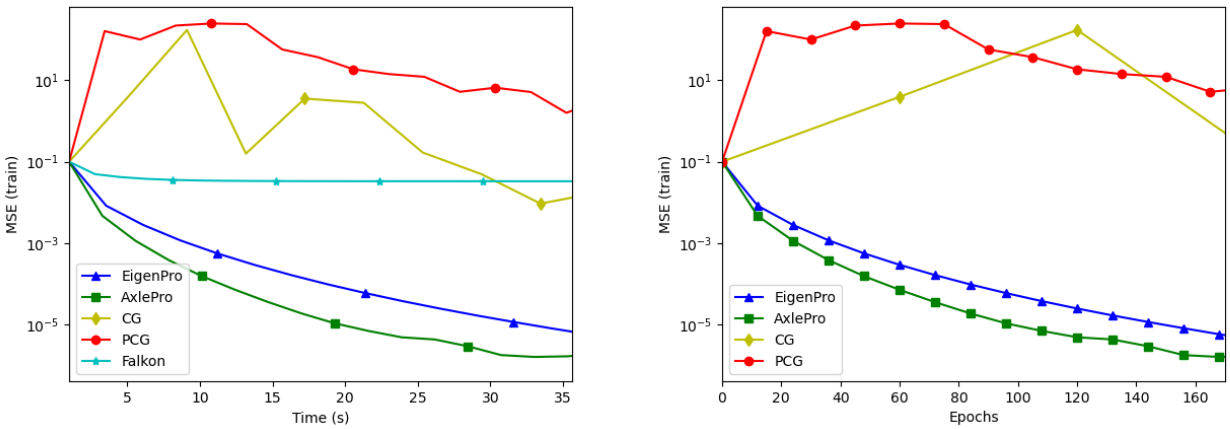

Figure 13. Convergence Comparison on CIFAR10 dataset with Myrtle5 kernel. We set the scaling factor of λ_n to be 3.

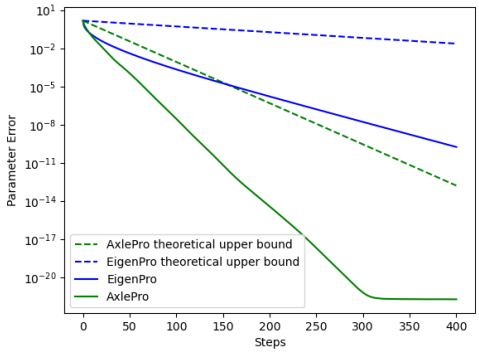

Figure 14. Comparison between experimental convergence rates and theoretical convergence rates. For both algorithms, we have convergence rates that are faster than theoretical upper bounds. Moreover, AxlePro is faster than EigenPro both theoretically and empirically.

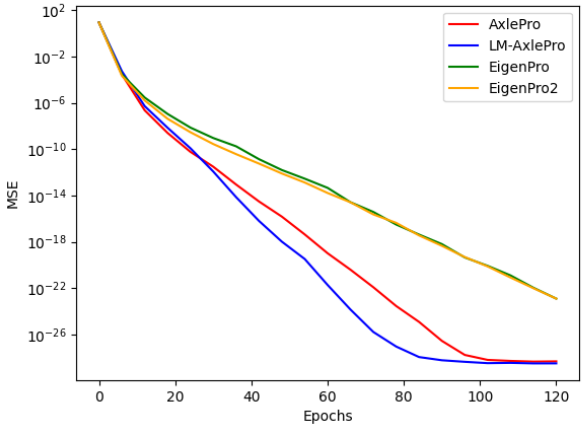

Figure 15. Comparing progress of algorithm 2 LM-AxlePro , algorithm 1 AxlePro, algorithm 3 EigenPro2, and algorithm 4 EigenPro1 with respect to number of steps on randomly generated dataset (n=20,000, m=100, s=600, q=200).