Simple, Practical, and Fast Dynamic Truncation Kernel Multiplication

Lianke Qin [1] **Somdeb Sarkhel** [2] **Zhao Song** [2] **Danyang Zhuo** [3]

Abstract

Computing the product of a kernel matrix and a vector is the most basic and important operation in high-performance machine learning and scientific computing. The speed for this calculation determines plays a critical role in the overall performance of machine learning training and inference. As dataset sizes rapidly increase, the dimension of the kernel matrix also increase accordingly, and this product computation is increasingly a performance bottleneck. In the meantime, our observation is that many popular kernel matrices are inherently sparse, due to natural data distributions. In this paper, we design an efficient data structure to approximate kernel matrix vector multiplication. Our data structure is a search tree which enables us to quickly extract those entries and calculate the multiplication results.

1. Introduction

Kernel method is an important class of machine learning techniques. It is widely used in classification (Elisseeff & Weston, 2001; Kashima et al., 2003; Weston et al., 2003; Rousu et al., 2006; Hoi et al., 2006), deep neural networks (Cho & Saul, 2009; Wilson et al., 2016b;a; Belkin et al., 2018), and computer vision (Tuzel et al., 2009; Jayasumana et al., 2013; Fang et al., 2021). When using the kernel method, we often need to compute the product of a kernel matrix and a vector. The speed for this computation determines the overall performance of the higher level machine learning and scientific computing tasks. The previous work on the fast multiple method (Greengard & Rokhlin, 1987; Greengard, 1988; Greengard & Rokhlin, 1988) designs algorithms with a $(\log(n/\epsilon))^{O(d)} n^{1+o(1)}$ running time complexity for ϵ-approximate matrix-vector multiplication for a number of kernel functions K, including when $\mathrm{K}(x,y) = \frac{1}{\|x-y\|_2^c}$ for a constant c and when $\mathrm{K}(x,y) = e^{-\|x-y\|_2^2}$.

The dimension of the kernel matrix is the same as the size of the dataset. Modern dataset has increasing numbers of samples. This makes the calculation of a kernel matrix and a vector increasingly slow. In the meantime, our observation is many data distributions naturally lead to sparse kernel matrices. For example, when data are clustered around several locations, the entry in the kernel matrix is non-negligible if only if the corresponding data points are in the same cluster. This raises an important question:

Can we speed up the kernel matrix vector multiplication time for sparse kernel matrices?

We have an affirmative answer. Our approach uses an efficient search tree data structure design. The search tree enables us to quickly locate non-negligible entries in a large kernel matrix. During the multiplication, we simply omit negligible entries to speed up the matrix vector computation. This approach enables us to compute the kernel vector multiplication efficiently when the kernel entries are highly sparse. This truncated matrix vector multiplication is motivated by activation functions like ReLU which sets the output as zero if the corresponding input values do not surpass certain threshold.

Our experiments show that our truncation kernel multiplication can bring up to $8.33\times$ speedup while maintaining $< 10\%$ L1 norm error rate in many kernels compared with the non-truncation kernel multiplication. Our paper makes the following contributions:

- We design a search tree based data structure to support fast and dynamic truncation kernel-vector multiplication with sublinear time complexity.
- We use experiments to quantify what type of kernels our approach works and show the tradeoff between the speedup improvement and accuracy drop of truncated matrix multiplication.

[*]Equal contribution [1]University of California, Santa Barbara [2]Adobe Research [3]Duke University. Correspondence to: Lianke Qin <lianke@ucsb.edu>, Somdeb Sarkhel <sarkhel@adobe.com>, Zhao Song <zsong@adobe.com>, Danyang Zhuo <danyang@cs.duke.edu>.

DyNN workshop at the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

2. Related Work

Kernel Methods When it comes to instance-based learners, kernel techniques are very similar. The i-th training data point (x_i, y_i) is remembered and a weight parameter w_i is learned for the training data, rather than learning a predetermined set of parameters corresponding to the features of their inputs, as is the case with traditional approaches. With the training inputs x_i and the unlabeled data point x', we can make educated guesses. There have been some new discoveries in the field of deep neural networks and kernels. (Daniely et al., 2016; Daniely, 2017; Chizat & Bach, 2018; Jacot et al., 2018; Brand et al., 2021; Song et al., 2021). Moreover, our work on truncated kernel multiplication is also relevant to the kernel regression problem. (Alaoui & Mahoney, 2015; Zhang et al., 2015; Avron et al., 2017; Zandieh et al., 2020).

Gaussian Kernels Gaussian kernels have been used in many machine learning problems (Wang et al., 2009; Li et al., 2019; Wenliang et al., 2019; Liao et al., 2020). Dziugaite et al. (Dziugaite et al., 2015) uses gaussian kernel to improve the generalization error during learning generative models from i.i.d. data with unknown distribution. Gaussian kernel is also used within Generative moment matching network (GMMN) to improve both the model expressiveness of GMMN and its computational efficiency (Li et al., 2017).

Kernel Matrix Vector Multiplication The prior work on the fast multiple method (Greengard & Rokhlin, 1987; Greengard, 1988; Greengard & Rokhlin, 1988) designs algorithms with time complexity of $(\log(n/\epsilon))^{O(d)} n^{1+o(1)}$ for ϵ-approximate adjacency matrix-vector multiplication for a number of kernel functions K, including when $\mathrm{K}(x,y) = \frac{1}{\|x-y\|_2^c}$ for a constant c and when $\mathrm{K}(x,y) = e^{-\|x-y\|_2^2}$. On the negative side, (Alman et al., 2020) proved the hardness result under exponential time hypothesis. (Huang et al., 2022) shows how to dynamically maintain the answers when data points are changing over the time. There are also many hardware accelerated sparse matrix multiplication work (Yang et al., 2018; Huang et al., 2020; Jain et al., 2020; Xie et al., 2021; Dai et al., 2022).

Inner Product Search Under the assumption that all the points are on unit sphere, then the ℓ_2 distance between x and y is equivalent to inner product between x and y. Recently, computing inner product exactly or approximately has been applied to many machine learning applications such as discrepancy (Song et al., 2022b), tree-width LP problems (Ye, 2021; Dong et al., 2021), sparsification (Song et al., 2022b), frank-wolfe method (Shrivastava et al., 2021a), reinforcement learning (Shrivastava et al., 2021b), and Fourier-signal interpolation (Song et al., 2022a).

3. Our Data Structures

We present a new data-structure for efficiently computing the truncated matrix vector multiplication in this section. We delay the algorithm description to Section A due to space reason.

3.1. Definitions

We define the kernel matrix in Definition 3.1.

Definition 3.1. *Given $x_1, \cdots, x_n \subset \mathbb{R}^d$ and $y_1, \cdots, y_m \subset \mathbb{R}^d$. We define matrix $\mathsf{K} \in \mathbb{R}^{n \times m}$ as follows:*

$$\mathsf{K}_{i,j} := f(x_i, y_j).$$

Then we define the truncated matrix vector multiplication in Definition 3.2.

Definition 3.2 (Truncated Matrix Vector Multiplication)**.** *Let K be defined as definition 3.1, for any query vector $v \in \mathbb{R}^m$ and a truncation threshold τ, the goal is to compute*

$$\sum_{j=1}^{m} v_j \cdot \mathsf{K}_{i,j} \cdot \mathbf{1}_{\langle x_i, y_j \rangle \geq \tau}, \forall i \in [n]$$

3.2. Our Theorem

In this section, we present our main theorem in Theorem 3.3 and prove it with individual lemmas for each operation of our data structure.

Theorem 3.3 (Main Theorem)**.** *Assume the time complexity of evaluating $f(x,y)$ is $\mathcal{T}_f$. There exists a data structure which uses $O(mn + d(m+n))$ spaces and supports the following operations:*

- INIT$(y_1, y_2, \cdots, y_m \in \mathbb{R}^d, x_1, x_2, \cdots, x_n \in \mathbb{R}^d)$. *Given $y_1, y_2, \cdots, y_m \in \mathbb{R}^d$ and n data points $x_1, x_2, \cdots, x_n \in \mathbb{R}^d$, the time complexity of* INIT *operation is $O(mn(d + \mathcal{T}_f))$.*
- UPDATE$(z \in \mathbb{R}^d, j \in [m])$. *Given $z \in \mathbb{R}^d$ and an index $j \in [m]$, the* UPDATE *operation runs in $O(n(d + \log(m) + \mathcal{T}_f))$ time.*
- QUERY$(i \in [n], \tau \in \mathbb{R})$. *Given an index $i \in [n]$ and a threshold $\tau \in \mathbb{R}$ as input and let K_i denote the number of entries of above τ in tree T_i, the* QUERY *operation runs in $O(K_i \log(m))$ time and output a set containing all y_j such that $\langle x_i, y_j \rangle \geq \tau$ in tree T_i.*
- MULTIPLY$(v \in \mathbb{R}^m, \tau \in \mathbb{R})$, *Given a vector $v \in \mathbb{R}^d$ and a threshold $\tau \in \mathbb{R}$ as input, and let K_i denote the number of entries of above τ in each tree T_i,* MULTIPLY *outputs the result of truncated matrix vector multiplication $\mathsf{K} \cdot v$ in $O(\sum_{i=1}^n K_i \log(m))$ time.*

Proof. By combining Lemma 3.4, Lemma 3.5 and Lemma 3.7, we can prove the correctness of QUERYSUB, QUERY and MULTIPLY operations in Theorem 3.3. □

3.3. Correctness of Query

We first need to prove the correctness of QUERYSUB in Lemma 3.4 and delay the proof to Section B.1.

Lemma 3.4 (Correctness of QUERYSUB)**.** *Given a threshold $\tau \in \mathbb{R}$ and a node $r \in T$, let $i \in [n]$ denote the index of the tree which contains node r as input, and let K_i denote the number of entries of above τ in tree T_i, the* QUERYSUB *operation outputs a set containing all $y_j \in \mathbb{R}^d$ such that $\langle x_i, y_j \rangle \geq \tau$ in the subtree whose root is r.*

With the above lemma, we can prove the correctness of QUERY in Lemma 3.5 an delay the proof to Section B.1.

Lemma 3.5 (Correctness of QUERY)**.** *Given an index $i \in [n]$ and a threshold $\tau \in \mathbb{R}$ as input, the* QUERY *outputs a set containing all y_j such that $\langle x_i, y_j \rangle \geq \tau$ in tree T_i.*

3.4. Correctness of Multiplication

In this section, we first prove the correctness of MULTIPLYSUB in Lemma 3.6 and delay the proof to Section B.2.

Lemma 3.6 (Correctness of MULTIPLYSUB)**.** *Given an index i, a vector $v \in \mathbb{R}^d$ and a threshold $\tau \in \mathbb{R}$ as input, and let K_i denote the number of entries of above τ in tree T_i,* MULTIPLYSUB *outputs the result of truncated vector inner product $\mathsf{K}_i \cdot v$.*

With the above lemma, we can prove the correctness of MULTIPLY in Lemma 3.7 and delay the proof to Section B.2.

Lemma 3.7 (Correctness of MULTIPLY)**.** *Given a vector $v \in \mathbb{R}^d$ and a threshold $\tau \in \mathbb{R}$ as input, and let K_i denote the number of entries of above τ in each tree T_i,* MULTIPLY *outputs the result of truncated matrix vector multiplication $\mathsf{K} \cdot v$.*

3.5. Running Time

In this section, we want to prove the time complexity of the operations in our data structure. We first prove the time complexity of INIT in the following lemma and delay the proof to Section B.3.

Lemma 3.8 (Time complexity of INIT)**.** *Given $y_1, y_2, \cdots, y_m \in \mathbb{R}^d$ and n data points $x_1, x_2, \cdots, x_n \in \mathbb{R}^d$ as input,* INIT *operation runs in $O(mn(d + \mathcal{T}_f))$ time.*

We prove the time complexity of UPDATE in Lemma 3.9 and delay the proof to Section B.3.

Lemma 3.9 (Time complexity of UPDATE)**.** *Given $z \in \mathbb{R}^d$ and an index $j \in [m]$ as input, the* UPDATE *operation runs in $O(n(d + \log(m) + \mathcal{T}_f))$ time.*

3.5.1. RUNNING TIME OF QUERY

In this section we prove the time complexity of QUERYSUB in Lemma 3.10 and QUERY in Lemma 3.11 respectively and delay the proof to Section B.3.

Lemma 3.10 (Time complexity of QUERYSUB)**.** *Given a threshold $\tau \in \mathbb{R}$ and a node $r \in T$ as input, let i denote the index of the tree which contains node r and let K_i denote the number of entries of above τ in tree T_i. The* QUERYSUB *operation runs in $O(\log(m))$ time.*

Lemma 3.11 (Time complexity of QUERY)**.** *Given an index $i \in [n]$ and a threshold $\tau \in \mathbb{R}$ as input and let K_i denote the number of entries of above τ in tree T_i, the* QUERY *operation runs in $O(K_i \log(m))$ time.*

3.5.2. RUNNING TIME OF MULTIPLICATION

In this section we prove the time complexity of MULTIPLYSUB in Lemma 3.12 and MULTIPLY in Lemma 3.13 respectively and delay the proof to Section B.3.

Lemma 3.12 (Time complexity of MULTIPLYSUB)**.** *Given an index i, a vector $v \in \mathbb{R}^d$ and a threshold τ as input and let K_i denote the number of entries of above τ in tree T_i, the* MULTIPLYSUB *operation output the truncated vector inner product $\mathsf{K}_i \cdot v$ in $O(K_i \log(m))$ time.*

Lemma 3.13 (Time complexity of MULTIPLY)**.** *Given a vector $v \in \mathbb{R}^d$ as input and let K_i denote the number of entries of above τ in each tree T_i, the* MULTIPLY *operation output the truncated matrix vector multiplication $\mathsf{K} \cdot v$ in $O(\sum_{i=1}^n K_i \log(m))$ time.*

4. Evaluation

Experiment setup. We evaluate our algorithm with $n = 2048$, $d = 64$ and $m = 2048$ randomly generated data points $\{x_i\}_{i=1}^n$ and $\{y_j\}_{j=1}^m$, where $\{x_i\}_{i=1}^n$ are generated by 32 random clusters and each cluster contains 64 data points. We run our simulation on an Intel i7-9700 and 64GB memory machine with Python 3.6.9 installed. We benchmark 5 different kernels including: (1) Gaussian kernel. (2) Polynomial kernel. (3) Laplacian kernel. (4) Rational kernel. (5) T-student. The code is available at https://github.com/brucechin/truncation_kernel_mul. We run MULTIPLY for 100 times to obtain the average time consumption and L1 and L2 norm error rate compared to non-truncated matrix vector multiplication per setting. We want to answer the following questions:

- **Q1:** For different truncation degree, what is the relationship between MULTIPLY running time speedup compared with truncating each kernel entry with threshold sequentially?

- **Q2:** For different kernel functions, what is the relationship between MULTIPLY accuracy compared with non-truncated kernel multiplication under different truncation percentages?

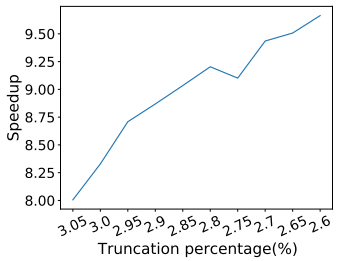


Figure 1: Speedup under different truncation percentage.

Q1. From Fig 1, we find that MULTIPLY achieves $8.33\times$ speedup compared with truncating each kernel entry with threshold sequentially. This speedup comes from efficiently locating the kernel indexes in logarithmic time for truncated kernel multiplication. As the truncation percentage decreases, the speedup grows. Because QUERY operation will early terminate for more recursive calls and yield to faster execution when the truncation percentage decreases. This speedup is independent from the kernel type.

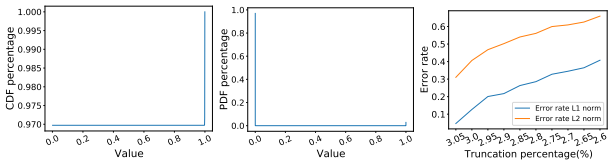
Q2. For Gaussian (Fig 2), T-student (Fig 3) and Laplacian (Fig 6) kernels, our MULTIPLY can achieve $< 10\%$ L1 norm error rate and $< 40\%$ L2 norm error rate when the truncation percentage is larger than 3.0%. For Polynomial kernel (Fig 4), our MULTIPLY can achieve $< 2\%$ L1 norm error rate and $< 20\%$ L2 norm error rate when the truncation percentage is larger than 3.0%. When the truncation percentage drops below 3.0%, the L1 and L2 error rates increase sharply as the truncation percentage decreases. This is because MULTIPLY leverages less kernel entries to compute the multiplication which yields to lower accuracy.

For Tanh(Fig 5), Multiquadratic(Fig 7) and Inverse multiquadratic(Fig 8) kernels, our MULTIPLY yields to bad multiplication accuracy compared to the non-truncated matrix-vector multiplication.

From the cumulative distribution function and probability density function figures of these kernels (From Fig 2 to Fig 9, part (a) and (b)), we find that for the kernels that yields low error rate with our truncation kernel multiplication, their probability density function turns out to be polarized and most of the kernel entries are small and can contribute little to the multiplication results. Our truncation kernel multiplication data structure can effectively filter out those negligible kernel entries and only use the important entries for the kernel multiplication to obtain speedup.

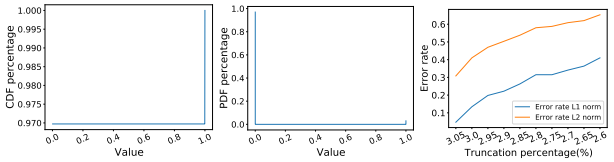
5. Conclusion

The most fundamental operation in machine learning is computing the product of a kernel matrix and a vector. The speed at which this computation is performed is critical for machine learning training and inference performance. As dataset sizes grow larger, the dimension of the kernel matrix grows larger as well, and this kernel multiplication becomes an increasingly significant performance bottleneck. Based on the observation that a large number of popular kernel matrices are inherently sparse, we design an efficient data structure for approximating kernel matrix vector multiplication. Our experiments demonstrate that when compared to baseline truncation kernel multiplication, our truncation kernel multiplication can achieve a speedup of up to $8.33\times$ while maintaining a 10% L1 norm error rate against non-truncated kernel multiplication.



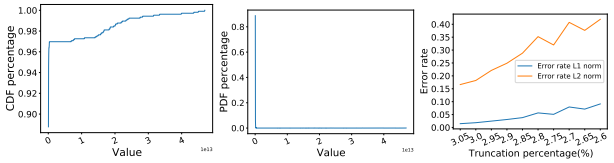
(a) (b) (c)

Figure 2: Gaussian kernel (a) CDF figure. (b) PDF figure. (c) L1 and L2 norm error rates under different truncation degree



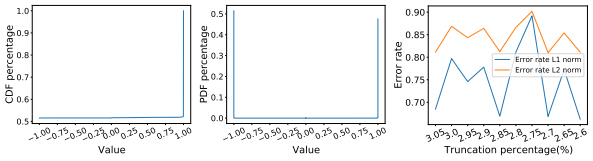
(a) (b) (c)

Figure 3: Tstudent kernel (a) CDF figure. (b) PDF figure. (c) L1 and L2 norm error rates under different truncation degree



(a) (b) (c)

Figure 4: Polynomial kernel (a) CDF figure. (b) PDF figure. (c) L1 and L2 norm error rates under different truncation degree



(a) (b) (c)

Figure 5: Tanh kernel (a) CDF figure. (b) PDF figure. (c) L1 and L2 norm error rates under different truncation degree

References

Alaoui, A. and Mahoney, M. W. Fast randomized kernel ridge regression with statistical guarantees. *Advances in neural information processing systems*, 28, 2015.

Alman, J., Chu, T., Schild, A., and Song, Z. Algorithms and hardness for linear algebra on geometric graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 541–552. IEEE, 2020.

Avron, H., Clarkson, K. L., and Woodruff, D. P. Faster kernel ridge regression using sketching and preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 38 (4):1116–1138, 2017.

Belkin, M., Ma, S., and Mandal, S. To understand deep learning we need to understand kernel learning. In *International Conference on Machine Learning*, pp. 541–549. PMLR, 2018.

Brand, J. v. d., Peng, B., Song, Z., and Weinstein, O. Training (overparametrized) neural networks in near-linear time. In *ITCS*, 2021.

Chizat, L. and Bach, F. A note on lazy training in supervised differentiable programming. *arXiv preprint arXiv:1812.07956*, 8, 2018.

Cho, Y. and Saul, L. Kernel methods for deep learning. *Advances in neural information processing systems*, 22, 2009.

Dai, G., Huang, G., Yang, S., Yu, Z., Zhang, H., Ding, Y., Xie, Y., Yang, H., and Wang, Y. Heuristic adaptability to input dynamics for spmm on gpus. *arXiv preprint arXiv:2202.08556*, 2022.

Daniely, A. Sgd learns the conjugate kernel class of the network. *Advances in Neural Information Processing Systems*, 30, 2017.

Daniely, A., Frostig, R., and Singer, Y. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. *Advances In Neural Information Processing Systems*, 29, 2016.

Dong, S., Lee, Y. T., and Ye, G. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2021.

Dziugaite, G. K., Roy, D. M., and Ghahramani, Z. Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.03906*, 2015.

Elisseeff, A. and Weston, J. A kernel method for multi-labelled classification. *Advances in neural information processing systems*, 14, 2001.

Fang, P., Harandi, M., and Petersson, L. Kernel methods in hyperbolic spaces. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10665–10674, 2021.

Greengard, L. *The rapid evaluation of potential fields in particle systems*. MIT press, 1988.

Greengard, L. and Rokhlin, V. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2): 325–348, 1987.

Greengard, L. and Rokhlin, V. On the evaluation of electrostatic interactions in molecular modeling. In *Proceedings of the Nobel Symposium on Structure and Dynamics in Biological Systems, Dec. 6-9, 1988*, 1988.

Hoi, S. C., Lyu, M. R., and Chang, E. Y. Learning the unified kernel machines for classification. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 187–196, 2006.

Huang, B., Song, Z., Weinstein, O., Zhang, H., and Zhang, R. A dynamic fast gaussian transform. *arXiv preprint arXiv:2202.12329*, 2022.

Huang, G., Dai, G., Wang, Y., and Yang, H. Ge-spmm: General-purpose sparse matrix-matrix multiplication on gpus for graph neural networks. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12. IEEE, 2020.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

Jain, A. K., Omidian, H., Fraisse, H., Benipal, M., Liu, L., and Gaitonde, D. A domain-specific architecture for accelerating sparse matrix vector multiplication on fpgas. In *2020 30th International conference on field-programmable logic and applications (FPL)*, pp. 127–132. IEEE, 2020.

Jayasumana, S., Hartley, R., Salzmann, M., Li, H., and Harandi, M. Kernel methods on the riemannian manifold of symmetric positive definite matrices. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 73–80, 2013.

Kashima, H., Tsuda, K., and Inokuchi, A. Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 321–328, 2003.

Li, C.-L., Chang, W.-C., Cheng, Y., Yang, Y., and Póczos, B. Mmd gan: Towards deeper understanding of moment matching network. *Advances in neural information processing systems*, 30, 2017.

Li, C.-L., Chang, W.-C., Mroueh, Y., Yang, Y., and Póczos, B. Implicit kernel learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2007–2016. PMLR, 2019.

Liao, Z., Couillet, R., and Mahoney, M. W. A random matrix analysis of random fourier features: beyond the gaussian kernel, a precise phase transition, and the corresponding double descent. *Advances in Neural Information Processing Systems*, 33:13939–13950, 2020.

Rousu, J., Saunders, C., Szedmak, S., and Shawe-Taylor, J. Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, 7:1601–1626, 2006.

Shrivastava, A., Song, Z., and Xu, Z. Breaking the linear iteration cost barrier for some well-known conditional gradient methods using maxip data-structures. NeurIPS'21, 2021a.

Shrivastava, A., Song, Z., and Xu, Z. Sublinear least-squares value iteration via locality sensitive hashing, 2021b.

Song, Z., Yang, S., and Zhang, R. Does preprocessing help training over-parameterized neural networks? *Advances in Neural Information Processing Systems*, 34, 2021.

Song, Z., Sun, B., Weinstein, O., and Zhang, R. Sparse fourier transform over lattices: A unified approach to signal reconstruction. http://arxiv.org/abs/2205.00658, 2022a.

Song, Z., Xu, Z., and Zhang, L. Speeding up sparsification with inner product search data structures. 2022b. URL https://arxiv.org/pdf/2204.03209.pdf.

Tuzel, O., Porikli, F., and Meer, P. Kernel methods for weakly supervised mean shift clustering. In *2009 IEEE 12th International Conference on Computer Vision*, pp. 48–55. IEEE, 2009.

Wang, J., Lu, H., Plataniotis, K. N., and Lu, J. Gaussian kernel optimization for pattern classification. *Pattern recognition*, 42(7):1237–1247, 2009.

Wenliang, L., Sutherland, D. J., Strathmann, H., and Gretton, A. Learning deep kernels for exponential family densities. In *International Conference on Machine Learning*, pp. 6737–6746. PMLR, 2019.

Weston, J., Zhou, D., Elisseeff, A., Noble, W., and Leslie, C. Semi-supervised protein classification using cluster kernels. *Advances in neural information processing systems*, 16, 2003.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. Deep kernel learning. In *Artificial intelligence and statistics*, pp. 370–378. PMLR, 2016a.

Wilson, A. G., Hu, Z., Salakhutdinov, R. R., and Xing, E. P. Stochastic variational deep kernel learning. *Advances in Neural Information Processing Systems*, 29, 2016b.

Xie, X., Liang, Z., Gu, P., Basak, A., Deng, L., Liang, L., Hu, X., and Xie, Y. Spacea: Sparse matrix vector multiplication on processing-in-memory accelerator. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 570–583. IEEE, 2021.

Yang, C., Buluc, A., and Owens, J. D. Design principles for sparse matrix multiplication on the gpu. In *European Conference on Parallel Processing*, pp. 672–687. Springer, 2018.

Ye, G. Fast algorithm for solving structured convex programs. Master's thesis, The University of Washington, 2021.

Zandieh, A., Nouri, N., Velingker, A., Kapralov, M., and Razenshteyn, I. Scaling up kernel ridge regression via locality sensitive hashing. In *International Conference on Artificial Intelligence and Statistics*, pp. 4088–4097. PMLR, 2020.

Zhang, Y., Duchi, J., and Wainwright, M. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *The Journal of Machine Learning Research*, 16(1):3299–3340, 2015.

Roadmap. We present our algorithm description in Section A. We present the missing proofs of the correctness and time complexity of our data structure in Section B. We present the additional evaluation figures in Section C.

A. Truncation Kernel Multiplication Algorithm

In this section, we provide our truncation kernel multiplication algorithm in Algorithm 1 and Algorithm 2.

Algorithm 1 Multiple tree

data structure MULTIPLE TREE
members
  $Y \in \mathbb{R}^{m \times d}$ ▷ m data points
  $X \in \mathbb{R}^{n \times d}$ ▷ n data points
  $\mathsf{K} \in \mathbb{R}^{n \times m}$ ▷ The kernel matrix in Definition 3.1
  Binary tree $T_1, T_2, \cdots, T_n$ ▷ n binary search trees
  $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$
end members
procedure INIT($y_1, y_2, \cdots, y_m \in \mathbb{R}^d, x_1, x_2, \cdots, x_n \in \mathbb{R}^d, f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$) ▷ Lemma 3.8
  $f \leftarrow f$
  for $i = 1 \to n$ **do**
    $x_i \leftarrow x_i$
  end for
  for $j = 1 \to m$ **do**
    $y_j \leftarrow y_j$
  end for
  for $i = 1 \to n$ **do** ▷ for data point, we create a tree
    for $j = 1 \to m$ **do**
      $u_j \leftarrow \langle x_i, y_j \rangle$
      $\mathsf{K}_{i,j} \leftarrow f(x_i, y_j)$
    end for
    $T_i \leftarrow$ MAKEMAXTREE($u_1, \cdots, u_m$) ▷ Each node stores the maximum value for his two children
  end for
end procedure
procedure UPDATE($z \in \mathbb{R}^d, j \in [m]$) ▷ Lemma 3.9
  $y_j \leftarrow z$
  for $i = 1 \to n$ **do**
    $l \leftarrow$ the l-th leaf of tree T_i
    l.value $= \langle z, x_i \rangle$
    $\mathsf{K}_{i,j} = f(x_i, y_j)$ ▷ Update the kernel
    while l is not root **do**
      $p \leftarrow$ parent of l
      p.value $\leftarrow \max\{p$.value, l.value$\}$
      $l \leftarrow p$
    end while
  end for
end procedure

B. Missing Proofs

In this section, we provide the missing proofs of the correctness and time complexity of our data structure.

B.1. Correctness of QUERY

Lemma B.1 (Restatement of Lemma 3.4)**.** *Given a threshold $\tau \in \mathbb{R}$ and a node $r \in T$, let $i \in [n]$ denote the index of the tree which contains node r as input, and let K_i denote the number of entries of above τ in tree T_i, the* QUERYSUB *operation*

Algorithm 2 Multiple trees

procedure QUERY($i \in [n], \tau \in \mathbb{R}$) ▷ Lemma 3.5 and Lemma 3.11
  QUERYSUB($\tau, \mathrm{root}(T_i)$)
end procedure
procedure QUERYSUB($\tau \in \mathbb{R}, r \in T$) ▷ Lemma 3.4 and Lemma 3.10
  if r is leaf **then**
    if $u_r \geq \tau$ **then**
      return r
    else
      return 0
    end if
  else
    $r_1 \leftarrow$ left child of r, $r_2 \leftarrow$ right child of r
    if r_1.value $\geq \tau$ **then**
      $S_1 \leftarrow$QUERYSUB(τ, r_1)
    end if
    if r_2.value $\geq \tau$ **then**
      $S_2 \leftarrow$QUERYSUB(τ, r_2)
    end if
  end if
  return $S_1 \cup S_2$
end procedure
procedure MULTIPLYSUB($i \in [n], v \in \mathbb{R}^m, \tau \in \mathbb{R}$) ▷ Lemma 3.6 and Lemma 3.12
  $R \leftarrow 0$, $S \leftarrow$ QUERY(i, τ)
  for $j \in S$ **do** ▷ Only compute the truncated entries.
    $R \mathrel{+}= v_j \cdot \mathsf{K}_{i,j}$
  end for
  return R
end procedure
procedure MULTIPLY($v \in \mathbb{R}^d, \tau \in \mathbb{R}$) ▷ Lemma 3.7 and Lemma 3.13
  $R \leftarrow \{\}$
  for $i = 1 \to n$ **do**
    R.APPEND(MULTIPLYSUB(i, v, τ))
  end for
  return R
end procedure

outputs a set containing all $y_j \in \mathbb{R}^d$ such that $\langle x_i, y_j \rangle \geq \tau$ in the subtree whose root is r.

Proof. We prove Lemma 3.4 by induction. The base case when the node $r \in T$ is the leaf node and the node height $i = 0$, the correctness of QUERYSUB is trivially true. Assume for the node height $i = 0, 1, \cdots, k$ we have proven Lemma 3.4 to be true. When the node height $i = k + 1$, we know $S_1 \leftarrow$QUERYSUB(τ, r_1) contains all y_j such that $\langle x_i, y_j \rangle \geq \tau$ in the left subtree of r and $S_2 \leftarrow$QUERYSUB(τ, r_2) contains all y_j such that $\langle x_i, y_j \rangle \geq \tau$ in the right subtree of r. Therefore, the $S_1 \cup S_2$ contains all all y_j such that $\langle x_i, y_j \rangle \geq \tau$ in the subtree whose root is r. This completes the proof. □

Lemma B.2 (Restatement of Lemma 3.5)**.** *Given an index $i \in [n]$ and a threshold $\tau \in \mathbb{R}$ as input, the* QUERY *operation outputs a set containing all y_j such that $\langle x_i, y_j \rangle \geq \tau$ in tree T_i.*

Proof. Because QUERY calls QUERYSUB at the root of tree T_i, from Lemma 3.4, we have that QUERY can output all y_j such that $\langle x_i, y_j \rangle \geq \tau$ in tree T_i correctly. □

B.2. Correctness of Multiplication

Lemma B.3 (Restatement of Lemma 3.6)**.** *Given an index i, a vector $v \in \mathbb{R}^d$ and a threshold $\tau \in \mathbb{R}$ as input, and let K_i denote the number of entries of above τ in tree T_i,* MULTIPLYSUB *outputs the result of truncated vector inner product $\mathsf{K}_i \cdot v$.*

Proof. From Lemma 3.5, we know that QUERY(i, τ) can correctly output a set S of size K_i containing all entries above τ. In MULTIPLYSUB, we can obtain the correct result of truncated vector product $\mathsf{K}_i \cdot v$ by accumulating $R[i] += v_j \cdot \mathsf{K}_{i,j}$ for all $j \in S$. This completes the proof. □

Lemma B.4 (Restatement of Lemma 3.7)**.** *Given a vector $v \in \mathbb{R}^d$ and a threshold $\tau \in \mathbb{R}$ as input, and let K_i denote the number of entries of above τ in each tree T_i,* MULTIPLY *outputs the result of truncated matrix vector multiplication $\mathsf{K} \cdot v$.*

Proof. With Lemma 3.6, we know that for $i \in [n]$, MULTIPLYSUB computes the correct vector inner product $\mathsf{K}_i \cdot v$. By combining the n scalar output of MULTIPLYSUB, we know that MULTIPLY computes the result truncated matrix vector multiplication $\mathsf{K} \cdot v$. This completes the proof. □

B.3. Running Time

Lemma B.5 (Restatement of Lemma 3.8)**.** *Given $y_1, y_2, \cdots, y_m \in \mathbb{R}^d$ and n data points $x_1, x_2, \cdots, x_n \in \mathbb{R}^d$ as input, the* INIT *operation runs in $O(mn(d + \mathcal{T}_f))$ time.*

Proof. We can view the INIT operation as having the following steps:

- It takes $O(mnd)$ time to compute vector inner dot product between two $\mathbb{R}^d$ vectors $\langle x_i, y_j \rangle$ for mn times.
- It takes $O(n \log(m))$ time to build binary tree T_i for n times.
- It takes $O(mn\mathcal{T}_f)$ time to compute $f(x_i, y_j)$ for mn times.

Thus, the total running time of INIT is

$$O(mnd) + O(n \log(m)) + O(mn\mathcal{T}_f) = O(mn(d + \mathcal{T}_f)).$$

□

Lemma B.6 (Restatement of Lemma 3.9)**.** *Given $z \in \mathbb{R}^d$ and an index $j \in [m]$ as input, the* UPDATE *operation runs in $O(n(d + \log(m) + \mathcal{T}_f))$ time.*

Proof. In UPDATE operation, we need to update all n trees $\{T_i\}_{i=1}^n$. We can view the tree update as having the following steps:

- It takes $O(nd)$ time to compute vector inner product between two $\mathbb{R}^d$ vectors $\langle z, x_i \rangle$ for n times.
- It takes $O(\log(m))$ time to update a tree of $O(\log(m))$ height from leaf node up to the root. There are n trees, so the total time is $O(n \log(m))$.
- It takes $O(n\mathcal{T}_f)$ time to compute $f(x_i, y_j)$ for n times.

Thus, the total running time of UPDATE is

$$O(nd) + O(n \log(m)) + O(n\mathcal{T}_f) = O(n(d + \log(m) + \mathcal{T}_f))$$

□

Lemma B.7 (Restatement of Lemma 3.10)**.** *Given a threshold $\tau \in \mathbb{R}$ and a node $r \in T$ as input, let i denote the index of the tree which contains node r and let K_i denote the number of entries of above τ in tree T_i. The* QUERYSUB *operation runs in $O(\log(m))$ time.*

Proof. The QUERYSUB operation calls itself recursively on its left and right child until meeting the leaf node. The depth of the tree is $O(\log(m))$, and the comparison check is $O(K_i)$ per level. Therefore, the overall time complexity of QUERYSUB is $O(K_i \log(m))$. □

Lemma B.8 (Restatement of Lemma 3.11)**.** *Given an index $i \in [n]$ and a threshold $\tau \in \mathbb{R}$ as input and let K_i denote the number of entries of above τ in tree T_i, the* QUERY *operation runs in $O(K_i \log(m))$ time.*

Proof. The QUERY call QUERYSUB at the root node of tree T_i, so the time complexity of QUERY is $O(K_i \log(m))$. □

Lemma B.9 (Restatement of Lemma 3.12)**.** *Given an index i, a vector $v \in \mathbb{R}^d$ and a threshold τ as input and let K_i denote the number of entries of above τ in tree T_i, the* MULTIPLYSUB *operation output the truncated vector inner product $\mathsf{K}_i \cdot v$ in $O(K_i \log(m))$ time.*

Proof. We can view the MULTIPLYSUB operation as having the following steps:

- It takes $O(K_i \log(m))$ to call QUERY operations.
- It takes $O(K_i)$ time to compute scalar multiplication $v_j \cdot \mathsf{K}_{i,j}$ for $O(K_i)$ times.

Thus, the total running time of MULTIPLYSUB is

$$O(K_i \log(m) + O(\log(m)) = O(K_i \log(m))$$

□

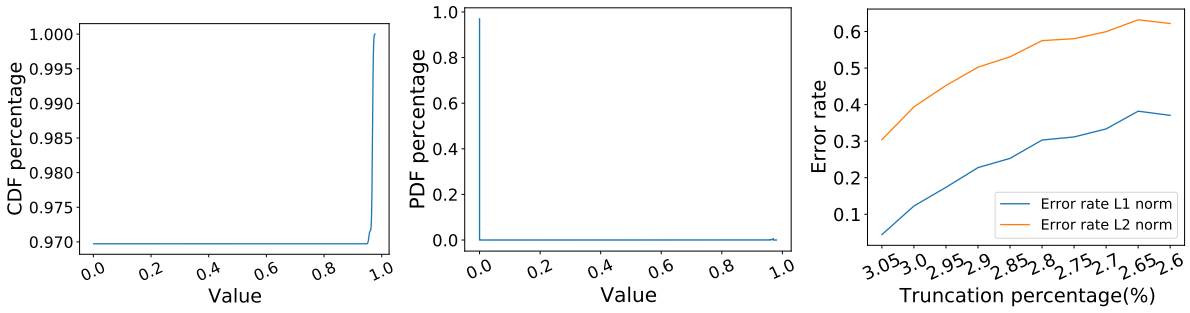
Lemma B.10 (Restatement of Lemma 3.13)**.** *Given a vector $v \in \mathbb{R}^d$ as input and let K_i denote the number of entries of above τ in each tree T_i, the* MULTIPLY *operation output the truncated matrix vector multiplication $\mathsf{K} \cdot v$ in $O(\sum_{i=1}^n K_i \log(m))$ time.*

Proof. We can view the MULTIPLY operation as executing MULTIPLYSUB for n times. From Lemma 3.12 we can know the overall time complexity of MULTIPLY is :

$$\sum_{i=1}^n O(K_i \log(m)) = O(\sum_{i=1}^n K_i \log(m))$$

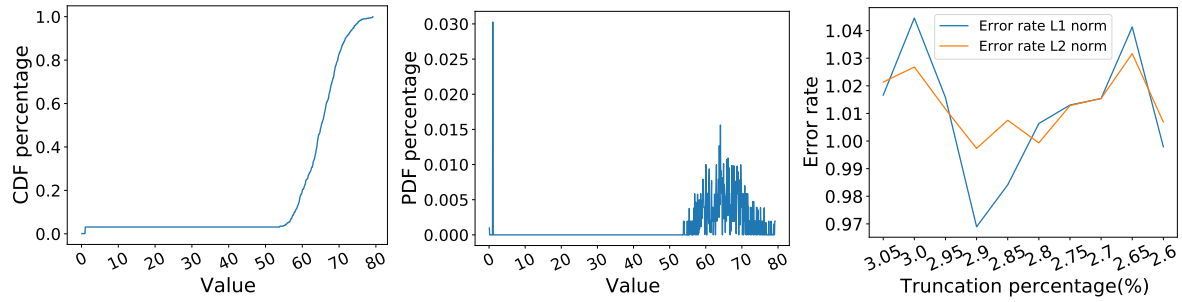
□

C. Additional Evaluation Figures



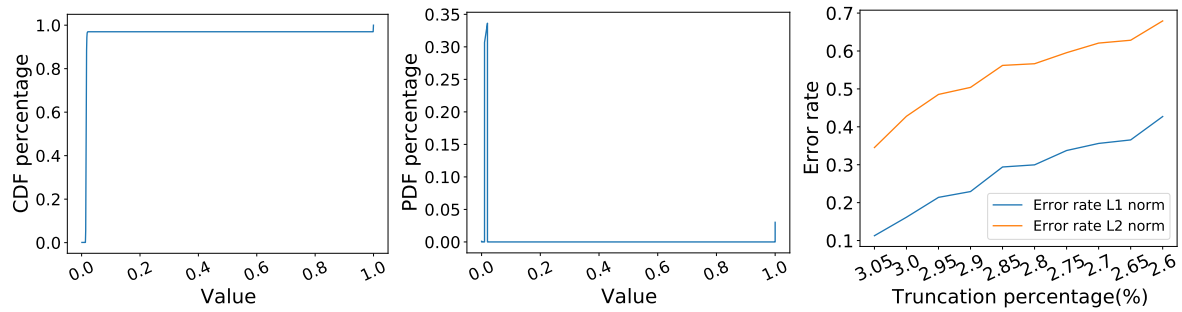
(a) (b) (c)

Figure 6: Laplacian kernel (a) cumulative distribution function (CDF) figure. (b) probability density function (PDF) figure. (c) L1 and L2 norm error rates under different truncation degree



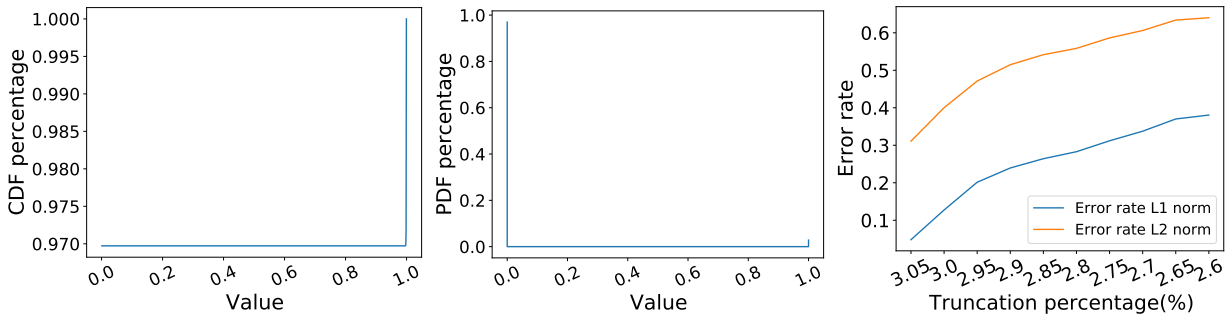
(a) (b) (c)

Figure 7: Multiquadratic kernel (a) CDF figure. (b) PDF figure. (c) L1 and L2 norm error rates under different truncation degree



(a) (b) (c)

Figure 8: Inverse multiquadratic kernel (a) CDF figure. (b) PDF figure. (c) L1 and L2 norm error rates under different truncation degree



(a) (b) (c)

Figure 9: Rational kernel (a) CDF figure. (b) PDF figure. (c) L1 and L2 norm error rates under different truncation degree