Approximate Architecture Evaluation with Accelerated Kernel Methods

Blake Bordelon
School of Engineering and Applied Science
Harvard University
Cambridge MA
blake_bordelon@g.harvard.edu

Jack Swisher
School of Engineering and Applied Science
Harvard University
Cambridge, MA
jswisher@college.harvard.edu

Xin Xu
School of Engineering and Applied Science
Harvard University
Cambridge, MA
xinxu@hsph.harvard.edu

***Abstract*—Kernel methods are powerful supervised learning algorithms amenable to theoretical analysis. We aim to exploit the equivalence of training wide neural networks and kernel machines to accelerate evaluation of different neural architectures. Unfortunately, the memory and compute requirements for large datasets render naive kernel algorithms intractable. We instead leverage stochastic optimization procedures with preconditioning to solve the kernel learning task efficiently in a manner that scales to larger datasets on GPUs. We apply our method to predict test risk of fully connected neural architectures of varying depth and show that the architecture evaluation is robust to choice of the preconditioning hyperparameter. Kernel regression solutions corresponding to fully connected neural architectures with ReLU nonlinearity are obtained in under a minute for CIFAR-100 on a GPU, giving a significant speedup over training the neural network directly with gradient descent. With these methods, we are able to achieve a 10% classification accuracy improvement over the previous state of the art neural tangent kernel result on CIFAR-10.**

***Index Terms*—neural tangent kernel, stochastic optimization, architecture search**

I. Introduction

Deep learning has revolutionized approaches to computer vision, natural language processing, and reinforcement learning. In many applications of artificial neural networks, choosing the right neural architecture for a given task can provide a significant performance advantage. Auto-ML and neural architecture search aim to automate the discovery of architectures in a task dependent manner. Efficient search demands accelerated architecture evaluation, which is the primary bottleneck in identifying the best neural architecture for a task.

II. Problem to Solve

A. Accelerating Architecture Search

Our goal is to speed up architecture evaluation by making use of an approximate correspondence between large neural networks and kernel regression algorithms. By efficiently solving a kernel regression problem, we can obtain an approximation to the generalization performance of a finite size neural network with the same architecture.

III. Background Motivation and Prior Work

A. Motivation

Kernel machines are powerful supervised learning algorithms that optimize a statistical loss over functions in a Hilbert space $\mathcal{H}$. Kernel regression problems are convex and can be thought of as infinite dimensional linear models, allowing extensive theoretical analysis. As we will discuss, they are powerful enough to fit a dataset of any size.

Unfortunately, the power of these methods comes at a cost. Naive applications of kernel methods scale poorly in memory and compute as the dataset size grows. Making kernel methods competitive for architecture evaluation requires identifying ways to reduce their compute and memory footprint.

B. Background: Neural Networks and Kernel Methods

Recently, an equivalence was established between training wide neural network models with gradient descent and performing kernel regression over a reproducing kernel Hilbert space for a particular kernel known as the Neural Tangent Kernel (NTK) [2]. Though the relation is exact only in the infinite width limit, large but finite width neural networks are very well approximated by NTK. This correspondence motivates exploration of ways to improve the scaling of kernel machines to study how neural networks of different architectures generalize.

C. Prior Work

Prior work established the connection between training wide networks with gradient descent and performing kernel regression with the NTK [3] [5]. The kernel eigenspectrum of NTK has been used to successfully explain generalization and training dynamics of deep NNs.

Scaling kernel methods to large datasets has also been a topic of recent interest. Eigenpro iteration and Falkon have been used to successfully train Gaussian and Laplace RBF kernels on many datasets including Imagenet with its $\approx 10^6$ training points [2] [4]. These methods rely on stochastic gradient descent (SGD) with preconditioning as we will discuss below. Both methods focus exclusively on mean square error cost function, however, which is ill-suited for classification problems. We will examine how to extend the preconditioning technique to the cross-entropy loss.

IV. Project Goals and Evaluation Metric

Our goals are the following

- Train kernel regression algorithms on CIFAR-10 and CIFAR-100 in time that is competitive with training real neural networks for a variety of depths using the same computational resource (Google Colab GPU).
- Obtain state of the art performance for kernel methods on these datasets by leveraging stochastic optimization procedures, preconditioning, and a cross-entropy objective.
- Evaluate the correspondence between classification error of the NTK model and finite width neural networks of the same depth.

V. Proposed approach, novelty and secret weapon

Our proposed approach includes training NTKs of varying depth, using the preconditioned minibatch SGD approach described in subsequent sections. Rather than using the traditional mean-square error loss, we extend the preconditioning technique to the logistic loss SGD updates which is more appropriate for classification. We will compare the resulting test risks for architectures of various depths to finite size neural networks trained on the same datasets.

VI. Intellectual Points

Kernel methods are powerful and theoretically well understood but historically have problems scaling to large datasets. Due to the NTK correspondence, they are particularly useful for understanding deep over-paramterized networks [5]. There are many theories about kernel regression that could be tested if the algorithms were made more efficient. Both training speed and generalization error depend crucially on the kernel eigenspectrum which is data and architecture dependent. Improving the speeed and efficiency of obtaining kernel regression solutions would allow us to better understand the interplay between these components of the supervised learning problem.

VII. Work Performed

A. Background: Kernel Regression Definition

Let $\mathcal{H}$ be a Reproducing Kernel Hilbert Space (RKHS) with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and let $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^p$ be a dataset with inputs $\mathbf{x}_i \in \mathbb{R}^d$ and targets $\mathbf{y}_i \in \mathbb{R}^c$. Kernel regression is defined by the solution to a supervised learning problem

$$\min_{f \in \mathcal{H}} \sum_{i=1}^p \ell(f(\mathbf{x}_i), \mathbf{y}_i) + \lambda \langle f, f \rangle_{\mathcal{H}} \tag{1}$$

for a convex loss $\ell(\cdot, \cdot)$. For concreteness our paper will only focus on least squares and cross entropy losses ($\ell(\mathbf{u}, \mathbf{v}) = ||\mathbf{u} - \mathbf{v}||_2^2$ and $\ell(\mathbf{u}, \mathbf{v}) = -\sum_{j=1}^c v_j \log \left(\frac{\exp(u_j)}{\sum_{j'} \exp(u_{j'})} \right)$ respectively). This learning problem has two parts: the first is a statistical objective that promotes goodness of fit whereas the second is a Hilbert norm on f that controls the complexity of the learned function. The $\lambda \to 0$ limit is referred to as kernel interpolation and is of great interest due to its connection to neural networks. In this limit, the optimization problem chooses the function with minimum norm that gives zero training error.

Though this problem appears to involve optimization over the infinite dimensional space $\mathcal{H}$, the problem is simplified by a fact known as the Representer Theorem. Let $K(\cdot, \cdot)$ be the Reproducing Kernel of $\mathcal{H}$ ($\langle g, K(\mathbf{x}, \cdot) \rangle_{\mathcal{H}} = g(\mathbf{x})$, then the solution to Eq. 1 has the form

$$f(\mathbf{x}) = \sum_{i=1}^p \alpha_i K(\mathbf{x}, \mathbf{x}_i) = \boldsymbol{\alpha}^\top \mathbf{k}(\mathbf{x}) \tag{2}$$

for some coefficients $\boldsymbol{\alpha} \in \mathbb{R}^{p \times c}$. We leave a proof of this common result to the appendix. The Representer Theorem simplifies the optimization problem in Eq. 1 to a finite dimensional problem in pc decision variables $\boldsymbol{\alpha}$. For least squares the solution has a simple form

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{Y}. \tag{3}$$

B. Computational and Memory Complexity of Naive Solution

The solution to Eq. 3 requires computing, storing and inverting the kernel Gram matrix $\mathbf{K} \in \mathbb{R}^{p \times p}$. The memory cost for 32-bit precision is $4p^2$ bytes while the cost of inversion is $\mathcal{O}(p^3)$. For even small datasets like MNIST and CIFAR-10, this requires about 10 Gigabytes of RAM. The time for inversion quickly become infeasible for large datasets. For Imagenet with $p \approx 10^6$ images the inversion time would require $\mathcal{O}(10^{18})$ operations and the amount of memory required would be a few terabytes. For this reason, the naive solution is computationally infeasible for the majority of deep learning tasks.

C. Full Batch Gradient Descent Updates

Solving a linear system with gradient descent could in principle provide an improvement in time complexity of the solution [1]. Rather than algebraically inverting the Gram matrix $\mathbf{K}$, we can iteratively update $\boldsymbol{\alpha}$ so that it converges to the desired solution. For $\lambda = 0$, we can use the simple update rule to solve $\mathbf{K}\boldsymbol{\alpha} = \mathbf{Y}$,

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} - \eta(\mathbf{K}\boldsymbol{\alpha}^{(t)} - \mathbf{Y}). \tag{4}$$

If this update is performed for T iterations, the time cost is now $\mathcal{O}(p^2 T)$ although the memory requirement is still $\mathcal{O}(p^2)$. The empirical question of how large T must be to provide a decent approximation to the true solution still remains. We note that the errors $\boldsymbol{r} = \boldsymbol{\alpha} - \mathbf{K}^{-1}\mathbf{Y}$ converge to zero exponentially,

$$\boldsymbol{r}^{(t)} = (\mathbf{I} - \eta \mathbf{K})^t \mathbf{Y} = \sum_k (1 - \eta \lambda_k)^t \boldsymbol{e}_k \boldsymbol{e}_k^\top \mathbf{Y}, \tag{5}$$

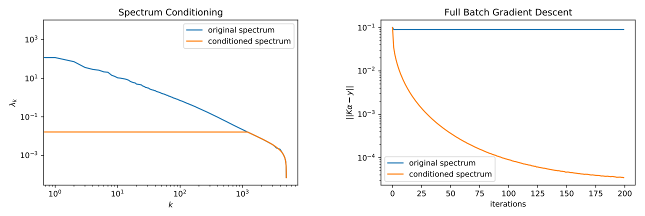

Fig. 1. Original and conditioned spectrum for full batch gradient descent on a 5000 data point subset of CIFAR-10. The original spectrum has a large range over 4 decades making convergence along the non-dominant eigendirections intractable. Preconditioning the spectrum so that the first $k = 1000$ eigenvalues are equal gives faster convergence.

provided that $\eta < 2/\lambda_1(\mathbf{K})$ assuming that the eigenvalues are ordered $\lambda_1 > \lambda_2 > ... > \lambda_p$. The convergence along each eigenvector $\boldsymbol{e}_k$ has a different exponential constant $(1 - \eta\lambda_k)$ which determines the number of iterations necessary to learn that mode. Therefore, the decay of the spectrum λ_k impacts the separation of timescales to learn different eigencomponents of the target data $\mathbf{Y}$.

An example spectrum for 2 layer ReLU NTK on a subset of CIFAR-10 is provided in Figure 1. Eigenvalues range over 4 decades, indicating a large range of time constants.

D. Preconditioning with Eigenpro

Two works have examined the acceleration of Kernel Gradient descent due to preconditioning [2] [4] but both focus on least squares loss rather than classification. The analysis begins by noting that multiplying both sides of a linear equation by an invertible matrix $\mathbf{P}$ does not change the solution.

$$\mathbf{PK}\boldsymbol{\alpha} = \mathbf{PY} \tag{6}$$

If we use Richardson iteration to solve this problem, we have an update rule of the form

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} - \eta\mathbf{P}(\mathbf{K}\boldsymbol{\alpha}^{(t)} - \mathbf{Y}). \tag{7}$$

Following Ma and Belkin [2], we choose $\mathbf{P}$ to set the first k eigenvalues to λ_k as shown in Figure 1. This allows us to choose a learning rate $\eta < 2/\lambda_k(\mathbf{K})$ giving an acceleration on the order of $\frac{\lambda_1}{\lambda_k}$. For the example in Figure 1, this corresponds to an acceleration on the order of 10^4.

E. Our Novel Extension to Cross Entropy Loss

We worked out the gradient descent update for a cross entropy loss as

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} - \eta\mathbf{K}(\sigma(\mathbf{K}\boldsymbol{\alpha}^{(t)}) - \mathbf{Y}), \tag{8}$$

where σ is the soft max function: $\sigma(\mathbf{v})_i = \dfrac{\exp(v_i)}{\sum_j \exp(v_j)}$. We note that though the fixed point equations for $\boldsymbol{\alpha}$ are nonlinear: $\mathbf{K}\sigma(\mathbf{K}\boldsymbol{\alpha}^*) = \mathbf{KY}$, they are unchanged by multiplying both sides of the equation by $\mathbf{P}$ as before. This gives a preconditioned gradient descent update $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} - \eta\mathbf{PK}(\sigma(\mathbf{K}\boldsymbol{\alpha}^{(t)}) - \mathbf{Y})$. To our knowledge, this is the first work examining preconditioning for cross-entropy GD updates for kernel regression.

	Mean Square Error	Cross Entropy
FC NTK	84%	**87%**
CNTK [5]	77%	-
Gauss RBF [6]	84%	-
Conv. NNGP [7]	63%	-

TABLE I
KERNEL REGRESSION CLASSIFICATION ACCURACY ON CIFAR-10.

F. Minibatch SGD

To reduce the memory costs and further accelerate convergence of gradient descent, we use minibatches to perform coordinate descent on $\boldsymbol{\alpha}$. This dramatically reduces the memory requirements and allows further data parallelism on the GPU. At each iteration, we sample a sub-block of s data points $\mathbf{r} \subset [p]^s$ and compute a s-row sub-block of the kernel matrix $\mathbf{K_r} \in \mathbb{R}^{s \times p}$ update of the form

$$\boldsymbol{\alpha}_\mathbf{r} \leftarrow \boldsymbol{\alpha}_\mathbf{r} - \eta(\mathbf{K_r}\boldsymbol{\alpha} - \mathbf{Y_r}) \tag{9}$$

$$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \mathbf{PK}_\mathbf{r}^\top(\mathbf{K_r}\boldsymbol{\alpha} - \mathbf{Y_r}) \tag{10}$$

for a least squares loss. Previous work [2] established that performing these updates at each iteration is a stochastic version of preconditioned stochastic coordinate descent on $\boldsymbol{\alpha}$. Memory and compute at each iteration is now reduced to $\mathcal{O}(ps + pk)$. We use an analogous minibatch update for the cross entropy loss.

VIII. Results

A. Full Batch GD

We first evaluate the time required to learn NTK regression solution with ℓ_2 loss with full batch gradient updates. Figure 1 shows a spectrum of a random 5000 data point subsample of CIFAR-10. A comparison of the training error dynamics for conditioned ($k = 1000$) and original spectra shows the dramatic improvement due to preconditioning in Figure 1.

B. Minibatch SGD on CIFAR-10(0)

We next use the cross-entropy loss to train NTK models of varying depth on CIFAR-10 and CIFAR-100. For these experiments we chose $s = 5000$ and used different preconditioners to verify the robustness of our results $k = 350, 500, 700$. We then train to 99% training accuracy ReLU NTKs of varying depths and calculate their performance on the test set. We record both the test classification error rate and the time required for convergence. Results are provided in Figure 4 for CIFAR-10 and Figure 5 for CIFAR-10 and CIFAR-100 respectively. We find that for CIFAR-10 there is a locally optimal depth around 5 layers but the best performance was attained for the depth-200 kernel. We also find that there is a depth that minimizes the time since there is a tradeoff between number of gradient steps and time spent computing kernel elements. For CIFAR-100, depth 20 NTK performed the best, both in terms of accuracy and training time.

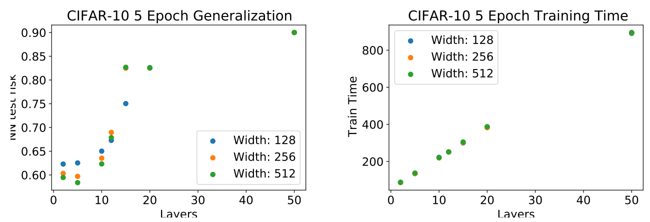

Fig. 2. CIFAR-10. Generalization error and training time for Finite Neural Network on CIFAR-10.

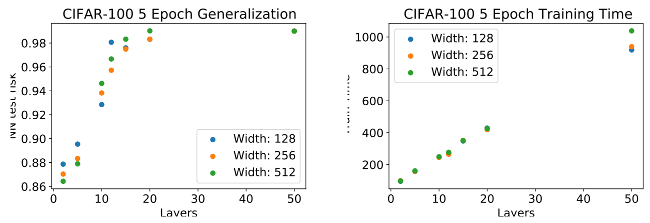

Fig. 3. CIFAR-100. Generalization error and training time for Finite Neural Network on CIFAR-100.

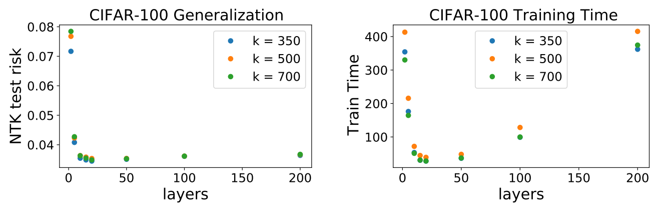

Fig. 5. Generalization error and Training time for Kernel Logistic Regression on CIFAR-100. The depth 20 ReLU NTK performed the best both in terms of training time and test accuracy.

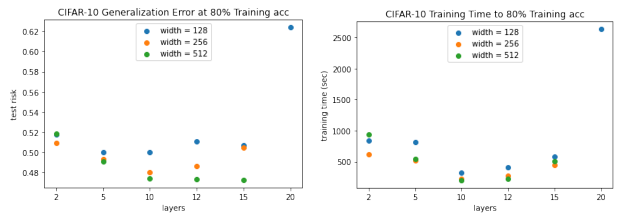

Fig. 6. Generalization error and Training time for Finite Neural Network on CIFAR-10. Fully connected neural networks are trained to 80% on the training data.

C. Finite Neural Networks Compared to NTK

When finite neural networks were trained for similar time to NTKs (5 epochs), the generalization error did not directly match the NTK predictions as seen in Figure 3 and Figure 4. On CIFAR-10, the best performing finite NN had depth 5, matching the local minimum of the NTK models, however the depth 50 finite NN did not perform well unlike the NTK model. This is likely a result of the few epochs employed in training. On CIFAR-100, this training deficiency is even more evident as the performance degrades as depth increases. In this case, no useful deductions can be made regarding the optimal depth when training a finite NN for few epochs.

When training the finite NNs to 80% accuracy on the training set, we observe in Figure 7 that on CIFAR-10 the generalization error of finite NNs does not match exactly with the generalization error of NTK models in Figure 5. However, in both figures we see that deeper networks achieve better performance. In Figure 8, we notice that depth 5 fully connected networks performed best on average across the various widths, while NTKs of depth 20 performed best in Figure 6. However, in both of these cases we expect longer training of the neural networks to cause the generalization curves to become even more similar.

Overall, these results suggest that the training dynamics of NTKs do not match exactly with finite NNs, even when trained to high training accuracy. However, they are able to provide a good approximation for how performance evolves over depth for NNs and can serve to guide architecture selection. Though the kernel method has an optimal depth for time complexity, time for finite NNs scale linearly with depth when trained for a fixed number of epochs. As a result, we observed that NTKs trained to convergence much faster than finite neural networks, especially for deeper networks, making them well suited for performance estimation, at least as a first pass to narrow the set of architectures to be later evaluated using grid search.

IX. Conclusion

Leveraging conditioned stochastic optimization can provide parallelism and speed improvements to kernel methods, making them competitive with neural networks both in terms of training time and test accuracy. Further, our use of SGD on the crossentropy loss allowed us improve state of the art

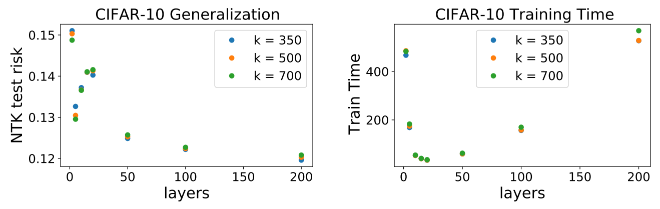

Fig. 4. CIFAR-10. Results are robust to different choices of preconditioning hyperparameter k. The best architecture obtains a roughly 10% improvement over the best stated kernel regression solution for CIFAR-10 [5]. We suspect this is a consequence of training with the cross entropy loss.

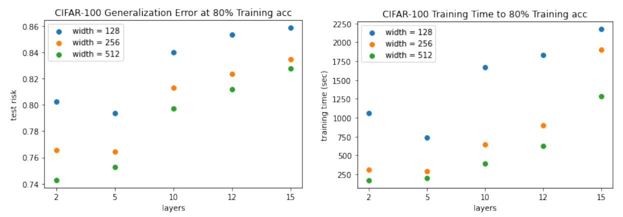

Fig. 7. Generalization error and Training time for Finite Neural Network on CIFAR-100. Fully connected neural networks are trained to 80% on the training data.

kernel classification accuracy by 10%. We were impressed that kernel methods could actually be extended to realistic datasets (CIFAR-10 and CIFAR-100) given the failures of the naive solution method. Further work could extend this method to even larger datasets (Imagenet etc) to evaluate the effect of depth on generalization. Further, we would like to find ways to accererate computation of convolutional NTK so that we can evaluate how shift invariance influences the spectrum.

References

[1] L. F. Richardson, "The Approximate Arithmetical Solution by Finite Differences of Physical Problems Involving Differential Equations, with an Application to the Stresses in a Masonry Dam," Phil. Trans. Roy. Soc. London, vol. 210, pp. 307-357, 1911.

[2] S. Ma Siyuan and M. Belkin, Diving into the shallows: a computational perspective on large-scale shallow learning,Advances in Neural Information Processing Systems 30, 2017, pp.3778–3787.

[3] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J.S. Dickstein, J. Pennington, "Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent," Arxiv, 2019.

[4] A. Rudi, L. Carratino and L. Rosasco, "FALKON: An Optimal Large Scale Kernel Method," Arxiv 2017.

[5] S. Arora, S. Du, W. Hu, Z. Li, R. Salakhutdinov, R. Wang, "On Exact Computation with an Infinitely Wide Neural Net," Arxiv 2019.

[6] S. Tu, R. Roelofs, S. Venkataraman, B. Recht, "Large Scale Kernel Learning Using Block Coordinate Descent," Arxiv 2016.

[7] R. Novak, L. Xiao, Y. Bahri, J. Lee, G. Yang, D. Abolaia, J. Pennington, J. Sohl-Dickstein, "Bayesian Deep Convolutional Networks with Many Channels are Gaussian Processes," International Conference on Learning Representations 2019.