Toward Large Kernel Models

Amirhesam Abedsoltan [1] **Mikhail Belkin** [2 1] **Parthe Pandit** [2]

Abstract

Recent studies indicate that kernel machines can often perform similarly or better than deep neural networks (DNNs) on small datasets. The interest in kernel machines has been additionally bolstered by the discovery of their equivalence to wide neural networks in certain regimes. However, a key feature of DNNs is their ability to scale the model size and training data size independently, whereas in traditional kernel machines model size is tied to data size. Because of this coupling, scaling kernel machines to large data has been computationally challenging. In this paper, we provide a way forward for constructing large-scale *general kernel models*, which are a generalization of kernel machines that decouples the model and data, allowing training on large datasets. Specifically, we introduce EigenPro 3.0, an algorithm based on projected dual preconditioned SGD and show scaling to model and data sizes which have not been possible with existing kernel methods. We provide a PyTorch based implementation which can take advantage of multiple GPUs.

1. Introduction

Deep neural networks (DNNs) have become the gold standard for many large-scale machine learning tasks. Two key factors that contribute to the success of DNNs are the large model sizes and the large number of training samples. Quoting from (Kaplan et al., 2020) "*performance depends most strongly on scale, which consists of three factors: the number of model parameters N (excluding embeddings), the size of the dataset D, and the amount of compute C used for training. Within reasonable limits, performance depends very weakly on other architectural hyperparameters such as depth vs. width*". Major community effort and great amount of resources have been invested in scaling models and data size, as well as in understanding the relationship between the number of model parameters, compute, data size, and performance. Many current architectures have hundreds of billions of parameters and are trained on large datasets with nearly a trillion data points (e.g., Table 1 in (Hoffmann et al., 2022)). Scaling both model size and the number of training samples are seen as crucial for optimal performance.

Recently, there has been a surge in research on the equivalence of special cases of DNNs and kernel machines. For instance, the Neural Tangent Kernel (NTK) has been used to understand the behavior of fully-connected DNNs in the infinite width limit by using a fixed kernel (Jacot et al., 2018). A rather general version of that phenomenon was shown in (Zhu et al., 2022). Similarly, the Convolutional Neural Tangent Kernel (CNTK) (Li et al., 2019) is the NTK for convolutional neural networks, and has been shown to achieve accuracy comparable to AlexNet (Krizhevsky et al., 2012) on the CIFAR10 dataset.

These developments have sparked interest in the potential of kernel machines as an alternative for DNNs. Kernel machines are relatively well-understood theoretically, are stable, somewhat interpretable, and have been shown to perform similarly to DNNs on small datasets (Arora et al., 2020; Lee et al., 2020; Radhakrishnan et al., 2022b), particularly on tabular data (Geifman et al., 2020; Radhakrishnan et al., 2022a). However, in order for kernels to be a viable alternative to DNNs, it is necessary to develop methods to scale kernel machines to large datasets.

The problem of scaling. Similarly to DNN, to achieve optimal performance of kernel models, it is not sufficient to just increase the size of the training set for a fixed model size, but the model size must scale as well. Fig. 1 illustrates this property on a small-scale example (see Appendix D.2 for the details). The figure demonstrates that the best performance cannot be achieved solely by increasing the dataset size. Once the model reaches its capacity, adding more data leads to marginal, if any, performance improvements. On the other hand, we see that the saturation point for each model is not achieved until the number of samples significantly exceeds the model size. This illustration highlights the need for algorithms that can independently scale dataset size and model size for optimal performance.

[1]Department of Computer Science and Engineering, and [2]Halicioglu Data Science Institute, UC San Diego, USA. Correspondence to: <aabedsoltan, parthepandit@ucsd.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

A Python package is available at github.com/EigenPro3

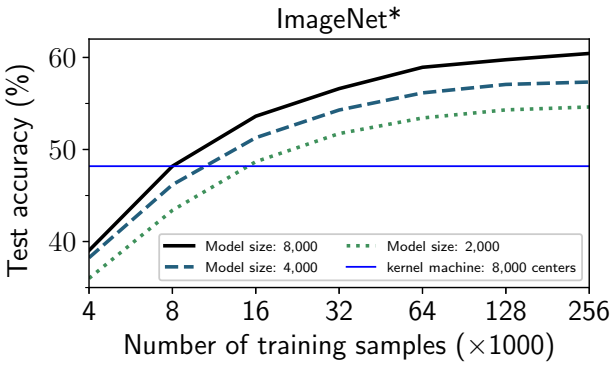

Figure 1: Increasing number of training samples for a fixed model size is helpful but insufficient for optimal performance. The model size must scale as well. See Appendix D.2 for details.

1.1. Main contributions

We introduce EigenPro 3.0 for learning general kernel models that can handle **large model sizes**. In our numerical experiments, we train models with up to 1 million centers on 5 million samples. To the best of our knowledge, this was not achievable with any other existing method. Our work provides a path forward to scaling both the model size and size of the training dataset, independently.

1.2. Prior work

A naive approach for training kernel machines is to directly solve the equivalent kernel matrix inversion problem. In general, the computational complexity of solving the kernel matrix inversion problem is $O(n^3)$, where n is the number of training samples. Thus, computational cost grows rapidly with the size of the dataset, making it computationally intractable for datasets with more than $\sim 10^5$ data points.

A number of methods have been proposed to tackle this issue by utilizing various iterative algorithms and approximations. We provide a brief overview of the key ideas below.

Gradient Descent(GD) based algorithms: Gradient descent (GD) based methods, such as Pegasos (Shalev-Shwartz et al., 2007), have a more manageable computational complexity of $O(n^2)$ and can be used in a stochastic setting, allowing for more efficient implementation. Preconditioned stochastic gradient descent based method, EigenPro (Ma & Belkin, 2017) was introduced to accelerate convergence of Pegasos. EigenPro is an iterative algorithm for kernel machines that uses a preconditioned Richardson iteration (Richardson, 1911). Its performance was further improved in EigenPro 2.0 (Ma & Belkin, 2019) by reducing the computational and memory costs for the preconditioner through the use of a Nyström extension (Williams & Seeger, 2000) and introducing full hardware (GPU) utilization by adaptively auto-tuning the learning rate.

However, Kernel machines are limited in scalability due to the coupling between the model and the training set. With modern hardware, these methods can handle just above one million training data points. A more in-depth discussion about EigenPro can be found in Section 2.1.

Large scale Nyström-approximate models: Nyström methods have been a popular strategy for applying kernel machines at scale starting with (Williams & Seeger, 2000). We refer the reader to Section 2 for a detailed description of the Nyström approximation. Methods such as NYTRO (Camoriano et al., 2016) and FALKON (Rudi et al., 2017) use Nyström approximation (NA) in combination with other techniques to improve performance. Specifically, NYTRO combines NA with gradient descent to improve the condition number, while FALKON uses NA in combination with the Conjugate Gradient method. While these methods allow for very large training sets, they are limited in terms of the model size due to memory limitations. For example, the largest models considered (Meanti et al., 2020) have about 100,000 centers. With 340GB RAM available to us on the Expanse cluster (Towns et al., 2014), we were able to run FALKON with at 256,000 model size (Figure 5). However, scaling to 512,000 model size, already requires over 1TB RAM which goes beyond specifications of most current high end servers.

Sparse Gaussian Process: Methods from the literature on Gaussian Processes, e.g. (Titsias, 2009), use so-called *inducing points* to control the model complexity. While several follow-ups such as (Wilson & Nickisch, 2015) and GPYTORCH (Gardner et al., 2018), and GPFlow (Matthews et al., 2017) have been applied in practice, they require quadratic memory in terms of the number of the inducing points, thus preventing scaling to large models. A closely related concept is that of *kernel herding* (Chen et al., 2010). The focus of these methods is largely on selecting "good" model centers, rather than scaling-up the training procedure. Our work is complementary to that line of research.

Random Fourier Features (RFFs): RFF is a popular method first introduced in (Rahimi & Recht, 2007) to approximate kernel machines using so-called "Random Fourier Features". However, it is generally believed that Nyström methods outperform RFF (Yang et al., 2012).

2. Preliminaries and Background

Kernel Machines: Kernel machines, (Schölkopf et al., 2002) are non-parametric predictive models. Given training data $(X, \boldsymbol{y}) = \left\{\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}\right\}_{i=1}^n$ a kernel machine is a model of the form

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

Here, $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a positive semi-definite symmetric kernel function (Aronszajn, 1950). According to the representer theorem (Kimeldorf & Wahba, 1970), the unique solution to the infinite-dimensional optimization problem

$$\arg\min_{f \in \mathcal{H}} \sum_{i=1}^{n} (f(\boldsymbol{x}_i) - y_i)^2 + \lambda \|f\|_{\mathcal{H}}^2 \tag{2}$$

has the form given in Eq. 1.

Here $\mathcal{H}$ is the (unique) reproducing kernel Hilbert space (RKHS) corresponding to K.

It can be seen that $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ in equation (1) is the unique solution to the linear system,

$$(K(X, X) + \lambda I_n)\boldsymbol{\alpha} = \boldsymbol{y}. \tag{3}$$

General kernel models are models of the form,

$$f(\boldsymbol{x}) = \sum_{i=1}^{p} \alpha_i K(\boldsymbol{x}, \mathbf{z}_i),$$

where $Z = \{z_i \in \mathbb{R}^d\}_{i=1}^p$ is the set of *centers*, which is not necessary the same as the training set. We will refer to p as the *model size*. Note that kernel machines are a special type of general kernel models with $Z = X$. Our goal will be to solve equation (2) with this new constraint on f.

Note that when $Z \subset X$, is a random subset of the training data, the resulting model is called a Nyström approximate (NA) model (Williams & Seeger, 2000).

In contrast to kernel machines, general kernel models, such as classical RBF networks (Poggio & Girosi, 1990), allow for the separation of the model and training set. General kernel models also provide explicit control over model capacity by allowing the user to choose p separately from the training data size. This makes them a valuable tool for many applications, particularly when dealing with large datasets.

Notation: In what follows, functions are lowercase letters a, sets are uppercase letters A, vectors are lowercase bold letters $\boldsymbol{a}$, matrices are uppercase bold letters $\boldsymbol{A}$, operators are calligraphic letters $\mathcal{A}$, spaces and sub-spaces are boldface calligraphic letters $\boldsymbol{\mathcal{A}}$.

Evaluations and kernel matrices: The vector of evaluations of a function f over a set $X = \{\boldsymbol{x}_i\}_{i=1}^n$ is denoted $f(X) := (f(\boldsymbol{x}_i)) \in \mathbb{R}^n$. For sets X and Z, with $|X| = n$ and $|Z| = p$, we denote the kernel matrix $K(X, Z) \in \mathbb{R}^{n \times p}$, while $K(Z, X) = K(X, Z)^\top$. Similarly, $K(\cdot, X) \in \mathcal{H}^n$ is a vector of functions, and we use $K(\cdot, X)\boldsymbol{\alpha} := \sum_{i=1}^n K(\cdot, \boldsymbol{x}_i)\alpha_i \in \mathcal{H}$, to denote their linear combination. Finally, for an operator $\mathcal{A}$, a function a, and a set $A = \{\boldsymbol{a}_i\}_{i=1}^k$, we denote the vector of evaluations of the output,

$$\mathcal{A}\{a\}(A) := (b(\boldsymbol{a}_i)) \in \mathbb{R}^k \qquad \text{where} \quad b = \mathcal{A}(a). \tag{4}$$

Definition 1 (Top-q eigensystem)**.** Let $\lambda_1 > \lambda_2 > \ldots > \lambda_n$, be the eigenvalues of a hermitian matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, i.e., for unit-norm $\boldsymbol{e}_i$, we have $\boldsymbol{A}\boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$. Then we call the tuple $(\Lambda_q, \boldsymbol{E}_q, \lambda_{q+1})$ the top-q eigensystem, where

$$\Lambda_q := \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_q) \in \mathbb{R}^{q \times q}, \text{ and} \tag{5}$$

$$\boldsymbol{E}_q := [\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_q] \in \mathbb{R}^{n \times q}. \tag{6}$$

Definition 2 (Fréchet derivative)**.** Given a function $J : \mathcal{H} \to \mathbb{R}$, the Fréchet derivative of J with respect to f is a linear functional, denoted $\nabla_f J$, such that for $h \in \mathcal{H}$

$$\lim_{\|h\|_{\mathcal{H}} \to 0} \frac{|J(f + h) - J(f) - \nabla_f J(h)|}{\|h\|_{\mathcal{H}}} = 0. \tag{7}$$

Since $\nabla_f J$ is a linear functional, it lies in the dual space $\mathcal{H}^*$. Since $\mathcal{H}$ is a Hilbert space, it is self-dual, whereby $\mathcal{H}^* = \mathcal{H}$. If f is a general kernel model, and L is the square loss for a given dataset $(X, \boldsymbol{y})$, i.e., $L(f) := \frac{1}{2}\sum_{i=1}^n (f(\boldsymbol{x}_i) - y_i)^2$ we can apply the chain rule, and using reproducing property of $\mathcal{H}$, and the fact that $\nabla_f \langle f, g \rangle_{\mathcal{H}} = g$, we get, that the Fréchet derivative of L, at $f = f_0$ is,

$$\nabla_f L(f_0) = \sum_{i=1}^{n} (f_0(\boldsymbol{x}_i) - y_i)\nabla_f f(\boldsymbol{x}_i) \tag{8}$$

$$= K(\cdot, X)(f_0(X) - \boldsymbol{y}). \tag{9}$$

Hessian operator: The Hessian operator $\nabla_f^2 L : \mathcal{H} \to \mathcal{H}$ for the square loss is given by,

$$\mathcal{K} := \sum_{i=1}^{n} K(\cdot, \boldsymbol{x}_i) \otimes K(\cdot, \boldsymbol{x}_i), \tag{10a}$$

$$\mathcal{K}\{f\}(\mathbf{z}) = \sum_{i=1}^{n} K(\mathbf{z}, \boldsymbol{x}_i) f(\boldsymbol{x}_i) = K(\mathbf{z}, X) f(X). \tag{10b}$$

Note that $\mathcal{K}$ is surjective on $\boldsymbol{\mathcal{X}}$, and hence invertible when restricted to $\boldsymbol{\mathcal{X}}$. Note that when $\boldsymbol{x}_i \overset{\text{i.i.d.}}{\sim} \mathbb{P}$, for some measure $\mathbb{P}$, the above summation, on rescaling by $\frac{1}{n}$, converges due to the strong law of large numbers as,

$$\lim_{n \to \infty} \frac{\mathcal{K}\{f\}}{n} = \mathcal{T}_K\{f\} := \int K(\cdot, \boldsymbol{x}) f(\boldsymbol{x}) \, \mathrm{d}\mathbb{P}(\boldsymbol{x}), \tag{11}$$

which is the integral operator associated with a kernel K. The following lemma relates the spectra of $\mathcal{K}$ and $K(X, X)$.

Proposition 1 (Nyström extension)**.** *For $1 \le i \le n$, let λ_i be an eigenvalue of $\mathcal{K}$, and ψ_i its unit $\mathcal{H}$-norm eigenfunction, $\mathcal{K}\{\psi_i\} = \lambda_i \psi_i$. Then λ_i is also an eigenvalue of $K(X, X)$. Moreover if $\boldsymbol{e}_i$, is a unit-norm eigenvector, $K(X, X)\boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$, we have,*

$$\psi_i = K(\cdot, X)\frac{\boldsymbol{e}_i}{\sqrt{\lambda_i}} = \sum_{j=1}^{n} K(\cdot, \boldsymbol{x}_j)\frac{e_{ij}}{\sqrt{\lambda_i}}. \tag{12}$$

We review EigenPro 2.0 which is a closely related algorithm for kernel regression, i.e., when $Z = X$.

2.1. Background on EigenPro

EigenPro 1.0, proposed in (Ma & Belkin, 2017), is an iterative solver for solving the linear system in equation (3) based on a preconditioned stochastic gradient descent in a Hilbert space,

$$f^{t+1} = f^t - \eta \cdot \mathcal{P}\left\{\nabla_f L(f^t)\right\}. \tag{13}$$

Here $\mathcal{P}$ is a preconditioner. Due to its iterative nature, EigenPro can handle $\lambda = 0$ in equation equation (3), corresponding to the problem of kernel interpolation, since in that case, the learned model satisfies $f(\boldsymbol{x}_i) = y_i$ for all samples in the training-set.

It can be shown that the following iteration in $\mathbb{R}^n$

$$\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t - \eta(\boldsymbol{I}_n - \boldsymbol{Q})(K(X,X)\boldsymbol{\alpha}^t - \boldsymbol{y}), \tag{14}$$

emulates equation (13) in $\boldsymbol{\mathcal{H}}$, see Lemma 7 in the Appendix. The above iteration is a preconditioned version of the Richardson iteration, (Richardson, 1911), with well-known convergence properties. Here, $\boldsymbol{Q}$ as a rank-q symmetric matrix obtained from the top-q eigensystem of $K(X,X)$, with $q \ll n$. Importantly $\boldsymbol{Q}$ commutes with $K(X,X)$.

The preconditioner, $\mathcal{P}$ acts to flatten the spectrum of the Hessian $\mathcal{K}$. In $\mathbb{R}^n$, the matrix $\boldsymbol{I}_n - \boldsymbol{Q}$ has the same effect on $K(X,X)$. The largest stable learning rate is then $\frac{2}{\lambda_{q+1}}$ instead of $\frac{2}{\lambda_1}$. Hence a larger q, allows faster training when $\mathcal{P}$ is chosen appropriately.

EigenPro 2.0 proposed in (Ma & Belkin, 2019), applies a stochastic approximation for $\mathcal{P}$ based on the Nyström extension. We apply EigenPro 2.0 to perform an inexact projection step in our algorithm.

3. Problem Formulation

In this work we aim to learn a general kernel model to minimize the square loss over a training set. We will solve the following infinite dimensional convex constrained optimization problem in a scalable manner.

$$\underset{f}{\text{minimize}}\ L(f) = \sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2, \tag{15a}$$

$$\text{subject to} \quad f \in \boldsymbol{\mathcal{Z}} := \text{span}\Big(\{K(\cdot, \mathbf{z}_j)\}_{j=1}^{p}\Big). \tag{15b}$$

The term "scalable" refers to both large sample size (n) and large model size (p). For instance Figure 3 shows an experiment with 5 million samples and the model size of 1 million. Our algorithm to solve this problem, EigenPro 3.0, is derived using a projected preconditioned gradient descent iteration.

4. EigenPro 3.0 derivation: Projected preconditioned gradient descent

In this section we derive EigenPro 3.0-Exact-Projection (Algorithm 1), a precursor to EigenPro 3.0, to learn general kernel models. This algorithm is based on a function space projected gradient method. However it does not scale well. In Section 5 we make it scalable by applying stochastic approximations, which finally yields EigenPro 3.0 (Algorithm 2).

We will apply a function-space projected gradient method to solve this problem,

$$f^{t+1} = \text{proj}_{\boldsymbol{\mathcal{Z}}}\left(f^t - \eta\mathcal{P}\left\{\nabla_f L(f^t)\right\}\right), \tag{16}$$

where $\text{proj}_{\boldsymbol{\mathcal{Z}}}(u) := \underset{f\in\boldsymbol{\mathcal{Z}}}{\text{argmin}}\ \|u - f\|^2_{\boldsymbol{\mathcal{H}}}$, and $\nabla_f L(f^t)$ is the Fréchet derivative at f^t as given in equation (8), $\mathcal{P}$ is a preconditioning operator given in equation (23), η is a learning rate. Note that the operator $\text{proj}_{\boldsymbol{\mathcal{Z}}} : \boldsymbol{\mathcal{H}} \to \boldsymbol{\mathcal{Z}}$ projects functions from $\boldsymbol{\mathcal{H}}$ onto the subspace $\boldsymbol{\mathcal{Z}}$, ensuring feasibility of the optimization problem.

Remark 1. Note that even though equation (16) is an iteration over functions which are infinite dimensional objects $\{f^t\}_{t\geq 0}$, we can represent this iteration in finite dimensions as $\{\boldsymbol{\alpha}^t\}_{t\geq 0}$, where $\boldsymbol{\alpha}_t \in \mathbb{R}^p$. To see this, observe that $f^t \in \boldsymbol{\mathcal{Z}}$, whereby we express it as,

$$f^t = K(\cdot, Z)\boldsymbol{\alpha}^t \in \boldsymbol{\mathcal{H}}, \qquad \text{for an } \boldsymbol{\alpha}^t \in \mathbb{R}^p. \tag{17}$$

Furthermore, the evaluation of f^t above at X, is

$$f^t(X) = K(X, Z)\boldsymbol{\alpha}^t \in \mathbb{R}^n. \tag{18}$$

4.1. Gradient

Due to equations (8) and (18) together, the gradient is given by the function,

$$\nabla_f L(f^t) = K(\cdot, X)(f^t(X) - \boldsymbol{y}) \tag{19a}$$

$$= K(\cdot, X)(K(X,Z)\boldsymbol{\alpha}^t - \boldsymbol{y}) \in \boldsymbol{\mathcal{X}} \tag{19b}$$

$$\boldsymbol{\mathcal{X}} := \text{span}(\{K(\cdot, \boldsymbol{x}_i)\}_{i=1}^{n}). \tag{19c}$$

Observe that the gradient does not lie in $\boldsymbol{\mathcal{Z}}$ and hence a step of gradient descent would leave $\boldsymbol{\mathcal{Z}}$, and the constraint is violated. This necessitates a projection onto $\boldsymbol{\mathcal{Z}}$. For finitely generated sub-spaces such as $\boldsymbol{\mathcal{Z}}$, the projection operation involves solving a finite dimensional linear system.

4.2. $\boldsymbol{\mathcal{H}}$-norm projection

Functions in $\boldsymbol{\mathcal{Z}}$ can be expressed as $K(\cdot, Z)\boldsymbol{\theta}$. Hence we can rewrite the projection problem in equation (16) as a minimization in $\mathbb{R}^p$, with $\boldsymbol{\theta}$ as the unknowns. Observe that,

$$\underset{f}{\text{argmin}}\ \|f - u\|^2_{\boldsymbol{\mathcal{H}}} = \underset{f}{\text{argmin}}\ \langle f, f\rangle_{\boldsymbol{\mathcal{H}}} - 2\langle f, u\rangle_{\boldsymbol{\mathcal{H}}}$$

Algorithm 1 EigenPro 3.0 Exact-Projection

Require: Data (X, y), centers Z, initialization $\boldsymbol{\alpha}^0$, preconditioning level q.

$(\Lambda, \boldsymbol{E}, \lambda_{q+1}) \leftarrow$ top-q eigensystem of $K(X, X)$
$\boldsymbol{Q} \leftarrow \boldsymbol{E}(\boldsymbol{I}_q - \lambda_{q+1}\Lambda^{-1})\boldsymbol{E}^\top \in \mathbb{R}^{n\times n}$
while Stopping criterion not reached **do**
  $\boldsymbol{g} \leftarrow K(X, Z)\boldsymbol{\alpha} - \boldsymbol{y}$
  $\boldsymbol{h} \leftarrow K(Z, X)(\boldsymbol{I}_n - \boldsymbol{Q})\boldsymbol{g}$
  $\boldsymbol{\theta} \leftarrow K(Z, Z)^{-1}\boldsymbol{h}$
  $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \eta\,\boldsymbol{\theta}$

since $\|u\|^2_{\boldsymbol{\mathcal{H}}}$ does not affect the solution. Further, using $f = K(\cdot, Z)\boldsymbol{\theta}$, we can show that

$$\langle f, f\rangle_{\boldsymbol{\mathcal{H}}} - 2\,\langle f, u\rangle_{\boldsymbol{\mathcal{H}}} = \boldsymbol{\theta}^\top K(Z, Z)\boldsymbol{\theta} - 2\boldsymbol{\theta}^\top u(Z). \quad (20)$$

This yields a simple method to calculate the projection onto $\boldsymbol{\mathcal{Z}}$.

$$\text{proj}_{\boldsymbol{\mathcal{Z}}}\{u\} = \underset{f\in\boldsymbol{\mathcal{Z}}}{\text{argmin}}\ \|f - u\|^2_{\boldsymbol{\mathcal{H}}} = K(\cdot, Z)\widehat{\boldsymbol{\theta}} \quad (21a)$$

$$= K(\cdot, Z)K(Z, Z)^{-1}u(Z) \in \boldsymbol{\mathcal{Z}}, \quad (21b)$$

where

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\mathbb{R}^p}{\text{argmin}}\ \boldsymbol{\theta}^\top K(Z, Z)\boldsymbol{\theta} - 2\boldsymbol{\theta}^\top u(Z) = K(Z, Z)^{-1}u(Z).$$

Notice that $\widehat{\boldsymbol{\theta}}$ above is linear in u, and $f^t(Z) = K(Z, Z)\boldsymbol{\alpha}^t$. Hence we have the following lemma.

Proposition 2 (Projection)**.** *The projection step in equation* (16) *can be simplified as,*

$$f^{t+1} = f^t - \eta\, K(\cdot, Z)K(Z, Z)^{-1}\times \left(\mathcal{P}\left\{\nabla_f L(f^t)\right\}(Z)\right) \in \boldsymbol{\mathcal{Z}}. \quad (22)$$

Hence, in order to perform the update, we only need to know $\mathcal{P}\left\{\nabla_f L(f^t)\right\}(Z)$, i.e., the evaluation of the preconditioned Fréchet derivative at the model centers. This can be evaluated efficiently as described below.

4.3. Preconditioner agnostic to the model

Just like with usual gradient descent, the largest stable learning rate is governed by the largest eigenvalue of the Hessian of the objective in equation (15), which is given by equation (10). The preconditioner $\mathcal{P}$ in equation (16) acts to reduce the effect of a few large eigenvalues. We choose $\mathcal{P}$ given in equation (23), just like (Ma & Belkin, 2017).

$$\mathcal{P} := \mathcal{I} - \sum_{i=1}^{q}\left(1 - \frac{\lambda_{q+1}}{\lambda_q}\right)\psi_i \otimes \psi_i \quad : \boldsymbol{\mathcal{H}} \to \boldsymbol{\mathcal{H}}. \quad (23)$$

Recall from Section 2 that ψ_i are eigenfunctions of the Hessian $\mathcal{K}$, characterized in Proposition 1. Note that this preconditioner is independent of Z. Since $\nabla_f L(f^t) \in \boldsymbol{\mathcal{X}}$, we only need to understand $\mathcal{P}$ on $\boldsymbol{\mathcal{X}}$. Let $(\Lambda_q, \boldsymbol{E}_q, \lambda_{q+1})$ be the top-q eigensystem of $K(X, X)$, see Definition 1. Define the rank-q matrix,

$$\boldsymbol{Q} := \boldsymbol{E}_q(\boldsymbol{I}_q - \lambda_{q+1}\Lambda_q^{-1})\boldsymbol{E}_q^\top \in \mathbb{R}^{n\times n}. \quad (24)$$

The following lemma outlines the computation involved in preconditioning.

Proposition 3 (Preconditioner)**.** *The action of $\mathcal{P}$ from equation* (23) *on functions in $\boldsymbol{\mathcal{X}}$ is given by,*

$$\mathcal{P}\left\{K(\cdot, X)\boldsymbol{a}\right\} = K(\cdot, X)(\boldsymbol{I}_n - \boldsymbol{Q})\boldsymbol{a}, \quad (25)$$

for all $\boldsymbol{a} \in \mathbb{R}^m$.

Since we know from equation (19) that $\nabla_f L(f^t) = K(\cdot, X)(K(X, Z)\boldsymbol{\alpha}^t - \boldsymbol{y})$, we have,

$$\mathcal{P}\left\{\nabla_f L(f^t)\right\}(Z) = K(Z, X)(\boldsymbol{I}_n - \boldsymbol{Q})(K(X, Z)\boldsymbol{\alpha}^t - \boldsymbol{y}).$$

The following lemma combines this with Proposition 2 to get the update equation for Algorithm 1.

Lemma 4 (Algorithm 1 iteration)**.** *The following iteration in $\mathbb{R}^p$ emulates equation* (16) *in $\boldsymbol{\mathcal{H}}$,*

$$\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t - \eta\, K(Z, Z)^{-1}K(Z, X)(\boldsymbol{I}_n - \boldsymbol{Q})\times (K(X, Z)\boldsymbol{\alpha}^t - \boldsymbol{y}). \quad (26)$$

Algorithm 1 does not scale well to large models and large datasets, since it requires $O(np \vee p^2)$ memory and $O(np \vee p^3)$ FLOPS. We now propose stochastic approximations that drastically make it scalable to both large models as well as large datasets.

Algorithm 2 EigenPro 3.0

Require: Data $(X, \boldsymbol{y})$, centers Z, batch size m, Nyström size s,, preconditioner level q.

Fetch subsample $X_s \subseteq X$ of size s
$(\Lambda, \boldsymbol{E}, \lambda_{q+1}) \leftarrow$ top-q eigensystem of $K(X_s, X_s)$
$\boldsymbol{C} = K(Z, X_s)\boldsymbol{E}(\Lambda^{-1} - \lambda_{q+1}\Lambda^{-2})\boldsymbol{E}^\top \in \mathbb{R}^{p\times s}$
while Stopping criterion is not reached **do**
  Fetch minibatch $(X_m, \boldsymbol{y}_m)$
  $\boldsymbol{g}_m \leftarrow K(X_m, Z)\boldsymbol{\alpha} - \boldsymbol{y}_m$
  $\boldsymbol{h} \leftarrow K(Z, X_m)\boldsymbol{g}_m - \boldsymbol{C}K(X_s, X_m)\boldsymbol{g}_m$
  $\boldsymbol{\theta} \leftarrow$ EigenPro 2.0$(Z, \boldsymbol{h})$
  $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \frac{n}{m}\eta\,\boldsymbol{\theta}$

Note: EigenPro 2.0$(Z, \boldsymbol{h})$ solves $K(Z, Z)\boldsymbol{\theta} = \boldsymbol{h}$ approximately
See Table 1 for a comparison of costs.

5. Upscaling via stochastic approximations

Algorithm 1 suffers from 3 main issues. It requires — (i) access to entire dataset of size $O(n)$ at each iteration, (ii)

Algorithm	Compution		Memory
	Setup	per iteration	
EigenPro 3.0	s^2q	$p(m+s)+s(m+q)+T_{\text{ep2}}$	s^2+sm+M_{ep2}
EigenPro 3.0 ExactProjection	nq^2+p^3	$np+nq$	$pn+n^2$
FALKON	p^3	np	p^2

Table 1: **Algorithm complexity.** Number of training samples n, number of model centers p, batch size m, Nyström sub-sample size s, preconditioner level q. Here T_{ep2} is the time it takes to run EigenPro 2.0 for the approximate projection. In practice we only run 1 epoch of EigenPro 2.0 for large scale experiments for which $T_{\text{ep2}} = O(p^2)$. Similarly, $M_{\text{ep2}} = O(p)$ is the memory rquired for running EigenPro 2.0. Cost of kernel evaluations and number of classes are assumed to be $O(1)$.

$O(n^2)$ memory to calculate the preconditioner $\boldsymbol{Q}$, and (iii) $O(p^3)$ for the matrix inversion corresponding to an exact projection. This prevents scalability to large n and p.

In this section we present 3 stochastic approximation schemes — stochastic gradients, Nyström approximated preconditioning, and inexact projection — that drastically reduce the computational cost and memory requirements. These approximations together give us Algorithm 2.

Algorithm 1 emulates equation (16), whereas Algorithm 2 is designed to emulate its approximation,

$$f^{t+1} = f^t - \tfrac{n}{m}\eta \cdot \widetilde{\text{proj}}_{\mathbb{Z}}\left(\mathcal{P}_s\left\{\widetilde{\nabla}_f L(f^t)\right\}\right), \tag{27}$$

where $\widetilde{\nabla}_f L(f^t)$ is a stochastic gradient obtained from a sub-sample of size m, $\mathcal{P}_s$ is a preconditioner obtained via a Nyström extension based preconditioner from a subset of the data of size s, and $\widetilde{\text{proj}}_{\mathbb{Z}}$ is an inexact projection performed using EigenPro 2.0 to solve the projection equation $K(Z,Z)\boldsymbol{\theta} = \boldsymbol{h}$.

Stochastic gradients: We can replace the gradient with stochastic gradients, whereby $\widetilde{\nabla}_f L(f^t)$ only depends on a batch $(X_m, \boldsymbol{y}_m)$ of size m, denoted $X_m = \{\boldsymbol{x}_{i_j}\}_{j=1}^m$ and $\boldsymbol{y}_m = (y_{i_j}) \in \mathbb{R}^m$,

$$\widetilde{\nabla}_f L(f^t) = K(\cdot, X_m)(K(X_m, Z)\boldsymbol{\alpha} - \boldsymbol{y}_m) \in \boldsymbol{\mathcal{X}}. \tag{28}$$

Remark 2. Here we need to scale the learning rate by $\frac{n}{m}$, to get unbiased estimates of $\nabla_f L(f^t)$.

Nyström preconditioning: Previously, we obtained the preconditioner $\mathcal{P}$ from equation (23), which requires access to all samples. We now use the Nyström extension to approximate this preconditioner, see (Williams & Seeger, 2000). Consider a subset of size s, $X_s = \{\boldsymbol{x}_{i_k}\}_{k=1}^s \subset X$. We introduce the Nyström preconditioner,

$$\mathcal{P}_s := \mathcal{I} - \sum_{i=1}^{s}\left(1 - \frac{\lambda_{q+1}^s}{\lambda_i^s}\right)\psi_i^s \otimes \psi_i^s. \tag{29}$$

where ψ_i^s are eigenfunctions of $\mathcal{K}^s := \sum_{k=1}^s K(\cdot, \boldsymbol{x}_{i_k}) \otimes K(\cdot, \boldsymbol{x}_{i_k})$. Note that $\mathcal{K}^s \approx \frac{s}{n}\mathcal{K}$ since both approximate $\mathcal{T}_K$ as shown in equation (11). This preconditioner was first proposed in (Ma & Belkin, 2019).

Next, we must understand the action of $\mathcal{P}_s$ on elements of $\boldsymbol{\mathcal{X}}$. Let $(\Lambda_q, \boldsymbol{E}_q, \lambda_{q+1})$ be the top-q eigensystem of $K(X_s, X_s)$. Define the rank-q matrix,

$$\boldsymbol{Q}_s := \boldsymbol{E}_q(\boldsymbol{I}_s - \lambda_{q+1}\Lambda_q^{-1})\Lambda_q^{-1}\boldsymbol{E}_q^\top \in \mathbb{R}^{s\times s}. \tag{30}$$

Lemma 5 (Nyström preconditioning)**.** *Let* $\boldsymbol{a} \in \mathbb{R}^m$*, and* X_m *chosen like in equation* (28)*, then,*

$$\mathcal{P}_s\{K(\cdot, X_m)\boldsymbol{a}\} = K(\cdot, X_m)\boldsymbol{a} - K(\cdot, X_s)\boldsymbol{Q}_s K(X_s, X_m)\boldsymbol{a}.$$

Consequently, using equation (28), we get,

$$\begin{aligned}\mathcal{P}_s\left\{\widetilde{\nabla}_f L(f^t)\right\}(Z) = \\ \Big(K(Z, X_m) - K(Z, X_s)\boldsymbol{Q}_s K(X_s, X_m)\Big)\times \\ \big(K(X_m, Z)\boldsymbol{\alpha}^t - \boldsymbol{y}_m\big) \in \mathbb{R}^p.\end{aligned} \tag{31}$$

Inexact projection: The projection step in Algorithm 1 requires the inverse of $K(Z,Z)$ which is computationally expensive. However this step is solving the $p \times p$ linear system

$$\begin{aligned} &K(Z,Z)\boldsymbol{\theta} = \boldsymbol{h} \\ &\boldsymbol{h} := \Big(K(Z, X_m) - K(Z, X_s)\boldsymbol{Q}_s K(X_s, X_m)\Big)\boldsymbol{g} \\ &\boldsymbol{g} := \big(K(X_m, Z)\boldsymbol{\alpha}^t - \boldsymbol{y}_m\big). \end{aligned} \tag{32}$$

Notice that this is the kernel interpolation problem EigenPro 2.0 can solve. This leads to the update,

$$\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t - \frac{n}{m}\eta\,\widehat{\boldsymbol{\theta}}^T \qquad \text{(EigenPro 3.0 update)}$$

where $\widehat{\boldsymbol{\theta}}^T$ is the solution to equation (32) after T steps of EigenPro 2.0 given in Algorithm 3 in the Appendix. Algorithm 2, EigenPro 3.0, implements the update above.

Remark 3 (Details on inexact-projection using EigenPro 2.0). We apply T steps of EigenPro 2.0 for the approximate projection. This algorithm itself applies a fast preconditioned SGD to solve the problem. The algorithm needs no hyper-parameters adjustment. However, you need to choose s and q. More details on this in the Appendix D.4.

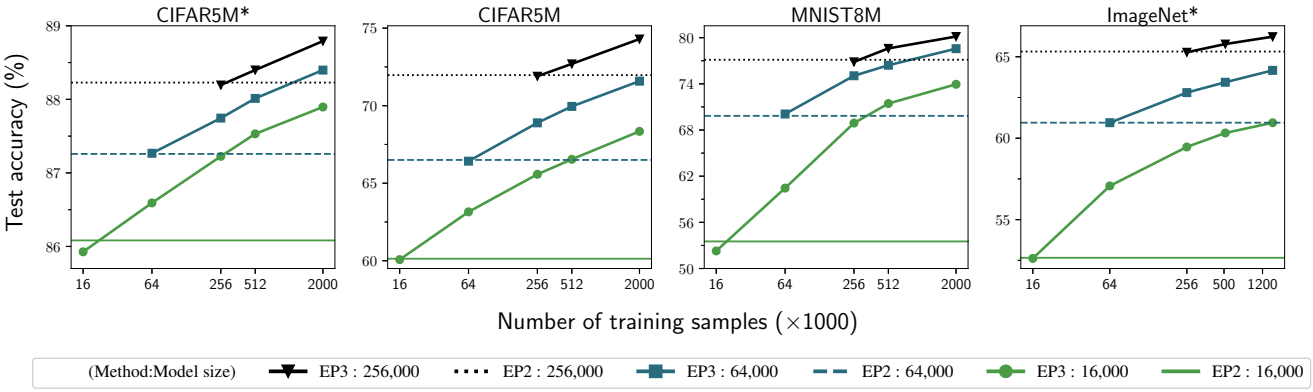

Figure 2: **(Scaling number of training samples)** Model centers are selected by random sub-sampling from the training data set. The baselines (lines without markers) are obtained from a standard kernel machine solved by EigenPro 2.0 over the centers and their corresponding labels. Lines with markers indicate the performance of kernel models trained with our algorithm (EigenPro 3.0) after 50 epochs.

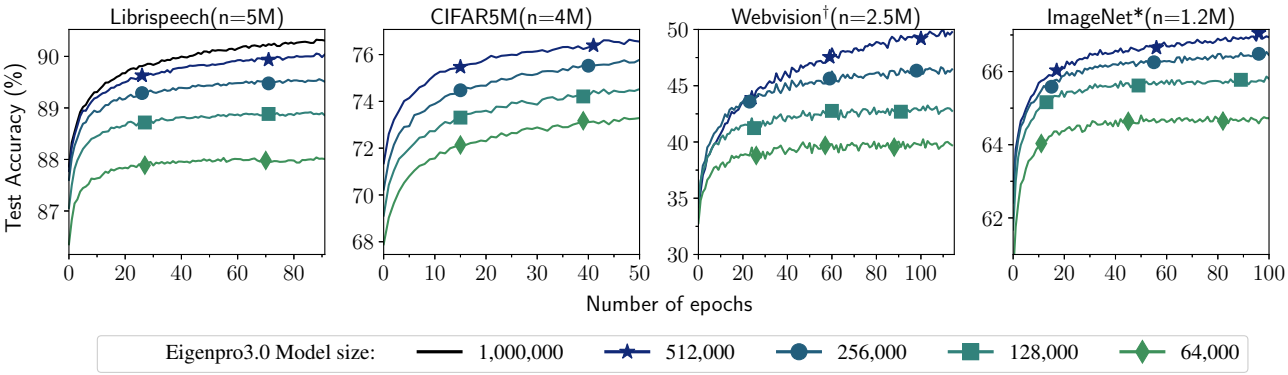

Figure 3: (**Scaling model size**) Performance of EigenPro 3.0 for different number of model centers, fixed number of data n.

Remark 4 (Decoupling). There are two preconditioners involved in EigenPro 3.0, a data preconditioner (for the stochastic gradient) which depends only on X, and a model preconditioner (for the inexact projection) which depends only on Z. This maintains the models decoupling from the training data.

Complexity analysis: We compare the complexity of the run-time and memory requirement of Algorithm 2 and Algorithm 1 with FALKON solver in Table 1.

6. Real data experiments

In this section, we demonstrate that our method can effectively scale both the size of the model and the number of training samples. We show that both of these factors are crucial for a better performance. We perform experiments on these datasets: (1) CIFAR10, CIFAR10* (Krizhevsky et al., 2009), (2) CIFAR5M, CIFAR5M* (Nakkiran et al., 2021), (3) ImageNet*, (Deng et al., 2009), (4) MNIST, (LeCun, 1998), (5) MNIST8M, (Loosli et al., 2007), (6) Fashion-MNIST, (Xiao et al., 2017), (7) Webvision†, (Li et al., 2017), and (8) Librispeech, (Panayotov et al., 2015). Details about datasets can be found in Appendix D. Our method can be implemented with any kernel function, but for the purpose of this demonstration, we chose to use the Laplace kernel due to its simplicity and empirical effectiveness. We treat multi-class classification problems as multiple independent binary regression problems, with targets from $\{0, 1\}$. The final prediction is determined by selecting the class with the highest predicted value among the K classes.

Scaling the number of training samples: Figure 2 illustrates that for a fixed model size, increasing the number of training samples leads to a significant improvement in performance, as indicated by the the lines with marker. As a point of comparison, we also include results from a standard kernel machine using the same centers, represented by the horizontal lines without markers. In this experiment, the centers were randomly selected from the data and they are a

*feature extraction using MobileNetV2

†feature extraction using ResNet-18

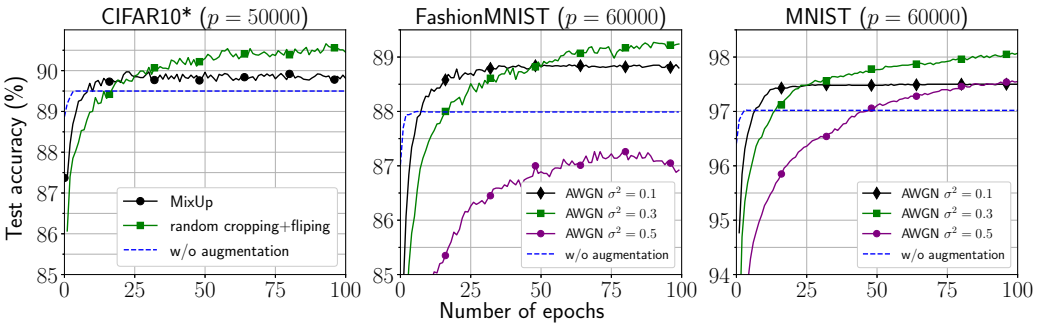

Figure 4: (**Data augmentation for kernel models**) Entire original dataset was used as as the model centers Z. The model was trained using the augmented set X (without the original data). MixUp and Crop+Flip augmentation was used for CIFAR10. Additive White Gaussian Noise (AWGN) augmentation was used for MNIST and FashionMNIST.

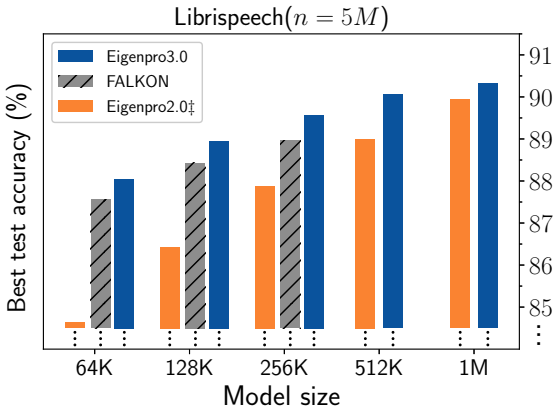

Figure 5: Comparison of EigenPro 3.0 (EP3) performance with FALKON and EigenPro 2.0 (EP2).
‡ For EigenPro 2.0 models size and number of training samples are the same. $n = 5$M was used for all other methods. FALKON could not be run for model size larger than 256K centers due to memory limitations. For each model size, EigenPro 2.0 and EigenPro 3.0 use the same centers.

subset of training data.

Scaling the model size: Figure 3 shows that, when the number of training samples is fixed, larger models perform significantly better.

To the best of our knowledge, FALKON is the only method capable of training general kernel models with model size larger than 100,000 centers. Our memory resources of 340GB RAM allowed us to handle up to 256,000 centers using FALKON. Figure 5 illustrates that our method outperforms FALKON by utilizing larger models. Moreover, for a fixed set of centers, training on greater number of data will boost the performance. Therefore, EigenPro 3.0 outperforms EigenPro 2.0 by training on more data.

Data augmentation for kernel models: Data augmentation is a crucial technique for improving the performance of deep networks. We demonstrate its benefits for general kernel models by selecting the model centers Z to be the original training data and the train set X to be the virtual examples generated through augmentation. To the best of our knowledge, this is the first implementation of data augmentation for kernel models at this scale. Figure 4 shows we have significant improvements in accuracy.

Data processing details: We performed our experiment on CIFAR10 with feature extraction, raw images of MNIST and FashionMNIST. For CIFAR10 augmentation, we apply random cropping and flipping before feature extraction. We also apply mix-up augmentation method from (Zhang et al., 2018) after feature extraction. For MNIST and FashionMNIST augmentation we added Gaussian noise with different variances. To the best of our knowledge, this is the first implementation of data augmentation for kernel models at this scale. Figure 4 shows we have significant improvements in accuracy.

7. Conclusions and Outlook

The remarkable success of Deep Learning has been in large part due to very large models trained on massive datasets. Any credible alternative requires a path to scaling model sizes as well as the size of the training set. Traditional kernel methods suffer from a severe scaling limitation as their model sizes are coupled to size of the training set. Yet, as we have seen in numerous experiments, performance improves with the amount of data far beyond the point where the amount of data exceeds the number of model centers. Other solvers, such as FALKON (Rudi et al., 2017), GPYTORCH (Gardner et al., 2018) GPFlow Matthews et al. (2017), allow for unlimited data but limit the model size.

In this work we provide a proof of concept showing that for kernel methods the barrier of limited model size can be overcome. Indeed, with a fixed model size our proposed algorithm, EigenPro 3.0 has no specific limitations on the number of samples it can use in training. As a simple illustration Fig. 6 demonstrates the results of a kernel machine trained on an augmented FashionMNIST dataset with 1.2×10^8 data samples. While increasing the model

Dataset	Model	$p = 100$	$p = 1000$	$p = 10000$
CIFAR10	$k-$means	**36.24**	**45.12**	**52.72**
($n = 50000$)	random	33.37 ± 0.50	44.19 ± 0.09	49.92 ± 0.08
CIFAR10*	$k-$means	**82.69**	**86.58**	**89.11**
($n = 50000$)	random	74.29 ± 0.44	84.38 ± 0.15	86.58 ± 0.06
MNIST	$k-$means	**91.89**	**95.96**	**97.69**
($n = 60000$)	random	87.24 ± 0.015	94.96 ± 0.102	97.31 ± 0.004
FashionMNIST	$k-$means	**78.66**	**85.55**	**88.13**
($n = 60000$)	random	76.24 ± 0.003	84.59 ± 0.069	87.84 ± 0.036

Table 2: (**Benefits of model flexibility**) Comparison between random centers selection and k-means clustering using EigenPro 3.0. Here p denotes the number of centers.
(* indicates a preprocessing step.)

size is more challenging, we have achieved 1 million centers and see no fundamental mathematical barrier to increasing the number of centers to 10 million and beyond. Furthermore, as EigenPro 3.0 is highly parallelizable, we anticipate future scaling to tens of millions of centers trained on billions of data points, approaching the scale of modern neural networks.

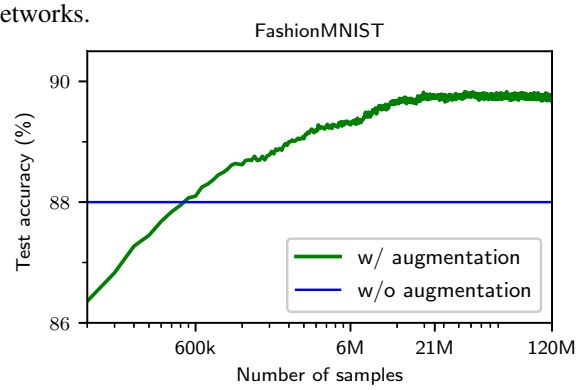

Figure 6: (**Training on 120 million samples**) The plot shows the results of training on 120 million data samples generated by adding Gaussian noise to the original 60,000 images. The model centers are the images.

This line of research opens a pathway for a principled alternative to deep neural networks. While in our experiments we focused primarily on Laplace kernels for their simplicity and effectiveness, recently developed kernels such as NTK, CNTK, and other neural kernels from (Shankar et al., 2020), can be used to achieve state-of-the-art performance on various datasets. Our approach is compatible with any choice of kernel, and furthermore, can be adapted to kernels that learn features, such as Recursive Feature Machines (Radhakrishnan et al., 2022a).
Finally, we note that while the set of centers Z can be arbitrary, the choice of centers can affect model performance. Table 2 (see also (Que & Belkin, 2016)) demonstrates that using centers selected via the k-means algorithm often results in notable performance improvements when the number of centers is smaller than the number of samples. Further research should explore criteria for optimal center selection, incorporating recent advances, such as Rocket-propelled Cholesky (Chen et al., 2022), into EigenPro 3.0 to improve both model selection and construction of preconditioners.

ACKNOWLEDGMENTS

We thank Giacomo Meanti for help with using FALKON, and Like Hui for providing us with the extracted features of the Librispeech dataset.
We are grateful for the support from the National Science Foundation (NSF) and the Simons Foundation for the Collaboration on the Theoretical Foundations of Deep Learning (https://deepfoundations.ai/) through awards DMS-2031883 and #814639 and the TILOS institute (NSF CCF-2112665). This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing. This work used NVIDIA V100 GPUs NVLINK and HDR IB (Expanse GPU) at SDSC Dell Cluster through allocation TG-CIS220009 and also, Delta system at the National Center for Supercomputing Applications through allocation bbjr-delta-gpu from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296. Prior to 09/01/2022, we used the Extreme Science and Engineering Discovery Environment (XSEDE) (Towns et al., 2014), which is supported by NSF grant number ACI-1548562, Expanse CPU/GPU compute nodes, and allocations TG-CIS210104 and TG-CIS220009.

References

Aronszajn, N. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

Arora, S., Du, S. S., Li, Z., Salakhutdinov, R., Wang, R., and Yu, D. Harnessing the power of infinitely wide deep nets on small-data tasks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkl8sJBYvH.

Baldi, P., Sadowski, P., and Whiteson, D. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):1–9, 2014.

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.

Chen, Y., Welling, M., and Smola, A. Super-samples from kernel herding. *UAI'10: Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Pages 109–116*, 2010.

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.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Gardner, J., Pleiss, G., Wu, R., Weinberger, K., and Wilson, A. Product kernel interpolation for scalable gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pp. 1407–1416. PMLR, 2018.

Geifman, A., Yadav, A., Kasten, Y., Galun, M., Jacobs, D., and Ronen, B. On the similarity between the laplace and neural tangent kernels. In *Advances in Neural Information Processing Systems*, 2020.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Vinyals, O., Rae, J. W., and Sifre, L. An empirical analysis of compute-optimal large language model training. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=iBBcRUlOAPR.

Hui, L. and Belkin, M. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=hsFN92eQEla.

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

Jurafsky, D. *Speech & language processing*. Pearson Education India, 2000.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

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.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Lee, J., Schoenholz, S., Pennington, J., Adlam, B., Xiao, L., Novak, R., and Sohl-Dickstein, J. Finite versus infinite neural networks: an empirical study. *Advances in Neural Information Processing Systems*, 33:15156–15172, 2020.

Li, W., Wang, L., Li, W., Agustsson, E., and Van Gool, L. Webvision database: Visual learning and understanding from web data. *arXiv preprint arXiv:1708.02862*, 2017.

Li, Z., Wang, R., Yu, D., Du, S. S., Hu, W., Salakhutdinov, R., and Arora, S. Enhanced convolutional neural tangent kernels. *arXiv preprint arXiv:1911.00809*, 2019.

Loosli, G., Canu, S., and Bottou, L. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, 2, 2007.

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. *International Conference on Machine Learning*, pp. 3325–3334, 2018.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017. URL http://jmlr.org/papers/v18/16-537.html.

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.

Nakkiran, P., Neyshabur, B., and Sedghi, H. The deep bootstrap framework: Good online learners are good offline generalizers. *International Conference on Learning Representations*, 2021.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5206–5210. IEEE, 2015.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Poggio, T. and Girosi, F. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

Que, Q. and Belkin, M. Back to the future: Radial basis function networks revisited. In *Artificial intelligence and statistics*, pp. 1375–1383. PMLR, 2016.

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*, 2022a.

Radhakrishnan, A., Stefanakis, G., Belkin, M., and Uhler, C. Simple, fast, and flexible framework for matrix completion with infinite width neural networks. *Proceedings of the National Academy of Sciences*, 119(16):e2115064119, 2022b.

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

Richardson, L. F. Ix. the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 210(459-470): 307–357, 1911.

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

Schölkopf, B., Smola, A. J., Bach, F., et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

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.

Titsias, M. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pp. 567–574. PMLR, 2009.

Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., Hazlewood, V., Lathrop, S., Lifka, D., Peterson, G. D., Roskies, R., Scott, J., and Wilkins-Diehr, N. Xsede: Accelerating scientific discovery. *Computing in Science & Engineering*, 16(05):62–74, sep 2014. ISSN 1558-366X. doi: 10.1109/MCSE.2014.80.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., Unno, Y., Enrique Yalta Soplin, N., Heymann, J., Wiesner, M., Chen, N., Renduchintala, A., and Ochiai, T. ESPnet: End-to-end speech processing toolkit. In *Proceedings of Interspeech*, pp. 2207–2211, 2018. doi: 10.21437/Interspeech.2018-1456. URL http://dx.doi.org/10.21437/Interspeech.2018-1456.

Wightman, R. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

Williams, C. and Seeger, M. Using the nyström method to speed up kernel machines. *Advances in neural information processing systems*, 13, 2000.

Wilson, A. and Nickisch, H. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International conference on machine learning*, pp. 1775–1784. PMLR, 2015.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yang, T., Li, Y.-F., Mahdavi, M., Jin, R., and Zhou, Z.-H. Nyström method vs random fourier features: A theoretical and empirical comparison. *Advances in neural information processing systems*, 25, 2012.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. mixup: Beyond empirical risk minimization. *International Conference on Learning Representations*, 2018.

Zhu, L., Liu, C., and Belkin, M. Transition to linearity of general neural networks with directed acyclic graph architecture. *Advances in neural information processing systems*, 2022.

APPENDICES

Table 3: Symbolic notation for EigenPro 3.0 in Algorithm 2. They satisfy $m < n$, and $q < s < n$.

Symbol	Purpose
n	Number of samples
m	Batch-size
p	Model size
s	Nyström approximation subsample size
q	Preconditioner level

A. Fixed point analysis

Here we provide a characterization of the fixed point of Algorithm 1.

Lemma 6. *For any dataset $X, \boldsymbol{y}$ and any choice of model centers Z, if the learning rate satisfies*

$$\eta < \frac{2}{\lambda_{\max}\left(K(Z,X)(I_n - \boldsymbol{Q})K(X,Z)\right)}$$

we have that

$$\lim_{t\to\infty} \boldsymbol{\alpha}_t = \left(K(Z,X)(I_n - \boldsymbol{Q})K(X,Z)\right)^{-1} K(Z,X)(I_n - \boldsymbol{Q})\boldsymbol{y}.$$

Furthermore, if $\boldsymbol{y} = K(X,Z)\boldsymbol{\alpha}^* + \boldsymbol{\xi}$, where ξ_i are independent centered with $\mathbb{E}\,|\xi_i|^2 = \sigma^2$, then

$$\begin{aligned}
\lim_{t\to\infty} \mathbb{E}\,\alpha_t &= \boldsymbol{\alpha}^* \\
\lim_{t\to\infty} \frac{\mathbb{E}\,\|\boldsymbol{\alpha}_t - \boldsymbol{\alpha}^*\|^2}{\sigma^2} &= \operatorname{tr}\left(\left(K(Z,X)(I_n - \boldsymbol{Q})K(X,Z)\right)^{-2} K(Z,X)(I_n - \boldsymbol{Q})^2 K(X,Z)\right) \\
&= n - \operatorname{trace}\left(\left(K(Z,X)(I_n - \boldsymbol{Q})K(X,Z)\right)^{-2} K(Z,X)\boldsymbol{Q}(I_n - \boldsymbol{Q})K(X,Z)\right)
\end{aligned}$$

B. Proofs of intermediate results

B.1. Proof of proposition 1

Proposition (Nyström extension)**.** *For $1 \le i \le n$, let λ_i be an eigenvalue of $\mathcal{K}$, and ψ_i its unit $\boldsymbol{\mathcal{H}}$-norm eigenfunction,* i.e., $\mathcal{K}\{\psi_i\} = \lambda_i \psi_i$. Then λ_i is also an eigenvalue of $K(X,X)$. Moreover if $\boldsymbol{e}_i$, is its unit-norm eigenvector, i.e., $K(X,X)\boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$, we have,

$$\psi_i = K(\cdot, X)\frac{\boldsymbol{e}_i}{\sqrt{\lambda_i}}. \tag{33}$$

Proof. Let $\psi \in \mathcal{H}$ be an eigenfunction of $\mathcal{K}$. Then by definition of $\mathcal{K}$ we have,

$$\lambda\psi = \mathcal{K}\{\psi\} = \sum_{i=1}^{n} K(\cdot, \boldsymbol{x}_i)\psi(\boldsymbol{x}_i). \tag{34}$$

As the result we can write ψ as below,

$$\psi = \sum_{i=1}^{n} \frac{\psi(\boldsymbol{x}_i)}{\lambda} K(\cdot, \boldsymbol{x}_i). \tag{35}$$

If we apply covariance operator to the both side of 35 we have,

$$\mathcal{K}\{\psi\} = \mathcal{K}\left\{\sum_{i=1}^{n} \frac{\psi(\boldsymbol{x}_i)}{\lambda} K(\cdot, \boldsymbol{x}_i)\right\} = \sum_{i,j=1}^{n} \frac{\psi(\boldsymbol{x}_i)}{\lambda} K(\boldsymbol{x}_i, \boldsymbol{x}_j) K(\cdot, \boldsymbol{x}_j) = \sum_{j=1}^{n} \psi(\boldsymbol{x}_j) K(\cdot, \boldsymbol{x}_j). \tag{36}$$

The last equation hold because of equation (34). If we define vector $\boldsymbol{\beta}$ such that $\boldsymbol{\beta}_i = \frac{\psi(x_i)}{\lambda}$, then 36 can be rewritten as,

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{\beta}_i K(\boldsymbol{x}_i, \boldsymbol{x}_j)K(\cdot, \boldsymbol{x}_i) = \lambda \sum_{i=1}^{n}\boldsymbol{\beta}_i K(\cdot, \boldsymbol{x}_i). \tag{37}$$

Compactly we can write 37 as below,

$$K(X,X)^2\boldsymbol{\beta} = \lambda K(X,X)\boldsymbol{\beta} \implies K(X,X)\boldsymbol{\beta} = \lambda\boldsymbol{\beta}.$$

The last implication holds because $K(X,X)$ is invertable. Thus $\boldsymbol{\beta}$ is an eigenvector of $K(X,X)$. It remains to determine the scale of $\boldsymbol{\beta}$.

Now, norm of ψ can be simplified as

$$\|\psi\|^2_{\boldsymbol{\mathcal{H}}} = \left\langle \sum_{i=1}^{n}\beta_i K(\cdot, \boldsymbol{x}_i), \sum_{j=1}^{n}\beta_j K(\cdot, \boldsymbol{x}_j)\right\rangle_{\boldsymbol{\mathcal{H}}} \tag{38}$$

$$= \sum_{i,j=1}^{n}\beta_i\beta_j \left\langle K(\cdot, \boldsymbol{x}_i), K(\cdot, \boldsymbol{x}_j)\right\rangle_{\boldsymbol{\mathcal{H}}} = \boldsymbol{\beta}^\top K(X,X)\boldsymbol{\beta} = \lambda \|\boldsymbol{\beta}\|^2. \tag{39}$$

Since ψ is unit norm, we have $\|\boldsymbol{\beta}\| = \frac{1}{\sqrt{\lambda}}$. This concludes the proof. □

B.2. Proof of lemma 5

Lemma (Nyström preconditioning)**.** *Let $\boldsymbol{a} \in \mathbb{R}^m$, then we have that,*

$$\mathcal{P}_s\left\{K(\cdot, X_m)\boldsymbol{a}\right\} = K(\cdot, X_m)\boldsymbol{a} - K(\cdot, X_s)\boldsymbol{Q}_s K(X_s, X_m)\boldsymbol{a}. \tag{40}$$

Where $Q_s = E_{s,q}(\boldsymbol{I}_n - \lambda_{s,q+1}\Lambda_{s,q}^{-1})\Lambda_{s,q}^{-1}E_{s,q}^\top$.

Proof. Recall that $\mathcal{P}_s := \mathcal{I} - \sum_{i=1}^{q}\left(1 - \frac{\lambda_{q+1}}{\lambda_q}\right)\psi_i \otimes \psi_i$. By this definition we can write,

$$\begin{aligned}
\mathcal{P}_s\left(K(\cdot, X_M)\boldsymbol{\alpha}\right) &= K(\cdot, X_M)\boldsymbol{\alpha} - \sum_{i=1}^{s}(1 - \frac{\lambda_{q+1}^s}{\lambda_i^s})\left\langle \psi_i^s, K(\cdot, X_M)\boldsymbol{\alpha}\right\rangle_{\boldsymbol{\mathcal{H}}}\psi_i^s \\
&= K(\cdot, X_M)\boldsymbol{\alpha} - \sum_{i=1}^{q}\frac{1}{\lambda_i^s}(1 - \frac{\lambda_{q+1}^s}{\lambda_i^s})\left\langle K(\cdot, X_s)\boldsymbol{e}_i, K(\cdot, X_M)\boldsymbol{\alpha}\right\rangle_{\boldsymbol{\mathcal{H}}} K(\cdot, X_s)\boldsymbol{e}_i \\
&= K(\cdot, X_M)\boldsymbol{\alpha} - \sum_{i=1}^{q}\frac{1}{\lambda_i^s}(1 - \frac{\lambda_{q+1}^s}{\lambda_i})\left\langle K(\cdot, X_s)\boldsymbol{e}_i, K(\cdot, X_M)\boldsymbol{\alpha}\right\rangle_{\boldsymbol{\mathcal{H}}} K(\cdot, X_s)\boldsymbol{e}_i \\
&= K(\cdot, X_M)\boldsymbol{\alpha} - \sum_{i=1}^{q}(1 - \frac{\lambda_{q+1}^s}{\lambda_i^s})K(\cdot, X_s)\boldsymbol{e}_i\boldsymbol{e}_i^\top K(X_s, X_M)\boldsymbol{\alpha}.
\end{aligned}$$

Note that we used proposition 1 for ψ. Now we can compactly write the last expression as below,

$$\begin{aligned}
\mathcal{P}_s\left(K(\cdot, X_M)\boldsymbol{\alpha}\right) &= K(\cdot, X_M)\boldsymbol{\alpha} - K(\cdot, X_s)E_{s,q}(\boldsymbol{I}_n - \lambda_{s,q+1}\Lambda_{s,q}^{-1})\Lambda_{s,q}^{-1}E_{s,q}^\top K(X_s, X_M)\boldsymbol{\alpha} \\
&= K(\cdot, X_M)\boldsymbol{\alpha} - K(\cdot, X_s)Q_s K(X_s, X_M)\boldsymbol{\alpha}.
\end{aligned}$$

This concludes the proof.

□

Algorithm 3 EigenPro 2.0$(X, \boldsymbol{y})$. Solves the linear system $K(X, X)\boldsymbol{\theta} = \boldsymbol{y}$

Require: Data $(X, \boldsymbol{y})$, Nyström size s, preconditioner level q
  $\boldsymbol{\alpha} \leftarrow \mathbf{0} \in \mathbb{R}^n$ ▷ initialization
  $X_s, (\boldsymbol{E}, \boldsymbol{D}), \eta, m \leftarrow$ EigenPro 2.0_setup(X, s, q)
  while Stopping criterion not reached **do**
    $\boldsymbol{\alpha} \leftarrow$ EigenPro 2.0_iteration$(X, \boldsymbol{y}, X_s, \boldsymbol{E}, \boldsymbol{D}, \boldsymbol{\alpha}, m, \eta)$
  return $\boldsymbol{\alpha}$

EigenPro2_setup(X, s, q)
Require: Data X, Nyström size s, preconditioner size q
  Fetch a subsample $X_s \subseteq X$ of size s
  $(\boldsymbol{E}, \Lambda_q, \lambda_{q+1}) \leftarrow$ top-q eigensystem of $K(X_s, X_s)$ ▷ $\boldsymbol{E} \in \mathbb{R}^{q\times s}, \Lambda = \text{diag}(\lambda_i) \in \mathbb{R}^{q\times q}$
  $\boldsymbol{D}_{ii} = \frac{1}{s\lambda_i}\left(1 - \frac{\lambda_{q+1}}{\lambda_i}\right)$
  $\beta \leftarrow \max_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) \in S$
  $m \leftarrow \min\left(\frac{\beta}{\lambda_{q+1}}, \mathsf{bs}_{\text{gpu}}\right)$ ▷ batch size*
  $\eta \leftarrow \begin{cases} \frac{\beta}{2m} & m < \frac{\beta}{\lambda_{q+1}} \\ \frac{0.99m}{\beta + (m-1)\lambda_{q+1}} & \text{otherwise} \end{cases}$ ▷ learning rate
  return $X_s, (\boldsymbol{E}, \boldsymbol{D}), \eta, m$

EigenPro2_iteration$(X, \boldsymbol{y}, X_s, \boldsymbol{E}, \boldsymbol{D}, \boldsymbol{\alpha}, m, \eta)$
Require: Data $(X, \boldsymbol{y})$, Nyström subset X_s, preconditioner $(\boldsymbol{E}, \boldsymbol{D})$, current estimate $\boldsymbol{\alpha}$, batchsize m
  Fetch minibatch $(X_m, \boldsymbol{y}_m)$ of size m
  $\boldsymbol{g}_m \leftarrow K(X_m, X)\boldsymbol{\alpha} - \boldsymbol{y}_m$ ▷ stochastic gradient
  $\boldsymbol{\alpha}_m \leftarrow \boldsymbol{\alpha}_m - \frac{\eta}{m}\boldsymbol{g}_m$ ▷ gradient step
  $\boldsymbol{\alpha}_s \leftarrow \boldsymbol{\alpha}_s + \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^\top K(X_s, X_m)\boldsymbol{g}_m$ ▷ gradient correction
  return Updated estimte $\boldsymbol{\alpha}$

C. Details on EigenPro 2.0

Lemma 7. *The iteration in $\mathbb{R}^n$*

$$\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^{t+1} - \eta(\boldsymbol{I}_n - \boldsymbol{Q})(K(X, X)\boldsymbol{\alpha}^t - \boldsymbol{y}), \tag{41}$$

where $Q = \boldsymbol{E}(\boldsymbol{I}_n - \lambda_{q+1}\Lambda_q^{-1})\boldsymbol{E}^\top$, emulates the following iteration in $\boldsymbol{\mathcal{H}}$.

$$f^{t+1} = f^t - \eta\mathcal{P}\left\{\nabla_f L(f^t)\right\}. \tag{42}$$

Proof. Recall that $\nabla_f L(f^t) = K(\cdot, X)(f^t(X) - \boldsymbol{y})$ from equation (8), and $f^t(X) = K(X, X)\boldsymbol{\alpha}^t$. from equation (18). We define $\boldsymbol{g}^t := f^t(X) - \boldsymbol{y} = K(X, X)\boldsymbol{\alpha}^t - \boldsymbol{y}$. Following steps of the proof in Appendix B.2 we have

$$\begin{aligned}
\mathcal{P}\{\nabla_f L(f^t)\} &= K(\cdot, X)\boldsymbol{g}^t - \sum_{i=1}^{q}(1 - \frac{\lambda_{q+1}}{\lambda_i})K(\cdot, X)\boldsymbol{e}_i^\top \boldsymbol{e}_i K(X, X)\boldsymbol{g}^t \\
&= K(\cdot, X)\boldsymbol{g}^t - K(\cdot, X)\boldsymbol{E}(\boldsymbol{I}_n - \lambda_{q+1}\Lambda_q^{-1})\Lambda^{-1}\boldsymbol{E}^\top K(X, X)\boldsymbol{g}^t \\
&\overset{(a)}{=} K(\cdot, X)\boldsymbol{g}^t - K(\cdot, X)\boldsymbol{E}(\boldsymbol{I}_n - \lambda_{q+1}\Lambda_q^{-1})\Lambda^{-1}\boldsymbol{E}^\top \boldsymbol{E}\Lambda\boldsymbol{E}^\top \boldsymbol{g}^t \\
&= K(\cdot, X)\boldsymbol{g}^t - K(\cdot, X)\boldsymbol{E}(\boldsymbol{I}_n - \lambda_{q+1}\Lambda_q^{-1})\boldsymbol{E}^\top \boldsymbol{g}^t \\
&= K(\cdot, X)\boldsymbol{g}^t - K(\cdot, X)\boldsymbol{Q}\boldsymbol{g}^t \\
&= K(\cdot, X)(\boldsymbol{I}_n - \boldsymbol{Q})\boldsymbol{g}^t.
\end{aligned}$$

Where (a) follows from $K(X,X) = \boldsymbol{E}\Lambda\boldsymbol{E}^\top$. Now since $f^t = K(\cdot, X)\boldsymbol{\alpha}^t$, equation (42) can be rewritten,

$$\begin{aligned} f^{t+1} &= K(\cdot, X)\boldsymbol{\alpha}^{t+1} - \eta K(\cdot, X)(\boldsymbol{I}_n - Q)\boldsymbol{g}^t \\ &= K(\cdot, X)(\boldsymbol{\alpha}^{t+1} - \eta(\boldsymbol{I}_n - Q)\boldsymbol{g}^t). \end{aligned}$$

Replacing $g^t = K(X,X)\boldsymbol{\alpha}^t - y$ leads to final update rule below,

$$f^{t+1} = K(\cdot, X)(\boldsymbol{\alpha}^{t+1} - \eta(\boldsymbol{I}_n - Q)(K(X,X)\boldsymbol{\alpha}^t - y)).$$

This concludes the proof. □

Thus each update constitutes a *stochastic gradient step* which consists updating m weights corresponding to a minibatch size m, followed by a *gradient correction* which consists of updating all n weights.

A higher preconditioner level q also allows for a higher optimal batch size m and hence better GPU utilization, see Ma et al. (2018) for details.

With this approximation, the gradient correction simplifies drastically, and only s weights need to be updated.

D. Details on experiments and implementation of Algorithm 2

D.1. Computational resources used

This work used the Extreme Science and Engineering Discovery Environment (XSEDE) (Towns et al., 2014). We used machines with 2x NVIDIA-V100 and 8x NVIDIA-A100 GPUs, with a V-RAM of 32GB and 40GB respectively, and 8x cores of Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz with a RAM of 100 GB.

D.2. Figure 1 experiment

We used Laplacian Kernel and (`sklearn.linear_model.Ridge`) solver from the Scikit-learn library Pedregosa et al. (2011) to solve the optimization problem $\|K(X,Z)\alpha - y\|^2 + \lambda \cdot \|\alpha\|^2$ for extracted features of ImageNet, using a pre-trained MobileNetv2 model obtained from the *timm* library Wightman (2019).

D.3. Datasets

We perform experiments on these datasets: (1) CIFAR10, Krizhevsky et al. (2009), (2) CIFAR5M, Nakkiran et al. (2021), (3) ImageNet, Deng et al. (2009), (4) MNIST, LeCun (1998), (5) MNIST8M, Loosli et al. (2007), (6) Fashion-MNIST, Baldi et al. (2014), (7) Webvision.Li et al. (2017), and (8) librispeech.

CIFAR5M. In our experiments, we utilized both raw and embedded features from the CIFAR5M data-set. The embedded features were extracted using a MobileNetv2 model pre-trained on the ImageNet data-set, obtained from *timm* library Wightman (2019). We indicate in our results when pre-trained features were used by adding an asterisk (*) to the corresponding entries.

ImageNet. In our experiments, we utilized embedded features from the ImageNet data-set. The embedded features were extracted using a MobileNetv2 model pre-trained on the ImageNet dataset, obtained from *timm* library Wightman (2019). We indicate in our results when pre-trained features were used by adding an asterisk (*) to the corresponding entries.

Webvision. In our experiments, we utilized embedded features from the Webvision data-set. The embedded features were extracted using a ResNet-18 model pre-trained on the ImageNet dataset, obtained from *timm* library Wightman (2019). Webvision data set contains 16M images in 5k classes. However, we only considered the first 2k classes.

Librispeech. Librispeech Panayotov et al. (2015) is a large-scale (1000 hours in total) corpus of 16 kHz English speech derived from audio books. We choose the subset train-clean-100 and train-clean-300 (5M samples) as our training data,

[*] $\mathsf{bs}_{\mathrm{gpu}}$ is the maximum batch-size that the GPU allows.

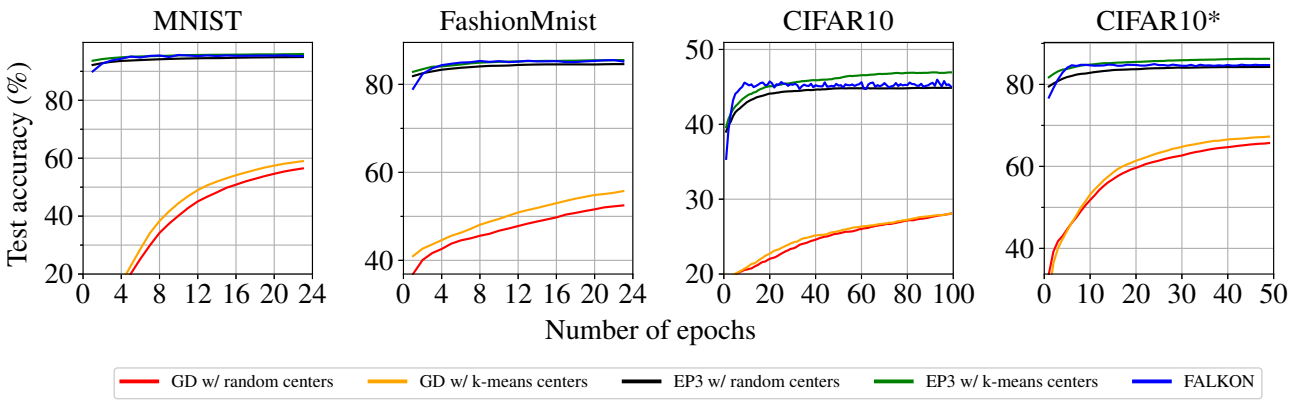

Figure 7: **(Large scale training.)** This figure shows the slow convergence of gradient descent given in (46) compared to our algorithm and FALKON from Rudi et al. (2017). Note that FALKON involves a matrix inverse for a projection operation and hence converges faster with respect to the number of epochs.

test-clean as our test set. The features are got by passing through a well-trained acoustic model (a VGG+BLSTM architecture in Hui & Belkin (2021)) to align the length of audio and text. It is doing a 301-wise classification task where different class represents different uni-gram Jurafsky (2000). The implementation of extracting features is based on the ESPnet toolkit Watanabe et al. (2018).

D.4. Choice of hyperparameters

We choose hyperparameters to minimize computation and maximize GPU utilization. The only hyperparameters that we need to set are s, q for outer gradient step, and σ, ξ for projection sub-problem. For σ, ξ, we used the same criteria as Ma & Belkin (2019) to optimally use GPU utilization. For s, q, we prefer larger q because as it is explained in Ma et al. (2018), larger q allows for larger learning rate and better condition number. However, in our algorithm we need to approximate the top q eigensystem of Nyström sub-samples matrix. We used Scipy Virtanen et al. (2020) library to approximate these eigensystem. The stability and precision of these approximations depends on how large is the ratio of $\frac{s}{q}$. Empirically we need this ratio to be larger than 10. On the other hand increasing s will increase setup cost, computation cost and memory cost. We take steps below to choose q and s,

1. We first choose s as big as our GPU memory allow
2. We choose $q \approx \frac{s}{10}$
3. We set batch size and learning rate automatically using the *new* top eigenvalue as it is explained in Ma & Belkin (2019) and Ma et al. (2018).

E. Classical approach to learning kernel models with GD

If you plug in the form of general kernel models into (2), we get

$$\underset{\boldsymbol{\alpha}}{\text{minimize}}\ L(\boldsymbol{\alpha}) = \sum_{i=1}^{n} L(\sum_{j=1}^{p} K(\boldsymbol{x}_i, \mathbf{z}_j)\alpha_j, y_i) + \lambda \left\langle \sum_{j=1}^{p} K(\cdot, \mathbf{z}_j), \sum_{j=1}^{p} K(\cdot, \mathbf{z}_j) \right\rangle_{\mathcal{H}} \tag{43}$$

$$= \sum_{i=1}^{n} L(K(X_i, Z)\boldsymbol{\alpha}, y_i) + \lambda \boldsymbol{\alpha}^\top K(Z, Z)\boldsymbol{\alpha}. \tag{44}$$

For the square loss this is equivalent to

$$\underset{\boldsymbol{\alpha}}{\text{minimize}}\ \|K(X, Z)\boldsymbol{\alpha} - \boldsymbol{y}\|^2 + \lambda \boldsymbol{\alpha}^\top K(Z, Z)\boldsymbol{\alpha}. \tag{45}$$

Gradient descent on this problem for the square loss yields the update equation,

$$\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t - \eta K(Z, X)((K(X, Z)\boldsymbol{\alpha}^t - \boldsymbol{y}) - \eta\lambda K(Z, Z)\boldsymbol{\alpha}. \tag{46}$$