UNIVERSITY OF CALIFORNIA

Los Angeles

Efficient Machine Learning by Leveraging Data Dependent Information

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

PEI-HUNG CHEN

2022

ABSTRACT OF THE DISSERTATION

Efficient Machine Learning by Leveraging Data Dependent Information

by

PEI-HUNG CHEN
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2022
Professor Cho-Jui Hsieh, Chair

As machine learning models grow much larger nowadays, recent research found that advances to improve accuracy might not be able to make neural networks applicable to all situations due to size and speed constraints. To make machine learning more applicable to all real-world applications, there is a need to obtain a small model size and faster inference speed. There are many explicit information and hidden data dependent distributions in the underlying data mining and machine learning problems. However, past research often focused on model parameters directly without considering the contextual information in the underlying problem. In this dissertation, we demonstrate how we can obtain a much more efficient machine learning systems via leveraging data dependent information. Specifically, we will show how both explicit and implicit data dependent information can be combined with many existing methods to obtain a much smaller model size and faster inference time. In addition, this data dependent information is ubiquitous and we can find it in many applications such as data mining, natural language processing, information retrieval and recommender system problems.

The dissertation of PEI-HUNG CHEN is approved.

Quanquan Gu

Sriram Sankararaman

Wei Wang

Cho-Jui Hsieh, Committee Chair

University of California, Los Angeles

2022

To my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Firstly, I would like to convey my sincere gratitude to my advisor, prof. Cho-Jui Hsieh, for picking up me during the Ph.D. program application process when I had no publication and almost no formal research experience at the time. I also thank him for giving me substantial space to reject continuing certain more publishable research topics and digress for a moment to study some topics which interested me more, despite that eventually no publication comes out through this thinking process.

Next, I would like to thank Dr. Si Si, who is my first internship supervisor and the major mentor of my first publication during Ph.D. program. From her, I learned how to frame the research problem, setup and organize experiments and write up a concise research paper. It's a very unforgettable research experience.

I want to thank my committee members, Prof. Quanquan Gu, Prof. Sriram Sankararaman, and Prof. Wei Wang for their advice and suggestions. The achievement of my Ph.D. journey and the work presented in this thesis could not have been possible without the help of my advisor and committee members. I also want to give a special thank to Prof. Wei Wang for giving many useful comments as a co-author of a publication. And thank Prof. Quanquan Gu for instructing an fascinating theory course at school, which inspires me to explore more theoretical topics of research.

I also want to thank all supervisors I met during internship: Bo Dai, Yang Li, Hsiang-fu Yu and Wei Wei. And million thanks for my awesome co-authors: Ciprian Chelba, Jyun-yu Jiang, Harold Li, Prof. Kai-Wei Chang and Prof. Inderjit Dhillon.

Last, I want to thank my family who supported me during the most difficulty time of Ph.D. study.

PUBLICATIONS

Publications Included in this Dissertation

P. H. Chen, S. Si, Y. Li, C. Chelba, and C.J. Hsieh. GroupReduce: Block-Wise Low-Rank Approximation for Neural Language Model Shrinking. In Neural Information Processing Systems (NeurIPS), 2018.

P. H. Chen, S. Si, S. Kumar, Y. Li, C.J. Hsieh. Learning to Screen for Fast Softmax Inference on Large Vocabulary Neural Networks. In Proceedings of the International Conference on Learning Representations (ICLR), 2019.

J.Y. Jiang*, **P. H. Chen***, C.J. Hsieh and W. Wang. Clustering and Constructing User Coresets to Accelerate Large-scale Top-K Recommender Systems. In Proceedings of the World Wide Web Conference (WWW), 2020. (* Equal Contribution)

Y.K. Ma*, **P. H. Chen***, C.J. Hsieh. MulCode: A Multiplicative Multi-way Model for Compressing Neural Language Model. To appear in EMNLP 2019. (* Equal contribution)

P. H. Chen, H. F. Yu, I. K. Dhillon and C.J. Hsieh. DRONE: Data-aware Low-rank Compression for Large NLP Models. To appear in Neural Information Processing Systems (NeurIPS), 2021.

Other publications which are not included in this dissertation for consistency of the topic.

L. H. Li, **P. H. Chen**, C.J. Hsieh, K. W. Chang. Efficient Contextual Representation Learning Without Softmax Layer. In Transactions of the Association for Computational, 2019.

M. Cheng, S. Singh, **P. H. Chen**, P. Y. Chen, S. Liu, C.J. Hsieh. Sign-OPT: A Query-Efficient Hard-label Adversarial Attack. In Proceedings of the International Conference on Learning Representations (ICLR), 2020.

P. H. Chen, W. Wei, C.J. Hsieh, Bo Dai. Overcoming Catastrophic Forgetting by Bayesian Generative Regularization. In Conference of Machine Learning (ICML), 2021.

CHAPTER 1

Introduction

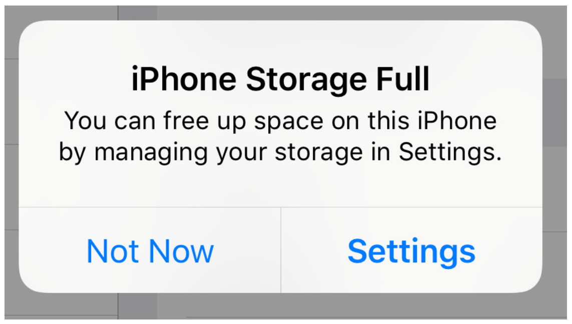
1.1 Motivations

We have witnessed a wave of machine learning development in the past decade. With the advent of neural networks [84], machine learning models have been increasingly applied in various domains such as computer vision [17, 122], natural language processing [15, 17, 36], information retrieval [3, 23] and data mining [8, 123]. However, in addition to using more training data to get a better accuracy, recent research found that advances to improve accuracy might not be able to make networks more efficient with respect to size and speed of models [63, 126]. In other words, to make machine learning models applicable to some applications, these models are required to be tailored into a more efficient forms. In literature, three aspects attracted most attention: model size, inference time and energy used. Let's dive into each of these aspects to see why they are important when deploying machine learning models in real world applications.

1.1.1 Model Size

As the machine learning models become more ubiquitous, we can observe that more and more machine learning models are deployed on devices. As shown in Figure1.1(a), there are many popular edge devices in our daily life. These edge devices might not have a large storage on device, so it's not uncommon that the storage is full as shown in Figure1.1(b). As noted in literature that modern neural network size grows in orders, it's presumable that these models won't be able to fit into all edge devices [24]. Thus, the wish to universally deploy machine learning models drive the need to make the model size smaller.

(a) Illustration that many emerging edge deployment of machine learning models. Images sources.


iPhone Storage Full
You can free up space on this iPhone by managing your storage in Settings.

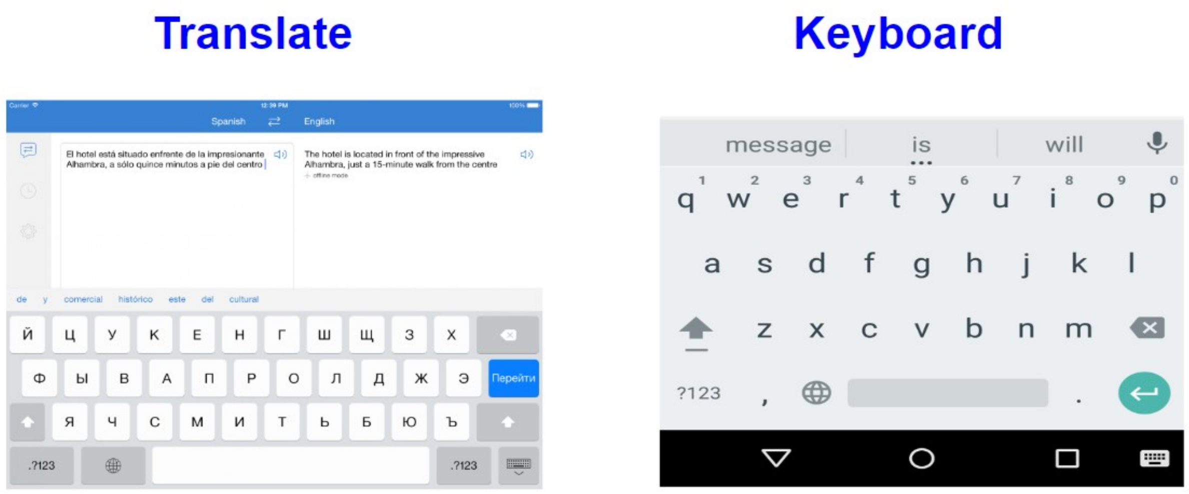
(b) Illustration that out of memory situation is not uncommon in modern daily life when accessing edge devices.

Figure 1.1: Illustration of importance of machine learning model size. Image sources for (a) https://www.google.com/glass/start/;https://home.google.com/welcome/; https://en.wikipedia.org/wiki/Pepper_(robot); https://www.apple.com/watch/. Image sources for (b) https://appleinsider.com/articles/19/08/15/how-to-stop-getting-the-storage-full-message-on-your-iphone.

1.1.2 Inference Time

In certain applications such as self-driving car or computer vision driven mobiles APPS, the image recognition tasks need to be carried out in a timely fashion on a resource-limited platform (e.g., a low-end MCU chip). As shown in Figure 1.2, in our daily mobile phone usage, we might use translate to communicate with others or use auto-complete when typing. We certainly want the responsive time to be negligible when using such services. In e-commercial scenario, servers need to respond to search request in milliseconds [46]. These application scenarios all require machine learning models to complete the inference step in a rather short period of time.

Moreover, in some machine learning classification tasks, the output space is in million scale. For example, in modern neural recommendation systems, users and items are represented by embedding vectors [83]. To find a recommendation, we have to perform an inner-product between each user and item. However, both users and items are in million-scale [164] such that even a full inference computation between all pairs of users and items becomes very challenging. In order to make such large-scale recommender systems practical, we have to accelerate the inference time of the machine learning models.



El hotel está situado enfrente de la impresionante Alhambra, a sólo quince minutos a pie del centro

The hotel is located in front of the impressive Alhambra, just a 15-minute walk from the centre

Figure 1.2: Illustration of fast inference requirement in daily mobile phone usages.

1.1.3 Energy Used

In our daily life usage, it's easy to meet out of battery situation as shown in Figure 1.3. As mobile APPs incorporate much more powerful machine learning models, we can expect the energy consumption will increase and that will cause the mobile usage duration to decrease. On a larger scale, there is no doubt that reducing energy costs and CO2 emissions has become the most important task globally [1]. Many information and communications technology companies are constantly installing more servers, and each server draws far more electricity than its earlier models [2]. These expansions will greatly harm the goal to ease the greenhouse effect. A plausible way to control this is to have the model consumes less energy so overall the whole server consumes less.

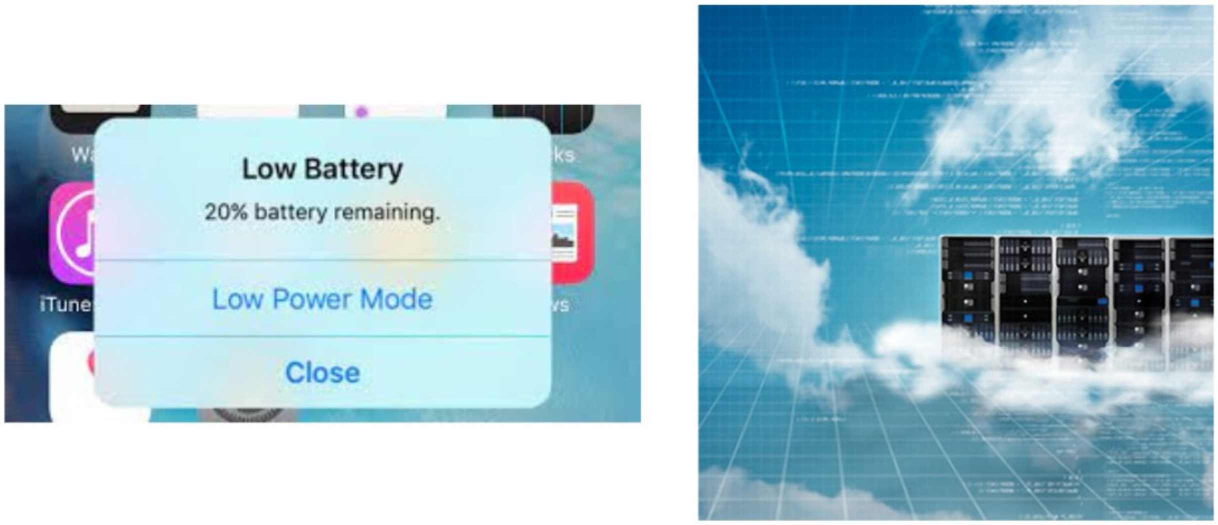


Figure 1.3: Illustration of why reducing energy consumption is an import task. It will influence both our daily life and industrial server deployment. Image source: https://stanfordmag.org/contents/carbon-and-the-cloud

1.1.4 Efficient Machine Learning

Based on the previous discussion, we can use Figure 1.4 to summarize what efficient machine learning is. We define efficient machine learning to be the union of the above mentioned three domains: machine learning model size, inference speed of machine learning models, and the energy consumed during usages of machine learning models. In the previous sections, we have motivated why these factors posted challenges to the current machine learning development. Consequently, achieving efficient machine learning becomes an important research topic and it's the main focus of this thesis. Before moving on, we point out two important aspects regarding efficient machine learning.

First, despite that these factors are correlated, they are not necessarily aligned with each other. Improving one factor might not contribute the improvement of another factor; even worse, for some cases to achieve one goal will actually be at the cost of another factor. For example, Sparse pruning [59], a very effective model compression approach, can greatly reduce the model size by setting some neuron connections to zero. However, once the links are pruned, the original model is no longer a dense matrix. This pruned structure won't

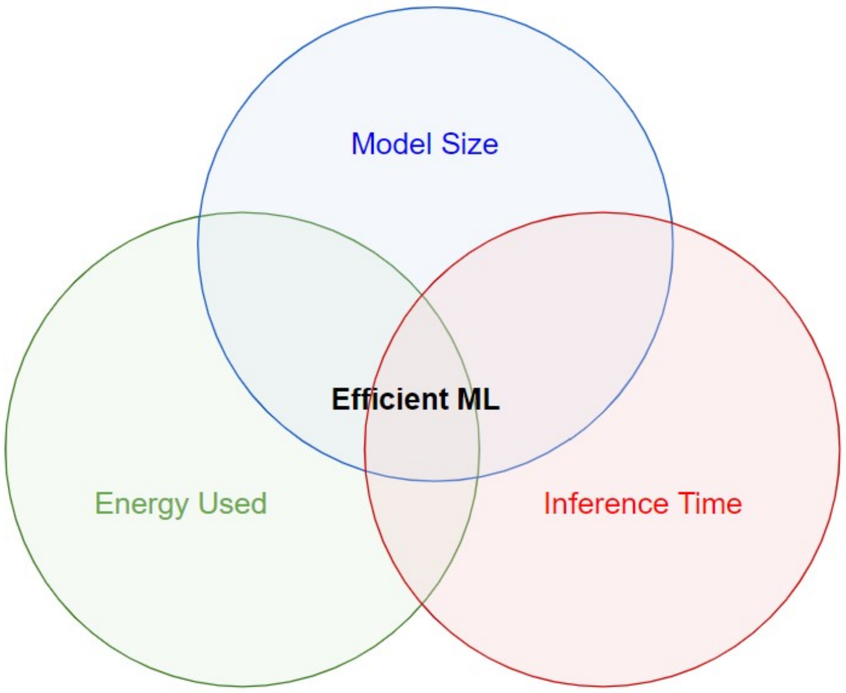


Figure 1.4: Definition of Efficient Machine Learning (EML). EML is composed of model size reduction, inference time speedup and less energy consumption.

be able to enjoy speedup brought by the well optimized BLAS matrix computation library. Thus, in practice, we will observe a longer inference time. On the other hand, low-rank based methods [125] try to approximate the original dense matrix by finding two low-rank matrices with smaller dimensions. It can simultaneously reduce the model size and keep the model matrices dense. As a result, we can observe a speedup ratio appropriate to the model size reduction ratio. Overall, there are some trade-offs we need to consider when designing efficient machine learning algorithms.

Second, in this thesis, we will focus on model compression and inference speedup only. The thesis will not discuss the energy consumption. One major reason is that the energy consumption of edge devices is highly related to its data-flow design [160]. This is more of how the hardware components arrange the data . Given the same dataflow structure, energy consumption will be highly dependent on number of processor cycles. Thus, we can effectively use inference time to approximate the energy consumption. Another reason is that measuring energy consumption requires special hardware equipment. It's not an easy task to measure energy consumption on CPUs or GPUs. In literature, most experimental comparisons were done on FPGA [151]. A straightforward way to measure energy consumption is to use Xilinx XPE toolkit [35] which will report the energy usage in a few seconds. However, this will limit our computations and models only be performed on FPGA, and this will set many restrictions . In sum, we focused more on algorithmically aspects of efficient machine learning, and use inference time and model size as an indicator of energy consumption.

1.2 Approach

In this thesis, we will demonstrate how we could incorporate data distributions into different existing compression methods in order to achieve efficient machine learning for different application scenarios. Traditional efficient machine learning methods focused on the model parameters directly. Most if not all methods neglected semantic or contextual information of the underlying data but only work with model parameters directly. For example, canonical low-rank SVD methods set a rank hyper-parameter of the weight matrix without considering how the matrix would be used in inference time. Similar situations happen in many inference speedup scenarios. A popular method to accelerate classification tasks is to approximate the computation of output softmax matrix, where each column vector in the matrix corresponds to the representation of an output class. Most classification model also use straightforward apply approximate top-k maximal inner-product search of the full softmax matrix directly. These methods all serve as generic methods which could be applied on all models in various machine learning tasks. However, there are more information hidden in the underlying data which can be very useful to achieve efficient machine learning. For example, frequency information is easily accessible in the training corpus of natural language processing application. As each column vector in the softmax matrix corresponds to a word in the corpus, we could use this frequency information to further process the matrix. And by saying data distribution, we are referring to a wide range of **input** data dependent features. In particular, we will heavily rely on the usage of hidden structures of output latent space of certain layer of neural networks in this thesis. These features can be detected by some machine learning techniques. And these structural information can eventually be used to accelerate the inference stage of large-scale machine learning problems. Most if not all of these ”structures” in the underlying problem are not explored before in machine learning research. In this thesis, the main approach to study efficient machine learning. We will demonstrate that these structures in the distribution could greatly help to reach efficient machine learning.

1.3 Thesis Contributions

The contributions of this thesis are summarized as follows:

- We showed that many modern Machine Learning problems have special structures in their problem setup. Specifically, there are many data or distribution related information hidden inside the problem which can be leveraged to achieve efficient Machine Learning.
- We demonstrated that this problem-specific data distribution information can be applied in combination to many different techniques to achieve machine learning. In particular, we will demonstrate that there are explicit information which can calculate from the data and there are implicit information which we can extract by some machine learning techniques. Both information could benefit existing efficient algorithms to further achieve a better performance.
- We will analyze some hardware software co-design issues in efficient machine learning. We will illustrate that not all methods could achieve both model reduction and inference time speedup simultaneously due to hardware and software acceleration issues. In addition, we will demonstrate that low-rank approximations in general could preserve the underlying computational model such that it could reduce the model memory usage while obtaining faster inference speed.
- We demonstrate that data distribution information could greatly help various application domains. We will show experimentally that on data mining, natural language

processing, information retrieval and recommender systems problems, using data dependent features could achieve much efficient machine learning systems.

1.4 Thesis Structure

The outline of the thesis is as follows. In first two chapters, we will focus on machine learning model size reduction problems. Specifically, we will use explicit information from the training corpus and propose simple observations from the data which can greatly help to obtain a much smaller model. In chapter 2, we will focus on combining data information with low-rank approximation methods, and in chapter 3, we will combine data information with compsotional methods. In chapter 4 and chapter 5, we will study the inference time speedup problems. In this two chapters, we switch directions toward using implicit data dependent information. We will show that on natural language processing tasks and recommender systems, these hidden structures in latent vectors spaces exist and we can learn an effective screening model to leverage the information in order to achieve a much faster inference time. In chapter 6, we will demonstrate a data-aware low-rank approximation method which can achieve both model compression and inference time speedup at the same time. A major contribution of chapter 6 is that this is a fairly generic method and it's applicable to any application regardless the characteristics of the underlying data format. In chapter 7, we will propose another generic tool which can achieve a much fast approximate top-K nearest neighbor search by using local data information. Last in chapter 8, we make a final remark on contributions, limitations and some interesting future directions.

CHAPTER 2

Low-rank Model Compression Methods by Leveraging Word Frequency Information.

2.1 Introduction

As we mentioned in the introduction, deep neural nets with a large number of parameters have a great capacity for modeling complex problems. However, the large size of these models is a major obstacle for serving them on-device where computational resources are limited. As such, compressing deep neural nets has become a crucial problem that draws an increasing amount of interest from the research community. Given a large neural net, the goal of compression is to build a light-weight approximation of the original model, which can offer a much smaller model size while maintaining the same (or similar) prediction accuracy.

We focused on some important Natural language processing (NLP) tasks such as language modeling (e.g., next word prediction) and machine translation. A modern neural language model often consists of three major components: one or more recurrent layers (often using LSTM), an embedding layer for representing input tokens, and a softmax layer for generating output tokens. The dimension of recurrent layers (e.g., LSTM), which corresponds to the hidden state, is typically small and independent of the vocabulary size of input/output tokens. In contrast, the dimension of the embedding and the softmax layers grow with the vocabulary size, which can easily be at the scale of hundreds of thousands. As a result, the parameter matrices of the embedding and softmax layers are often responsible for the major memory consumption of a neural language model. For example, DE-EN Neural Machine Translation task has roughly a vocabulary size around 30k and around 80% of the memory is used to store embedding and softmax matrices. Furthermore, the One Billion Word language modeling task has a vocabulary size around 800k, and more than 90% of the memory footprint is due to storing the embedding and softmax matrices. Therefore, to reduce the size of a neural language model, it is highly valuable to compress these layers, which is the focus of our paper.

There have been extensive studies for compressing fully connected and convolutional networks [34, 58, 59, 64, 125, 155, 165]. The mainstream algorithms from these work such as low-rank approximation, quantization, and pruning can also be directly applied to compress the embedding and softmax matrices. However, these methods only focused on the model parameters itself. They didn't consider the underlying task to perform inference. One important aspect that has not been well explored in the literature is that the embedding matrix has several specific properties that do not exist in a general weight matrix of other types of machine learning models. Each column of the input embedding and softmax matrix represents a token, which implies that on a given training or test set the parameters in that column are used with a frequency which obeys Zipf's law distribution.

By exploiting these structures, we propose GroupReduce, a novel method for compressing the embedding and softmax matrices using block-wise, weighted low-rank approximation. Our method starts by grouping words into blocks based on their frequencies, and then refines the clustering iteratively by constructing weighted low-rank approximation for each block. This allows word vectors to be projected into a better subspace during compression. Our experiments show that GroupReduce is more effective than standard low-rank approximation methods for compressing these layers. It is easy-to-implement and can handle very large embedding and softmax matrices.

2.2 Related Work

In this section, we will firstly review some general model compression techniques on compressing convolutional neural networks (CNN) in computer vision applications. These techniques provide some basic baselines which can be applied to NLP tasks too. Next we will review some prior experimental results on applying these general techniques to NLP tasks.

2.2.1 General Model Compression Methods

Low-rank matrix/tensor factorization. To compress a deep net, a natural direction is to approximate each of its weight matrices, W, by a low-rank approximation of the matrix using SVD. Based on this idea, [125] compressed the fully connected layers in neural nets. For convolution layers, the kernels can be viewed as 3D tensors. Thus, [34, 66] applied higher-order tensor decomposition to compress CNN. In the same vein, [63] developed another structural approximation. [79] proposed an algorithm to select rank for each layer. More recently, [165] reconstructed the weight matrices by using sparse plus low-rank approximation.

Pruning. Algorithms have been proposed to remove unimportant weights in deep neural nets. In order to do this, one needs to define the importance of each weight. For example, [88] showed that the importance can be estimated by using the Hessian of loss function. [59] considered adding ℓ_1 or ℓ_2 regularization and applied iterative thresholding approaches to achieve very good compression rates. Later on, [58] demonstrated that CNNs can be compressed by combining pruning, weight sharing and quantization.

Quantization. Storing parameters using lower precision representations has been used for model compression. Recently, [64] showed that a simple uniform quantization scheme can effectively reduce both the model size and the prediction time of a deep neural net. [95] showed that non-uniform quantization can further improve the performance. Recently, several advanced quantization techniques have been proposed for CNN compression [32, 159].

2.2.2 Model Compression for RNN/LSTM

Although model compression has been studied extensively for CNN models, less works have focused on the compression for recurrent neural nets (RNNs), another widely-used category of deep models in NLP applications. Since RNN involves a collection of fully connected layers, many of the aforementioned approaches can be naturally applied. For example, [64] applied their quantization and retraining procedure to compress a LSTM (a popular type of RNN) language model on Penn Tree Bank (PTB) dataset. [145] applied a matrix/tensor factorization approach to compress the transition matrix of LSTM and GRU, and tested their algorithm on image and music classification problems (which does not need word embedding matrices). [99, 118] proposed pruning algorithms for LSTM models compression.

Among the previous work, we found only [64, 99] tried to compress the word embedding matrix in NLP applications. [64] showed that the quantization-plus-retraining approach can only achieve less than 3 times compression rate on PTB data with no performance loss. [99] showed that for word-level LSTM models, the pruning approach can only achieve 87% sparsity with more than 5% performance loss. This means roughly 26% parameters over the original model since this approach also needs to store the index for non-zero locations. Very recently, [86] compressed the word embeddings computed by the word2vec algorithm and applied to similarity/analogy task and Question Answering. [134] applied compositional coding to compress the input embedding matrix of LSTM, but it is challenging to compress the softmax (output) layer matrix using the same algorithm. As a result, the overall compressed model from this approach is still large. One main issue of the approach is that multiple words share the same coding, which makes these words indistinguishable in the output layer during inference.

These previous results indicate that compressing embedding matrices in natural language tasks is a difficult problem—it is extremely challenging to achieve 4 times compression rate without sacrificing performance. In this paper, we will show that instead of only treating the embedding or the softmax parameters as a pure matrix, by exploiting the inherent structure of natural languages, GroupReduce algorithm could achieve much better compression rates.

Table 2.1: The size of each layer in the model. The number in parenthesis shows the ratio respective to the entire model size.

Models	vocabulary size	dimension	model size	embedding layer(s)	softmax layer	LSTM cell
PTB-Small	10k	200	17.7MB	7.6MB(42.9%)	7.6MB(42.9%)	2.5MB(14.2%)
PTB-Large	10k	1500	251MB	57MB(22.7%)	57MB(22.7%)	137MB(54.6%)
NMT: DE-EN	30k	500	195 MB	115 MB (59.0%)	47MB(24.1%)	33MB(16.9%)
OBW-BigLSTM	793k	1024	6.8GB	3.1GB (45.6%)	3.1GB(45.6%)	0.6GB(8.8%)

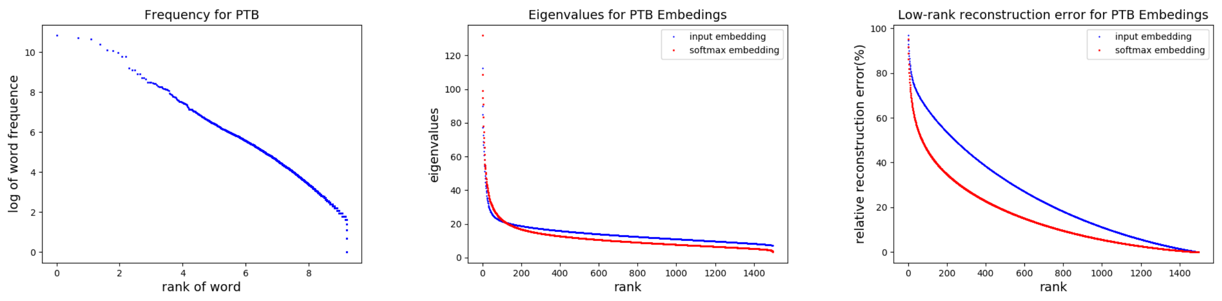
2.3 Problem Statement

Assume the word embedding matrix has size N-by-D, where N is the vocabulary size and D is the embedding dimension. We will use $A \in \mathcal{R}^{N \times D}$ to denote the embedding matrix (either input or softmax layer), and each row of A corresponds to the embedding vector of a word, i.e., the vector representation of the word. Our goal is to compress the embedding matrix A so that it uses less memory while achieving similar prediction performance. For a typical language model, especially the one with a large vocabulary size, the large memory size of the model is mostly due to the need to store the input and output word embedding matrices. In Table 2.2, we show an anatomy of memory consumption for several classic models trained on the publicly available datasets. We can see that for three out of four setups, embedding matrices contribute more than 75% of the overall memory usage. For example, in bigLSTM model that achieved start-of-the-art performance on OBW, more than 90% of memory is used to store two (input and output) word-embedding matrices. Thus, for deep neural net models alike, the main challenge to serve them on-device is to store tremendous memory usage of word embedding matrices. As such, it is highly valuable to compress these word embedding matrices.

2.4 Methods

2.4.1 Conventional Low-rank Approximation

Given a word embedding matrix A, a standard way to compress A while preserving the information is to perform low-rank approximation over A. A low-rank approximation can



(a) Frequency (b) Eigenvalues (c) Reconstruction error

Figure 2.1: Illustration on Penn Treebank (PTB) dataset with the vocabulary size to be 10k and the model's embedding dimension to be 1500. (a): log of word frequency vs rank of the word. One word' rank is defined as the log of number of words that occurs less than it. We can clearly observe the power law distribution of the word frequency; (b) x-axis shows the rank of approximatiion, and y-axis shows the eigenvalues. Here eigenvalues for two embedding matrices are from the input embedding layer and softmax layer; we can see the eigenvalues are very large. (c) low-rank reconstruction error based on singular value decomposition for the two embedding matrices. This in other way shows that the vanilla SVD may not work well for the embedding matrix.

be acquired by using singular value decomposition (SVD), which achieves the best rank-k approximation:

$$A \approx USV^T, \tag{2.1}$$

where $U \in \mathcal{R}^{N \times k}, V \in \mathcal{R}^{D \times k}$ where $k < \min(D, N)$ is the target rank, and S is a diagonal matrix of singular values. After the rank-k low-rank approximation, the memory footprint for A reduces from $O(ND)$ to $O(Nk + Dk)$.

There are two issues for using vanilla SVD to compress an embedding matrix. First, the rank of the SVD is not necessarily low for an embedding matrix. For example, Figure 2.1b shows that all the eigenvalues of the PTB word embedding matrices are quite large, which leads to poor reconstruction error of low-rank approximation in Figure 2.1c. Second, the SVD approach considers A as a regular matrix, but in fact each row of A corresponds to the embedding of a word, which implies additional structure that we can further exploit under the language model case.

Table 2.2: The size of each layer in the model. The number in parenthesis shows the ratio respective to the entire model size.

Models	vocabulary size	dimension	model size	embedding layer(s)	softmax layer	LSTM cell
PTB-Small	10k	200	17.7MB	7.6MB(42.9%)	7.6MB(42.9%)	2.5MB(14.2%)
PTB-Large	10k	1500	251MB	57MB(22.7%)	57MB(22.7%)	137MB(54.6%)
NMT: DE-EN	30k	500	195 MB	115 MB (59.0%)	47MB(24.1%)	33MB(16.9%)
OBW-BigLSTM	793k	1024	6.8GB	3.1GB (45.6%)	3.1GB(45.6%)	0.6GB(8.8%)

2.4.2 The Word Frequency

One important statistical property of natural languages is that the distribution of word frequencies can be approximated by a power law. That means a small fraction of words occur many times, while many words only appear few times. Figure 2.1a shows the power-law distribution of word frequency in the PTB datasets.

In the previous compression methods, none of them takes the word frequency into consideration when approximating the embedding matrix. Intuitively, to construct a good compressed model with low-rank approximation under the limited memory budget, it is important to enforce more frequent words to have better approximation. Thus, we considered two strategies to exploit the frequency information in low-rank approximation: weighted low-rank approximation and block low-rank approximation.

2.4.3 Improved Low-rank Approximation by Exploiting Frequency

In this subsection, we introduce GroupReduce: a Block-Wise Low-Rank Approximation for Neural Language Model compression method that incorporates word frequency information into the compressing process.

Weighted low-rank approximation. Firstly, we introduce a weighted low-rank approximation to compress the embedding matrix A. This will be used to replace original SVD and serves as the basic building block of our proposed algorithm. The main idea is to assign a different weight for each word's approximation and penalize more for the higher frequency words when constructing low-rank approximation. Mathematically, for the i-th word's frequency to be q_i, we want to approximate the embedding A by minimizing

$$\min_{U \in R^{N\times k}, V \in R^{D\times k}} \sum_{i=1}^{N}\sum_{j=1}^{D} q_i (A_{ij} - U_i V_j^T)^2 \tag{2.2}$$

where k is the reduced rank; A_{ij} is i-th word's j-th feature; $U \in R^{N\times k}, V \in R^{D\times k}$; U_i and V_j are i-th and j-th row of U and V respectively. Note that here we do not require U, V to be orthonormal.

Although it is known that weighted SVD with element-wise weights does not have a closed-form solution [139], in our case elements in the same row of A are associated with the same weights, which leads to a simple solution. Define $Q = \text{diag}(\sqrt{q_1}, \ldots, \sqrt{q_N})$, then the optimization problem of (2.2) is equivalent to

$$\min_{U \in R^{N\times k}, V \in R^{D\times k}} \|QA - QUV^T\|_F^2. \tag{2.3}$$

Therefore, assume all the q_i are nonzeros, we can solve (2.2) by conducting low-rank approximation of QA. Assume $[\bar{U}, \bar{S}, \bar{V}] = \text{svd}(QA)$, then $(U^*, V^*) = (Q^{-1}\bar{U}\bar{S}, \bar{V})$ will be a solution of (2.2). Therefore solving Eq.(2.2) is easy and the solution can be immediately computed from SVD of QA.

Block low-rank approximation. As can be seen from Figure 2.1b, the embedding matrix is in general not low-rank. Instead of constructing one low-rank approximation for the entire matrix, we can consider block-wise low-rank approximation–each block has its own approximation to achieve better compression. A similar strategy has been exploited in [135] for kernel approximation (symmetric PSD matrix). Mathematically, suppose we partition the words into c disjoint blocks $\mathcal{V}_1, \cdots, \mathcal{V}_c$, and each $\mathcal{V}_p$ contains a set of words. For each block $\mathcal{V}_p$ and its corresponding words' embedding $A_{\mathcal{V}_p}$ in A, we can generate a low-rank approximation with rank k_p as $A_{\mathcal{V}_p} \approx U^p(V^p)^T$ for $A_{\mathcal{V}_p}$. Then block low-rank approximation for A is represented as:

$$A = [A_{\mathcal{V}_1}, A_{\mathcal{V}_2}, \cdots, A_{\mathcal{V}_c}] \approx [U^1(V^1)^T, U^2(V^2)^T, \cdots, U^c(V^c)^T]. \tag{2.4}$$

The challenges for Eq (2.4) is on how to construct the clustering structure. Intuitively, we want similar frequency words to be grouped in the same block, so we can assign different ranks for different blocks based on their average frequency. For higher frequency words' clusters, we can provide more ranks/budget for better approximation. Meanwhile, we want to make sure the approximation error to be small for words under the same memory budget. Therefore, in this paper we consider two factors, word frequency and reconstruction performance, when constructing the partition. Next, we will explain how to construct the partition.

Block weighted low-rank approximation. To take both matrix approximation as well as frequency information into account when forming the block structure in Eq (2.4), we propose to refine the blocks after initializing the blocks from frequency grouping to achieve lower reconstruction error. In the refinement stage, we move the words around by simultaneously learning a clustering structure as well as low-rank approximation inside each cluster for the word embedding matrix.

Mathematically, given an embedding matrix A, we first initialize the blocks by frequency grouping, and then jointly learn both the clustering $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_c$ and low-rank embeddings for each block U^p, V^p simultaneously by minimizing the following clustering objective:

$$\min_{\{\mathcal{V}_p\}_{p=1}^c, \{U^p\}_{p=1}^c, \{V^p\}_{p=1}^c} \sum_{p=1}^{c} \|Q_{\mathcal{V}_p} A_{\mathcal{V}_p} - Q_{\mathcal{V}_p} U^p (V^p)^T\|_F^2, \tag{2.5}$$

where $Q_{\mathcal{V}_p} = \text{diag}_{j \in \mathcal{V}_p}(\sqrt{q_1}, \ldots, \sqrt{q_j})$. Intuitively, the inner part aims to minimize the weighted low-rank approximation error for one cluster, and outer sum is searching for the partitions so as to minimize the overall reconstruction error.

Optimization: Eq.(2.5) is non-convex. In this paper, we use alternating minimization to minimize the above objective. When fixing the clusters assignment, we use weighted SVD to solve for U^p and V^p for each $A_{\mathcal{V}_p}$. To solve for U^p and V^p, as mentioned above in Eq(2.2), we can perform SVD over $Q_{\mathcal{V}_p} A_{\mathcal{V}_p}$ to obtain the approximation. The time complexity is the same with traditional SVD on $A_{\mathcal{V}_p}$.

To find the clustering structure, we first initialize the clustering assignment by frequency,

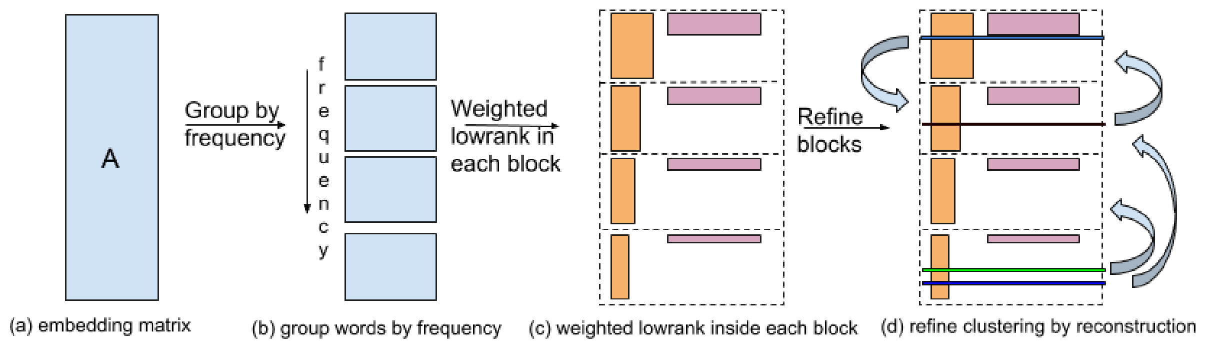

A
Group by frequency
frequency
Weighted lowrank in each block
Refine blocks
(a) embedding matrix
(b) group words by frequency
(c) weighted lowrank inside each block
(d) refine clustering by reconstruction

Figure 2.2: Illustration of our method. Given an embedding matrix A in (a), we first group the words by their frequency (step (b)), and then perform weighted-SVD inside each group as shown in Eq.2.2(step (c)). Finally we refine the clustering by considering the low-rank reconstruction error of words as in Eq.2.5(step (d)).

and then refine the block structure by moving words from one cluster to another cluster if the moves can decrease the reconstruction error Eq (2.5). To compute the reconstruction error reduction, we will project each A_i into each basis V^p and see how much reconstruction error will improve. So if

$$\|A_i - V^p(V^p)^T A_i\| > \|A_i - V^{\bar{p}}(V^{\bar{p}})^T A_i\|, \tag{2.6}$$

then we will move i-th word A_i from the p-th cluster to the $\bar{p}$-th cluster. By this strategy, we will decrease the restructure error.

The overall algorithm, GroupReduce is in Figure (2.2) illustrates our overall algorithm. First, we group the words into c blocks based on frequency. After that, we perform weighted lowrank approximation Eq (2.2) for each block, and then solve Eq (2.5) to iteratively refine the clusters and obtain block-wise approximation based on reconstruction error.

There are some implementation details for Algorithm A.1. After initial grouping, we assign different ranks to different blocks based on the average frequency of words inside that cluster—the rank k_p for block p is proportional to the average frequency of words inside that cluster. Suppose the block with smallest frequency is assigned with rank r, then the rank of cluster p is $\frac{f_p}{f_c}r$, where f_c is the average frequency for the block with least frequency words. r

Algorithm 2.1: GroupReduce: Block-Wise Low-Rank Approximation for Neural Language Model Shrinking

```
Input: Embedding matrix $A$; number of clusters $c$; the smallest rank $r$; the
       maximal number of iterations $t_{max}$; minimal size of the candidate set $m_{min}$;
Output: Compact representation $\bar{A}$
1  Initialize clusters of words as $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_c$ by clustering on the frequency of words;
2  Compute the desired rank for each cluster based on the average frequency for that
     cluster and $r$;
3  for $p = 1, \cdots, c$ do
4      Compute the rank-$k_p$ weighted lowrank for each sub-matrix $A_{\mathcal{V}_p}$ as
         $A_{\mathcal{V}_p} \approx U^p (V^p)^T$;
5  for $t = 1, \cdots, t_{max}$ do
6      $M = []$;
7      for $i = 1, \cdots, N$ do
8          Compute the reconstruction error for $i$-th word $A_i$,
             $e^i = min_{p=1 \cdots c} \|A_i - V^p (V^p)^T A_i\|_2^2$ ;
9          Find the cluster with smallest reconstruction error $g_i : min_{p=1 \cdots c} e^i_p$;
10         if $g_i \neq \pi_i$ ($\pi_i$ is the original cluster index for i-th word) then
11             put $i$ into the candidate set $M$;
12     Choose the top $m$ words in $M$ that with least reconstruction error;
13     move $m$ words (we choose 10% in the paper) into clusters with smallest
         reconstruction error;
14     if $m < m_{min}$ then
15         Stop and output;
16     for $p = 1, \cdots, c$ do
17         if Cluster $\mathcal{V}_p$ changes then
18             Compute the rank-$k_p$ weighted lowrank from Eq (2.2) for each sub-matrix
                 $A_{\mathcal{V}_p}$ as $A_{\mathcal{V}_p} \approx U^p (V^p)^T$;
19 Output: $\bar{A} = [U^1 (V^1)^T, \cdots, U^c (V^c)^T]$
```

is related to the budget requirement. This dynamic rank assignment can significantly boost the performance, as it assigns more ranks to high-frequency words and approximates them better.

In Table 2.3, we compare the effectiveness of different strategies in our algorithm. We test on PTB-Small setting with statistics shown in Table 2.2. Every method in the table has the same compression rate, and we report perplexity number. We compare using vanilla SVD, weighted SVD, weighted SVD for each block (10 blocks), assigning different ranks for

Table 2.3: PTB-small with 5 blocks and 5 times compression rate. We add the proposed strategies one-by-one to see the effectiveness of each of them using the perplexity as the performance metric. Notice that in practice, when applying GroupReduce, we will keep certain percentage of most frequent words uncompressed. But numbers in this table is obtained without preserving any frequent words.

vanilla SVD	weighted-SVD	block SVD	block weighted-SVD	block weighted-SVD with dynamic rank	refinement
161.44	155.10	143.88	135.19	129.63	127.26

different blocks, and refining the blocks. We can see that all the operations involved can improve the final performance and are necessary for our algorithm. The overall memory usage to represent A after our algorithm is $O(Nk + ckD)$, where N is the vocabulary size; c is the number of clusters; k the average rank of each cluster.

2.5 Experiments

2.5.1 Datasets and Pretrained Models

We evaluate our method (GroupReduce) on two tasks: language modeling (LM) and neural machine translation (NMT). For LM, we evaluate GroupReduce on two datasets: Penn Treebank Bank (PTB) and One-billion-Word Benchmark (OBW). OBW is introduced by [21], and it contains a vocabulary of 793,471 words with the sentences shuffled and the duplicates removed. For NMT, we evaluate our method on the IWSLT 2014 German-to-English translation task [19]. On these three benchmark datasets, we compress four models with the models details shown in Table 2.2. All four models use a 2-layer LSTM. Two of them (OBW and NMT) are based on exiting model checkpoints and the other two (based on PTB) are trained from scratch due to the lack of publicly released model checkpoint.

We train a 2-layer LSTM-based language model on PTB from scratch with two setups: PTB-Small and PTB-Large. The LSTM hidden state sizes are 200 for PTB-Small and 1500 for PTB-Large, so are their embedding sizes. For OBW, we use the "2-LAYER LSTM-8192-1024" model shown in Table 1 of [76]. For NMT, we use the PyTorch checkpoint provided by OpenNMT [82] to perform German to English translation tasks. We verified that all these four models achieved benchmark performance on the corresponding datasets as reported in the literature. We then apply our method to compress these benchmark models.

For experiments using BLEU scores as performance measure, we report results when the BLEU scores achieved after compression is within 3 percent difference from original score. For experiments using perplexity (PPL) as measure such as PTB dataset, we target 3 percent drop of performance too. For OBW dataset, since it has larger vocaburary size, we report results within 10 percent difference from original PPL. For each method in Table 2.4, 3.3 and 2.6, we tested various parameters and report the smallest model size of the compression fulfilling above criteria. Certainly, the compression rate and corresponding performance will be a spectrum. The more we compress, the larger the performance drop. We plot this trade-off on PTB-Large in the supplementary. Number of clusters will impact the compression rate. In the experiment, we set the number of clusters to be 5 for PTB and IWSLT datasets, and 20 for the OBW dataset. We show the performance of GroupReduce with different numbers of clusters under the PTB-Large setting in the supplementary.

Note that the goal of this work is to compress an existing model to a significantly-reduced size while maintaining accuracy (e.g., perplexity or BLEU scores), rather than attempting to achieve higher accuracy. It is possible that there are models that could achieve higher accuracy, in which case our method can be applied to compress these models as well.

2.5.2 Comparison with Low-Rank and Pruning

We compare GroupReduce with two standard model compression strategies: low-rank approximation and pruning.These two techniques are widely used for language model compression, such as [99, 100, 118] We compress both input embedding and softmax matrices. For the low-rank approximation approach, we perform standard SVD on the embedding and softmax matrices and obtain the low-rank approximation. For pruning, we set the entires whose magnitude is less than a certain threshold to zero. Note that storing the sparse matrix requires to use the Compressed Sparse Row or Compressed Sparse Column format, the memory usage is thus 2 times the number of non-zeros in the matrix after pruning. After

Table 2.4: Embedding compression results on three datasets comparing our method GroupReduce with Low-rank and Pruning. Compression rate is compared to both input embedding and softmax layer. For example, 10x means approximated embedding uses 10 times smaller memory compared to original input layer and softmax layer.

Model	Metric	Original	Low-rank	Pruning	GroupReduce
PTB-Small	Embedding Memory	1x	2x	2x	5x
	PPL(before retrain)	112.28	117.11	115.9	115.24
	PPL(after retrain)	–	113.83	113.78	113.77
PTB-Large	Embedding Memory	1x	5x	3.3x	10x
	PPL(before retrain)	78.32	84.63	84.23	82.86
	PPL(after retrain)	–	80.04	78.38	79.16
OBW-bigLSTM	Embedding Memory	1x	2x	1.14x	6.6x
	PPL(before retrain)	31.04	39.41	128.31	32.47
	PPL(after retrain)	–	38.03	84.11	32.50
NMT: DE-EN	Embedding Memory	1x	3.3x	3.3x	10x
	BLEU(before retrain)	30.33	29.65	25.96	29.48
	BLEU(after retrain)	–	29.96	29.34	29.96

approximation, we retrain the rest of parameters by SGD optimizer with initial learning rate 0.1. Whenever, the validation perplexity does not drop down, we decrease the learning rate to an order smaller. As shown in Table 2.4, GroupReduce can compress both the input embedding and softmax layer 5-10 times without losing much accuracy. In particular, GroupReduce compress 6.6 times on the language model trained on OBW benchmark, which saves more than 5 GB memory.

Notice that GroupReduce achieves good results even before retraining. This is important as retraining might be infeasible or take a long time to converge. We experimented with different learning rates and retrained for 100k steps (about 3 hours), but we observe that all the retraining scheme of OBW-bigLSTM model after approximation do not lead to significant improvement on accuracy. One reason is that to retrain the model, we need to keep the approximated embedding matrices fixed and re-initialize other parameters, and train these parameters from scratch as done in [134]. On OBW-bigLSTM, it will take more than 3 weeks for the retraining process. It is not practical if the goal is to compress model within a short period of time. Therefore, performance before retraining is important and GroupReduce in general obtains good results.

2.5.3 Comparison with Quantization

As noted in the related work, quantization has been shown to be a competent method in model compression [64]. We implement b-bit quantization by equally spacing the range of a matrix into 2^b intervals and use one value to represent each interval. For example, 4-bit quantization will transform original matrix into matrix with 16 distinct values.

We need to point out that quantization is not orthogonal to other methods. In fact, GroupReduce can be combined with quantization to achieve a better compression rate. We firstly approximate the embedding or the softmax matrices by GroupReduce to obtain low rank matrices of each block, and then apply 4 or 8 bits quantization on these low rank matrices. After retraining, quantized GroupReduce could achieve at least 26 times compression for both input embedding and softmax matrix in OBW as shown in Table 3.3. In addition, comparisons to other coding schemes including deep compositional coding [134] and dictionary coding [29] are shown in the supplementary.

2.5.4 Overall Compression

Results above have shown GroupReduce is an effective compression method when the frequency information is given. We need to point out that part of the model (e.g., LSTM cells) cannot leverage this information as the transition matrices in LSTM cell do not correspond to the representation of a word. We adopt simple quantized low-rank approximation of LSTM to compress this part. Specifically, we first compute SVD of LSTM matrix to obtain 2 times compression, and quantize the entries of low-rank matrices by using only 16 bits. In total the model would be 4 times smaller. However, we found out for OBW-bigLSTM model, LSTM matrix does not have a clear low-rank structure. Even slight compression of LSTM part will cause performance significantly drop. Therefore, we only apply 16-bit quantization on OBW-bigLSTM to have a 2 times compression on LSTM cells. Overall compression rate is shown in Table 2.6. With the aid of GroupReduce, we can achieve over 10 times compression on both language modeling and neural machine translation task.

Table 2.5: Embedding compression results on three datasets comparing our method Quantized GroupReduce with traditional Quantization. 10x means approximated embedding uses 10 times smaller memory compared to original input embedding layer and softmax layer.

Model	Metric	Original	Quantization	Quantized GroupReduce
PTB-Small	Embedding Memory	1x	8x	40x
	PPL(before retrain)	112.28	132.5	146.59
	PPL(after retrain)	–	112.94	112.45
PTB-Large	Embedding Memory	1x	8x	40x
	PPL(before retrain)	78.32	116.54	88.67
	PPL(after retrain)	–	80.72	80.68
OBW-bigLSTM	Embedding Memory	1x	4x	26x
	PPL(before retrain)	31.04	32.63	34.43
	PPL(after retrain)	–	33.86	33.60
NMT: DE-EN	Embedding Memory	1x	8x	24x
	BLEU(before retrain)	30.33	27.41	29.29
	BLEU(after retrain)	–	30.19	29.65

Table 2.6: Compression rate of overall model compression using Quantized GroupReduce. Compression rate shown in the column 4-6 is compared to the corresponding part of the model.

Models	Original PPL/BLEU	PPL/BLEU after approximation	input layer	softmax layer	LSTM cell	Overall Compression
NMT: DE-EN	30.33(BLEU)	29.68(BLEU)	24x (45.9%)	24x(31.8%)	4x(22.3%)	11.3x
OBW-BigLSTM	31.04(PPL)	33.61(PPL)	26x (45.6%)	26x(45.6%)	2x(8.8%)	12.8x

2.6 Summary

In this section, we demonstrate the power of leveraging data information to achieve the efficient machine learning. Specifically, we propose a novel compression method for neural language models by incorporating the statistical property of words in language to form block-wise low-rank matrix approximations for embedding and softmax layers. Experimental results show that our method can significantly outperform traditional compression methods such as low-rank approximation and pruning. In particular, on the OBW dataset, our method combined with quantization achieves 26 times compression rate for both the embedding and softmax matrices, which saves more than 5GB memory usage. It provides practical benefits when deploying neural language models on memory-constrained devices, which is a very important property of efficient machine learning.

CHAPTER 3

Multiplicative Multi-way Model with Frequency Information for Further Model Compression

3.1 Introduction

In chapter 2, we demonstrated that data information, specifically frequency information, could help to greatly improve the performance of vanilla low-rank method. In fact, the power of this information is that it can be combined with various other efficient model compression methods. In addition to the computationally inspired low-rank method, there are some semantically meaningful compositional methods such as deep compositional models [28, 134] which has shown to be an effective model compression method. In this section, we propose MulCode, a novel and effective deep neural compressor which combines the good part of compositional method and frequency data information. MulCode uses a multiplicative composition instead of addition. The multiplicative factors introduce a larger capacity empowering the encoding of more complicated semantic. From the perspective of semantic composition, this allows introducing new information to the base codebook vectors (sub-elements to compose a word vector), taking frequency information of each token into consideration. Technically, MulCode embraces the research line of dense matrix decomposition that has been investigated in many applications [111, 121]. It trains the composition with weighted loss proportional to the frequency of each word. MulCode also adopts an adaptive way of constructing codebooks based on the frequency of each word.

3.2 Related Work

This section also focused on model compression. Thus, a majority of related literature survey is basically the same as the related work section in chapter 2. Please refer to section 2.2 for a complete review of existing model compression methods. The remaining of the section will review the literature on neural compositional models. In essence, neural compositional models represent the original word vector with a set of discrete pseudo words and thus decompose the word vector to the addition of the corresponding pseudo word vectors learned in an end-to-end manner. These methods are seen as embracing the long-existing yet still, the most popular assumption [42] on compositionality of semantics which states that the meaning of an element (e.g., a word, phrase, or sentence) can be obtained by taking the addition of its constituent parts.

Additive Composition One of the most popular assumptions about the composition of semantics is called the additive composition stating that the meaning of a unit (e.g., word, phrase, or sentences) can be obtained by summing up the meaning of its constituents. At the word level, a word might be decomposed into a set of subword units. For example, "disagree" = "dis"+"agree". Alternatively, a word can also be represented by a set of relevant words, e.g., "king" = "man" + "crown". [51] has validated this particular assumption for a popular word embedding approach (i.e., Skip-Gram [112]).

Based on this assumption, [29] created a codebook by splitting the vocabulary into two disjoint sets based on the word frequency: the most frequent words and the rest. Less frequent words are represented using a sparse linear combination of the vectors of more frequent words. Instead of using an explicit set of words, [134] designed the codebooks in a more data-driven fashion where the selection of pseudo words and code vectors are learned automatically using the Gumbel-Softmax trick [67]. Following the same approach, [28] propose additional training objective to integrate the learning of discrete codes with the training of the language model. Our method, based on the multiplicative composition rather than addition, follows this research line. The effectiveness of multiplicative composition is

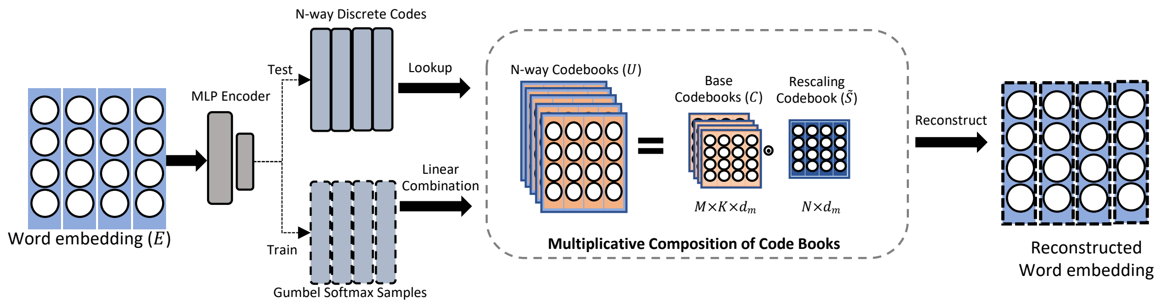


Figure 3.1: Overall architecture of the multi-way multiplicative compressor

verified in some language modeling tasks [113, 114].

3.3 Method

3.3.1 Multi-way Multiplicative Codes

Compositional models start with defining a set of d_m dimensional vectors, which serve as basic codes to compose the targeted embedding matrix. These vectors could further be separated into M groups and each group contains K vectors. We call each group a codebook, and each d_m dimensional vector in the codebook a codeword.

An 1-way codebook then is defined as a $R^{1 \times M \times K \times d_m}$ tensor. Correspondingly, we define N-way codebooks as $U \in R^{N \times M \times K \times d_m}$, where each of the N ways consists of a set of M codebooks, each codebook contains K words, and each codeword is associated with a d_m dimensional vector representing its semantics.

Since U is a high-order tensor which is not memory-friendly, we model the composition of U as applying a customized multiplicative operator on two tensors $C \in R^{M \times K \times d_m}$ and $S \in R^{N \times M \times d_m}$ as

$$U = C \odot S,$$

where $\odot$ is a multiplicative operator defining that

$$U_{i,j,k} = C_{j,k} \circ \sigma(S_{i,j}),$$
$$\forall i \in [1, N], j \in [1,M], \text{and}\, k \in [1, K],$$

$\sigma(\cdot)$ stands for the tangent hyperbolic function, and $\circ$ is the Hadamard product (entry-wise product). Tensor C is referred to as the base codebook, consisting of M codebooks where each codebook contains K codeword vectors of d_m dimensions. S is called rescaling codebooks. Each codeword in the base codebook is injected with new meanings by a code vector in the rescaling codebook.

Despite rescaling codebook uses much less memory (O(NMd_m)) than simply creating a N times larger set of vectors (O($NMKd_m$)), using rescaling codebook still introduces additional costs of the memory. We propose to further reduce the cost by allowing rescaling code vectors to be shared. We replace S with a $\tilde{S} \in R^{N\times d_m}$. That is, for each of the N way codebook, we use only one d_m-dimensional vector to rescale the base codebook C. The number of parameters in S can thus be reduced and the computation of U becomes

$$U_{i,j,k} = C_{j,k} \circ \sigma(\tilde{S}_i),$$
$$\forall i \in [1, N], j \in [1,M], \text{and}\, k \in [1, K].$$

Now, with the N-way codebook defined by C and $\tilde{S}$, given an embedding matrix E with V vocabularies (i.e. $E \in R^{V\times d_m}$), we could represent E by finding an encoding $Q \in R^{V\times N\times M}$ to compose E from C and $\tilde{S}$. Q_i contains encoding of corresponding vocabulary V_i. From each of N-way n, and each of M codebooks m, $Q_{i,n,m}$ indicates which codeword to compose.

Let the word vector for the ith word be e_i. We could construct e_i by

$$\begin{aligned} e_i \approx \hat{e}_i &= \sum_{n=1}^{N}\sum_{m=1}^{M} U_{n,m,Q_{i,n,m}} \\ &= \sum_{n=1}^{N}\sum_{m=1}^{M} C_{m,Q_{i,n,m}} \odot \sigma(\tilde{S}_n). \end{aligned}$$

The N-way discrete code could be learned in an end-to-end manner by using the Gumbel-Softmax trick [67]. We first compute an encoding vector for the original word vector e_i by feeding it to a neural network

$$a_i = Softmax(\sigma(W^\top \phi(W'^\top e_i + b) + b')),$$

where W, W', b and b' are the parameters of the network, and ϕ is the softplus function. a_i represents a real value tensor in a shape $N \times M \times K$, which is then fed to the Gumbel-Softmax to generate a continuous approximation of drawing discrete samples with respect to the last dimension

$$\hat{D}_i = Softmax((\log a_i + g)/\tau),$$

where g is a random noise vector sampled from Gumbel distribution, and τ is the Softmax temperature controlling how close is the sample vector to a uniform distribution. $\hat{D}_i$ is an approximately one-hot out of K drawing for each way N and codebook M. We can then compute approximation of each dimension of e_i as

$$\hat{e}_{i,d} = U_{:,:,:,d} \odot \hat{D}_i$$

Note that during training $\hat{D}_i$ is generated as a continuous approximation of the N-way discrete code. At the testing phase, the fixed encoding Q_i of vector e_i is directly computed as $Q_{i,n,m} = \arg\max_k a_{n,m,k}$. More details on Gumbel-trick could be found in [67].

3.3.2 Group Adaptive Coding

Given the rescaling codebooks $\tilde{S}$ is small, the memory consumption mainly consists of two parts: code vectors C and discrete codes Q. On the one hand, the code vectors normally account for the major memory usage when dealing with relatively smaller vocabulary size. On the other hand, the size of discrete codes grows linearly with the size of vocabulary and the logarithm of K. To further reduce the memory usage, we propose to use codes of adaptive length and dimensions to deal with the linear dependency with vocabulary size. The intuition is to encode frequent words with a longer code length to achieve a relative lower reconstruction loss while representing rare words with fewer codes. To achieve this, we first sort the words according to their frequency and then split words into fixed number of groups G. The ith group could access only $\gamma_{G_i} = M \times (1 - \frac{i-1}{|G|})$ codebooks.

In the same time, we store the low-rank version of code vectors for some codebooks that were mainly used for representing rare words.

$$C_{i,j} = W_{G_i} c_i,$$

where $c_i \in R^{d_{G_i}}$ and $W_{G_i} \in R^{d_m \times d_{G_i}}$ is the linear transformation matrix to be shared by all the codebooks in the ith group. We resolve rank d_{G_i} in an intuitive way as shown in Algorithm 3.1.

In practice, high frequency words tend to get accesses to codebooks with higher rank while less frequent words can only access codebooks of lower dimension. Thanks to the long tail of rare words, this could actually help save considerable memory space. Normally, using a low compression rate results in $d_{G_i} \ll \gamma_{G_i} \times K$ so that it is guaranteed to reduce the number of parameters in the base codebook.

3.3.3 Prior-weighted Reconstruction Loss

According to Zipf's law, the frequency of words conforms to a power law distribution. It means that word which falls to the long tail may rarely appear in the sentence. It motivates

Algorithm 3.1: Algorithm to resolve the dimension of adaptive codebooks towards achieving a targeted compression rate.

```
1  Sort set G according to frequency;
2  $d'$: the minimum dimension ;
3  $\delta$ : targeted compression rate;
4  $f_{G_i}$ : frequency of group $G_i$;
5  $o_{G_i}$ : bits consumed by discrete code matrix $Q$ for each word in $G_i$;
6  $n \leftarrow \delta \times V \times d_m \times 32(bits)$ ;
7  for $i \leftarrow 1$ to $|G|$ do
8      compute a ratio based on frequency $\Delta_{G_i} \leftarrow \frac{f_{G_i}}{\sum_{i=1}^{|G|} f_{G_i}}$;
9      calculate the dimension;
10     $d_{G_i} \leftarrow (n - o_{G_i}) \times \Delta_{G_i}/(\gamma_{G_i} + d_m)$;
11     ensure dimension is valid within $(d', d_m)$;
12     $d_{G_i} \leftarrow \min(\max(d_{G_i}, d'), d_m)$;
13     update $n$ by subtracting used bits;
14     $n \leftarrow n - o_{G_i} - (\gamma_{G_i} - \gamma_{G_{i+1}} + d_m) \times d_{G_i}$
15 end
```

the modification on the learning objective so that the compressor will be more focused on words with high frequency. In this paper, we use the distribution of the words in the training set as a prior knowledge to guide the learning of the compressor. In addition, similar to [96] we also want to let each of the N way encoding focus on different aspects of the original word embedding. The proposed training objective function then becomes

$$\begin{aligned}\mathcal{L} = &\sum_{i=1}^{V}(-\log \hat{p}_i \|\hat{e}_i - e_i\|_2^2 \\ &+ \lambda\|\hat{v}_i\hat{v}_i^\top - I\|_2^2 + \lambda\|\hat{v}_i^\top\hat{v}_i - I\|_2^2),\end{aligned}$$

where $\hat{p}_i$ represents the empirical distribution of word V_i in the training set, e_i is original word vector, $\hat{e}_i$ is the reconstructed vector and $\hat{v}_i \in R^{N\times d_m}$ is a compilation of reconstrcuted vectors from all N-way codebooks in a matrix form (i.e. $\hat{v}_{i,n} = \sum_{m=1}^{M} C_{m,Q_{i,n,m}} \odot \sigma(\tilde{S}_n)$ is the reconstructed vector from nth-way codebook). The new objective function thus focuses on high-frequency words and in the meanwhile allows the N-way coding to encode different information.

Table 3.1: Statistics of data sets. The number in parenthesis shows the ratio of embedding layers (input plus Softmax) respective to the entire model size

Models	Vocab. Size	Dimension	Model Size
PTB-small	10k	200	17.7MB(85.8%)
PTB-large	10k	1500	251MB(45.4%)
OBW	793k	1024	6.8GB(91.2%)
NMT: DE-EN	30k	500	195MB(83.1%)

3.4 Experiment

3.4.1 Data Sets and Models

Following the experimental protocol of [25], we evaluate our proposed method with two important NLP tasks: language modeling and machine translation. Table 3.1 summarizes the key characteristics of the four data sets and models. PTB-small is using 2-layer LSTM-based language model built on Penn Tree Bank (PTB) data set. The vocabulary size is 10k. Input embedding layer and output Softmax layer are both set to 200 dimensions. PTB-large is trained on the same PTB data set as PTB-small except the dimensions for input and output are increased to 1500. Neural machine translation (NMT: DE-EN) is a Seq2seq model initialized using Pytorch checkpoints provided by Open-NMT [81]. The model is to perform German to English translation tasks on IWSLT-14 [19] data set. One billion words (OBW) uses a 2-layer LSTM (referred to as ”2-LAYER LSTM-8192-1024” by [76]) trained on the OBW data set and the vocabulary size is 793,471. For LSTM-based language models, the input embedding matrix and Softmax embedding matrix account for the major memory usage (up to 91.2%). Therefore, we target compressing both the input and Softmax embedding matrices.

3.4.2 Implementation Details

We compress both input embedding and Softmax matrices. We trained MulCode by using Adam optimizer with learning rate 0.001. For PTB data set, we group the vocabulary using 3 groups and 8 groups for OBW. To resolve the dimension of codebooks for adaptive coding, we use targeted compression rate $\delta = 0.2$ for PTB-small and $\delta = 0.05$ for the rest three models[1].

After approximation, we retrain the rest of parameters by SGD optimizer with initial learning rate 0.01. Whenever the validation perplexity does not drop down, we decrease the learning rate to an order smaller. We did not include results of fine-tuning on OBW for that the re-training process takes too long (few days) which is not compliant with our motivation to compress the given pre-trained embeddings.

The compression rate and corresponding performance could certainly be plotted as a spectrum graph. The more we compress, the larger the performance drop. In this paper, as far as BLEU score is concerned, we report results of compressed models when the BLEU falls within 3 percent difference from the original score. For PTB data set, we target 3 percent drop of perplexity (PPL) after retrain. For OBW data set, since it has a larger vocabulary size, we report results within 10 percent difference from the PPL achieved by the uncompressed model. For each method we tested various parameters and report the smallest model size of the compression fulfilling above criteria.

Notice that some previous methods compress model directly during training phase [78, 153]. In contrast, our problem setup follows [25, 28, 134] that given a pre-trained model, we want to compress the model with limited fine-tuning.

3.4.3 Comparison with Baseline Models

We refer to our proposed method as MulCode (Mul stands for both multi-way and multiplicative composition). We mainly compare with two state-of-the-art baseline compressors targeting compressing the embedding layer.

1. **GroupReduce** We refer to the results reported in [25]. Also note that GroupReduce is the method introduced in chapter 2.

[1]The original language models can be downloaded from `https://github.com/mayktian/MulCode.git`

2. **DeepCode** The additive composition model by [134]. We use the pytorch code[2] released by [134] to produce the results.

Table 3.2: Compressed model evaluation with 3 language models and 1 machine translation model. Memory usage is reported as compared with the original input embedding and Softmax embedding. For example, 30x means the compressed embeddings use only 1/30 the memory space of the original embedding.

Model	Metric	Original	GroupReduce	DeepCode	MulCode
PTB-small	Memory	1X	4X	3.9X	6.5X
	PPL(before retrain)	112.28	115.38	120.05	115.38
	PPL(after retrain)	-	113.81	115.57	113.34
PTB-large	Memory	1X	8X	3.8X	17.1X
	PPL(before retrain)	78.32	84.79	84.73	83.85
	PPL(after retrain)	-	79.83	81.80	79.89
NMT: DE-EN	Memory	1X	8X	5.7X	17X
	BLEU(before retrain)	30.33	29.31	26.58	29.48
	BLEU(after retrain)	-	29.96	29.37	30.08
OBW	Memory	1X	6.6X	5.7X	18.2X
	PPL	31.04	32.47	99.59	32.66

Table 3.2 summarizes the comparison between the proposed methods and state-of-the-art baselines for the four benchmark data sets and LSTM models. MulCode manages to compress the input embedding layer and Softmax embedding layer 6 to 18 times without suffering a significant loss in the performance.

In comparison, all the baseline models achieve much lower compression rate with PTB-small which has only 200 dimensions. It is reasonable since embedding layers of PTB-small contains less redundant information and thus can be hardly compressed. As compared with DeepCode, MulCode achieves much higher compression rate[3] for all the four models. MulCode also consistently and significantly outperforms GroupReduce.

Table 3.3: Results of compressing all input and Softmax embedding layers on three data sets. 28.2x means approximated embedding uses 28.2 times smaller memory compared to original input embedding layer and Softmax layer..

Model	Metric	Original	Quantization	Quantized MulCode
PTB-small	Memory	1X	6.4X	16.3X
	PPL(before retrain)	112.28	115.81	116.33
	PPL(after retrain)	-	114.14	113.34
PTB-large	Memory	1X	6.4X	28.2X
	PPL(before retrain)	78.32	81.69	81.55
	PPL(after retrain)	-	79.22	78.91
NMT: ED-EN	Memory	1X	6.4X	41.38X
	BLEU(before retrain)	30.33	27.41	29.32
	BLEU(after retrain)	-	30.19	29.94
OBW	Memory	1X	6.4X	30.8X
	PPL(before retrain)	31.04	32.63	34.36

3.4.4 Comparison with Quantization

Quantization has been proven to be a strong baseline. In fact, the discrete coding of MulCode can be considered equivalent to a trainable quantization. On the other hand, we need to point out that quantization is not orthogonal to MulCode. MulCode could be combined with quantization to achieve better performance. Specifically, the $M \times K$ base codebooks as well as the rescaling codebook could be quantized to further reduce the memory usage. We summarize the results in Table 3.3. Quantized MulCode could achieve more than 30.8 times compression for both input embedding and Softmax matrix in OBW. In addition, on machine translation task, it achieves 41.38X with BLEU score drops around only 1% after retraining. In particular, we observe that the effect of retraining is more prominent for MulCode and simple quantization compared to GroupReduce. This implies that local precision lost for low-rank basis in GroupReduce is more difficult to be recovered. In contrast, the collective information of MulCode due to the compositional property is more robust when imprecise local vectors present.

[2]`https://github.com/zomux/neuralcompressor`

[3]Aligned with what has been mentioned by [134] in their rebuttal, we found the DeepCode unable to work with the Softmax layer of OBW no matter how hard we have tuned it (99.5 as shown in Table 3.2).

3.4.5 Selection of M,N and K

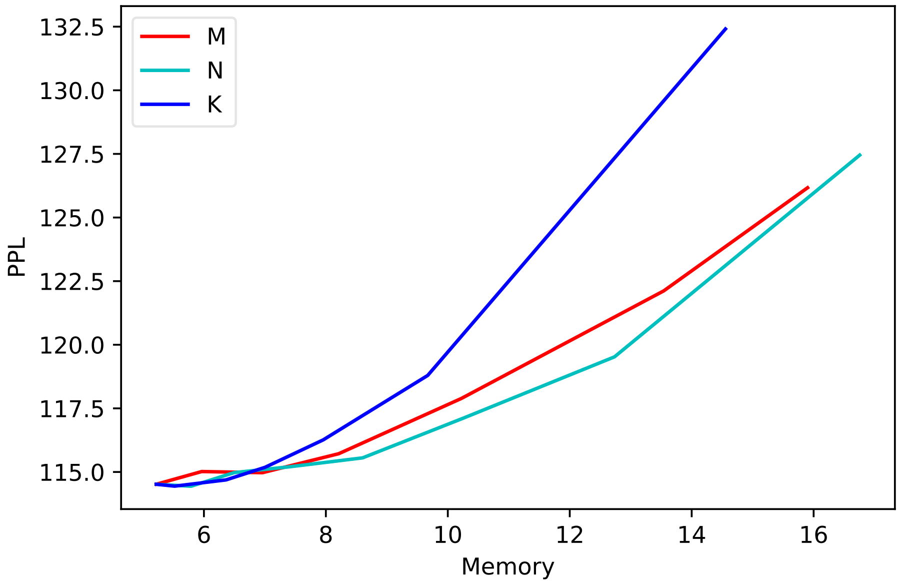


Figure 3.2: Influence of M, N, K for PTB-small.

To understand how the choices of M, N, K would affect the performance of the compressed model, we test the proposed model for PTB-small with varying setting of M, N, and K. The departure point of this experiment is using $M = 32, K = 32, N = 8$. We then adjust the three parameters one at a time while fixing the other two. In order to plot the results in a single figure, we set the x-axis as the memory usage instead of different values of the three parameters. As shown in Figure 3.2, adjusting the values of M, N, and K have similar effects on the PPL when the compression rate is low ($\leq$7X). With a larger compression rate, the performance becomes much more sensitive to the change of K. It suggests that it is safer to maintain a high value for K while tuning the rest two parameters for the purpose of securing a reasonable performance of the compressed model.

3.4.6 Understanding Composed Codes

At the core of our approach is the multiplicative composition from two sets of codebooks. Hence, it is interesting to investigate what has been encoded in the N-way codebooks. One assumption is that each code in the base codebooks encodes a mix of information which can be disentangled by the rescaling codebook. We encode all the words of OBW corpus by a $32 \times 8 \times 4$ codes. We compute the hamming distance of example query words (shown in Table 3.4) for each of the N-way codes. We select the top ones with the smallest hamming distance from the 10,000 most frequent words that are likely to have low reconstruction errors.

Word	1	2	3	4
tomorrow	today	Sunday	prompt	Tuesday
beautiful	elegant	iconic	great	fields
soccer	football	hockey	Ghana	basketball
where	when	experiencing	who	learn
bank	company	police	companies	banks
halt	stop	halted	afternoon	cover
like	just	Like	such	is

Table 3.4: Most similar word computed using Hamming distance for each of the N-way codings.

Since the N-way codings are generated by selecting from the base codebooks and modified by rescaling codebooks, each channel can be seen as meaningful subspace. It shows that each of the N-way codings might have encoded a different subspace of the original meaning of words, including tenses (e.g., halt v.s. halted), plurals (e.g., bank v.s. banks), synonyms (e.g., soccer v.s. football), co-occurrence (like v.s. just), topical relatedness (soccer v.s. hockey). It verifies that the multiplicative composition used in our approach is able to introduce new information to the base codebook.

3.5 Summary

In this section, we propose MulCode, a novel compression method for neural language models. MulCode applies multiplicative factors on end-to-end learned codebooks. MulCode considers the frequency information in the corpus by adding weighted loss according to the importance. At the same time, the coding scheme is made adaptive based on the frequency information. Experimental results show MulCode outperforms the state-of-the-art compression methods by a large margin. In particular, on the IWSLT-14 data set, MulCode combined with quantization achieves 41.38 times compression rate for both the embedding and Softmax matrices. MulCode is yet another demonstration that frequency information could greatly help to improve the compression results. Data information will facilitate deployment of large neural language models on memory-constrained devices.

CHAPTER 4

Fast Softmax Inference on Large Vocabulary Neural Networks via Learning to Screen Latent Distribution

4.1 Introduction

In this section, we will switch gear to discuss another important factor of efficient machine learning: inference time. In particular, we will demonstrate the usefulness of "hidden input structure" in this section. In previous sections, we achieve a much smaller model size by a simple yet powerful observation: frequency information is significant in achieving efficient machine learning. It's an **explicit statistic** related to the underlying data, and we can easily compute it. However, there are other types of **"structures"** in the data distribution, which doesn't have an explicit meaning. But it's related to the input variations and could be captured by some method such as clustering. In this section, we will show how clustering can be used to find this **"input structure"** hidden inside the model in order to achieve faster inference time.

In previous sections, we mentioned that neural networks have been widely used in many natural language processing (NLP) tasks, including neural machine translation [143], text summarization [124] and dialogue systems [91]. In these applications, a neural network (e.g. LSTM) summarizes current state by a context vector, and a softmax layer is used to predict the next output word based on this context vector. The softmax layer first computes the "logit" of each word in the vocabulary, defined by the inner product of context vector and weight vector, and then a softmax function is used to transform logits into probabilities. For most applications, only top-k candidates are needed, for example in neural machine translation where k corresponds to the search beam size. In this procedure, the computational complexity of softmax layer is linear in the vocabulary size, which can easily go beyond 10K. Therefore, the softmax layer has become the computational bottleneck in many NLP applications at inference time.

Our goal is to speed up the prediction time of softmax layer. In fact, computing top-k predictions in softmax layer is equivalent to the classical Maximum Inner Product Search (MIPS) problem—given a query, finding k vectors in a database that have the largest inner product values with the query. In neural language model prediction, context vectors are equivalent to queries, and weight vectors are equivalent to the database. MIPS is an important operation in the prediction phase of many machine learning models, and many algorithms have been developed [7, 55, 119, 133, 161]. Surprisingly, when we apply recent MIPS algorithms to LSTM language model prediction, there's not much speedup if we need to achieve $> 98\%$ precision (see experimental section for more details). This motivates our work to develop a new algorithm for fast neural language model prediction.

In natural language, some combinations of words appear very frequently, and when some specific combination appears, it is almost-sure that the prediction should only be within a small subset of vocabulary. This observation leads to the following question: Can we learn a faster "screening" model that identifies a smaller subset of potential predictions based on a query vector? In order to achieve this goal, we need to design a learning algorithm to exploit the distribution of context vectors (queries). This is quite unique compared with previous MIPS algorithms, where most of them only exploit the structure of database (e.g., KD-tree, PCA-tree, or small world graph) instead of utilizing the query distribution.

We propose a novel algorithm (L2S: learning to screen) to exploit the distribution of both context embeddings (queries) and word embeddings (database) to speed up the inference time in softmax layer. To narrow down the search space, we first develop a light-weight screening model to predict the subset of words that are more likely to belong to top-k candidates, and then conduct an exact softmax only within the subset. The algorithm can be illustrated in Figure 4.1.

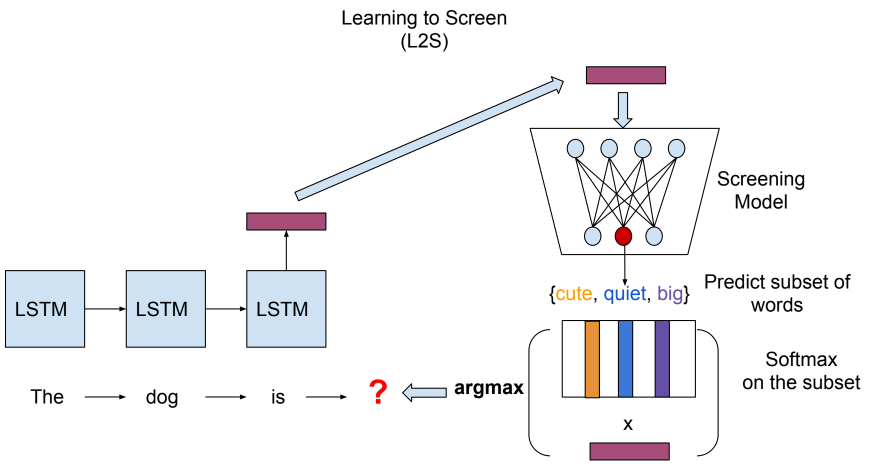


Figure 4.1: Illustration of the proposed algorithm.

In sum, we propose a screening model L2S to exploit the clustering structure of context features. All the previous neural language models only consider partitioning the embedding matrix to exploit the clustering structure of the word embedding to achieve prediction speedup. On the other hand, we consider the hidden space generated by neural network models. To make prediction for a context embedding, after obtaining cluster assignment from screening model, L2S only needs to evaluate a small set of vocabulary in that cluster. Therefore, L2S can significantly reduce the inference time complexity from $O(Ld)$ to $O((r+\bar{L})d)$ with $\bar{L} \ll L$ and $r \ll L$ where d is the context vector' dimension; L is the vocabulary size, r is the number of clusters, and $\bar{L}$ is the average word/candidate size inside clusters. We propose to form a joint optimization problem to learn both screening model for clustering as well as the candidate label set inside each cluster simultaneously. Using the Gumbel trick [68], we are able to train the screening network end-to-end on the training data.

4.2 Related Work

We summarize previous works on speeding up the softmax layer computation.

4.2.1 Algorithms for Speeding up Softmax in the Training Phase

Many approaches have been proposed for speeding up softmax training. [70, 115] proposed importance sampling techniques to select only a small subset as "hard negative samples" to conduct the updates. The hierarchical softmax-based methods [54, 116] use the tree structure for decomposition of the conditional probabilities, constructed based on external word semantic hierarchy or by word frequency. Most hierarchical softmax methods cannot be used to speed up inference time since they only provide a faster way to compute probability for a target word, but not for choosing top-k predictions as they still need to compute the logits for all the words for inference. One exception is the recent work by [54], which constructs the tree structure by putting frequent words in the first layer—so in the prediction phase, if top-k words are found in the first layer, they do not need to go down the tree. We provide comparison with this approach in our experiments.

4.2.2 Algorithms for Maximum Inner Product Search (MIPS)

Here, we only briefly review some important aspects of MIPS. More detailed discussion can be found in chapter 7. Given a query vector and a database with n candidate vectors, MIPS aims to identify a subset of vectors in the database that have top-k inner product values with the query. Top-k softmax can be naturally approximated by conducting MIPS. Here we summarize existing MIPS algorithms:

- Hashing: [119, 133] proposed to reduce MIPS to nearest neighbor search (NNS) and then solve NNS by Locality Sensitive Hashing (LSH) [65].
- Database partitioning: PCA tree [138] partitions the space according to the directions of principal components and shows better performance in practice. [7] shows tree-based approaches can be used for solving MIPS but the performance is poor for high dimensional data.
- Graph-based algorithm: [102, 103] recently developed an NNS algorithm based on small world graph. The main idea is to form a graph with candidate vectors as nodes and

edges are formed between nearby candidate vectors. The query stage can then done by navigating in this graph. [168] applies the MIPS-to-NNS reduction and shows graph-based approach performs well on neural language model prediction.

- Direct solvers for MIPS: Some algorithms are proposed to directly tackle MIPS problem instead of transforming to NNS. [55, 156] use quantization-based approach to approximate candidate set. Another Greedy MIPS algorithm is recently proposed in [161], showing significant improvement over LSH and tree-based approaches.

4.2.3 Algorithms for Speeding up Softmax in Inference Time.

MIPS algorithms can be used to speed up the prediction phase of softmax layer, since we can view context vectors as query vectors and weight vectors as database. In the experiments, we also include the comparisons with hashing-based approach (LSH) [65], partition-based approach (PCA-tree [138]) and Greedy approach [161]. The results show that they perform worse than graph-based approach [168] and are not efficient if we want to keep a high precision.

For NLP tasks, there are two previous attempts to speed up softmax layer prediction time. [131] proposed to approximate the weight matrix in the softmax layer with singular value decomposition, find a smaller candidate set based on the approximate logits, and then do a fine-grained search within the subset. [168] transformed MIPS to NNS and applied graph-based NNS algorithm to speed up softmax. In the experiments, we show our algorithm is faster and more accurate than all these previous algorithms. Although they also have a screening component to select an important subset, our algorithm is able to learn the screening component using training data in an end-to-end manner to achieve better performance.

4.3 Problem Formulation

Softmax layer is the main bottleneck when making prediction in neural language models. We assume L is the number of output tokens, $W \in \mathbb{R}^{d \times L}$ is the weight matrix of the softmax layer, and $b \in \mathbb{R}^L$ is the bias vector. For a given context vector $h \in \mathbb{R}^d$ (such as output of LSTM), softmax layer first computes the logits

$$x_s = w_s^T h + b_s \quad \text{for} \quad s = 1, \cdots, L \tag{4.1}$$

where w_s is the s-th column of W and b_s is the s-th entry of b, and then transform logits into probabilities $p_s = \frac{e^{x_s}}{\sum_{l=1}^{L} e^{x_l}}$ for $s = 1, \cdots, L$. Finally it outputs the top-k candidate set by sorting the probabilities $[p_1, \cdots, p_L]$, and uses this information to perform beam search in translation or predict next word in language model. The efficient machine learning challenge is to speed up this sfotmax process.

4.4 Method

To speedup the computation of top-k candidates, all the previous algorithms try to exploit the structure of $\{w_s\}_{s=1}^{L}$ vectors, such as low-rank, tree partitioning or small world graphs [54, 131, 168]. However, in NLP applications, there exists strong structure of context vectors $\{h\}$ that has not been exploited in previous work. In natural language, some combinations of words appear very frequently, and when some specific combinations appear, the next word should only be within a small subset of vocabulary. This hidden structure is also data dependent and thus we can treat it as a function of data distribution. However, this structure does not have an explicit meaning as we did in previous sections. Consequently there is no simple way to calculate the statistic of such information. A intuitive yet powerful tool to explore structure in data is clustering. The motivation to use clustering to exploit such input dependent structure is illustrated below.

The Prediction Process. Suppose the context vectors are partitioned into r disjoint clusters and similar ones are grouped in the same partition/cluster, if a vector h falls into one of the cluster, we will narrow down to that cluster's label sets and only compute the logits of that label set. This screening model is parameterized by clustering weights $v_1, \ldots, v_r \in \mathbb{R}^d$ and label candidate set for each cluster $c_1, \ldots, c_r \in \{0,1\}^L$. To predict a hidden state h, our algorithm first computes the cluster indicator

$$z(h) = \arg\max_t v_t^T h, \tag{4.2}$$

and then narrows down the search space to $C(h) := \{s \mid c_{z(h),s} = 1\}$. The exact softmax is then computed within the subset $C(h)$ to find the top-k predictions (used in language model) or compute probabilities used for beam search in neural machine translation. As we can see the prediction time includes two steps. The first step has r inner product operations to find the cluster which takes $O(rd)$ time. The second step computes softmax over a subset, which takes $O(\bar{L}d)$ time where $\bar{L}$ ($\bar{L} \ll L$) is the average number of labels in the subsets. Overall the prediction time for a context embedding h is $O((r+\bar{L})d)$, which is much smaller than the $O(Ld)$ complexity using the vanilla softmax layer. Figure 4.1 illustrates the overall prediction process.

However, how to learn the clustering parameter $\{\mathrm{v}_t\}_{t=1}^r$ and the candidate sets $\{\mathrm{c}_t\}_{t=1}^r$? We found that running spherical kmeans on all the context vectors in the training set can lead to reasonable results, but can we learn even parameters to minimize the prediction error? In the following, we propose an end-to-end procedure to learn both context clusters and candidate subsets simultaneously to maximize the performance.

Learning the clustering. Traditional clustering algorithms such as kmeans on Euclidean space or cosine similarity have two drawbacks. First, they are discrete and non-differentiable, and thus hard to use with back-propagation in the end-to-end training process. Second, they only consider clustering on $\{h_i\}_{i=1}^N$, without taking the predicted label space into account. In this paper, we consider learning the partition through Gumbel-softmax trick. We will briefly summarize the technique and direct the reader to [68] for further details on these techniques. In Table 4.4, we compare our proposed method to traditional spherical-kmeans to show that it can further improve the performance.

First, we turn the deterministic clustering in Eq(4.2) into a stochastic process: the probability that h belongs to cluster t is modeled as

$$P(t|h) = \frac{\exp(v_t^T h)}{\sum_{j=1}^{r} \exp(v_j^T h)}, \ \forall t, \quad \text{and } z(h) = \arg\max_t P(t|h). \tag{4.3}$$

However, since argmax is a discrete operation, we cannot combine this operation with final objective function to find out better clustering weight vectors. To overcome this, we can re-parameterize Eq(4.3) using Gumbel trick. Gumbel trick provides an efficient way to draw samples z from the categorical distribution calculated in Eq(4.3):

$$m(h) = \text{one\_hot}(\arg\max_j [g_j + \log P(j|h)]), \tag{4.4}$$

where each g_j is an i.i.d sample drawn from Gumbel$(0, 1)$. We then use the Gumbel softmax with temperature = 1 as a continuous, differentiable approximation to argmax, and generate r-dimensional sample vectors $p = [p_1, \cdots, p_r]$ which is approximately one-hot $m(h)$ with

$$p_t = \frac{\exp(\log(P(t|h)) + g_t)}{\sum_{j=1}^{r} \exp(\log(P(j|h)) + g_j)}, \forall t \in \{1, \ldots, r\}. \tag{4.5}$$

Using the Straight-Through (ST) technique proposed in [68], we denote $\bar{p} = p + (\text{one\_hot}(\arg\max_j p_j) - p)$ as the one-hot representation of p and assume back-propagation only goes through the first term. This enables end-to-end training with the loss function defined in the following section.

Learning the candidate set for each cluster. For a context vector h_i, after getting into the partition t, we will narrow down the search space of labels to a smaller subset. Let c_t be the label vector for t-th cluster, we define the following loss to penalize the mis-match between correct predictions and the candidate set:

$$\ell(h_i, y_i) = \sum_{s:y_{is}=1} (1 - c_{ts})^2 + \lambda \sum_{s:y_{is}=0} (c_{ts})^2, \tag{4.6}$$

where $y_i \in \{0,1\}^L$ is the 'ground truth' label vector for h_i that is computed from the exact softmax. We set y to be the label vector from full softmax because our goal is to approximate full softmax prediction results while having faster inference (same setting with [131, 168]). The loss is designed based on the following intuition: when we narrow down the candidate set, there are two types of loss: 1) When a candidate s ($y_{is} = 1$) is a correct prediction but not in the candidate set ($c_{ts} = 0$), then our algorithm will miss this label. 2) When a candidate j ($y_{is} = 0$) is not a correct prediction but it's in the candidate set ($c_{ts} = 1$), then our algorithm will waste the computation of one vector product. Intuitively, 1) is much worse than 2), so we put a much smaller weight $\lambda \in (0,1)$ on the second term.

The choice of true candidate set in y can be set according to the application. Throughout this paper, we set y to be the correct top-5 prediction (i.e., positions of 5-largest x_s in Eq(4.1). $y_{is} = 1$ means s is within the correct top-5 prediction of h_i, while $y_{is} = 0$ means it's outside the top-5 prediction.

Final objective function: We propose to learn the partition function (parameterized by $\{v_t\}_{t=1}^r$) and the candidate sets ($\{c_t\}_{t=1}^r$) simultaneously. The joint objective function will be:

$$\begin{aligned} \underset{\substack{\mathrm{v}_1,\cdots,v_r \\ c_1,\cdots,c_r}}{\text{minimize}} \quad & \sum_{i=1}^{N}\Big(\sum_{s:y_{is}=1} (1 - c_{\bar{p}(h_i),s})^2 + \lambda \sum_{s:y_{is}=0} (c_{\bar{p}(h_i),s})^2\Big) \\ \text{s.t.} \quad & c_t \in \{0,1\}^L \quad \forall t = 1, \ldots, r \\ & \bar{L} \leq B \quad \forall i = 1, \ldots, N \end{aligned} \tag{4.7}$$

,where N is the number of samples, $\bar{L}$ is the average label size defined as $\bar{L} = \frac{(\sum_{i=1}^{N}\sum_{s=1}^{L} c_{\bar{p}(h_i),s})}{N}$, $\bar{p}(h_i)$ is the index for where $\bar{p}_t = 1$ for $t = 1, \cdots, r$; B is the desired average label/candidate size across different clusters which could be thought as prediction time budget. Since $\bar{L}$ is related to the computation time of proposed method, by enforcing $\bar{L} \leq B$ we can make sure label sets won't grow too large and desired speed-up rate could be achieved. Note that $\bar{p}(h_i)$ is for clustering assignment, and thus a function of clustering parameters $v_1, \cdots, v_r$ as shown in Eq(4.3).

Optimization. To solve the optimization problem in Eq (4.7), we apply alternating minimization. First, when fixing the clustering (parameters $\{\mathrm{v}_t\}$) to update the candidate sets (parameters $\{\mathrm{c}_t\}$), the problem is identical to the classic "Knapsack" problem—each $\mathrm{c}_{t,s}$ is an item, with weight proportional to number of samples belonging to this cluster, and value defined by the loss function of Eq(4.7), and the goal is to maximize the value within weight capacity B. There is no polynomial time solution with respect to r, so we apply a greedy approach to solve it. We sort items by the value-capacity ratio and add them one-by-one until reaching the upper capacity B.

When fixing $\{c_t\}$ and learning $\{v_t\}$, we convert the cluster size constraint to objective function by Lagrange-Multiplier:

$$\begin{aligned} \underset{v_1,\cdots,v_r}{\text{minimize}} \quad & \sum_{i=1}^{N}\Big(\sum_{j:y_{is}=1}(1-c_{\bar{p}(h_i)s})^2+\lambda\sum_{j:y_{is}=0}(c_{\bar{p}(h_i)s})^2\Big)+\gamma\max(0,\bar{L}-B) \qquad (4.8)\\ \text{s.t.} \quad & c_t\in\{0,1\}^L \quad \forall t=1,\ldots,r, \end{aligned}$$

and simply use SGD since back-propagation is available after applying Gumbel trick. To deal with $\bar{L}$ in the mini-batch setting, we replace it by the moving-average, updated at each iteration when we go through a batch of samples. The overall learning algorithm is given in Algorithm A.1.

4.5 Experiments

We evaluate our method on two tasks: Language Modeling (LM) and Neural Machine Translation (NMT). For LM, we use the Penn Treebank Bank (PTB) dataset [108]. For NMT, we use the IWSLT 2014 German-to-English translation task [19] and IWSLT 2015 English-Vietnamese data [101]. All the models use a 2-layer LSTM neural network structure. For IWSLT-14 DE-EN task, we use the PyTorch checkpoint provided by OpenNMT [82]. For IESLT-15 EN-VE task, we set the dimension of hidden size to be 200, and the rest follows the default training hyperparameters of OpenNMT. For PTB, we train a 2-layer LSTM-based language model on PTB from scratch with two setups: PTB-Small and PTB-Large.

Algorithm 4.1: Training Process for Learning to Screen (L2S)

```
  Input: Context vectors $\{h_i\}_{i=1}^N$ (e.g., from LSTM); trained network's softmax
         layer's weight $W$ and basis vector $b$.
  Output: Clustering parameters $v_t$ and candidate label set $c_t$ for each cluster for
          $t = 1, \cdots, r$.
1 Hyperparameter: Number of clusters $r$; prediction time budget $B$; regularization
    terms $\lambda$ and $\gamma$; number of iterations $T$.
2 Compute ground true label vector $\{y_i\}_{i=1}^N$ with only top-$k$ non-zeros entries. The
    top-$k$ labels for each context vector $h_i$ are generated by computing and then
    sorting the values in $x_i = W^T h_i + b$;
3 Initialize cluster weights $\{v_t\}_{t=1}^r$ using spherical kmeans over $\{h_i\}_{i=1}^N$ ;
4 Initialize the label set for each cluster $\{c_t\}_{t=1}^r$ to be zeros;
5 for $j = 1, \cdots, T$ do
6     Fixing $\{c_t\}_{t=1}^r$ and learning the clustering parameters $\{v_t\}_{t=1}^r$ in Eq(4.8) by SGD
        with Gumbel trick;
7     Fixing $\{v_t\}_{t=1}^r$ and learning the labels set $c_t$ $t = 1, \cdots, r$ by solving the
        "Knapsack" problem using Greedy approach;
8 return $c_t$,$v_t$ for all $t = 1, \cdots, r$.
```

The LSTM hidden state sizes are 200 for PTB-Small and 1500 for PTB-Large, so are their embedding sizes. We verified that all these models achieved benchmark performance on the corresponding datasets as reported in the literature. We then apply our method to accelerate the inference of these benchmark models.

4.5.1 Competing Algorithms

We include the following algorithms in our comparisons:

- L2S (Our proposed algorithm): the proposed learning-to-screen method. Number of clusters and average label size across clusters will be the main hyperparameters affecting computational time. We could control the tradeoff of time and accuracy by fixing the number of clusters and varying the size constraint B. For all the experiments we set parameters $\lambda = 0.0003$ and $\gamma = 10$. We will show later that L2S is robust to different numbers of clusters.
- FGD [168]: transform the softmax inference problem into nearest neighbor search (NNS) and solve it by a graph-based NNS algorithm.

- SVD-softmax [131]: a low-rank approximation approach for fast softmax computation. We vary the rank of SVD to control the tradeoff between prediction speed and accuracy.
- Adaptive-softmax [54]: a variant of hierarchical softmax that was mainly developed for fast training on GPUs. However, this algorithm can also be used to speedup prediction time (as discussed in Section 2), so we include it in our comparison. The tradeoff is controlled by varying the number of frequent words in the top level in the algorithm.
- Greedy-MIPS [161]: the greedy algorithm for solving MIPS problem. The tradeoff is controlled by varying the budget parameter in the algorithm.
- PCA-MIPS [7]: transform MIPS into Nearest Neighbor Search (NNS) and then solve NNS by PCA-tree. The tradeoff is controlled by varying the tree depth.
- LSH-MIPS [119]: transform MIPS into NNS and then solve NNS by Locality Sensitive Hashing (LSH). The tradeoff is controlled by varying number of hash functions.

We implement L2S, SVD-softmax and Adaptive-softmax in numpy. For FGD, we use the C++ library implemented in [14, 104] for the core NNS operations. The last three algorithms (Greedy-MIPS, PCA-MIPS and LSH-MIPS) have not been used to speed up softmax prediction in the literature and they do not perform well in these NLP tasks, but we still include them in the experiments for completeness. We use the C++ code by [161] to run experiments for these three MIPS algorithms.

Since our focus is to speedup the softmax layer which is known to be the bottleneck of NLP tasks with large vocabulary, we only report the prediction time results for the softmax layer in all the experiments. To compare under the same amount of hardware resource, all the experiments were conducted on an Intel Xeon E5-2620 CPU using a single thread.

4.5.2 Performance Comparisons

To measure the quality of top-k approximate softmax, we compute Precision@k (P@k) defined by $|A_k \cap S_k|/k$, where A_k is the top-k candidates computed by the approximate algorithm and S_k is the top-k candidates computed by exact softmax. We present the results for $k = 1, 5$. This measures the accuracy of next-word-prediction in LM and NMT. To measure

Table 4.1: Comparison of softmax prediction results on three datasets. Speedup is based on the original softmax time. For example, 10x means the method's prediction time is 10 times faster than original softmax layer prediction time. Computation of full softmax per step is 4.32 ms for PTB-Large, 0.32 ms for PTB-Small and 4.83 ms for NMT: DE-EN.

	PTB-Small			PTB-Large			NMT: DE-EN		
	Speedup	P@1	P@5	Speedup	P@1	P@5	Speedup	P@1	P@5
L2S (Our Method)	**10.6x**	**0.998**	**0.990**	**45.3x**	**0.996**	**0.982**	**20.4x**	**0.989**	**0.993**
FGD	1.3x	0.980	0.989	6.9x	0.975	0.979	6.7x	0.987	0.981
SVD-softmax	0.8x	0.987	0.99	2.3x	0.988	0.981	3.4x	0.98	0.985
Adaptive-softmax	1.9x	0.972	0.981	4.2x	0.974	0.937	3.2x	0.982	0.984
Greedy-MIPS	0.5x	0.998	0.972	1.8x	0.945	0.903	2.6x	0.911	0.887
PCA-MIPS	0.14x	0.322	0.341	0.5x	0.361	0.326	1.3x	0.379	0.320
LSH-MIPS	1.3x	0.165	0.33	2.2x	0.353	0.31	1.6x	0.131	0.137

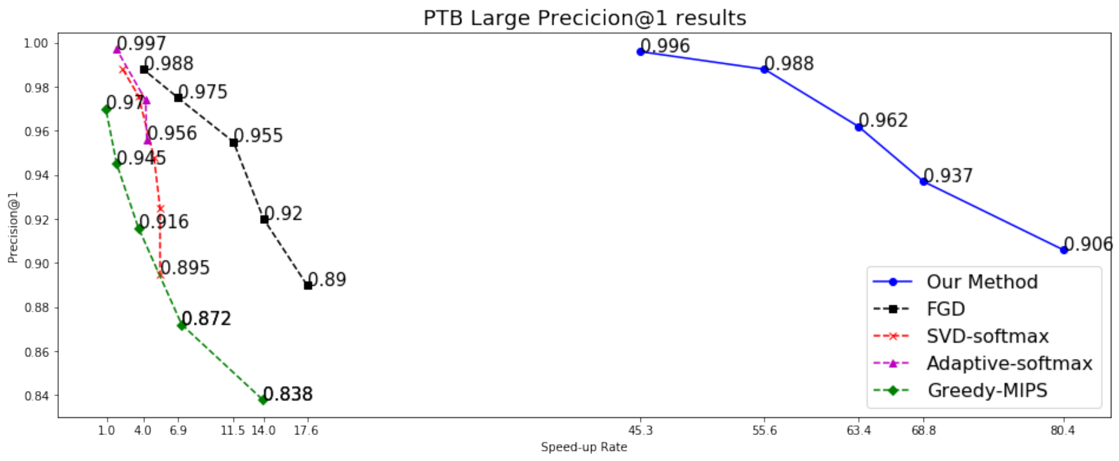


Figure 4.2: Precision@1 versus speed-up rate of PTB Large Setup.

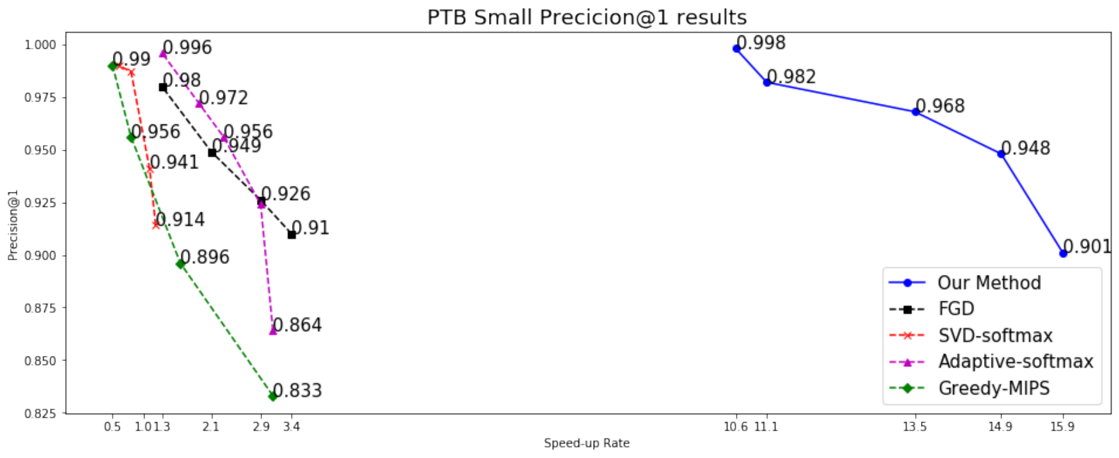


Figure 4.3: Precision@1 versus speed-up rate of PTB Small Setup.

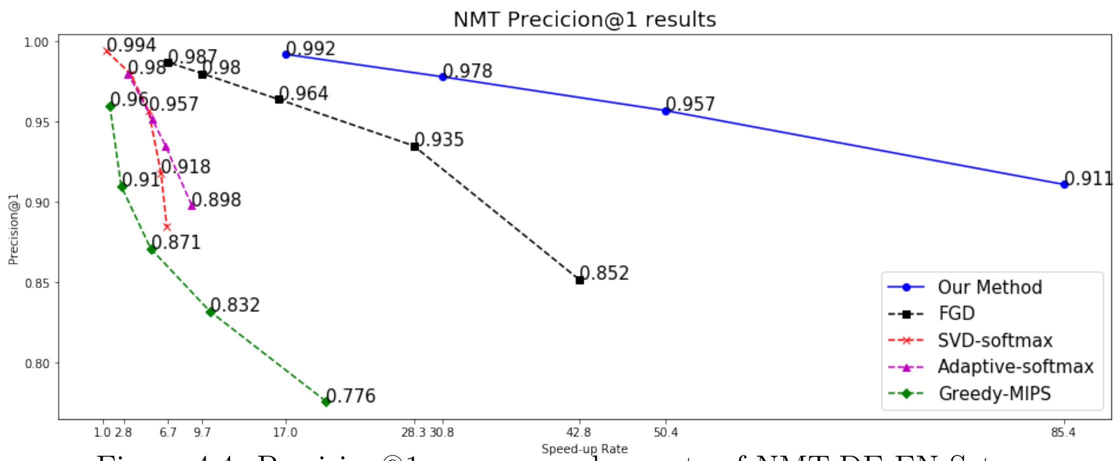


Figure 4.4: Precision@1 versus speed-up rate of NMT:DE-EN Setup.

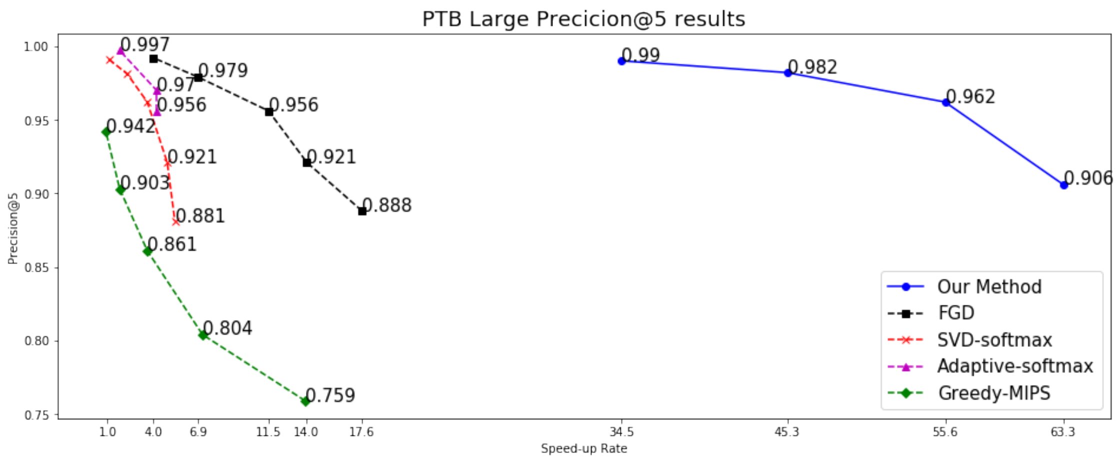


Figure 4.5: Precision@5 versus speed-up rate of PTB Large Setup.

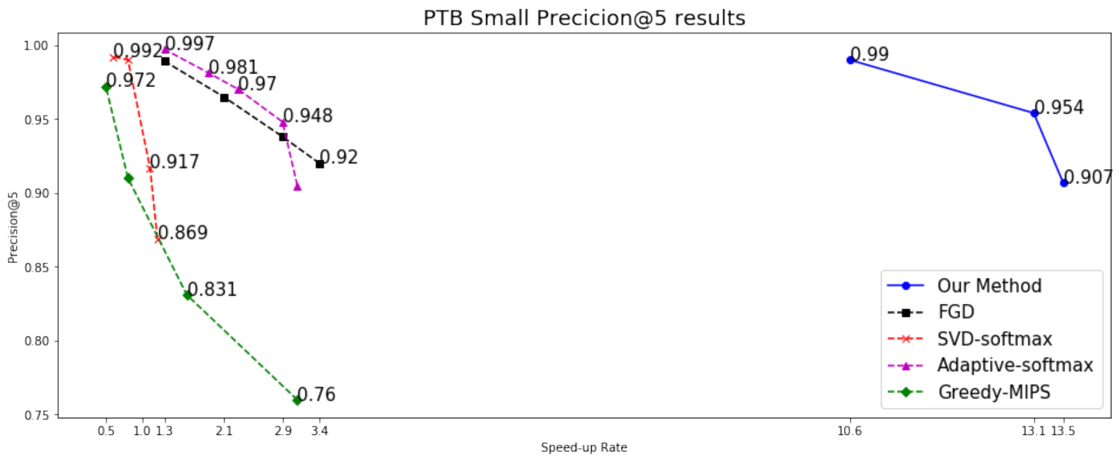


Figure 4.6: Precision@5 versus speed-up rate of PTB Small Setup.

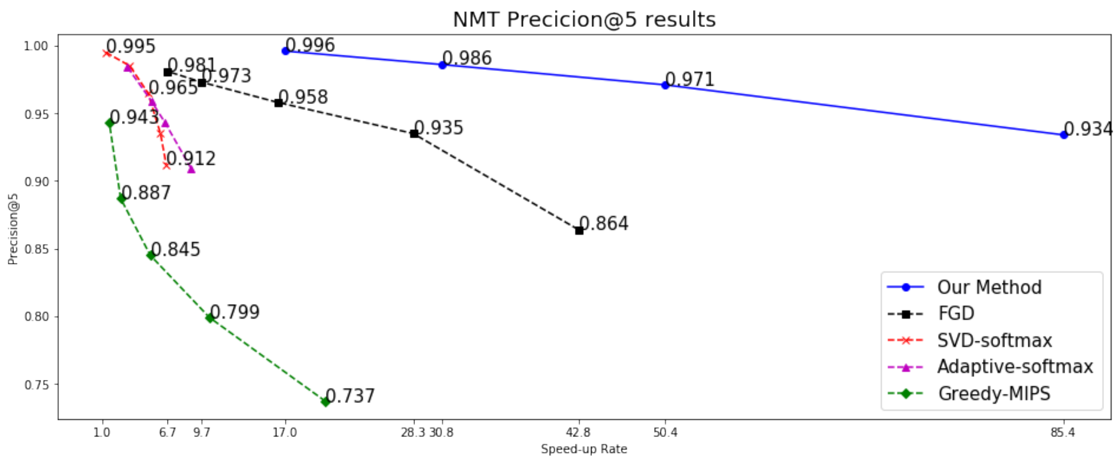


Figure 4.7: Precision@5 versus speed-up rate of NMT:DE-EN Setup.

the speed of each algorithm, we report the speedup defined by the ratio of wall clock time of the exact softmax to find top-k words divided by the wall clock time of the approximate algorithm.

For each algorithm, we show the prediction accuracy vs speedup over the exact softmax in Figure 4.2, 4.3, 4.4, 4.5, 4.6, 4.7. We do not show the results for PCA-MIPS and LSH-MIPS in the figures as their curves run outside the range of the figures. Some represented results are reported in Table 4.1. These results indicate that the proposed algorithm significantly outperforms all the previous algorithms for predicting top-k words/tokens on both language model (next word prediction) and neural machine translation.

Next, we measure the BLEU score of the NMT tasks when incorporating the proposed algorithm with beam search. We consider the common settings with beam size = 1 or 5, and report the wall clock time of each algorithm excluding the LSTM part. We only calculate log-softmax values on reduced search space and leave probability of other vocabularies not in the reduced search space to be 0. From the precision comparison, since FGD shows better performance than other completing methods in Table 4.1, we only compare our method with state-of-the-art algorithm FGD in Table 4.2 in terms of BLEU score. Our method can achieve more than 13 times speed up with only 0.14 loss in BLEU score in DE-EN task with beam size 5. Similarly, our method can achieve 20 times speed up in EN-VE task with only 0.08 loss in BLEU score. In comparison, FGD can only achieve less than 3-6 times speed up

Table 4.2: Comparison of BLEU score results vs prediction time on DE-EN and EN-VE task. Speedup is based on the original softmax time.

Model	Metric	Original	FGD	Our method
NMT: DE-EN	Speedup Rate	1x	2.7x	14.0x
Beam=1	BLEU	29.50	29.43	29.46
NMT: DE-EN	Speedup Rate	1x	2.9x	13.4x
Beam=5	BLEU	30.33	30.13	30.19
NMT: EN-VE	Speedup Rate	1x	6.4x	12.4x
Beam=1	BLEU	24.58	24.28	24.38
NMT: EN-VE	Speedup Rate	1x	4.6x	20x
Beam=5	BLEU	25.35	25.26	25.27

Table 4.3: L2S with different number of clusters.

Number of Clusters	50	100	200	250
Time in ms	0.12	0.17	0.14	0.12
P@1	0.997	0.998	0.998	0.994
P@5	0.988	0.99	0.99	0.98

over exact softmax to achieve a similar BLEU score. We also compare our algorithm with other methods using perplexity as a metric in PTB-Small and PTB-Large as shown in Table 4.5. We observe more than 5 times speedup over using full softmax without losing much perplexity (less than 5% difference).

4.5.3 Selection of the Number of Clusters

Finally, we show the performance of our method with different number of clusters in Table 4.3. When varying number of clusters, we also vary the time budget B so that the prediction time including finding the correct cluster and computing the softmax in the candidate set are similar. The results indicate that our method is quite robust to number of clusters. Therefore, in practice we suggest to just choose the number of clusters to be 100 or 200 and tune the "time budget" in our loss function to get the desired speed-accuracy tradeoff.

Table 4.4: Comparison of L2S to spherical-KMEANS clustering.

	PTB-Small			PTB-Large			NMT: DE-EN		
	Speedup	P@1	P@5	Speedup	P@1	P@5	Speedup	P@1	P@5
Our Method	10.6x	0.998	0.990	45.3x	0.999	0.82	20.4x	0.989	0.993
Sphereical-kmeans	4x	0.988	0.992	6.9x	0.992	0.971	13.8x	0.991	0.993
FGD	1.3x	0.980	0.989	6.9x	0.975	0.979	6.7x	0.987	0.981

4.5.4 Comparison to Spherical-KMEANS initialization

Since we firstly initialize parameters in our method by Shperical-KMEANS, we also show in Table 4.4 that L2S can further improve over the baseline clustering methods. Notice that even the basic Spherical-KMEANS can outperform state-of-the-art methods. This shows that clustering structure of context features is a key to perform fast prediction.

4.5.5 Perplexity Results

Finally, we go beyond top-k prediction and apply our algorithm to speed up the perplexity computation for language models. To get perplexity, we need to compute the probability of each token appeared in the dataset, which may not be within top-k softmax predictions. In order to apply a top-k approximate softmax algorithm for this task, we adopt the low-rank approximation idea proposed in [131]. For tokens within the candidate set, we compute the logits using exact inner product computation, while for tokens outside the set we approximate the logits by $\tilde{W}h$ where $\tilde{W}$ is a low-rank approximation of the original weight matrix in the softmax layer. The probability can then be computed using these logits. For all the algorithms, we set the rank of $\tilde{W}$ to be 20 for PTB-Small and 200 for PTB-Large. The results are presented in Table 4.5. We observe that our method outperforms previous fast softmax approximation methods for computing perplexity on both PTB-small and PTB-large language models.

Table 4.5: Comparison of Perplexity results vs prediction time on PTB dataset.

Model	Metric	Original	SVD-softmax	Adaptive-softmax	FGD	Our method
PTB-Small	Speedup Rate	1x	0.84x	1.69x	0.95x	5.69x
	PPL	112.28	116.64	121.43	116.49	115.91
PTB-Large	Speedup Rate	1x	0.61x	1.76x	2.27x	8.11x
	PPL	78.32	80.30	82.59	80.47	80.09

4.5.6 Qualitative Results

We select some translated sentences of DE-EN task shown in Table 4.6 to demonstrate that our algorithm can provide similar translations but with faster inference time.

4.6 Summary

In this section, we proposed a new algorithm for fast softmax inference on large vocabulary neural language models. The main idea is to use a light-weight screening model to predict a smaller subset of candidates, and then conduct exact search within that subset. By forming a joint optimization problem, we are able to learn the screening network end-to-end using the Gumbel trick. Notice that the clustering screening process is done on the output of the LSTM neural model instead of the softmax matrix, which demonstrated that hidden structures are presented in the LSTM output space, which is data-dependent. Therefore, we again demonstrate the usefulness of data distribution to further achieve efficient machine learning.

Table 4.6: Qualitative comparison of our method to full softmax computation. The accelerated model used is the same as reported in Table 4.2.

Full-softmax	Our method
you know , one of the great <unk> at travel and one of the **pleasures** at the <unk> research is to live with the people who remember the old days , who still feel their past in the wind , touch them on the rain of <unk> rocks , taste them in the bitter sheets of plants .	you know, one of the great <unk> at travel and one of the **joy** of the <unk> research is to live **together** with the people who remember the old days , who still feel their past in the wind , touch them on the rain of <unk> rocks , taste them in the bitter sheets of plants.
it 's the symbol of all **that** we are , and what we're capable of as astonishingly <unk> species .	it's the symbol of all of **what** we are , and what we're capable of as astonishingly <unk> species .
when **any of you were** born in this room , there were 6,000 languages **talking** on earth .	when **everybody was born** in this room , there were 6,000 languages **spoken** on earth .
a continent is always going to leave out , because the **idea** was that in sub-saharan africa there was no religious faith , and of course there was a <unk> , and <unk> is just the **remains** of these very profound religious thoughts that <unk> in the tragic diaspora of the <unk> .	a continent is always going to leave out , because the **presumption** was that in sub-saharan africa there was no religious faith , and of course there was a <unk> , and <unk> is just the **cheapest** of these very profound religious thoughts that <unk> in the tragic diaspora of <unk> <unk> .
so , the fact is that , in the 20th century, in 300 years , it is not going to be remembered for its wars or technological innovation , but rather than an era where we were present , and the massive destruction of biological and cultural diversity on earth either on earth is either **active or <unk>**. so the problem is not the change .	so , the fact is that , in the 20th century , in 300 years , it is not going to be remembered for its wars or technological innovation , but rather than an era where we were present , and the massive destruction of biological and cultural diversity on earth either on earth is either **<unk> or passive**. so the problem is not the change .

CHAPTER 5

Faster Recommender Systems via User Latent Structures

5.1 Introduction

In the previous section, we showed that hidden data distribution can help to achieve faster inference time by learning a clustering structures on top of latent space generated by neural network models. In this section, we wan to demonstrate that such a hidden/latent structure is ubiquitous. In particular, this structure also exists in latent spaces learned in the recommender systems, the most important application in today's e-commerce applications. Building large-scale personalized recommender systems has already become a core problem in many online applications since the explosive growth of internet users in the recent decade. For example, user-item recommender systems achieve many successes in e-commerce markets [97] while link prediction in social networks can be treated as a variant of recommender systems [8, 144]. To establish recommender systems, latent factor models for collaborative filtering have become popular because of their effectiveness and simplicity. More precisely, each user or item can be represented as a low-dimensional vector in a latent space so that the inner products between user and item vectors are capable of indicating the user-item preferences. Furthermore, these latent vectors can then be learned by optimizing a loss function with sufficient training data. For instance, matrix factorization [83] has been empirically shown to outperform conventional nearest-neighbor based approaches in a wide range of application domains [40].

After obtaining user and item latent vectors, to make item recommendations for each user, recommender systems need to calculate the inner products for all user-item pairs. Although learning user and item latent vectors is efficient and scalable for most existing models, recommender systems can take an enormous amount of time in evaluating all user-item pairs. More specifically, the time complexity of learning latent vectors is only proportional to the number of user-item pairs in the training data which is a small subset of all possible user-item pairs, but finding the top recommendations entails examining all $O(mn)$ inner products between all m users and n items. As a result, the quadratic complexity becomes a hurdle for large-scale recommender systems. For example, it can take more than a day to compute and rank all preference scores, and consequently the systems cannot be updated on a daily basis [41]. In order to make large-scale recommender systems practical, it is critical to accelerate the process of computing and ranking the inner products of user and item latent vectors, in order to efficiently obtain the top-K recommendations for all users.

To accelerate the computation of inner products, the maximum inner product search (MIPS) [119, 133, 162] is one of the feasible approaches. Locality sensitive hashing (LSH) [65] and PCA tree [138] may be applied to solve MIPS after reducing the problem to nearest-neighbor search. To reduce the computation for making recommendations for a given user, one may find a small group of candidate items whose latent vectors have large inner products with the user's latent vector using clustering algorithms [28], or sort entries of each dimension in the latent vectors separately by some greedy algorithms [41, 162]. In essence, most of the existing MIPS algorithms adopt a two-stage strategy, decomposing the computation into a preparation process and a prediction process. In the preparation stage, these methods will construct suitable data structures [162] or reduce the number of ranking candidates [28], and these prepared data structures are used to conduct efficient maximum inner product search for query vectors in the inference stage. However, most of these existing MIPS algorithms have the following two weaknesses, making them often impractical for real applications: (1) they only focus on optimizing the inference speed for a given user at the cost of considerable preparation time, but for recommender systems, the overall execution time (including both preparation and inference time) matters more because the system needs to be re-trained frequently as new data arrive. (2) All the MIPS approaches aim to quickly identify the top item set for *any* query vector. However, in recommender systems queries are not arbitrary vectors. They are user latent factors and usually have very strong *clustering structure*, which is ignored in most of the MIPS algorithms.

In order to speed up the overall execution time, our main idea is to exploit the relationships between users. More precisely, users with similar latent factors are more likely to share similar item preferences which may be reflected by their high inner products. However, existing methods for accelerating recommender systems do not consider user relationships and the distribution of user latent vectors. For instance, existing greedy strategies [41, 162] only consider the values of item latent vectors. Some studies based on proximity graphs [105, 168] and clustering algorithms [28] also solely reduce the search space of items. In the inference stage, these approaches treat the recommendation to each user as an independent query to the data structures and algorithms. As a consequence, it can be extremely time-consuming, especially with myriad users and enormous spaces of candidate items.

We propose a novel model for clustering and navigating for top-K recommenders (CANTOR) that leverages the knowledge of user relationships to accelerate the process of generating recommendations for all users with a given latent factor model. CANTOR consists of two stages: preparation and prediction. In the preparation stage, we aim to cluster users sharing similar interests into affinity groups and compute a small set of preferred items for each affinity group. More specifically, the user vectors (generated from a given latent factor model) are used in clustering affinity groups. To further accelerate the preparation time, a user coreset of few representative vectors are derived for each affinity group, and are used to obtain a small set of preferred items for users in this group by an efficient approximate nearest neighbor search algorithm. Finally, in the prediction stage, the top-K recommendations for each user can be retrieved by ranking these preferred items of the corresponding affinity group, which can be done much more efficiently than evaluating and ranking all items.

Our contributions are three-fold: (1) To the best of our knowledge, this is the first work to focus on the preparation time and user relationships for accelerating the prediction process of large-scale top-K recommender systems. (2) Clustering users into affinity groups based on the distribution of user latent vectors provides significant speedup of the prediction process, compared to conventional approaches that independently deal with each user. The representative vectors of the affinity groups offer a theoretically guaranteed precision for users with similar preferences. Approximate nearest neighbor search is applied to efficiently retrieve the satisfactory recommendations for each user from a small set of candidate items. (3) Experiments conducted on six publicly available datasets demonstrate that CANTOR can significantly accelerate large-scale top-K recommender systems for both item recommendation and personalized link prediction. An in-depth analysis then indicates the robustness and effectiveness of the proposed framework.

5.2 Problem Statement

In this section, we first introduce the notations and then formally define the objective of this work. Suppose that we have an incomplete $m \times n$ one-class matrix $\mathbb{R} = \{R_{ij}\} \in \{0, 1\}^{m\times n}$, where m and n are the numbers of users and items in the system. $R_{ij} = 1$ if user i prefers item j in the training data; otherwise, $R_{ij} = 0$. Based on $\mathbb{R}$, a matrix factorization based algorithm learns d-dimensional user and item latent vectors, denoted by $\mathbb{P} \in \mathbb{R}^{m\times d}$ and $\mathbb{Q} \in \mathbb{R}^{n\times d}$ respectively, where $\hat{\mathbb{R}} = \mathbb{P}\,\mathbb{Q}^T \in \mathbb{R}^{m\times n}$ reflects the underlying preferences. To compute top-K recommendations for each user, we need to find items with the K highest scores among $\hat{\boldsymbol{R}}(i) = \{\hat{R}_{ij'} \mid j' \in 1 \dots m\}$. Note that $m = n$ for personalized link prediction in social networks, where the goal is to suggest other users as recommended items.

Although matrix factorization models can be learned expeditiously when $\mathbb{R}$ is sparse, inferring the top-K recommendations requires computing and sorting the scores $\hat{R}_{ij}$ of all items j for each user i. As a result, the inference process can be time-consuming with an $O(nmd)$ time complexity which becomes intractable when n and m are large. To address this problem, our goal is to speed up the inference time of top-K recommenders with a high precision. More specifically, given the trained matrices $\mathbb{P}$ and $\mathbb{Q}$, we aim to propose an efficient approach that approximates the top-K recommended items for each user.

5.3 Constructing User Coresets for Top-K Recommender Systems

In this section, we present CANTOR for accelerating top-K recommender systems, starting with several key preliminary ideas.

5.3.1 Preliminary

In order to leverage the relationship between users, we first formally define the *affinity groups* of users in recommender systems as follows:

Definition 5.1. (Affinity Group) An *affinity group* $\mathbb{A}_t$ is a set of users sharing similar interests in items. Even though any similarity metrics may be used, we adopt cosine similarity as the metric to define the affinity groups.

By this definition, the sets of satisfactory recommendations should be similar for users in the same affinity group. This suggests that the top recommendations for all users in an affinity group are confined to a small subset of the items and such item subset can be learned by examining only a few carefully selected users in the group, leading to the following definition of the *preferred item set*.

Definition 5.2. (Preferred Item Set) A *preferred item set* c for an affinity group is a set of (potentially) satisfactory items for the users in the group, and the size of the preferred item set is usually much smaller than the total number of items, i.e., $|\,\mathrm{c}\,| \ll n$.

Therefore, we only need to examine the preferred item set to generate top recommendations, leading to significant time saving overs the alternative of examining all items.

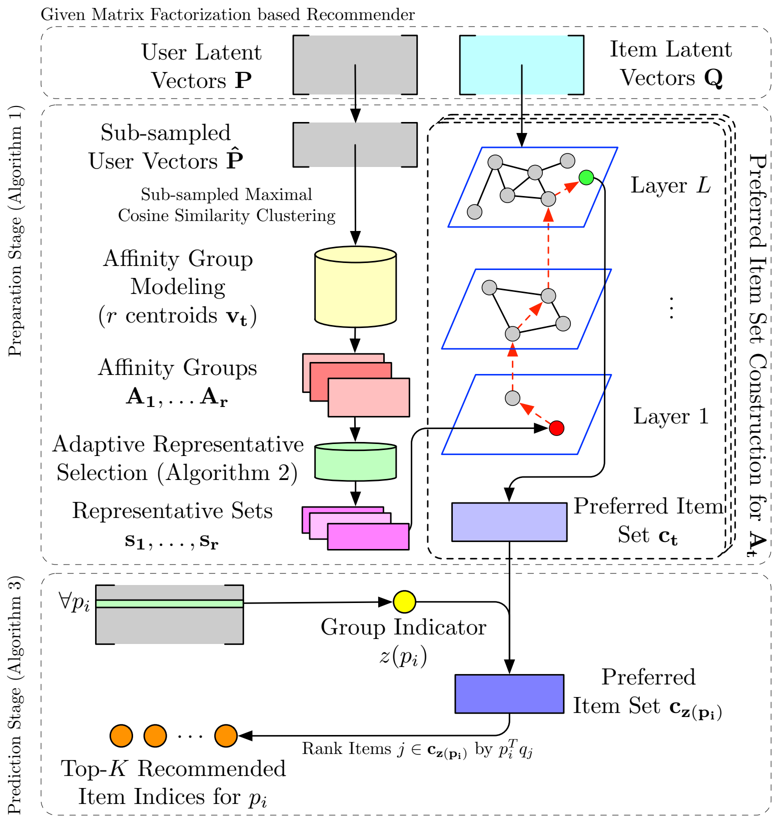
In order to robustly generate the preferred item set for each affinity group, we generate a few representatives from the group to compute the preferred item set. This is statistically more robust than using only the "centroid" user in the latent space, and is more computationally efficient than using all users in the group.

Definition 5.3. (User Coreset of an Affinity Group) A *δ-user coreset* s_t of an affinity group $\mathbb{A}_t$ is a (small) set of latent representative vectors to preserve the item preference of the users in $\mathbb{A}_t$ such that $\forall q \in \mathbb{Q}, i \in \mathbb{A}_t$:

$$\left|p_i q^T - \mathcal{N}_{\mathrm{s}_t}(p_i)\, q^T\right| \leq \delta,$$

where $\mathcal{N}_{\mathrm{s}_t}(p_i) \in \mathrm{s}_t$ is the nearest coreset representative for p_i; $\delta > 0$ is a small enough constant.

The user interests in the affinity group can be well captured by the representative vectors in the user coreset. Note that the representative vectors do not have to be identical to actual user latent vectors in the group.



Given Matrix Factorization based Recommender
User Latent Vectors $\mathbf{P}$
Item Latent Vectors $\mathbf{Q}$
Preparation Stage (Algorithm 1)
Sub-sampled User Vectors $\hat{\mathbf{P}}$
Sub-sampled Maximal Cosine Similarity Clustering
Affinity Group Modeling (r centroids $\mathbf{v_t}$)
Affinity Groups $\mathbf{A_1}, \ldots \mathbf{A_r}$
Adaptive Representative Selection (Algorithm 2)
Representative Sets $\mathbf{s_1}, \ldots, \mathbf{s_r}$
Layer L
Layer 1
Preferred Item Set $\mathbf{c_t}$
Preferred Item Set Construction for $\mathbf{A_t}$
Prediction Stage (Algorithm 3)
$\forall p_i$
Group Indicator $z(p_i)$
Preferred Item Set $\mathbf{c_{z(p_i)}}$
Rank Items $j \in \mathbf{c_{z(p_i)}}$ by $p_i^T q_j$
Top-K Recommended Item Indices for p_i

Figure 5.1: The general framework of the proposed clustering and navigating for top-K recommenders (CANTOR).

5.3.2 Framework Overview

Figure 5.1 shows the general framework of CANTOR. The framework consists of two stages: preparation and prediction. In the preparation stage as shown in Algorithm 5.1, the m user latent vectors $\mathbb{P}$ are first sub-sampled as $\hat{\mathbb{P}}$ and clustered into r affinity groups $\mathbb{A}_t$ with a centroid vector v_t computed from the corresponding user vectors $\mathbb{P}_t$, where $t = 1 \ldots r$. For each affinity group $\mathbb{A}_t$, we aim at deriving a small user coreset s_t. To do so, we propose an adaptive representative selection method (Algorithm 5.2) to greedily construct a set cover of user latent vectors for each affinity group after mathematically proving that the set covers can be the coresets of affinity groups. Finally, a small preferred item set c_t can be obtained by approximate nearest neighbor search using its coreset s_t for each affinity group. In the prediction stage (Algorithm 5.4), CANTOR first locates the corresponding affinity group $\mathbb{A}_t$ for each user and then ranks the small number of items in the preferred item set c_t, thereby efficiently providing satisfactory recommendations.

5.3.3 Preparation Stage

To overcome the hurdle of extremely long preparation time experienced by conventional methods, we propose to exploit similarities between user vectors in the latent space for acceleration as shown in Algorithm 5.1.

Affinity Group Modeling by User Clustering. Most of the conventional algorithms only rely on similarities of item latent vectors [131] and proximity graphs [54, 168] to accelerate the recommendation, and have not used the relationships between users and the distributions of user latent vectors in this endeavor. To exploit the knowledge of user relationships, we propose a clustering based framework to model user affinity groups.

Let r be the number of affinity groups for all m users, where r is a hyperparameter in CANTOR. We partition all m users into r disjoint clusters as the affinity groups $\mathbb{A} = \{\mathbb{A}_t \mid t = 1 \ldots r\}$ based on the user latent vectors $\mathbb{P} = \{p_i \mid i = 1 \ldots m\}$. In addition, each affinity group $\mathbb{A}_t$ has a centroid vector $v_t \in \mathbb{R}^d$ in the latent space. Each user i with the latent vector

```
Algorithm 5.1: Preparation Process for CANTOR
  Input: User latent vectors $\mathbb{P}$; item latent vectors $\mathbb{Q}$; degree of each user $deg_{i=1}^{m}$; the
         number of desired recommendation $K$
  Output: Centroid vectors $v_t$ and preferred item sets $\mathrm{c}_t$ for each affinity cluster $\mathbb{A}_t$
          for $t = 1 \ldots r$.
1 Hyperparameter: Number of affinity groups $r$; small world graph search size efs.;
   number of sub-sampled users $u$;
2 $\hat{P}$ = Multinomial_Sampling($\mathbb{P}$, $deg_{i=1}^{m}$, $u$); $\hat{P} \in \mathbb{R}^{u \times d}$ ;
3 $v_1, \cdots, v_r = 0$; $I = 0, I \in \mathbb{R}^{u \times 1}$ ;
4 repeat
5     for $i = 1, \cdots, r$ do
6         $v_i = \sum_{j \in \{j | I[j] = i\}} \hat{\mathbb{P}}[j]$ ;
7         $v_i = v_i \ / \ \|v_i\|_2$ ;
8     $\mathrm{I} = \arg\max_t v_t^T \hat{\mathbb{P}}$ ;
9 until Convergence;
10 G = CreateProximityGraph($\mathbb{Q}$, efs);
11 $\mathrm{c}_1, \ldots, \mathrm{c}_r = \emptyset, \ldots, \emptyset$ ;
12 $\mathrm{I} = \arg\max_t v_t^T \hat{\mathbb{P}}$ ;
13 for $i = 1, \cdots, r$ do
14     $\hat{\mathbb{P}}_i = \left\{ p_j \mid p_j \in \hat{\mathbb{P}}, I[j] = i \right\}$ ;
15     $\mathrm{s}_i$ = AdaptiveClustering($\hat{\mathbb{P}}_i$, $\epsilon$, $w$) ;
16     for $q \in \mathrm{s}_i$ do
17         $\hat{I}_i$ = QueryProximityGraph(G, s, $K$) ;
18         $\mathrm{c}_i = \mathrm{c}_t \cup \hat{I}_i$ ;
19 return $\mathrm{c}_t$,$v_t$ for all $t = 1, \cdots, r$.
```

p_i belongs to $\mathbb{A}_{z(p_i)}$, where $z(p_i)$ is the affinity group indicator represented as:

$$z(p_i) = \arg\max_r v_r^T p_i. \tag{5.1}$$

Let $C(p_i, K)$ be the top-K preferred items for user i which is defined as:

$$\{j \mid p_i^T q_j \geq p_i^T q_{j'}, \forall j' \notin C(p_i, K) \text{ and } |C(p_i, K)| = K\},$$

where $q_j \in \mathbb{Q}$ is the latent vector of item j. Intuitively, if users i and k are in the same affinity group, their preferred sets $C(p_i, K)$ and $C(p_k, K)$ may have substantial overlap because of their similar interests. This motivates us to compute a preferred item set c_t for users in the same affinity group $\mathbb{A}_t$ so that each c_t contains only a small subset of all n items, i.e., $|\,\mathrm{c}_t\,| \ll n$. Instead of computing the inner products between p_k and all item latent factors $q \in \mathbb{Q}$, we can narrow down the candidate set to be c_t, and only evaluate the items in c_t to find the top-K predictions for user k.

Since our task is to accelerate the maximum inner product search, the centriod vector v_t for each affinity group $\mathbb{A}_t$ can then be updated by the maximum cosine similarity criteria as:

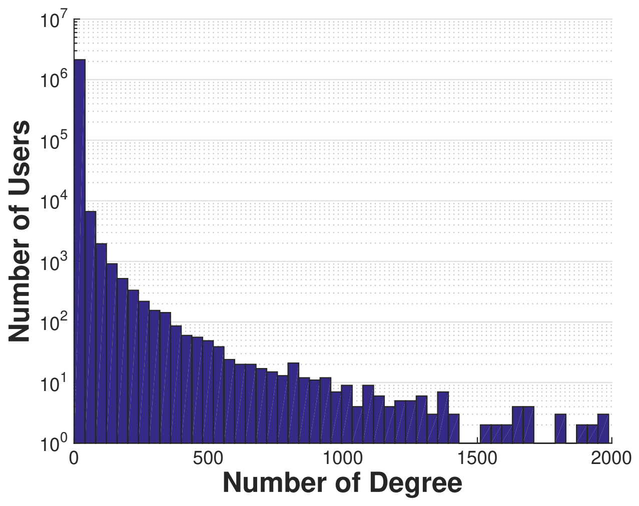
$$v_t = \frac{\sum_{i=1}^{|\,\mathbb{P}_t\,|} \mathbb{P}_{ti}}{\|\sum_{i=1}^{|\,\mathbb{P}_t\,|} \mathbb{P}_{ti}\,\|_2}, \tag{5.2}$$

where $\mathbb{P}_t = \{p_i \mid z(p_i) = t\}$ contains the latent vectors of users that belong to the affinity group $\mathbb{A}_t$. Therefore, each affinity group $\mathbb{A}_t$ can obtain a centroid vector v_t by iteratively running Equations (5.1) and (5.2). However, iteratively performing Equations (5.1) and (5.2) can still cost a long computational time when the number of users m is large. To address this issue, we propose to sub-sample a portion of the m user latent vectors to learn the centroid vectors. Moreover, we sample the latent vectors based on the degree distribution in the one-class matrix $\mathbb{R}$. For example, Figure 5.2a shows that degree distribution of users usually follows a power-law distribution. Hence, instead of using a uniform sampling, we sample user i with a probability proportional to a log function of its degree as:

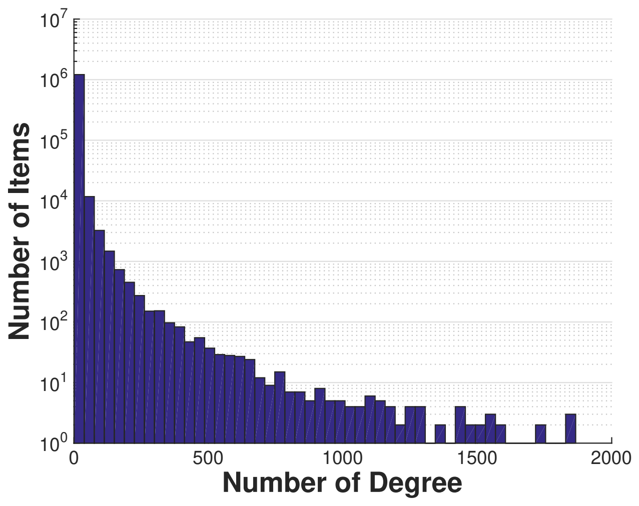
$$P(X = i) \propto \log \sum_{j=1}^{n} R_{ij}, \tag{5.3}$$

where X denotes the random variable of the target sampling process. We will later show in Theorem 5.2 that error of approximation based on sub-sampling will be asymptotically bounded.

After learning the centroids $v_1, \cdots, v_r \in \mathbb{R}^d$ and the corresponding user latent vectors $\mathbb{P}_1, \cdots, \mathbb{P}_r$ for r affinity groups $\mathbb{A}_1, \cdots, \mathbb{A}_r$, the preferred item set c_t for each group $\mathbb{A}_t$ can be constructed so that user vectors $\mathbb{P}_t$ only need to search over this set of preferred items for top recommendations. However, the naïve approach to generate c_t would require $O(nd)$



(a) User Distribution



(b) Item Distribution

Figure 5.2: The distributions of users and items over different degrees in the Amazon dataset.

operations to examine all n items in order to derive the top candidates for each user in $\mathbb{A}_t$. Each affinity group $\mathbb{A}_t$ would need $O(|\mathbb{P}_t|nd)$ operations for considering all $|\mathbb{P}_t|$ users in the group to construct the preferred item set c_t.

Coreset Construction as Finding a Set Cover. To accelerate the process of constructing the preferred item set c_t for an affinity group A_t, we want to find a δ-user coreset of A_t, and use it only instead of whole $\mathbb{A}_t$ to construct c_t. We achieve this by first defining the idea of ϵ-set cover, and then show that each ϵ-set cover corresponds to a δ-coreset.

Definition 5.4. (ϵ-Set Cover) s_t is an ϵ-cover of $\mathbb{P}_t$ if $\exists \mathcal{N}_{\mathrm{s}_t}(p) \in \mathrm{s}_t$ so that $\mathcal{N}_{\mathrm{s}_t}(p)p^T \geq \epsilon$ for all $p \in \mathbb{P}_t$, where ϵ is a real number, and $\mathcal{N}_{\mathrm{s}_t}(p_i) \in \mathrm{s}_t$ denotes the nearest vector in s_t of p_i.

Theorem 5.1. Given an ϵ-cover s_t of $\mathbb{A}_t$, there exists a δ such that ϵ-cover s_t is a δ-user coreset of the affinity group $\mathbb{A}_t$.

Proof. Without loss of generality, we assume that vectors in $\mathbb{A}_t$, $\mathbb{Q}$, and s_t have unit norms.

$\forall q \in \mathbb{Q}, i \in \mathbb{A}_t$, we have:

$$\begin{aligned}
|p_i q^T - \mathcal{N}_{\mathrm{s}_t}(p_i)\, q^T| &= \left|(p_i - \mathcal{N}_{\mathrm{s}_t}(p))q^T\right| \\
&\overset{(a)}{\leq} \sqrt{d}\|p_i - \mathcal{N}_{\mathrm{s}_t}(p_i)q^T\|_2 \leq \sqrt{d}\|p_i - \mathcal{N}_{\mathrm{s}_t}(p_i)\|_2 \leq \sqrt{d}\|p_i - \mathcal{N}_{\mathrm{s}_t}(p_i)\|_2^2 \\
&= \sqrt{d}\left(\|p_i\|_2^2 + \|\mathcal{N}_{\mathrm{s}_t}(p_{i_j})\|_2^2 - 2\mathcal{N}_{\mathrm{s}_t}(p_i)p_i^T\right) \overset{(b)}{\leq} \sqrt{d}\,[2-2\epsilon] = \delta,
\end{aligned}$$

where we define $\delta = \sqrt{d}\,[2-2\epsilon]$. (a) follows from the fact that $\|\cdot\|_1 \leq \sqrt{d}\|\cdot\|_2$, where d is the dimension of the vector. (b) follows from the condition of theorem. □

Therefore, we could construct a user coreset with an arbitrarily small δ by finding a cover with a greater ϵ.

Another nice property is that we could find an ϵ-set cover on sampled subset of $\mathbb{P}$ and generalize asymptotically with bounded error. Denote $\mathbb{P}_{\mathbb{A}_t}$ to be same sampling process of $\mathbb{P}$ generating user vectors p_i belonging to $\mathbb{A}_t$. We will have following result:

Theorem 5.2. For an affinity group $\mathbb{A}_t$, given any query q, an ϵ-cover of k samples $\{p_i\}$ drawn from $\mathbb{P}_{\mathbb{A}_t}$ would satisfy following inequality with probability at least $1-\gamma$:

$$\min_i \left(\left|\mathcal{N}_{\mathrm{s}_t}(p_i)\, q^T - p_t q^T\right|\right) \leq \delta + \sqrt{\frac{2\log(1/\gamma)}{k}}.$$

Proof. Since s_t is a ϵ set cover of p_is, there exist a δ such that s_t is a δ-user coreset of p_is. Therefore, for any given query q and vector p_t sampled from $\mathbb{P}_{\mathbb{A}_t}$, we have

$$\begin{aligned}
|\mathcal{N}_{\mathrm{s}_t}(p_i)q^T - p_t q^T| &= |\mathcal{N}_{\mathrm{s}_t}(p_i)q^T - p_i q^T + p_i q^T - p_t q^T| \\
&\leq |\mathcal{N}_{\mathrm{s}_t}(p_i)q^T - p_i q^T| + |p_i q^T - p_t q^T| \leq \delta + |p_i q^T - p_t q^T|
\end{aligned}$$

Since p_i and p_t follow the same distribution, p_i and p_t will have same expectation value and we have:

$$\begin{aligned}
\mathbb{E}[|\mathcal{N}_{\mathrm{s}_t}(p_i)q^T - p_t q^T|] &\leq \mathbb{E}[\delta + |p_i q^T - p_t q^T|] \\
&= \delta + \mathbb{E}[|p_i q^T - p_t q^T|] \\
&\overset{(a)}{\leq} \delta + |\,\mathbb{E}[p_i q^T] - \mathbb{E}[p_t q^T]| \\
&= \delta,
\end{aligned}$$

where (a) follows the Jensen's inequality. Therefore, by Hoeffding's inequality, with probability at least 1 - γ,

$$\frac{1}{k}\sum_{i=1}^{k} \left|\mathcal{N}_{\mathrm{s}_t}\left(p_i\right) q^T - p_t q^T\right| \leq \delta + \sqrt{\frac{2\log\left(1/\gamma\right)}{k}}.$$

By the fact that for any set S, $\min(S) \leq \mathrm{mean}(S)$, we will have:

$$\begin{aligned}
\min_i \left(\left|\mathcal{N}_{\mathrm{s}_t}(p_i)q^T - p_t q^T\right|\right) &\leq \frac{1}{k}\sum_{i=1}^{k} \left|\mathcal{N}_{\mathrm{s}_t}(p_i)q^T - p_t q^T\right| \\
&\leq \delta + \sqrt{\frac{2\log(1/\gamma)}{k}},
\end{aligned}$$

□

Theorem 5.2 indicates that we could construct an ϵ-cover of sub-sampled vectors to have an asymptotically guaranteed difference of inner-product values to true distributions within the same affinity group. Consequently, our task becomes finding an ϵ-cover of all $\mathbb{A}_t$s and constructing the preferred item set c_t of it. Hence, we propose a fast adaptive representative selection method to efficiently construct an ϵ-cover with sub-sampled user latent vectors for each affinity group as summarized in Algorithm 5.2. For each affinity group $\mathbb{A}_t$, the adaptive representative selection method is applied to obtain a few representative ϵ-cover s_t. If there exists at least one user whose latent vector has cosine similarity lower than ϵ to all representative vectors, the algorithm iteratively generates more representatives until every user has high cosine similarity to at least one representative vector. As a result, the

Algorithm 5.2: Adaptive Representative Selection

```
   Input: User latent vectors for an affinity group $\mathbb{P}$, the number of iterations $T$, the
          threshold $\epsilon$, the number of new representatives $w$ ;
   Output: Representative vectors s.
 1 Initialize $s = \emptyset$ ;
 2 $I = \arg\max_t s^T \mathbb{P}$ ;
 3 repeat
 4     for $i = 1 \ldots |s|$ do
 5         $s_i = \sum_{j \in \{j | I[j]=i\}} \mathbb{P}[j]$ ;
 6         $s_i = s_i \,/\, \| s_i \|_2$ ;
 7     $I = \arg\max_t s^T \mathbb{P}$ ;
 8     Outliers $= \{j | s^T_{I[j]} \mathbb{P}_j < \epsilon\}$ ;
 9     for $j \in$ Outliers do
10         Draw $i$ from $1 \ldots w$ ;
11         $I[j] = |s| + i$ ;
12     if Outliers $\neq \emptyset$ then
13         Append $w$ vectors to s ;
14 until Outliers $= \emptyset$;
15 Outliers $= \{j | s^T_{I[j]} \mathbb{P}_j < \epsilon\}$ ;
16 Append $\mathbb{P}_{\text{Outliers}}$ to s ;
17 return s.
```

number of ϵ-cover $|s_t|$ must be less than or equal to the number of user vectors in the cluster $|\mathbb{P}_t|$, and in practice, $|s_t| \ll |\mathbb{P}_t|$ in most cases. Note that the adaptive representative selection method is applied on each affinity group $\mathbb{A}_t$ independently. Next, the ϵ-cover s_t will be utilized to construct the preferred item set to reduce complexity from $O(|\mathbb{P}_t|nd)$ to $O(|s_t|nd)$.

Proximity Graph Navigation for Preferred Item Set Construction. To avoid examining all n items ($O(nd)$ complexity) in preferred item set construction, we apply an approximate nearest neighbor search (ANNS) method to accelerate the computation. We adopt a model based on proximity graphs [105, 168] which has shown the state-of-the-art performance in ANNS. Specifically, a proximity graph is generated in which item vectors are nodes and nodes of similar item vectors are connected by edges. Since the item degree in recommender systems tends to follow a power-law distribution as illustrated in Figure 5.2b, this proximity graph has the small world properties [13] with sparse edges that offer high

```
Algorithm 5.3: QueryProximityGraph
  Input: Hierarchical small world graph $G$; the query vector $q$; the number of the
         output approximate nearest neighbors $K$
  Output: $K$ nearest vectors in $G$
1 $p$ = Randomly select an entry node in $G$ ;
2 for $l = 1$ to $L$ do
3     $p = \arg\max_{\boldsymbol{r} \in \{p' | p' \, \mathbb{E}(p,l)\}} q^T \boldsymbol{r}$;
4 return $K$ Nearest Nodes in $\mathbb{E}(p, L)$ to $q$ ;
```

reachability between nodes. Hence, we apply the model of hierarchical navigable small world graphs [103, 105] to obtain the preferred item set c_t for each affinity group $\mathbb{A}_t$.

To construct the proximity graph of item vectors $\mathbb{Q}$ as a hierarchical small world graph G, we iteratively insert the item vectors into the graph, where each node q has a list $\mathbb{E}(q)$ of at most *efs* approximate nearest neighbors that could be dynamically updated when inserting other item vectors, where *efs* is a hyperparameter. In addition, the edges in the graph are organized as a hierarchy so that edges connecting items that have a high inner product value of their corresponding item vectors are at the bottom layers and edges connecting items whose vectors have low inner product values are at the top layers, thereby shrinking the search spaces for nearest neighbors. Let $L(e)$ denote the corresponding layer of edge e. Given two edges e_i and e_j, if $L(e_i) > L(e_j)$, then the nodes connected by edge e_i has a smaller inner product score than that of edge e_j. For simplicity, let $\mathbb{E}(q, l)$ denote the list of nodes connected to node q by edges in the l-th layer. Finally, the hierarchical small world graph G of item vectors $\mathbb{Q}$ can be constructed in $O(dn \log n)$ [105, 168], where n is the total number of items; *efs* is treated a constant hyperparameter. Note that *efs* controls the trade-off between efficiency and accuracy for searching nearest neighbors because it decides the size of search space and the potential coverage of real nearest neighbors.

The hierarchical small world graph G provides the capability of efficiently querying K nearest neighbors of a vector q with a hierarchical greedy search algorithm. More specifically, we can greedily traverse the graph G by navigating the query vector from the bottom layer to the top layer to derive K approximate nearest neighbors to q as shown in Algorithm 5.3

Algorithm 5.4: Prediction Process for CANTOR

```
Input: User latent vectors $p_i$; item latent vectors $\mathbb{Q}$; Number of top
       recommendations $K$
Output: The indices of estimated top-$K$ recommendations for the user $i$.
1 $z(p_i) = \arg\max_t v_t^T p_i$ ;
2 logits = $p_i^T \, \mathbb{Q} \left[\mathrm{c}_{z(p_i)}\right]$ ;
3 topIndices = argsort(logits, K) ;
4 return topIndices.
```

with a $O(d \log n)$ time complexity for each query. For each affinity group $\mathbb{A}_t$, we perform a small world graph query to approximate $C(\mathrm{s}_{t,i}, K)$ for each representative vector $\mathrm{s}_{t,i} \in \mathrm{s}_t$. The preferred item set c_t can then be constructed by taking the union operation to individual top-k sets as

$$\underset{t}{\mathrm{c}} = \bigcup_{i=1}^{|s_t|} C(s_{t,i}, K). \tag{5.4}$$

5.3.4 Prediction Stage

To predict top recommendations for a user with the latent vector p_i, CANTOR relies on the clustering model parameterized by the centroid vector $v_t \in \mathbb{R}^d$ and the preferred item set c_t for each affinity group $\mathbb{A}_t$. More precisely, we first compute the affinity group indicator $z(p)$ as:

$$z(p_i) = \arg\max_r v_r^T p_i, \tag{5.5}$$

and evaluate full vector matrix product $p^T \mathbb{Q}_I$ over the corresponding item vectors of the preferred item set $\mathbb{Q}_I$, $I = \{j | j \in \mathrm{c}_{z(p_i)}\}$. The computed results are then sorted to provide the final top-K recommendations for the user. Algorithm 5.4 shows the procedure of the prediction process.

Table 5.1: The statistics of six experimental datasets. Note that the personalized link prediction problem can be mapped to an item recommendation problem by treating each user as an item and recommending other users to a user in a similar way to that of recommending items to a user, and in this case the numbers of users and items are equal.

Task	Item Recommendation		
Dataset	MovieLens	Last.fm	Amazon
#(Users)	138,493	359,293	2,146,057
#(Items)	26,744	160,153	1,230,915
Task	Personalized Link Prediction		
Dataset	YouTube	Flickr	Wikipedia
#(Users)	1,503,841	1,580,291	1,682,759
#(Items)	1,503,841	1,580,291	1,682,759

5.4 Experiments

In this section, we conduct extensive experiments and in-depth analysis to demonstrate the performance of CANTOR.

5.4.1 Experimental Settings

Experimental Datasets. We evaluate the performance in two common tasks: item recommendation and personalized link prediction, using six publicly available real-world large-scale datasets as shown in Table 5.1. For the task of item recommendation, the MovieLens 20M dataset (MovieLens) [60] consists of 20-million ratings between users and movies; the Last.fm 360K dataset (Last.fm) [18] contains the preferred artists of about 360K users; the dataset of Amazon ratings (Amazon) includes ratings between millions of users and items [85]. For the task of personalized link prediction, we follow the previous study [41] to construct three social networks among users: YouTube, Flickr, and Wikipedia [85]. Note that four of the six experimental datasets, Amazon, YouTube, Flickr, and Wikipedia, are available in the Koblenz Network Collection [85].

Evaluation Metrics. To measure the quality of an approximate algorithm for top-K recommendation we evaluate the top-K approximated recommendations with Precision@K

(P@K), which is defined by

$$\frac{1}{m}\sum_{i}\frac{|Y_K^i \cap S_K^i|}{K},$$

where Y_K^i and S_K^i are the top-K items computed by the approximate algorithm and full inner-product computations for user i; m is the number of users. To measure the speed of each algorithm, we report the speedup defined by the ratio of wall clock time consumed by the full set of $O(mn)$ inner products to find the top-K recommendations divided by the wall clock time of the approximate algorithm.

Baseline Methods. To evaluate our proposed CANTOR, we consider the following five algorithms as the baseline methods for comparison.

- ϵ-approximate link prediction (ϵ-Approx) [41] sorts entries of the latent factor for each dimension to construct a guaranteed approximation of full inner products.
- Greedy-MIPS (GMIPS) [162] is a greedy algorithm for solving the MIPS problem with a trade-off controlled by varying a computational budget parameter in the algorithm.
- SVD-softmax (SVDS) [131] is a low-rank approximation approach for fast softmax computation. We vary the rank of SVD to control the trade-off between prediction speed and accuracy.
- Fast Graph Decoder (FGD) [168] directly applies small world graph on all items $\mathbb{Q}$ and navigates to derive recommended items with user latent vectors as queries on the proximity graph. It also serves a direct baseline of only using proximity graph navigation.
- Learning to Screen (L2S) [28] is the first clustering-based method on fast prediction in NLP tasks with the state-of-the-art results on inference time but suffers from long preparation time. CANTOR is inspired by the clustering step in L2S, thus L2S serves as a direct baseline. In our implementation, random sub-sampling is applied to choose a subset of users for training L2S.

Note that [162] has shown that Greedy-MIPS outperforms other MIPS algorithms including LSH-MIPS [119, 133], Sampling MIPS [9] and PCA-MIPS [7], so we omit those other

MIPS algorithms in our comparisons. Although bandit-based methods [49, 92, 93] have elegant mathematical properties and theoretical bounds, we did not include them originally because they generally perform worse than other methods in practical cases. For example, SCLUB [93], which is one of the state-of-the-art bandit-based approaches, only achieves 0.81x and 0.62x speedups on the Amazon and Wikipedia datasets with the official implementations. This is because bandit-based methods independently manipulate each dimension and cannot benefit from low-level optimization for linear algebra operations.

Implementation Details. For each dataset, the LIBMF library [31] is used to train a non-negative MF (NMF) model. More specifically, the number of dimensions for latent vectors is 10 while the models are trained with all data for 100 iterations. Note that we adopt NMF models because of the restrictions of ϵ-Approx, but CANTOR does not have any limitation on matrix types. We implement CANTOR in Python with NumPy optimized by BLAS [12]. For the baseline methods, the implementations of GMIPS, SVDS, FGD, and L2S are provided by the original authors and highly-optimized while we utilize an efficient C++ implementation of ϵ-Approx. All experiments were run on a 64-bit Linux Ubuntu 16.04 server with 512 GB memory and single thread regime on an Intel® Xeon® CPU E5-2698 v4 2.2 GHz.

5.4.2 Performance Comparison

To fairly compare the performance, for each dataset, we tune the parameters such that each method can roughly achieve 0.99 P@1 accuracy. Table 5.2 shows the efficiency and the precision scores of CANTOR and all baseline methods on six datasets. Note that since the open-sourced library of GMIPS does not provide the breakdown of execution time into preparation and prediction time, the reported time includes both preparation and prediction processes. Among the baseline methods, FGD performs the best because it exploits the state-of-the-art algorithm for approximate nearest neighbor search to retrieve recommendations for each user. Although L2S is the most efficient baseline in the inference process, its preparation process is slow so that the overall speedup is further degraded. SVDS can

Table 5.2: Comparisons of top-K recommendation results on six datasets in two tasks. Note that P@K measures the precision of approximating the top-K recommendations of full inner-product computations. SU indicates the ratio of speedup based on the original full inner product time of inferring top-K recommendations. For example, 9.4x means the computation time of the method is 9.4 times faster than the full inner product computation time. PT means the preparation time and IT represents the inference time in prediction process. The time units of *seconds*, *minutes*, and *hours* are represented as s, m, and h, respectively. Computation time of the full inner product method for each dataset is 71s (MovieLens), 1,017s (Last.fm), 92,828s (Amazon), 56,824s (Youtube), 71,653s (Flickr), and 72,723s (Wikipedia).

Task	Item Recommendation														
Dataset	MovieLens					Last.fm					Amazon				
Method	SU	PT	IT	P@1	P@5	SU	PT	IT	P@1	P@5	SU	PT	IT	P@1	P@5
ϵ-Approx	0.7x	0.19s	99.00s	0.753	0.671	0.5x	1.40s	36.78m	0.378	0.467	0.2x	23.42s	107.34h	0.529	0.559
GMIPS	3.9x	N/A	18.41s	1.000	0.972	2.3x	N/A	7.55m	0.997	0.966	1.8x	N/A	14.57h	0.993	0.952
SVDS	1.0x	0.10s	69.00s	1.000	1.000	0.9x	0.10s	19.25m	0.984	0.984	1.3x	5.32s	19.46h	0.952	0.953
FGD	2.8x	4.94s	20.10s	1.000	0.999	10.9x	0.49m	1.07m	0.997	0.988	19.7x	42.76m	35.83m	0.986	0.977
L2S	3.0x	22.15s	1.72s	1.000	1.000	9.0x	1.77m	0.12m	0.993	0.980	21.2x	71.02m	1.86m	0.988	0.979
CANTOR	**9.4x**	6.17s	1.36s	1.000	0.999	**37.1x**	0.37m	0.09m	0.999	0.998	**29.0x**	52.13m	1.26m	0.994	0.991
Task	Personalized Link Prediction														
Dataset	YouTube					Flickr					Wikipedia				
Method	SU	PT	IT	P@1	P@5	SU	PT	IT	P@1	P@5	SU	PT	IT	P@1	P@5
ϵ-Approx	0.1x	0.3m	129.2h	0.364	0.432	0.4x	0.29m	53.44h	0.545	0.581	0.2x	0.39m	130.61h	0.374	0.480
GMIPS	1.4x	N/A	11.12h	0.987	0.965	2.0x	N/A	10.10h	0.987	0.962	3.6x	N/A	5.64h	0.991	0.974
SVDS	1.0x	0.03m	15.30h	0.965	0.963	1.4x	0.03m	14.00h	0.952	0.946	1.4x	0.03m	14.83h	0.949	0.944
FGD	44.8x	10.28m	10.85m	0.989	0.981	37.5x	17.61m	14.25m	0.985	0.980	93.7x	4.18m	8.76m	0.990	0.985
L2S	6.9x	135.93m	0.79m	0.984	0.968	8.3x	142.84m	0.58m	0.989	0.980	22.4x	53.38m	0.84m	0.988	0.968
CANTOR	**112.7x**	7.75m	0.65m	0.993	0.985	**54.7x**	21.31m	0.53m	0.994	0.990	**355.1x**	2.45m	0.97m	0.995	0.991

efficiently decompose the preference matrix as its preparation process, but it still requires to examine all items many times to achieve sufficient accuracy so that the acceleration is unsatisfactory. In addition, it is worth noting that, although ϵ-Approx theoretically needs fewer multiplications than the full evaluation, it actually does not provide any acceleration in practice. Similar to bandit-based methods, this is because each dimension is independently processed so that the model cannot benefit from any low-level optimization for linear algebra operations.

Our approach CANTOR significantly outperforms all of the baseline methods in accelerating the overall execution time to provide top-K recommendations in all datasets. More specifically, CANTOR has similar inference time for the prediction process to that of L2S (that also reduces the candidate item sets for less computation) but the preparation process of CANTOR is much faster. This is because similarities between user latent vectors are well leveraged to avoid unnecessary and redundant computation.

5.5 Summary

In this chapter, we demonstrated that the implicit data dependent information could also be found in other applications. We proposed a new method for accelerating large-scale top-K recommender systems by generalizing the clustering methods in order to capture the implicit information in user latent space. The proposed method tried to find the implicit information by first clusters users into affinity groups, and for each group there are only a limited number of preferred items need to be examined for the users in the affinity group. In particular, the proposed method achieves 355x and 29x speedup on the largest Wikipedia and Amazon datasets in two tasks while the accuracy scores still remain to be 99% for both P@1 and P@5. These results show that implicit data dependent information also exist in recommender system, and the proposed method CANTOR can leverage this implicit information to achieve efficient machine learning.

CHAPTER 6

Data-aware Low-rank Approximation: When Model Compression meets Inference Time Speedup

6.1 Introduction

In previous chapters, we introduced two ways to leverage data distributions. For explicit information such as word frequency, we can use low-rank or compositional methods to obtain a much better model compression. For implicit information hidden in a latent space, we learn a screening to select more pertinent groups to accelerate the inference step. However, these methods in general can't achieve both desiderata at the same time. For compression scenario, compostional coding needs time to reconstruct the approximated embedding vectors, so it won't be able to accelerate the inference. For fast inference applications, we need to store the additional screening model, which apparently won't reduce the model size. The only exception is the GroupReduce method. GroupReduce builds multiple clusters over the softmax matrix, and for each cluster GroupReduce combines frequency information with low-rank methods, which has the potential to speed up inference simultaneously. In particular, we know that BLAS library has greatly optimized linear algebra computation, and low-rank methods preserve this attribute and it achieved superior inference speed over sparse methods. However, the biggest problem with GroupReduce is that it requires existence of explicit information such as frequency information. This method cannot be applied to any given matrices. In this chapter, we introduce an idea of data-aware low-rank approximation, which combines both ends of model compression and inference time speedup. We resorts to the low-rank method since it's a method which can achieves these two goals simultaneously.

We also bring the implicit data distribution into the low-rank method. Specifically, we observe that the learned representation of each layer lies in a low-dimensional space. Based on this observation, we propose a provably optimal low-rank decomposition of weight matrices, which has a simple closed form solution that can be efficiently computed.

To deal with efficiency issues, most existing work resorts to adjusting the model structure or distillation. For instance, [80] used locality-sensitive hashing to accelerate dot-product attention, [87] used repeating model parameters to reduce the size and [167] applied a pre-defined attention pattern to save computation. A large body of prior work focused on variants of distillation has also been explored [22, 72, 98, 127, 141, 142, 142, 158, 170]. These methods require a specific design of model architecture, or a long training stage and thus it is less straightforward to combine these methods with each other.

We explore a simpler acceleration method to speed up inference time which can be applied to most existing architectures. As shown in Figure 6.1, matrix multiplication (feed-forward layer) is a fundamental operation which appears many times in the Transformer [146], the backbone architecture of the BERT model. In fact, the underlying computation of both multi-head attention layers and feed-forward layers is matrix multiplication. Therefore, instead of resorting to the complex architecture redesign approaches, we aim to investigate whether low-rank matrix approximation, a classical and simple model compression approach, can be used to accelerate Transformers. Despite its successful application to CNNs [132, 137, 165], at first glance, low-rank compression does not appear to work for BERT since the matrices in both feed-forward layers and attention layers **are not low rank** (see Figure 6.2). Therefore, even the optimal low-rank approximation (e.g., by SVD) will lead to very large reconstruction error. This is probably why low-rank approximation has not been successfully used in BERT compression.

In this paper, we propose a novel low-rank approximation algorithm to compress the weight matrices even though they are not low-rank. The main idea is to exploit the data distribution. In NLP applications, the latent features (features fed into each matrix mulitplication layer) usually indicate some information extracted from natural sentences, and they

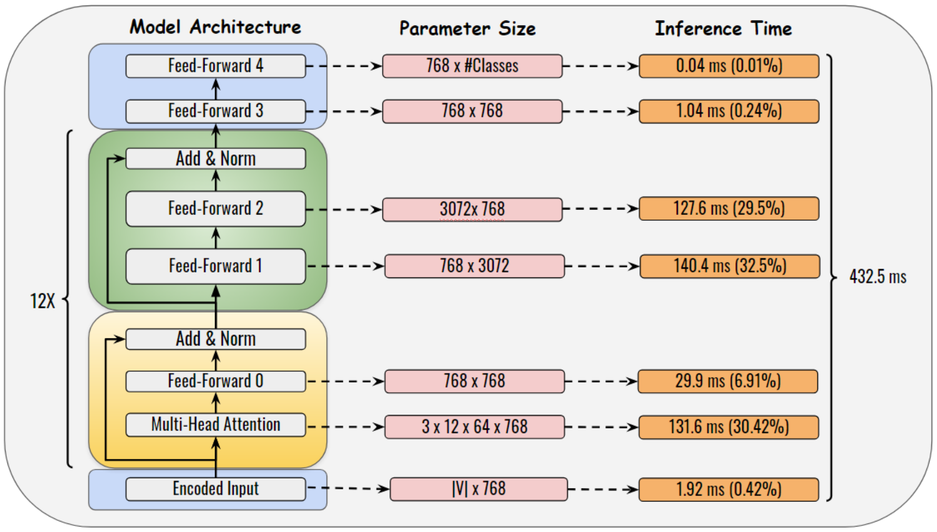


Figure 6.1: Illustration of the BERT-base computational model. $|V|$ (at bottom of the Parameter Size column) denotes the number of tokens in the model. #Classes (at top of the Parmeter Size column) denotes the number of classes in the down-stream classification task. Input encoding, Feed-forward 3 and Feed-forward 4 are computed only once and thus do not contribute much to overall time. The inference time (in milliseconds) listed here is based on the inference time measured on a CPU.

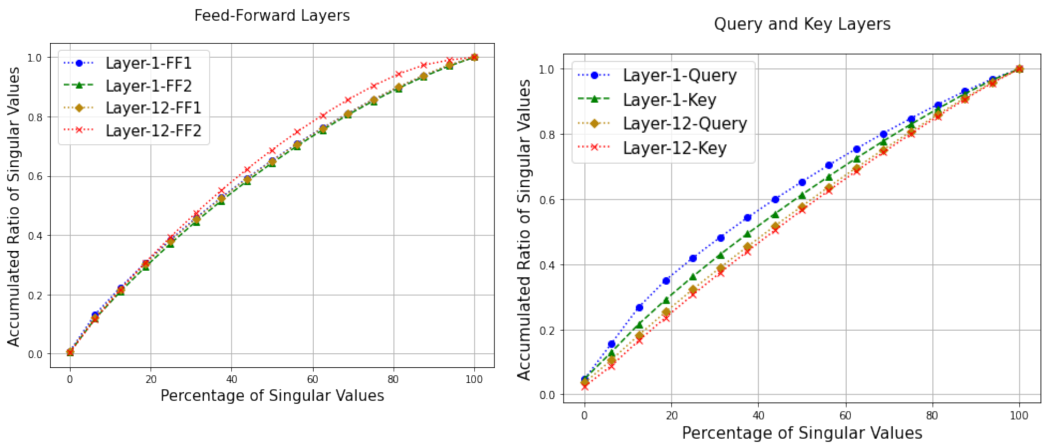


Figure 6.2: Illustration of the empirical observation that weight matrices in BERT model are not low-rank. The X-axis represents what percentage of singular values; the Y-axis represents sum of singular values connected to the selected ranks divided by sum of all singular values. Ideally, a low-rank structure will have a larger area under the curve, meaning that a small percentage of the singular values can explain their total sum. We observe that the sum of the top 50% of the ranks only accounts about 60% of all singular values for matrices in the BERT model. This shows that the matrices do not have a clear low-rank structure.

often lie in a subspace with a low intrinsic dimension [28, 114, 132]. Therefore, in most of the matrix-vector products, even though the weight matrices are not low-rank, the input vectors lie in a low-dimensional subspace, allowing dimension reduction with minimal degraded performance. We mathematically formulate this generalized low-rank approximation problem which includes the data distribution term and provide a closed-form solution for the optimal rank-k decomposition of the weight matrices. By leveraging the data distribution idea, we propose DRONE (**d**ata-awa**r**e l**o**w-ra**n**k compr**e**ssion). Our decomposition significantly outperforms the SVD under the same rank constraint, and can successfully accelerate the BERT model without sacrificing too much test performance. In addition to compressing standard models, DRONE can also be used on distilled BERT models to further improve the compression rate. For example, DRONE alone achieves 1.92x speedup on the MRPC task with only 1.5% loss in accuracy, and when combined with distillation, DRONE achieves over 12.3x speedup on various natural language inference tasks.

6.2 Related Work

Fast inference is important for deploying NLP models in various applications. Generally speaking, inference efficiency can be enhanced by hardware [130] or lower-level instruction optimization [120]. On the other hand, the main focus of the current research is on using algorithmic methods to reduce computational complexity. These methods can be mainly categorized into two aspects: attention complexity reduction and model size reduction.

Attention Complexity Reduction

Attention mechanism is the building block of transformer models and has attracted the most attention of researchers recently in the NLP field [146]. Pre-training on large corpus of BERT, a transformer-based model, has contributed to state-of-the-art performance on various tasks after fine-tuning [36]. Attention on sequences of length L is $O(L^2)$ in both computational and memory complexity, which yields long inference time when the sequence is long. Thus, researchers have focused on reducing the complexity of the attention module. [80] used locality-sensitive hashing to reduce the complexity to $O(L \log L)$. [30, 167] pre-defined an attention map to have a constant computational time. [53] progressively eliminated the redundant context vectors within the attended sequence to improve efficiency of attention in the last few layers of the model. [150] proposed to train the low-rank attention by choosing a rank $r \ll L$. This is similar to our work in the sense of leveraging low-rank structures. But our method does not require training the model from scratch and can be applied to different modules other than attention. In fact, most of the above methods require special modules and thus need to train the proposed models from scratch. This prohibits the usage of a large body of publicly available open models for faster research progress. More importantly, these methods mainly focus on the long sequence scenario. As shown in Figure 4.1, we have found out that attention module is actually not the main inference bottleneck of inference time in common usage. In most, if not all, models of common usages, two layers of large feed-forward layer are appended after the attention module which incurs much more computational time. Attention complexity reduction only works when a long sequence is used but in current practice this is unusual. Thus, in many tasks accelerating the attention module itself does not contribute to a significant reduction of overall inference time.

Model Size Reduction

Inference speed is also related to model compression. In principle, smaller models lead to reduction in the number of operations and thus faster inference time. [128] explored pruning methods on BERT models to eliminate redundant links, and there is a line of research on pruning methods [27, 52, 58, 59]. Quantization methods [38, 64, 95, 166] convert the 32 bits float models into fewer-bits fixed-point representation and make model prediction faster with fixed point accelerator. [87] reduce the model size by sharing encoder parameters. A large body of prior work focused on variants of knowledge distillation [22, 72, 98, 127, 141, 142, 142, 158, 170]. These methods use different strategies to distill information from a teacher network and reduce the number of layers [127] or hidden dimension size [72]. Further, a hybrid compression method by combining matrix factorization, pruning and knowledge distillation is proposed by [106]. Notice that [106] performed SVD for some components and in this paper we propose an improvement over SVD by leveraging input distribution to each layer. Idea of using input distribution to compress model has also been explored in PCN method [147], which is perhaps the closest to our work. However, DRONE differs from PCN in following three aspects. First, PCN only considers input distribution but not weight matrix and thus it's a special case of DRONE (i.e., W be an identity matrix in equation (6.3)). Second, PCN merely does dimension reduction whereas our formulation achieves dimension reduction and low-rank approximation simultaneously. Last, PCN does not guarantee the obtained transformation is the optimal; whereas, DRONE formulates an approximation optimization problem and we provide the optimal solution. Other forms of low-rank learning strategies including initialization and structure pruning were also explored in the literature [77, 152], and we will compare to these baseline methods. Among the above-mentioned methods, quantization requires hardware accelerator to maximally reduce the inference time. Pruning methods can only reduce the model size, but the inference time might not be reduced due to the limitation of sparse operations. Only algorithmic methods such as distillation serve as a more generic inference time accelerating method. We want to emphasize that our method is orthogonal to these distillation methods. In fact, the proposed method is an acceleration method that is applicable to all components in most NLP models. In Section 6.4, we show that DRONE can be combined with the distilled models to further improve the performance.

6.3 Proposed Method

We now introduce an algorithm for improving efficiency of matrix multiplication. The computation of feed-forward (FF) layer in the attention models can be described as:

$$h = Wx + b, \tag{6.1}$$

$$o = \sigma(h), \tag{6.2}$$

where $W \in \mathbb{R}^{d_2 \times d_1}$ and $b \in \mathbb{R}^{d_2}$ are model parameters, $x \in \mathbb{R}^{d_1}$ is the latent representation of a token, and $h \in \mathbb{R}^{d_2}$ is the intermediate representation before the activation function, $\sigma(\cdot)$ is the activation function, and $o \in \mathbb{R}^{d_2}$ is the output. Assuming the sequence length is L, all the token representations $x_1, \dots, x_L \in \mathbb{R}^{d_1}$ will pass through this same operation, so in practice the whole FF layer can be computed by a matrix-matrix product $W[x_1, \dots x_L] + b$, and the computation of the bias term b would be broadcast to all L input tokens. In practice we will normally have $L \ll \max(d_1, d_2)$ (e.g., $L = 128$, $d_2 = 3072$). Notice that applying $\sigma(\cdot)$ on h element-wisely costs $O(Ld_2)$, which is much smaller than the cost of computing Wx ($O(Ld_2d_1)$). Therefore, in this paper we focus on reducing the cost of computing Wx to accelerate the computation. A standard way to accelerate this computation is to perform low-rank approximation on W. A low-rank approximation can be obatined by using singular value decomposition (SVD), which achieves the best rank-k approximation in terms of Frobenius norm and we can write W as:

$$W = USV^T \approx U_{W,k} {V_{W,k}}^T,$$

with orthogonal matrices $U \in \mathbb{R}^{d_2 \times d_2}$, $V \in \mathbb{R}^{d_1 \times d_1}$ and a diagonal matrix $S \in \mathbb{R}^{d_2 \times d_1}$. $U_{W,k} \in \mathbb{R}^{d_2 \times k}$ and $V_{W,k} \in \mathbb{R}^{d_1 \times k}$ are the rank-k approximation matrices by taking $U_{W,k} = US_k^{\frac{1}{2}}$, $V_{W,k} = VS_k^{\frac{1}{2}}$, where $S_k^{\frac{1}{2}}$ is the square-root of the first k entries of the diagonal matrix S. Given such an approximation, we can simplify the computation in (6.1) by

$$h = Wx + b \approx U_{W,k} {V_{W,k}}^T x + b.$$

After conducting rank-k approximation, the computational complexity reduces from $O(d_2d_1)$ to $O((d_1 + d_2)k)$. When k is small enough, low-rank approximation not only accelerates the computation [132] but also compresses the model size [125]. However, as shown in Figure 6.2, matrices in FF layer of BERT do not show obvious low-rank structures. We observe that choosing top 50% rank (e.g., $k = 0.5 \min(d_1, d_2)$) can only achieve around 60% of the accumulation ratio of singular values, which implies large matrix approximation error. In the meantime, the complexity is still about $O(d_2d_1)$ and there is no enhancement of speed.

Even though the matrices in the model are not low-rank, we now provide an illustrative example to show that a low-rank computation could still exist when data distribution lies in a lower intrinsic dimension. Suppose we have a matrix W defined as below and the input x lies in a subspace:

$$W = \begin{bmatrix} 7 & 0 & 2 & 3 & 1 \\ 9 & 6 & 7 & 5 & 0 \\ 6 & 1 & 8 & 0 & 3 \\ 4 & 3 & 2 & 1 & 4 \\ 1 & 2 & 2 & 1 & 2 \end{bmatrix}, \quad x \in \operatorname{span}\left(\begin{bmatrix} 2 \\ 2 \\ 5 \\ 5 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 6 \end{bmatrix} \right).$$

In this case, W is a full-rank matrix so there is no lossless low-rank approximation of W. On the other hand, the input data x lies in a 2-dimensional subspace so that we could construct the following low-rank approximation:

$$\underbrace{\begin{bmatrix} 7 & 0 & 2 & 3 & 1 \\ 9 & 6 & 7 & 5 & 0 \\ 6 & 1 & 8 & 0 & 3 \\ 4 & 3 & 2 & 1 & 4 \\ 1 & 2 & 2 & 1 & 2 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 5 & 2 \\ 5 & 2 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}}_{x}$$

$$= \underbrace{\begin{bmatrix} 43 & 23 \\ 90 & 39 \\ 66 & 41 \\ 45 & 37 \\ 29 & 21 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} -1 & -1 & 0.5 & 0.5 & 0 \\ -0.5 & 0 & 0 & 0 & 0.25 \end{bmatrix}}_{V^T} \underbrace{\begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 5 & 2 \\ 5 & 2 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}}_{x},$$

which gives a rank-2 matrix UV^T where $W \neq UV^T$ but $Wx = UV^Tx$ for any x in the low dimensional space. This shows that even if W cannot be approximated, it is still possible to construct a good low-rank decomposition, and the key is to exploit the space of input vectors.

6.3.1 DRONE: Data-aware Low-rank Compression

Assuming the input x of the FF layer follows some distribution, instead of minimizing the approximation error of the weight matrix (for which SVD is optimal), we want to minimize the approximation error of the outputs. Denoting X as the $\mathbb{R}^{d_1 \times n}$ matrix where columns of X capture the empirical distribution of the input (when n is large), our goal is to find projection matrix $V_{X,k} \in \mathbb{R}^{d_1 \times k}$ and recovery matrix $U_{X,k} \in \mathbb{R}^{d_2 \times k}$ such that the output is well approximated. We rewrite (6.1) as:

$$\begin{aligned} h = WX + b &\approx WU_{x,k}{V_{x,k}}^T X + b \\ &= (WU_{x,k}){V_{x,k}}^T X + b = W_{X,k}{V_{x,k}}^T X + b, \end{aligned}$$

where $W_{X,k} = WU_{x,k}$. Intuitively, when X lies in a lower-dimensional space, we could find such a pair by PCA decomposition on X to project X onto the subspace that explains the most variance. In this way, instead of considering the decomposition of W, we leverage the distribution of X to complete the low-rank approximation.

However, the best way is to consider the properties of both W and X simultaneously, and we can mathematically present this desideratum by the following optimization problem:

$$\min_M \|WX - WMX\|_F^2, \quad \text{s.t.} \quad \text{rank}(M) = k, \tag{6.3}$$

where M is the desired rank-k transformation which maximally preserves the results of the matrix multiplication. In the theorem below, we show that there exists a closed-form, optimal solution for the above optimization problem. Before stating the theorem, we first introduce some notation. Assuming $\text{rank}(W) = r$ and $\text{rank}(X) = t$, we can write $W = U_W S_W V_W^T$ and

$X = U_X S_X V_X^T$ such that

$$U_W = \begin{bmatrix} U_{W,r} & \bar{U}_{W,r} \end{bmatrix}, S_W = \begin{bmatrix} S_{W,r} & 0 \\ 0 & 0 \end{bmatrix}, V_W = \begin{bmatrix} V_{W,r} & \bar{V}_{W,r} \end{bmatrix}$$
$$U_X = \begin{bmatrix} U_{X,t} & \bar{U}_{X,t} \end{bmatrix}, S_X = \begin{bmatrix} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix}, V_X = \begin{bmatrix} V_{X,t} & \bar{V}_{X,t} \end{bmatrix}.$$

In other words, the decomposition $U_W S_W V_W^T$ and $U_X S_X V_X^T$ are the full-SVD decompositions of W and X, respectively. The matrices $U_{W,r}$,$V_{W,r}$, $U_{X,t}$,$V_{X,t}$ denote corresponding row spaces and column spaces, while $\bar{U}_{W,r}$, $\bar{V}_{W,r}$, $\bar{U}_{X,t}$ and $\bar{V}_{X,t}$ are null spaces. With this notation, we are ready to state the theorem.

Theorem 6.1. Assume rank$(W) = r$ and rank$(X) = t$. The closed form solution M^* of the optimization problem (6.3) is

$$M^* = V_{W,r} S_{W,r}^{-1} Z_k S_{X,t}^{-1} U_{X,t}^T, \tag{6.4}$$

where Z_k is the rank-k truncated SVD of $Z = S_{W,r} V_{W,r}^T U_{X,t} S_{X,t}$.

Proof. We firstly consider the unconstrained problem:

$$\begin{aligned} M^* &= \underset{M}{\operatorname{argmin}} \|WX - WMX\|_F^2 \\ &= \underset{M}{\operatorname{argmin}} \|U_W^T W X V_X - U_W^T W M X V_X\|_F^2 \\ &= \underset{M}{\operatorname{argmin}} \|S_W V_W^T U_X S_X - S_W V_W^T M U_X S_X\|_F^2, \end{aligned}$$

where the second equality holds due to the fact that U_W and V_X are orthonormal matrices.

Note that we could expand the term $S_W V_W^T U_X S_X$ as:

$$\begin{aligned}
S_W V_W^T U_X S_X &= \begin{bmatrix} S_{W,r} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_{W,r}^T \\ \bar{V}_{W,r}^T \end{bmatrix} \begin{bmatrix} U_{X,t} & \bar{U}_{X,t} \end{bmatrix} \begin{bmatrix} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} S_{W,r} V_{W,r}^T & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U_{X,t} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} S_{W,r} V_{W,r}^T U_{X,t} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix}.
\end{aligned}$$

Similarly, we will have

$$S_W V_W^T M U_X S_X = \begin{bmatrix} S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix}.$$

Therefore, we continue above unconstrained problem as:

$$\begin{aligned}
M^* &= \underset{M}{\operatorname{argmin}} \| S_W V_W^T U_X S_X - S_W V_W^T M U_X S_X \|_F^2 \\
&= \underset{M}{\operatorname{argmin}} \| \begin{bmatrix} S_{W,r} V_{W,r}^T U_{X,t} S_{X,t} - S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix} \|_F^2 \\
&= \underset{M}{\operatorname{argmin}} \| S_{W,r} V_{W,r}^T U_{X,t} S_{X,t} - S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t} \|_F^2. \\
&= \underset{M}{\operatorname{argmin}} \| Z - S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t} \|_F^2.
\end{aligned}$$

The above minimization problem obtains the optimal value if $S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t}$ equals the rank-k truncated SVD of Z by the fundamental property of SVD decomposition. Thus, we will have:

$$\begin{aligned}
& Z_k = S_{W,r} V_{W,r}^T M^* U_{X,t} S_{X,t} \\
& \implies M^* = V_{W,r} S_{W,r}^{-1} Z_k S_{X,t}^{-1} U_{X,t}^T.
\end{aligned}$$

□

We note that since Z_k is the rank-k truncated SVD of Z, we could also write Z_k as $U_{Z,k}V_{Z,k}^T$ by distributing the top-k singular values of Z into left or right singular matrices. Thus the original computation can be rewritten as:

$$WX \approx (WV_{W,r}S_{W,r}^{-1}U_{Z,k})(V_{Z,k}^T S_{X,t}^{-1}U_{X,t}^T)X = U^* V^{*T} X, \tag{6.5}$$

where $U^* = WV_{W,r}S_{W,r}^{-1}U_{Z,k}$ and $V^{*T} = V_{Z,k}^T S_{X,t}^{-1}U_{X,t}^T$ are two rank-k matrices, and we will replace W by $U^* V^{*T}$.

6.3.2 Extension to Dot-product Attention

Although the optimization problem in (6.3) is proposed for feed-forward computation, in this section we show that it can also be applied to the dot-product part of the attention module. The key computation in the attention layer is to compute pairwise similarity between queries and keys of the sequence:

$$O = (Q\bar{Y})^T(KY), \tag{6.6}$$

where $\bar{Y} \in \mathbb{R}^{d_1 \times n}$ is the batch query data, $Q \in \mathbb{R}^{d_2 \times d_1}$ is the query transformation matrix, $Y \in \mathbb{R}^{d_1 \times m}$ is the batch key data, $K \in \mathbb{R}^{d_2 \times d_1}$ is the key transformation matrix and n, m are query and key batch sizes, respectively. We can again see that the desired low-rank approximation is the solution of the following optimization problem:

$$\min_M \|(Q\bar{Y})^T(KY) - (Q\bar{Y})^T M(KY)\|_F^2, \text{s.t. } \operatorname{rank}(M) = k. \tag{6.7}$$

With $Q\bar{Y} = W$ and $K\bar{Y} = X$, we get the following corollary from Theorem 6.1 directly.

Corollary 6.1. Assume $\operatorname{rank}(Q\bar{Y}) = r$ and $\operatorname{rank}(KY) = t$. Let $Q\bar{Y} = U_W S_W V_W^T$ and $KY = U_X S_X V_X^T$ be the SVD decomposition of $Q\bar{Y}$ and KY respectively. The closed form solution M^* of the optimization problem (6.7) is given by

$$M^* = V_{W,r} S_{W,r}^{-1} Z_k S_{X,r}^{-1} U_{X,r}^T, \tag{6.8}$$

where Z_k is the rank-k truncated SVD of $Z = S_{W,r} V_{W,r}^T U_{X,t} S_{X,t}$.

Proof. By denoting $W = (Q\bar{Y})^T$ and $X = KY$, and Theorem 1 could be applied to obtain the optimal solution M^*. □

6.3.3 Overall Algorithm

We have shown that the proposed DRONE method is a generic acceleration module applicable to all parts of neural language models. We summarize the DRONE on feed-forward layer in Algorithm A.1. Since in practice we don't have the exact distribution of X, we use training data to calculate the low-rank approximations as described in Algorithm A.1. The attention map calculation can be done by the same procedure with $W = (Q\bar{Y})^T$ and $X = KY$ as given by the Corollary 6.1.

To accelerate the whole model, we need to select appropriate ranks for each component. However, since the approximation of one component affects the distribution of overall representations, the optimal rank for the model requires a complete search of all possible combinations of rank values, which is infeasible in practice. We thus resort to a simplified approach as shown in Algorithm 6.2. A more detailed description is provided in the Appendix A.2. In short, as the changes of lower layer parameters will cause the distribution of representation shifts in upper layers, we approximate each component one-by-one in their topological order of the model. In another words, we approximate the model from the lower layers toward the higher layers. Within each layer, we follow the topological order of underlying modules. We provide a total allowed increase of loss ratio r as an input to the Algorithm 6.2. The hyper-parameter r depends on the efficiency and efficacy trade-off which users are willing to pay. The larger the value r, the faster approximation we get at the cost of lower accuracy. We then distribute r into each module $R_{l,i}$ (allowed loss increase

Algorithm 6.1: Data-Aware Low-rank Compression of feed-forward layer.

```
Input: rank $k$, training data $D_{train}$, Original weight matrix $W$, Prediction Model $M$.
Output: Low-rank Approximation $U^*, V^*$.
1 X = {}
2 for all batches $x_b$ in $D_{train}$ do
3     Feed the batch of training data $x_b$ into $M$ and extract the representation $x$. $x$ is the representation which will be multiplied with $W$ as in (6.1).
4     Append $x$ to X.
5 Given $X$,$k$ and $W$, solve the optimal low-rank matrices $U^*$,$V^*$ by (6.5).
```

ratio of i-th module of l-th layer in Algorithm 6.2). The distribution from r to each $R_{l,i}$ is based on the observed inference time of each module $E_{l,i}$ (observed empirical inference time of i-th module of l-th layer in Algorithm 6.2). The longer a module takes to compute, the more budget is allocated. Overall, total allowed loss r and the distributed loss ratio for each module $R_{l,i}$ fulfil the equality $(1 + r) = \prod_l \prod_i (1 + R_{l,i})$. For each module, if the approximation with certain rank used won't increase the loss over the ratio $(1 + R_{l,i})$, we will use that rank to approximate the module and move on to the next module. The pseudo code is also provided in the Appendix to illustrate the process.

6.4 Experimental Results

6.4.1 Experimental Setup

We evaluate DRONE on both LSTM and transformer-based BERT models. For LSTMs, we train a 2-layer LSTM-based language model from scratch with hidden sizes 1500 on Penn Treebank Bank (PTB) dataset. For BERT models, we evaluate the pre-trained BERT models on GLUE tasks. Various pre-trained models are offered in the open source platform [154]. For BERT models, we use BERT-base models and it contains 12 layers of the same model structure without sharing parameters. Each layer contains an attention module with hidden size 768 and 12 channels, a small 768×768 Feed-forward (FF) layer followed by 2 larger FF layers (768×3072 and 3072×768). As shown in Figure 4.1, these four components consume the most computational time in the BERT-base models.

Algorithm 6.2: Overall Low-rank Model Approximation Algorithm.

```
Input: training data $D_{train}$, original weight matrix $W$. prediction Model $M$, total
       allowed loss increase ratio $r$, Observed inference time $E$, Search grids of
       ranks for each module $G$, original Training loss $L$.
Output: Low-rank Approximation $U^*, V^*$.
# Distribute allowed ratio r into each module by $E$
$E_{min} \leftarrow \text{argmin}_{l,i} E_{l,i}$
$E_{l,i} \leftarrow \frac{E_{l,i}}{E_{min}}$
$E_b \leftarrow exp(\frac{log(1+r)}{\sum_{l,i} E_{l,i}})$
$R_{l,i} \leftarrow E_b^{E_{l,i}} - 1$
for $l = 1, \cdots, total\ layers$ do
    for $module\ m_i \in M_l$ do
        $W_{l,i} \leftarrow$ $l$-th layer parameter of module $m_i$
        (e.g., 2nd feed-forward matrix in first layer.)
        for $i = 1, \cdots, |G_{l,i}|$ do
            $k \leftarrow G_{l,i}$
            $U, V \leftarrow$ Algorithm A.1 $(k, D_{train}, W_{l,i}, M)$
            $\hat{M} \leftarrow M$ with $W_{l,i}$ replaced by $U, V$.
            Evaluate new loss $L_{new} = \hat{M}(D_{train})$
            if $L_{new}/L < 1 + R_{l,i}$ then
                $M \leftarrow \hat{M}$
                break;
```

For the baseline methods, most of the existing work pertaining to low-rank approximation [106, 132] leverages SVD in part of the compression procedure. Therefore, our baseline comparison will be the SVD approximation, and our work aims to provide an improvement over SVD. We also include the state-of-the-art distillation methods TinyBERT [72] in the comparison and show that the proposed method can be combined with it to further improve the performance. TinyBERT reduces the model into 4 layers of attention dimension 312 with 12 channels, and the FF layers are downsized to 312×1200. As we mentioned above, all the approximation methods need to consider efficiency and efficacy trade-off. In this paper, we follow previous literature [28, 72] and report the approximation results with about 3% loss in accuracy to compare the performance of all methods.

In real-world applications, NLP models are mostly evaluated on mobile devices or servers with multiple hardware accelerators. Thus, we measure the inference speed on both CPU

Table 6.1: The experimental results of running pret-rained BERT-base model on natural language inference tasks (Glue dataset). Each task has its own metric for performance measurement. Accuracy (SST-2, QNLI, RTE and WNLI), F1/Accuracy (MRPC and QQP), Matthew's correlation (CoLA), Matched accuracy/Mismatched accuracy (MNLI) and Person/Spearman correlation (STS-B) are used respectively. All the DRONE results are within 3% accuracy loss and show that DRONE can accelerate the whole BERT model across different tasks and devices.

Methods	MNLI	QQP	SST-2	QNLI	MRPC	RTE	CoLA	STS-B
Original	84.3	90.9	92.3	91.4	89.5	72.6	53.4	87.8
SVD	74.4	50.8	73.1	52.2	63.8	47.3	12.4	33.6
DRONE	82.0	89.4	90.0	88.5	86.7	70.0	52.5	85.8
DRONE-Retrain	82.6	90.1	90.8	89.3	88.0	71.5	53.2	87.8
CPU Speedup Ratio	1.60x	1.25x	1.64x	1.20x	1.92x	1.31x	1.33x	1.52x
GPU Speedup Ratio	1.28x	1.38x	1.45x	1.28x	1.56x	1.33x	1.29x	1.57x

(Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz) and GPU (GeForce GTX 1080 Ti) devices. All the experiments are repeated 10 times. The average single sequence prediction inference time in milliseconds is reported in the results. We want to emphasize that unlike many of the literature [30, 80, 150], which reported speedup only in the attention layers, our results reflect end-to-end speedup including both attention module and feed-forward layers. To perform the approximation, empirically we found randomly sub-sample 10% of the training data suffices to provide good results. Using more data can only provide limited performance boost but comes at a higher cost of longer preprocessing time. Thus, we will use 10% random sample of the training data to perform the experiments. After the proposed data-aware low-rank distribution, we slightly fine-tune the model to further improve the performance. We use a relatively smaller learning rate 10^{-7} and retrain 1 epoch on the sub-sampled training data to complete the fine-tuning procedure.

6.4.2 Results of BERT Models on GLUE Dataset

We summarize the results of DRONE on GLUE tasks in Table 6.1. Detailed inference time of each component of the compressed Transformer model is listed in the Appendix. Detailed inference time of an uncompressed BERT-BASE model can be found in Figure 4.1. We observe that each task exhibits different difficulty. The best acceleration we can achieve is nearly twice as fast (1.92x) with less than 2% accuracy loss after retraining (on the MRPC). In addition, DRONE achieves 1.52x acceleration without accuracy loss on the STS-B task.

By applying the same selected rank for each module with SVD method, we can observe that the performance drops significantly. This shows that the matrices within the model is generally not low-rank; thus the direct low-rank approximation without considering data distribution does not work. On GPU, we see that the acceleration is more or less the same as on CPU except MNLI and MRPC tasks. This is due to the fact that GPU uses massive parallelism and low-rank approximation introduces a sequential computation which might hinder the speedup depending on the size of the matrices and ranks used. To resolve this problem, low-level cuda code optimization is needed and system researchers have studied the problem [129], which is out of the scope of the present work. Despite this, we can still observe that DRONE performs better on QQP, RTE and STS-B and it provides about 1.5x acceleration for various tasks on GPU. An example of ranks used in SST-2 is listed in Appendix A.5.

6.4.3 Combination with Model Size Reduction Methods

Our proposed DRONE is a general low-rank approximation technique, and it is complementary to many other model compression methods. To illustrative its power, we now demonstrate that DRONE can be combined together distillation. The discussion of combination with Quantization is left in the Appendix. Distillation methods compress the underlying model into a smaller one without losing much accuracy. Distilled models are much smaller in number of layers or hidden dimension, resulting in a smaller model size and faster inference time. As shown in the Table 6.2, TinyBERT, one of the most competitive distillation methods, indeed achieves good performance within 3% accuracy loss for some of the GLUE tasks. Due to the fact that the computation inside the distilled model is still full matrix computation, DRONE can be applied to find data-aware low-rank approximation of these smaller matrices. Results are summarized in Table 6.2. As we can see combining DRONE with the distillation method further reduces the inference time without sacrificing accuracy. In particular, on the SST-2 task DRONE + TinyBERT speeds the inference time from 11.7x to 15.3x on CPU while achieving the same accuracy as the TinyBERT. Similarly, DRONE

+ TinyBERT speedups GPU results with 10.9x STS-B and 9.7x on SST-2 with competitive performance. These results again show that the proposed method has the potential to be applied under various scenarios and hardware devices to achieve a better model inference time speedup.

Table 6.2: The average inference time (in milliseconds) in comparison to distilled models on CPU and GPU. The unit is in millisecond. The results show that DRONE can be combined with distillation to further improve the performance. Compared to the state-of-the-art distillation method, the speedup ratio increases from 11.4x to 14.2x on STS-B and from 11.7x to 15.3x on SST-2.

Tasks	Models	CPU-speedup	GPU-speedup	Accuracy (%)
STS-B	BERT	1x	1x	87.8
	TinyBERT	11.4x	8.6x	86.9
	DRONE +TinyBERT	**14.2x**	**10.9x**	**87.0**
RTE	BERT	1x	1x	72.6
	TinyBERT	1.8x	1.9x	70.8
	DRONE +TinyBERT	**2.1x**	**2.2x**	**71.7**
MRPC	BERT	1x	1x	89.5
	TinyBERT	11.6x	7.8x	86.3
	DRONE +TinyBERT	**12.3x**	**8.6x**	**86.7**
SST-2	BERT	1x	1x	92.3
	TinyBERT	11.7x	8.4x	**90.7**
	DRONE +TinyBERT	**15.3x**	**9.7x**	**90.7**

Table 6.3: Illustration of SVD fine-tuning on MRPC, RTE, CoLA and STS-B. Using the same rank as the proposed DRONE method, SVD accuracy will drop significantly after the approximation. After fine-tuning done on the SVD approximation, the accuracy could be recovered for some tasks (e.g., MRPC), but SVD + Retrain still perform much worse than DRONE across all the tasks.

Models	MRPC	RTE	CoLA	STS-B
BERT	89.5	72.6	53.4	87.8
DRONE-Retrain	**88.0**	**71.5**	**53.2**	**87.8**
SVD	63.8	47.3	12.4	33.6
SVD-Retrain	85.8	63.5	24.4	66.3

6.4.4 Comparison to Structured Pruning Methods.

Instead of compressing model after training to accelerate inference time, another line of research called Structured Pruning tried to learn the low-rank structure during training of the model to save both training and inference time simultaneously. This raises the question if post-processing such as DRONE is necessary if no further compression is required once we can get a small model after training. Thus, it's worth comparing DRONE with the state-of-the-art Structure Pruning method [152]. In [152], attention modules are not approximated by low-rank matrices. To make the comparison fair, we apply DRONE on base models except the attention module and keep others the same. For MRPC, DRONE achieves 1.58x speedup with performance drop from 89.5 to 89.4. [152] achieves 1.43x with performance 88.61. For SST-2, DRONE achieves 1.41x speedup without sacrificing performance (92.3). [152] also achieves about 1.41x speedup with the performance 92.09. Thus, we can see that DRONE performs better than structured pruning.

One further question is that if we can use the idea of DRONE to do fast training? We conducted DRONE on the pre-trained RTE task before fine-tuning to get a low-rank structure, and then apply the regular end-to-end training over this compressed model. We found out this procedure with directly using the same rank as in our experiments (with 1.38x speedup) can only achieve 68.2 accuracy. But if we reduce the compression ratio into 1.2x training time speedup, this procedure can give us 72.9 accuracy. On the other hand, the same procedure with SVD as the initilialization of low-rank structure can only get 52.1 accuracy. This preliminary experiment shows that in addition to inference time acceleration, DRONE also has the potential to be applied in the training, but directly transport DRONE into training can not lead to the optimal result. How to improve low-rank training is an interesting future direction.

6.4.5 Additional Experiments on Large-Scale Models and Language Generation Tasks

Concerns might be raised that if DRONE can also be generalized to other scenarios such as larger models or other NLP tasks. To validate DRONE on larger models. We conduct experiments on RTE dataset with BERT-LARGE model. BERT-LARGE doubled number of layers and the dimension is increased from 768 to 1024. Average inference time for a data increased to 1405ms and it achieves accuracy 74. The overall result is 72.9 with inference time 1018ms (1.38x speedup). This result is comparable to our BERT-BASE result (1.31x speedup). To validate DRONE on other NLP tasks, We conducted the method on the machine translation task via OpenNMT. It provides a 2-layer transformer model on en-de translation. On the transformer part, DRONE achieves 1.76x speedup with BLEU from 33.47 to 33.26. Through these two additional experiments, we can validate that the proposed DRONE is generic in the sense that so long as the underlying model is composed of matrix computation, DRONE can compress the model regardless of model sizes and target tasks.

6.4.6 Can we directly learn low-rank structures by end-to-end training?

From an optimization perspective, a natural question to ask is whether the same optimal low-rank structure could be learned by end-to-end fine-tuning once the rank is decided. We conduct experiments on 4 tasks to verify this, and the results are summarized in Table 6.3. We start by performing DRONE on the task to achieve the desired accuracy, and perform SVD with the same set of ranks. Accuracy of SVD drops significantly for all tasks. We then fine-tune hyper-parameters as in [154][4] to fine-tune the above SVD results. After fine-tuning, the accuracy improves across all tasks, but none of it can reach the same performance as DRONE. This shows that due to the difficulty of optimizing a non-convex objective function, fine-tuning the SVD result may not achieve the best low-rank result. On the other hand, the proposed DRONE method under the the optimization problem (6.3) can obtain the provably optimal low-rank approximation at a much lower computational cost than the fine-tuned SVD.

6.4.7 Pre-processing of DRONE is not too costly

DRONE accelerates inference speed at the cost of a pre-processing step. Thus, It's natural to ask if DRONE will take long pre-processing time. Given the rank, prep-rocessing of DRONE has a one-time distribution extraction plus low-rank solving of equation (6.3). Depending on training data size, first stage takes 2 mins (RTE) to 20 mins (MNLI). Second stage has

[4]https://huggingface.co/transformers/v2.1.1/examples.html#glue

2 SVD computations and is about 5-10 mins. The matrix size involved is limited as we subsample training data. SVD costs about 3 mins and retrain costs from 5 mins to 2 hours depending on training data size. Thus, pre-processing of DRONE is about the same order as SVD but with much better performance. Distillation such as TinyBERT firstly has a general distillation of BERT-base on large data used to train original BERT, followed by task-specific distillation. Despite task-specific one is rather fast (15 mins to 2hrs), first stage takes a few days for a single GPU. Overall, DRONE is not costly compared to other methods.

6.5 Summary

In this chapter, we propose DRONE, a data-aware low-rank approximation, to achieve a better low-rank approximation in BERT models. DRONE leverages the fact that data distribution in NLP tasks usually lies in a lower-dimensional subspace. By considering the data distribution, we propose a data-aware low-rank approximation problem and provide a closed-form solution. Empirical results validate that DRONE can significantly outperform the vanilla-SVD method, and can achieve at least 20% acceleration with less than 3% accuracy loss. When DRONE is combined with distillation methods, it further achieves up to 15.3 times acceleration with less than 2% accuracy loss. Most importantly, this is a low-rank based method so it enjoys the model compression and inference time speedup simultaneously.

CHAPTER 7

Fast Graph-base Approximate Nearest Neighbor Search by Local Data Information

7.1 Introduction

In previous sections, we mostly focused on processing machine learning models directly. We demonstrated that through using both explicit and implicit information, machine learning models can be greatly compressed and the inference step could be largely accelerated. On the other hand, in modern e-commerce system or recommender systems, finding top-K elements within a database is an important step in the whole search pipeline. This operation is so generic that it also appears in many computer vision, natural language processing and machine mining applications. In addition, once the representation of the underlying objects are determined, it's no longer related to the model. Therefore, we also need to consider this case of processing "representations vectors" directly instead of processing models. In this chapter, we will show that data distribution can again help to accelerate this process directly.

K-Nearest Neighbor Search (KNNS) is a fundamental problem in machine learning [11], and is used in various applications in computer vision, natural language processing and data mining [28, 110, 123]. Further, most of the neural embedding-based retrieval and recommendation algorithms require KNNS in the inference phase to find items that are nearest to a given query [169]. Formally, consider a dataset D with n data points $\{d_1, d_2, ..., d_n\}$ where each data point has m-dimensional features. Given a query $q \in \mathbb{R}^m$, KNNS algorithms return the K closest points in D under a certain distance measure (e.g., $L2$ distance

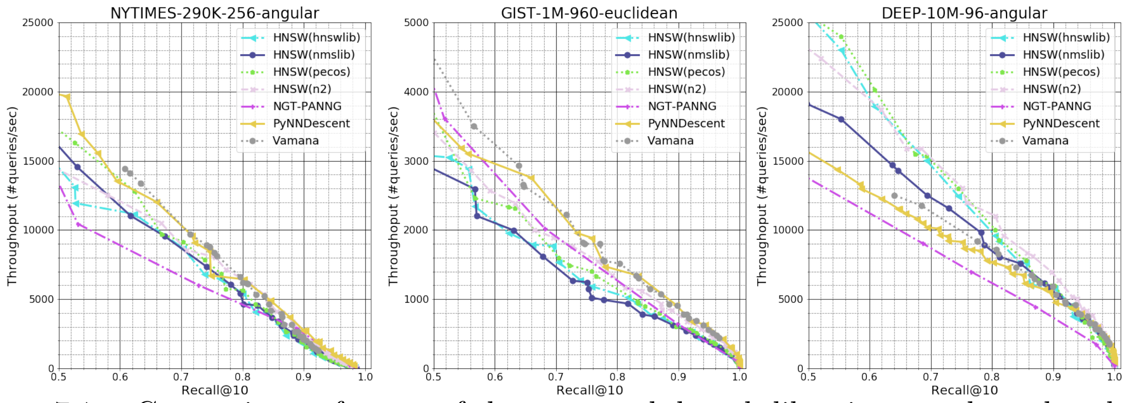


Figure 7.1: Comparison of state-of-the-art graph-based libraries on three benchmark datasets. Throughput versus recall@10 curve is used as the metric, where a larger area under the curve corresponds to a better method. We can observe no single method outperforms the rest on all datasets.

$\|\cdot\|_2$). Despite its simplicity, the cost of finding exact nearest neighbors is linear in the size of a dataset, which can be prohibitive for massive datasets in real time applications. It is almost impossible to obtain exact K-nearest neighbors without a linear scan of the whole dataset due to a well-known phenomenon called curse of dimensionality [65]. Thus, in practice, an exact KNNS becomes time-consuming or even infeasible for large-scale data. To overcome this problem, researchers resort to Approximate K-Nearest Neighbor Search (AKNNS). An AKNNS method proposes a set of K candidate neighbors $T = \{t_1, \cdots, t_K\}$ to approximate the exact answer. Performance of AKNNS is usually measured by recall@K defined as $\frac{|T \cap A|}{K}$, where A is the set of ground-truth K-nearest neighbors of q in the dataset D. Most AKNNS methods try to minimize the search time by leveraging pre-computed data structures while maintaining high recall [69]. There is a large body of AKNNS literature [16, 110, 148]; most of the efficient AKNNS methods can be categorized into three categories: quantization methods, space partitioning methods and graph-based methods. In particular, graph-based methods receive extensive attention from researchers due to their competitive performance. Many papers have reported that graph-based methods are among the most competitive AKNNS methods on various benchmark datasets [5, 16, 47, 148].

Graph-based methods work by constructing an underlying search graph where each node corresponds to a data point in D. Given a query q and a current search node c, at each step, an algorithm will only calculate distances between q and all neighboring nodes of c.

Once the local search of c is completed, the current search node will be replaced with an unexplored node whose distance is the closest to q among all unexplored nodes. Thus, neighboring edge selection of a data point plays an important role in graph-based methods as it controls the complexity of the search space. Consequently, most recent research is focused on how to construct different search graphs or design heuristics to prune edges in a graph to achieve efficient searches [47, 69, 105, 140]. Despite different methods having their own advantages, there is no clear winner among these graph construction approaches on all datasets. Following a recent systematic evaluation protocol [5], we evaluate performance by comparing throughput versus recall@10 curves, where a larger area under the curve corresponds to a better method. As shown in Figure 7.1, many graph-based methods achieve similar performance on three benchmark datasets. A method (e.g., PyNNDescent [37]) can be competitive on a dataset (e.g., GIST-1M-960) while another method (e.g., HNSW [105]) performs better on the other dataset (e.g., DEEP-10M-96). These results suggest there might not be a single graph construction method that works best, which motivates us to consider the research question: *Other than improving an underlying search graph, is there any other strategy to improve search efficiency of all graph-based methods?*.

In this chapter, instead of proposing yet another graph construction method, we show that for a given graph, part of the computations in the inference phase can be substantially reduced. Specifically, we observe that after a few node updates, most of the distance computations will not influence the search update. This suggests the complexity of distance calculation during an intermediate stage can be reduced without hurting performance. Based on this observation, we propose FINGER, **F**ast **IN**ference for **G**raph-based approximated nearest neighbor s**E**a**R**ch, which reduces computational cost in a graph search while maintaining high recall. Main contributions of this chapter are summarized as follows:

- We provide an empirical observation that most of the distance computations in the prevalent best-first-search graph search scheme do not affect final search results. Thus, we can reduce the computational complexity of many distance functions.

- Leveraging this characteristic, we propose an approximated distance based on using local data distribution. Specifically, we model angles between neighboring vectors using low-rank bases. In addition, angles of neighboring vectors in a graph tend to be distributed as a Gaussian distribution, and we propose a distribution matching scheme to achieve a better distance approximation.

- We provide an open source efficient C++ implementation of the proposed algorithm FINGER on the popular HNSW graph-based method. HNSW-FINGER outperforms many popular graph-based AKNNS algorithms in wall-clock time across various benchmark datasets by 20%-60%.

7.2 Related Work

There are three major directions in developing efficient approximate K-Nearest-Neighbours Search (AKNNS) methods. The first direction is still to traverse all elements in a database but reduce the complexity of each distance calculation; quantization methods represent this direction. The second direction is to partition search space into regions and only search data points falling into matched regions. This includes tree-based methods [136] and hashing-based methods [20]. The third direction is graph-based methods which construct a search graph and convert the search into a graph traversal.

Quantization Methods compress data points and represent them as short codes. Compressed representations consume less storage and thus achieve more efficient memory bandwidth usage [56]. In addition, the complexity of distance computations can be reduced by computing approximate distances with the pre-computed lookup tables. Quantization can be done by random projections [94], or learned by exploiting structure in the data distribution [107, 117]. In particular, the seminal Product Quantization method [71] separates data feature space into different parts and constructs a quantization codebook for each chunk. Product Quantization has become the cornerstone for most recent quantization methods [39, 56, 109, 157]. There is also work focusing on learning transformations in accordance with product quantization [48]. Most recent quantization methods achieve competitive results on various benchmarks [56, 74].

Space Partition Methods includes hashing-based and tree-based methods. Hashing-based Methods generate low-bit codes for high dimensional data and try to preserve the similarity among the original distance measure. Locality sensitive hashing [50] is a representative framework that enables users to design a set of hashing functions. Some data-dependent hashing functions have also been designed [62, 149]. Nevertheless, a recent review [16] reported the simplest random-projection hashing [20] actually achieves the best performance. According to this review, the advantage of hashing-based methods is simplicity and low memory usage; however, they are significantly outperformed by graph-based methods. Tree-based Methods learn a recursive space partition function as a tree following some criteria. When a new query comes, the learned partition tree is applied to the query and the distance computation is performed only on relevant elements falling in the same sub-tree. Representative methods are KD-tree [136] and R^*-tree [10]. It is observed in previous studies that tree-based methods only work for low-dimensional data and their performances drop significantly for high-dimensional problems [16].

Graph-based Methods date back to theoretical work in graph theory [6, 33, 89]. However, these theoretical guarantees only work for low-dimensional data [6, 89] or require expensive ($O(n^2)$ or higher) index building complexity [33], which is not scalable to large-scale datasets. Recent works are mostly geared toward approximations of different proximity graph structures to improve nearest neighbor search. There is a series of works on approximating K-nearest-neighbour graphs [44, 57, 61, 73]. Most recent works approximate monotonic graph [45] or relative neighbour graph [4, 105]. In essence, these methods first construct an approximated K-nearest-neighbour graph and prune redundant edges by different criteria inspired by different proximity graph structures. Some other works mixed the above criteria with other heuristics to prune the graph [47, 69]. Some pruning strategies can even work on randomly initialized dense graphs [69]. According to various empirical studies [5, 16, 61], graph-based methods achieve very competitive performance among all AKNNS methods. Despite concerns about scalability of graph-based methods due to their larger memory usage

[39], it has been shown that graph-based methods can be deployed in billion scale commercial usage [45]. In addition, recent studies also demonstrated that graph-based AKNNS can scale quite well on billion-scale benchmarks when implemented on SSD hard-disks [26, 69]. In this work, we aim at demonstrating a generic method to accelerate the inference speed of graph-based methods so we will mainly focus on in-memory scenarios.

7.3 Methods

7.3.1 Observation: Most distance computations do not contribute to better search results

Once a search graph is built, graph-based methods use a greedy-search strategy (Algorithm 7.1) to find relevant elements of a query in a database. It maintains two priority queues: candidate queue that stores potential candidates to expand and top results queue that stores current most similar candidates (line 1). At each iteration, it finds the current nearest point in the candidate queue and explores its neighboring points. An upper-bound variable records the distance of the furthest element from the current top results queue to the query q (line 4). The search will stop when the current nearest distance from the candidate queue is larger than the upper-bound (line 5), or there is no element left in the candidate queue (line 2). The upper-bound not only controls termination of the search but also determines if a point will present in the candidate queue (line 11). An exploring point will not be added into the candidate queue if the distance from the point to the query is larger than the upper-bound. Thus, upper-bound plays an important role as we need to spend computational resources on distance calculation (dist function in line 11) but it might not influence search results if the distance is larger than the upper-bound. Empirically, as shown in Figure **??**, we observe in two benchmark datasets that most of the explorations end up having a larger distance than the upper-bound. Especially, starting from the mid-phase of a search, over 80 % of distance calculations are larger than the upper-bound. Using greedy graph search will inevitably waste a significant amount of computing time on non-influential operations. [90]

also found this phenomenon and proposed to learn an early termination criterion by an ML model. Instead of only focusing on the near-termination phase, we propose a more general framework by incorporating the idea of reducing the complexity of distance calculations into a graph search. The fact that most distance computations do not influence search results suggests that we don't need to have exact distance computations. A faster distance approximation can be applied in the search.

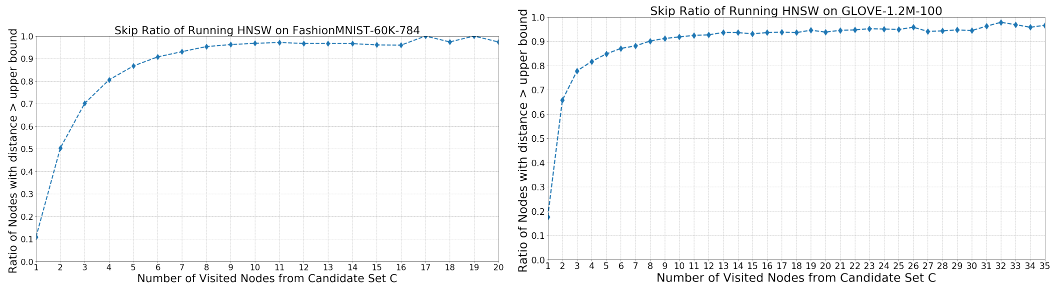


Figure 7.2: Illustration of the empirical observation that most points in a database will have distance to query larger than the upper-bound. (a) FashionMNIST-60K-784 dataset (b) Glove-1.2M-100 dataset. Starting from the 5th step of greedy graph search (i.e., running line 2 in Algorithm 7.1 five times), both experiments show more than 80% of data points will be larger than the current upper-bound.

Algorithm 7.1: Greedy Graph Search

```
Input: graph $G$, query $q$, start point $p$, distance dist(), number of nearest points to
       return $efs$
Output: top results queue $T$
1  candidate queue $C = \{p\}$ , currently top results queue $T = \{p\}$ , visited $V = \{p\}$
2  while $C$ is not empty do
3      cur ← nearest element from $C$ to $q$ (i.e., current nearest point to expand)
4      ub ← distance of the furthest element from $T$ to $q$ (i.e., upper bound of the
         candidate search)
5      if dist(cur, q) $>$ ub then
6          return $T$
7      for point $n \in$ neighbour of cur in $G$ do
8          if $n \in V$ then
9              continue
10         V.add($n$)
11         if dist(n, q) $\leq$ ub or $|T| \leq efs$ then
12             $C$.add(n)
13             $T$.add(n)
14             if $|T| > efs$ then
15                 remove furthest point to $q$ from $T$
16             ub ← distance of the furthest element from $T$ to $q$ (i.e., update ub)
17 return $T$
```

7.3.2 Modeling Distribution of Neighboring Residual Angles

Given a query q and the current nearest point to q in the candidate queue c, in Line 7 of Algorithm 7.1, we will expand the search by exploring neighbors of c. Consider a specific neighbor of c called d, we have to compute distance between q and d in order to update the search results. Here, we will focus on the $L2$ distance (i.e., $Dist = \|q - d\|_2$). The derivations of inner-product and angle distance are provided in the Supplementary A. As shown in the previous section, most distance computations will not contribute to the search in later stages, we aim at finding a fast approximation of $L2$ distance. A key idea is that we can use local data distribution to represent the neighboring node. Specifically, we can use the center node c to represent q (and d) as a vector along c (i.e., projection) and a vector orthogonal to c (i.e., residual):

$$q = q_{proj} + q_{res}, \quad q_{proj} = \frac{c^T q}{c^T c} c, \quad q_{res} = q - q_{proj}. \tag{7.1}$$

In other words, we treat each center node as a basis and project the query and its neighboring points onto the center vector so query and data can be written as $q = q_{proj} + q_{res}$ and $d = d_{proj} + d_{res}$ respectively. With this formulation, the squared $L2$ distance can be written as:

$$\begin{aligned} Dist^2 &= \|q - d\|_2^2 = \|q_{proj} + q_{res} - d_{proj} - d_{res}\|_2^2 = \|(q_{proj} - d_{proj}) + (q_{res} - d_{res})\|_2^2 \\ &= \|(q_{proj} - d_{proj})\|_2^2 + \|(q_{res} - d_{res})\|_2^2 + 2(q_{proj} - d_{proj})^T (q_{res} - d_{res}) \\ &\overset{(a)}{=} \|(q_{proj} - d_{proj})\|_2^2 + \|(q_{res} - d_{res})\|_2^2 \\ &= \|(q_{proj} - d_{proj})\|_2^2 + \|q_{res}\|_2^2 + \|d_{res}\|_2^2 - 2q_{res}^T d_{res}, \end{aligned} \tag{7.2}$$

where (a) comes from the fact that projection vectors are orthogonal to residual vectors so the inner product vanishes. For d_{proj} and d_{res}, we can pre-calculate these values after the search graph is constructed. For q_{proj}, notice that center node c is extracted from the candidate queue (line 3 of Algorithm 7.1). That means we must have already visited c before. Thus, $\|q - c\|_2$ has been calculated and we can get $q^T c$ by a simple algebraic manipulation:

$$q^T c = \frac{\|q\|_2^2 + \|c\|_2^2 - \|q - c\|_2^2}{2}.$$

Therefore, the only uncertain term in Eq. (7.2) is $q_{res}^T d_{res}$. If we can estimate this term with less computational resources, we can obtain a fast yet accurate approximation of $L2$ distance. Since we don't have direct access to the distribution of q and thus q_{res}, we hypothesize we can instead use the distribution of residual vectors between neighbors of c to approximate the distribution of $q_{res}^T d_{res}$ term. The rationale behind this is as we only approximate $q_{res}^T d_{res}$ when q and c are close enough (i.e., c is selected in line 3 of Algorithm 7.1), both q and d could be treated as near points in our search graph and thus interaction between q_{res} and d_{res} might be well approximated by ${d'_{res}}^T d_{res}$, where d' is another neighbouring point of c and d'_{res} is its residual vector. Empirically, as shown in the left column of Figure 7.3, angles between residual vectors of sampled neighbors (i.e., $d, d' \in \text{neighbor}(c)$) distributes like a Gaussian. In particular, compared to the distribution of direct inner-product ${d'_{res}}^T d_{res}$ (right column of Figure 7.3), the distribution $\cos(d'_{res}, d_{res})$ is less-skewed and thus more alike Gaussian. This motivates us to design an efficient approximator of $\cos(q_{res}, d_{res})$ and obtain $q_{res}^T d_{res}$ by $\|q_{res}\|_2 \|d_{res}\|_2 \cos(q_{res}, d_{res})$.

7.3.3 FINGER: Fast Inference by Low-rank Angle Estimation and Distribution Matching

Low-rank Estimation Motivated by the above derivations, we aim at finding an efficient estimation of angles between all pairs of neighboring residual vectors. In AKNNS literature, a popular method for estimating this is Locality Sensitive Hashing (LSH) and its variants. In particular, Random Projection-based LSH (RPLSH) [20] is reported to achieve good average performance on various benchmark datasets [16]. RPLSH samples r random vectors from Normal distribution to form a projection matrix $P \in \mathbb{R}^{r \times m}$, where m is the dimension of data and query. $L2$ distance between two vectors $x, y \in \mathbb{R}^m$ can be approximated by the distance in projected space, and the error,

$$\left\| \|Px - Py\|_2^2 - \|x - y\|_2^2 \right\|_2,$$

is bounded probabilistically [75]. We can further binarize the projection results to form a compact representation, and the angle between x and y can be approximated by hamming

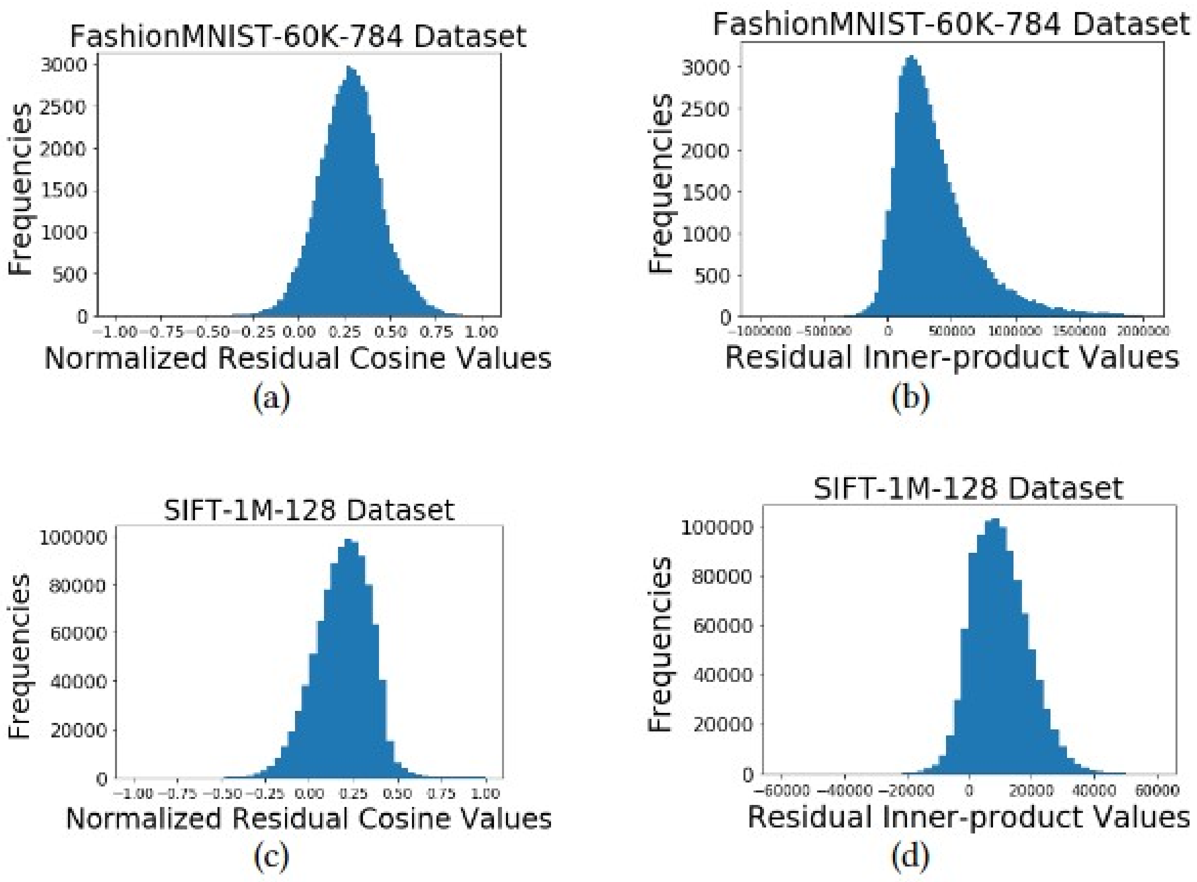


Figure 7.3: Illustration of the empirical observation that normalized cosine values of neighboring residual vectors distribute as a Gaussian distribution on FashionMNIST-60K-784 and SIFT-1M-128. Left column (a) and (c): angles of neighbouring residual pairs distribute alike Gaussian. Right column (b) and (d): un-normalized inner-product values between neighbouring residual pairs are more skewed.

distance of the signed results: hamm(sgn(Px),sgn(Py))$\frac{\pi}{r}$. However, there is an immediate disadvantage with this approach. Random projection guarantees worst case performance [43] and it is oblivious of the data distribution. Since we can sample abundant neighboring residual vectors from the training database, we can leverage the data information to obtain a better approximation. Formally, given an existing search graph $G = (D, E)$ where D are nodes in the graph corresponding to data points and E are edges connecting data points, we collect all residual vectors into $D_{res} \in \mathbb{R}^{m\times N}$, where N is total number of edges in G (i.e., $|E|$); and we assume D_{res} spans the whole space which residual vectors lie in. The approximation problem can be formulated as the following optimization problem:

$$\underset{P\in\mathbb{R}^{r\times m}}{\operatorname{argmin}} \mathbb{E}_{x,y\sim D_{res}} \left\| \|Px - Py\|_2^2 - \|x - y\|_2^2 \right\|_2, \tag{7.3}$$

where we aim at finding an optimal P minimizing the approximating error over the residual pairs D_{res} from training data. It's not hard to see that the Singular Value Decomposition (SVD) of D_{res} will provide an answer to the above optimization problem, and thus we can use SVD to find better r lower-dimensions to estimate the angle of neighboring residual vectors.

Proposition 7.1. Given a residual vector matrix $D_{res} \in \mathbb{R}^{m\times N}$, and denoting $D_{res} = USV^T$ as the Singular Value Decomposition of D_{res}. $U_{1:r}$, the first r columns of U is an optimal solution of optimization problem Eq. (7.3).

Proof. We can firstly construct all possible pairs of $\frac{N(N-1)}{2}$ combinations of sample of vectors x, y from D_{res} and compile all $\frac{N(N-1)}{2}$ pairs into two matrices X and Y. With this notation, we can rewrite the original optimization into matrix form:

$$\begin{aligned}
&\underset{P\in\mathbb{R}^{r\times m}}{\operatorname{argmin}} \mathbb{E}_{x,y\sim D_{res}} \| \|Px - Py\|_2^2 - \|x - y\|_2^2 \|_2 \\
&= \underset{P\in\mathbb{R}^{r\times m}}{\operatorname{argmin}} \| \|PX - PY\|_F^2 - \|X - Y\|_F^2 \|_2^2 \\
&= \underset{P\in\mathbb{R}^{r\times m}}{\operatorname{argmin}} (\|PX - PY\|_F^2 - \|X - Y\|_F^2)^2,
\end{aligned}$$

where $\|\cdot\|_F$ denotes matrix frobenius norm. By introducing the matrix notation, we then explicitly write out the overall objective function without the sampling. We can further denote $Z = X - Y$. Z matrix then denotes all possible pairs of vector difference from our original distribution. The objective function can then further be written into:

$$\begin{aligned}
&\operatorname*{argmin}_{P\in\mathbb{R}^{r\times m}}(\|PX - PY\|_F^2 - \|X - Y\|_F^2)^2 \\
&= \operatorname*{argmin}_{P\in\mathbb{R}^{r\times m}}(\|PZ\|_F^2 - \|Z\|_F^2)^2 \\
&= \operatorname*{argmin}_{P\in\mathbb{R}^{r\times m}}(\|PU_zS_zV_z^T\|_F^2 - \|U_zS_zV_z^T\|_F^2)^2,
\end{aligned}$$

where $U_zS_zV_z^T$ denotes the SVD decomposition of Z. By the basic properties of SVD decomposition, we know that $\|U_zS_zV_z^T\|_F^2 = \|S_z\|_F^2$ as U_z and V_z are unitary matrices. $\|S_z\|_F^2$ equals sum of square of singular values of Z. Similarly, $\|PU_zS_zV_z^T\|_F^2 = \|PU_zS_z\|_F^2$. Thus it's not hard to see that the objective function is to find a projection direction which will result the minimal difference between the projected S_z and full sum of squared eigenvalues of S_z. Thus, the optimal answer is the top r directions as of columns of matrix U_z as it will cancel out the top r square of eigenvalues of S_z which happens to be the largest ones.

The remaining thing is to show that SVD of Z is essentially the same as SVD of D_{res}. Notice that both X and Y are just duplicating and re-ordering of D_{res}. So both X,Y share the same basis of D_{res}. Denote SVD results of $D_{res} = USV^T$. We can then represent $X = USV_x^T$ and $Y = USV_y^T$. Consequently, we can also represent the SVD of Z as $Z = X - Y = USV_x^T - USV_y^T = US(V_x - V_y)^T$ so we can see that it shares the same basis as D_{res} and the proof is complete.

□

Distribution Matching In addition to efficient low-rank estimation of angles, we further propose a distribution matching method to improve the performance. Despite as discussed in Section 7.3.2 that angles between neighbouring residual vectors tend to be distributed alike Gaussian, this attribute only partially transfers to the distribution of angles approximated

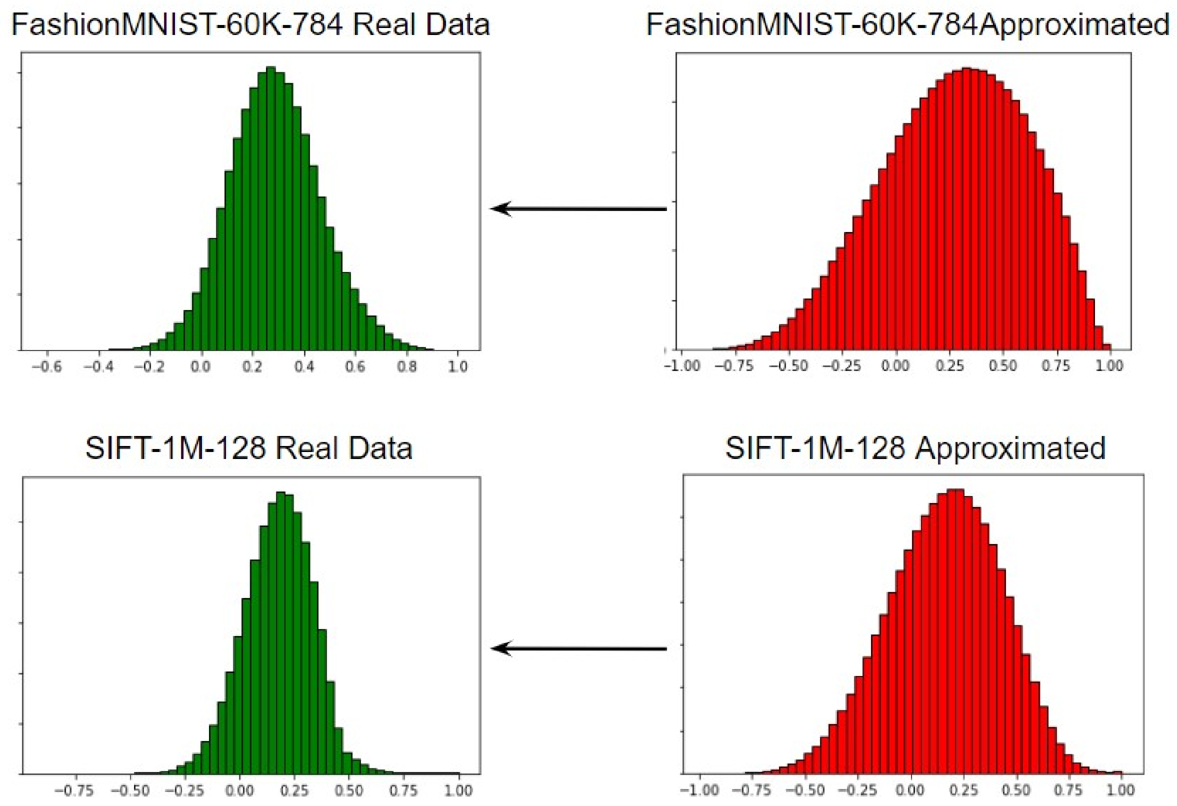


Figure 7.4: Illustration of Distribution Matching. In the left column, we show correct angle distributions of FashionMNIST-60K-784 and SIFT-1M-128. In the right column, we show angles of neighbouring residual pairs calculated by low-rank approximation ($r = 16$). Our goal is to transform approximated results (in red) into real ones (in green).

by low-rank computations as shown in Figure 7.4. Although the approximated distribution still looks alike Gaussian, its distribution is slightly skewed. Furthermore, its mean is shifted and its variance is larger than the real data distribution. To mitigate this, we propose to transform the approximated distributions into real data distributions by matching their mean and variance. Formally, assume angles of neighboring residual vectors follows a Gaussian distribution $\mathcal{N}(\mu, \sigma)$, and the approximated angles distributes as $\mathcal{N}(\hat{\mu}, \hat{\sigma})$. Given a residual pair x and y with a low-rank projection matrix P, we can calculate the approximated angle $\hat{t} = \cos(Px, Py)$. Under our assumption that it comes from a draw of $\mathcal{N}(\hat{\mu}, \hat{\sigma})$, the value can be transformed by $t = (\hat{t} - \hat{\mu})\frac{\sigma}{\hat{\sigma}} + \mu$. The transformed angle estimation t then follows $\mathcal{N}(\mu, \sigma)$ as desired. Parameters $\mu, \sigma, \hat{\mu}, \hat{\sigma}$ can be estimated by using training data.

Overall Algorithm Construction of FINGER can be summarized in Algorithm 7.2. Our aim is to provide a generic acceleration for all graph-based search. Thus, we can build the search index from any existing graph G. FINGER first iterates through all nodes in the graph. For each node, FINGER samples a pair of distinct nodes from its neighbors. In addition, we also calculate the residual vector of one sampled point and store it for later usage. We hypothesize the collected residual vectors D_{res} spans the residual space, and we can find its optimal low-rank approximation by SVD. Once the low-rank projection P is ready, we can estimate the mean and variance of angle distribution and approximated distribution respectively (i.e., line 9, 10 in Algorithm 7.2). Certainly, this distribution matching scheme would still produce error. We further compute the average $L1$ error between real and approximated angles to serve as an error correction term. With this information saved in a search index, Algorithm 7.3 approximates the distance between a query q and a data point d. Notice that we explicitly write out the projection matrix P and center node c in Algorithm 7.3 to make it easier to understand the full approximation workflow. In practice, the projected residual vector Pd_{res} can be pre-computed and stored.

Detailed computation of Approximate Distance As mentioned in the main text, we explicitly write out the projection matrix P and center node c in Algorithm 7.3 to make it easier to understand the full approximation workflow. In practice, the projected residual vector Pd_{res} can be pre-computed and stored in the search index to save inference time. Pq_{res} can also be easily calculated by

$$Pq_{res} = P(q - q_{proj}) = Pq - Pq_{proj} = Pq - \frac{c^T q}{c^T c} Pc.$$

Pq needs only be calculated once for whole graph search so the cost is limited when the rank r is not large. As we point out in Section 7.3.1, $c^T q$ must already be calculated when we explore neighbours of c. $c^T c$ and Pc can again be pre-computed and stored. Thus, the cost of Pq_{res} is limited to a low-dimensional vector subtraction. To apply FINGER, we only need to slightly modify Algorithm 7.1. Specifically, we replace distance function dist(n,q) in line 5 of

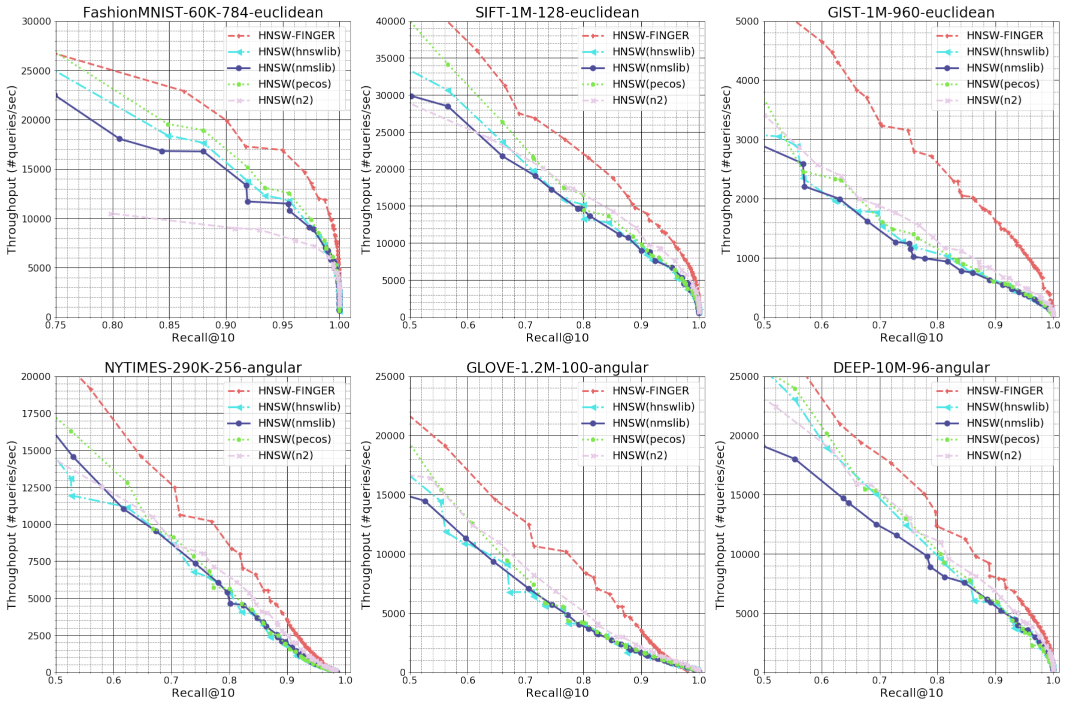


Figure 7.5: Experimental results of graph-based methods. Throughput versus Recall@10 chart is plotted for all datasets. Top row presents datasets with $L2$ distance measure and bottom row presents datasets with angular distance measure. We can observe a significant performance gain of FINGER over all existing graph-based methods.

Algorithm 7.1 with the approximation (i.e., Algorithm 7.3). If the approximated distance is larger than the upper-bound variable, the search continues. Otherwise, we calculate precise distance between q and d and update the candidate queue correspondingly. This will make sure that all distance information in candidates set C is correct so the algorithm won't terminate too early. The complete modified algorithm is listed in the Supplementary B.2, and the selection of rank r is discussed in Supplementary B.4.

7.4 Experimental Results

7.4.1 Experimental Setups

Baseline Methods We compare FINGER to the most competitive graph-based and quantization methods. We include different implementations of the popular HNSW methods such as NMSLIB [105], n2[5], PECOS [163] and HNSWLIB [105]. Other graph construction

[5]https://github.com/kakao/n2/tree/master

methods include NGT-PANNG [140] , VAMANA(DiskANN) [69] and PyNNDescent [37]. Since our goal is to demonstrate FINGER can improve search efficiency of an underlying graph, we mainly include these competitive methods with good python interface and documentation. For quantization methods, we compare to the best performing ScaNN [56] and Faiss-IVFPQFS [74]. In experiments, we combine FINGER with HNSW as it is a simple and prevalent method. The implementation of HNSW-FINGER is based on a modification of PECOS as its codebase is easy to read and extend. Pre-processing time and memory footprint are discussed in the Supplementary B.5.

Algorithm 7.2: Construction of FINGER

```
Input: graph $G = (D, E)$, rank $r$
Output: projection matrix $P$ and distribution parameters $\mu, \sigma, \hat{\mu}, \hat{\sigma}$, $\epsilon$
1  $D_{res} = \{\}$, $S = \{\}$ for $c \in D$ do
2      Sample $d$,$d' \in$ neighbors of $c$ and add it to $S$
3      Calculate $d_{res}$, $D_{res}$.add($d_{res}$)
4  Calculate SVD $U, S, V = \text{SVD}(D_{res})$
5  $P = U^T_{1:r}$, $X = \{\}$, $Y = \{\}$
6  for pair $d, d' \in S$ do
7      X.add(cos($d$,$d'$)), Y.add(cos($Pd$,$Pd'$))
8  $N =$ size of $X$,
9  $\mu_x = \frac{1}{N} \sum_{x \in X} x$, $\sigma_x = \frac{1}{N} \sum_{x \in X} (x - \mu_x)^2$ ,
10 $\mu_y = \frac{1}{N} \sum_{y \in Y} y$, $\sigma_y = \frac{1}{N} \sum_{y \in Y} (y - \mu_y)^2$
11 $\epsilon = \frac{1}{N} \sum_{i=1}^{N} \left|(Y_i - \mu_y)\frac{\sigma_x}{\sigma_y} + \mu_x - X_i\right|$
12 return $P$, $\mu, \sigma, \hat{\mu}, \hat{\sigma}$,$\epsilon$
```

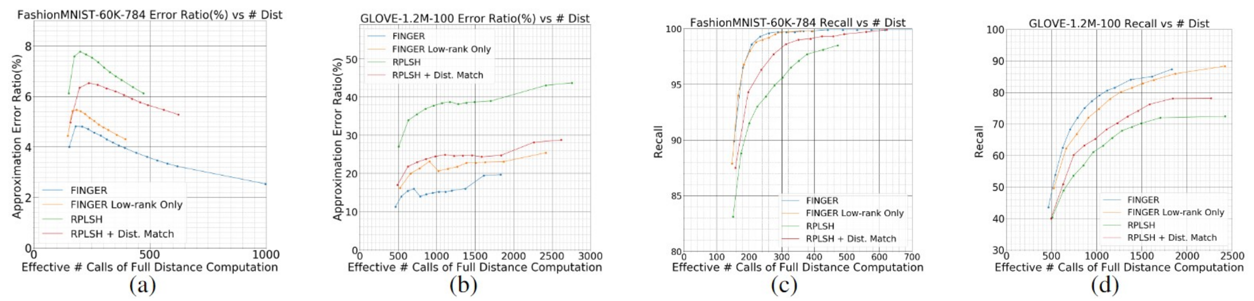


Figure 7.6: Results of ablation studies on FashionMNIST-60K-784 and GLOVE-1.2M-100. (a) and (b) show approximation error(%) vs effective number of full distance calls. FINGER achieves smaller error than RPLSH. (c) and (d) show recall@10 vs effective number of full distance calls. FINGER achieves higher recalls.

Algorithm 7.3: Approximate Distance Function

```
Input: query $q$, projection matrix $P$, center node $c$, data point $d \in$ neighbors of $c$,
       distribution parameters $\mu, \sigma, \hat{\mu}, \hat{\sigma}$, $\epsilon$
Output: $t$, the approximated distance between $q$ and $d$
1 compute $q_{res}$ and $d_{res}$ with $c$ and Eq. 7.1
2 compute $\hat{t} = \cos(Pq_{res}, Pd_{res})$
3 $t = (\hat{t} - \hat{\mu})\frac{\sigma}{\hat{\sigma}} + \mu$, t = t + $\epsilon$, return t
```

7.4.2 Improvements of FINGER over HNSW

In Figure 7.5, we demonstrate how FINGER accelerates the competitive HNSW algorithm on all datasets. Since FINGER is implemented on top of PECOS, it's important for us to check if PECOS provides any advantage over other HNSW libraries. Results verify that across all 6 datasets, the performance of PECOS does not give an edge over other HNSW implementations, so the performance difference between FINGER and other HNSW implementations could be mostly attributed to the proposed approximate distance search scheme. We observe that FINGER greatly boosts the performance over all different datasets and outperforms existing graph-based algorithms. FINGER works better not only on datasets with large dimensionality such as FashionMNIST-60K-784 and GIST-1M-960, but also works for dimensionality within range between 96 to 128. This shows that FINGER can accelerate the distance computation across different dimensionalities. Results of comparison to most

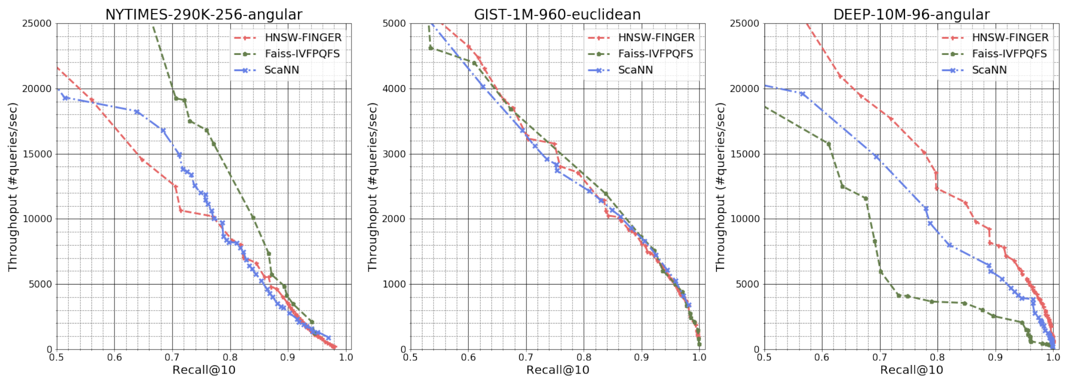


Figure 7.7: Comparisons to competitive quantization methods. Throughput versus Recall@10 chart is plotted for three datasets. We can observe each method has its pros and cons and there is no single method which performs best on all datasets.

competitive graph-based methods are shown in Figure B.1 of the Supplementary B.3. Briefly speaking, HNSW-FINGER outperforms most state-of-the-art graph-based methods except FashionMNIST-60K-784 where PyNNDescent achieves the best and HNSW-FINGER is the runner-up. Notice that FINGER could also be implemented over other graph structures including PyNNDescent. We chose to build on top of HNSW algorithm only due to its simplicity and popularity. Studying which graph-based method benefits most from FINGER is an interesting future direction. Here, we aim at empirically demonstrating approximated distance function can be integrated into the greedy search for graph-based methods to achieve a better performance.

7.4.3 Ablation Study

We conduct an ablation study to see the effectiveness of each component of FINGER. First, we compare FINGER to the popular random projection locality hashing (RPLSH) for angle estimation. Since we have greatly optimized C++ implementation of FINGER, a direct comparison on wall-clock time won't be fair. Instead, we compare two schemes by counting the effective number of distance function calls. We collect the number of full distance calls and approximate distance calls separately, and combine them into an effective number of distance calls. For example, if we call full m-dimensional distance a times and b times of r-dimensional approximate computations, we would have an effective distance calls of

$a + b\frac{r}{m}$ times. We firstly analyze estimation quality by approximation error defined as $\frac{|t-\hat{t}|}{|t|}$ where t is the true cosine angle value and $\hat{t}$ is the approximated value. Ideally, we could expect a better approximation scheme results in a smaller approximation error. Certainly, a smaller approximation doesn't necessarily yield better recall. Thus, we will also analyze the performance based on recall. Results of trade-off between approximation error (%) and effective number of distance calls are shown in Figure 7.6(a) for FashionMNIST-60K-784 and 7.6(b) for GLOVE-1.2M-100. Corresponding results of recall vs effective distance calls are shown in Figure 7.6(c) and 7.6(d). We can see FINGER achieves smaller approximation errors compared to RPLSH on both datasets, which shows that FINGER is indeed a better low-rank approximation given data distribution. We also observe smaller approximation error transfers to higher recalls on both datasets. In addition, we apply distribution matching on RPLSH and found out this will greatly improve RPLSH. This shows distribution matching is a generic method that improves the performance of all different angle estimation methods. But even with the aid of distribution matching, RPLSH cannot achieve similar performance as FINGER and this shows the superiority of SVD results. Since FINGER consists of a low-rank approximation module plus a distribution matching module, we are interested in studying their own effectiveness. We conduct similar analysis on full method and low-rank only version of FINGER shown in Figure 7.6. Low-rank approximation alone still provides a much better angle estimation compared to RPLSH. Even without the distribution matching scheme, low-rank angle estimation outperforms RPLSH with distribution matching. We also observe limited difference between FINGER and FINGER without distribution matching in FashionMNIST-60K-784. However, the difference is more significant when it comes to GLOVE-1.2M-100 which still shows effective distribution matching.

7.4.4 Comparison to Quantization Results

In addition to graph-based method, we are also interested in seeing the performance of HNSW-FINGER compared to the state-of-the-art quantization methods. Results of comparisons to quantization methods are shown in Figure 7.7. As we can observe, there is no single method achieving the best performance over all tasks. Faiss-IVFPQFS performs well on NYTIMES-290K-256 but fails on DEEP-10M-96. ScaNN performs consistently well on all datasets but it doesn't achieve top performance on anyone. HNSW-FINGER performs competitively on GIST-1M-960 and DEEP-10M-96 but worse on NYTIMES-290K-256. These results showed that quantization provides some advantages over graph-based methods but the advantage is not consistent across datasets. Studying how to combine the advantage of quantization methods with FINGER and graph-based methods is an interesting future direction.

7.5 Summary

In this work, we propose FINGER, a fast inference method for graph-based AKNNS. FINGER approximates distance function in graph-based method by estimating angles between neighboring residual vectors. FINGER constructs low-rank bases to estimate residual angles and use distribution matching to achieve a better precision. The approximated distance can be used to bypass unnecessary distance evaluations, which translates into a faster searching. Empirically, FINGER on top of HNSW is shown to outperform all existing graph-based methods.

This work mainly focuses on accelerating existing models with approximate computations. It doesn't directly touch any controversial part of the data and thus it's unlikely providing any negative social impact. When used correctly with positive information to spread. It can help to accelerate the propagation as the work accelerates the inference speed.

CHAPTER 8

Conclusion

8.1 Summaries

This thesis focused on two pillars of efficient machine learning, namely model size reduction problem and inference time speedup problem. This thesis explores a number methods to achieve fundamental efficient machine learning. These methods share certain similarities and thus We can draw some general conclusions and summarize as follows.

Firstly, data distribution is extremely useful to tackle the efficient machine learning problem. In particular, "data distribution" doesn't restrict to using underlying data input directly, it refers to various type of features dependent on the input so the applicability Through the thesis, we have shown both explicit distribution such as frequency information, and implicit information such as clusters learned from latent vector spaces.

Second, efficient machine learning is hardware dependent. The design of efficient machine learning will require a software hardware co-design. We illustrated in this thesis that many operations are algorithmically simple but require special hardware to achieve the desired speed. On the other hand, certain methods such as sparse pruning will destruct the computational structure such that certain hardware related acceleration will become unavailable. Therefore, we have to understand the characteristic of the underlying hardware before choosing the tool to solve the efficient machine learning problem.

Third, we might not be able to achieve both model reduction and inference time speedup simultaneously. We have illustrated this important conclusion in chapter 6. Thus, it's important to consider the most important factor in advance when solving efficient machine learning problems. We also provide a generic method to achieve both attributes in the same time: the low-rank based methods. By using data-aware low-rank approximation, we can get the model size greatly reduced and enjoy the corresponding speedup in inference stage.

Last, the applicability of data distribution based methods is fairly general. We demonstrated that the technique can be applied to various scenarios including data mining, information retrieval, natural language processing and recommender systems. We could expect that more domains have this characteristics.

8.2 Limitations

Despite that we have discussed many aspect of efficient machine learning, there are still certain parts of the topics we didn't cover. In addition, some of the discussed methodologies are not immediately applicable to all situations. We summarize some limitations of the thesis as follows.

First, this thesis doesn't discuss the energy consumption issues. Although, energy usage is greatly related to the inference time, there is no guarantee that a faster inference would always leads to a small energy consumption. In addition, energy consumption is mostly measured at a larger scale (e.g., server clusters), the effect of inference time speedup might only be one of influencing factors so we can't simply draw any conclusions just based on the inference time results.

Second, this thesis mostly focused on inference stage only. In modern machine learning community, using tens or hundreds of GPUs to train the model is not uncommon. Even worse, many models are based on hyper-parameter tuning which is build on multiple rounds of trail-and-error. This leads to a significant training time and energy consumption. To really achieve the efficient machine learning, we also need to consider the training time usage. But this topic is not covered in the thesis which is a great limitation.

Third, most part of the thesis requires data dependent information. To obtain this information, we might have to add many codes in the state-of-the-art implementations of latest machine learning libraries. This task is challenging and currently there is no automated way of obtaining this. Therefore, the applicability is limited as to apply these methods, it requires some experts who are good at both machine learning libraries and machine learning algorithms. This might not be practical for certain industries or application domains.

8.3 Future Work

Following issues are plans for immediate future works which could either resolve above limitations or push the existing methods further.

First and foremost, I plan to expand the methods into training phase. As we mentioned above, the training phase usually costs a lot of energy. To really achieve efficient machine learning, considering the training energy preservation is also important. Could we leverage the latent structures appeared in the early phase of training becomes an interesting direction to explore in future.

Second, it's interesting to keep exploring different types of compressing methods. In particular, tensor-based decomposition methods, a generalization of low-rank approximation, could be an interesting object to study. We could be expecting tensor-based method to possess similar properties as low-rank methods which enjoy both compression and speedup. Tensor-based methods also have the potential to achieve a much better compression due to its more compact representation.

Third, the thesis focused on the discussion of efficient machine learning "models". Whereas, data is assumed to be fixed once it's given. It's possible that within data, there are already rich structures that we could exploit. Discussion of efficient machine learning via data compression is also an attractive way to resolve the problem. The interactions between exploiting data and exploiting model could also lead to better solutions to achieve efficient machine learning.

APPENDIX A

Appendix in Data-aware Low-rank Approximation

A.1 An algorithm to Search of Ranks under DRONE

The input to Algorithm 6.2 consists of training data, the model with all parameters of weight matrices and original training loss. In addition, a pre-defined search grid is also necessary. Taking $W \in R^{768\times768}$ as an example, we can perform a grid search for a proper low rank k over $[1, 768]$ such as $\{96, 192, 288, 384, \ldots, 768\}$. The finer the grid, the more compressed model we could get at the cost of longer running time of the DRONE method. With these input parameters, we firstly distribute the total allowed loss into each individual module. We then iteratively apply Algorithm A.1 following the computational sequence illustrated in Figure 4.1. For each module, we search the rank k by going through the grid. If the approximated result will not increase the allowed loss increase ratio of the component, we will end the search and tie the found rank to the component and move on. The procedure will continue until all components are compressed. The whole process could guarantee us that the final loss L' of the compressed model $\hat{M}$ would not be greater than $(1 + r)L$, where L is the original loss before approximation.

A.2 Efficiency and Efficacy Trade-off Graph

In this paper, we mentioned that we report the result of 3% accuracy drop as the performance of the baseline methods and DRONE. However, as we mentioned above that all the approximation methods need to consider efficiency and efficacy trade-off. 3% is chosen according to the literature. Here, we show two exemplar graph on MRPC and SST-2 task to

Algorithm A.1: Overall Low-rank Model Approximation Algorithm

```
Input: training data $D_{train}$, original weight matrix $W$. prediction Model $M$, total
       allowed loss increase ratio $r$, Observed inference time $E$, Search grids of
       ranks for each module $G$, original Training loss $L$;
Output: Low-rank Model $\hat{M}$
1  # Distribute allowed ratio r into each module by $E$ ;
2  $E_{min} \leftarrow \text{argmin}_{l,i} E_{l,i}$ ;
3  $E_{l,i} \leftarrow \frac{E_{l,i}}{E_{min}}$ ;
4  $E_b \leftarrow exp(\frac{log(1+r)}{\sum_{l,i} E_{l,i}})$ ;
5  $R_{l,i} \leftarrow E_b^{E_{l,i}} - 1$ ;
6  for $l = 1, \cdots, total\ layers$ do
7      for $module\ m_i \in M_l$ do
8          $W_{l,i} \leftarrow$ $l$-th layer parameter of module $m_i$ ;
9          (e.g., 2nd feed-forward matrix in first layer.) ;
10         for $i = 1, \cdots, |G_{l,i}|$ do
11             $k \leftarrow G_{l,i}$ ;
12             $U, V \leftarrow$ Algorithm A.1 $(k, D_{train}, W_{l,i}, M)$ ;
13             $\hat{M} \leftarrow M$ with $W_{l,i}$ replaced by $U, V$. ;
14             Evaluate new loss $L_{new} = \hat{M}(D_{train})$ ;
15             if $L_{new}/L < 1 + R_{l,i}$ then
16                 $M \leftarrow \hat{M}$ ;
17                 break; ;
```

demonstrate two facts. First, it's indeed a trade-off between the efficiency and efficacy as the speedup ratio goes higher at the cost of lower accuracy. Second, we want to point out that this trade-off relationship is not linear, and different task might have different characteristics. Thus, in the real application, users need to decide what's the best cutoff to use. We also want to point out that this 3% accuracy drop comparison is fair to all baseline methods. We could have chose another cutoff like 1% accuracy with lower speedup ratio to report, but this won't help too much when comparing different baseline methods.

Figure A.1: Illustration of efficiency and efficacy trade-off. Each point in this graph represents a specific ratio of training loss increase after approximation.

A.3 Detailed results

A.3.1 LSTM result

A 2-layer LSTM model is composed of two large matrices layers and one large softmax layer. Additional processing time includes applying activation functions, softmax function and computing the updated hidden representation. The detailed inference time in each layer is summarized in Table A.1. We could observe that the overhead of the computation will be greatly incurred on GPU. Thus, despite the matrix is much smaller and well approximated by DRONE, the overall acceleration on GPU is less.

A.3.2 Transformer result

For BERT models, we use BERT-base models and it contains 12 layers of the same model structure without sharing parameters. Each layer contains an attention module with hidden size 768 and 12 channels, a small 768×768 Feed-forward (FF) layer followed by 2 larger FF layers (768×3072 and 3072×768). As shown in Figure 4.1, these four components consume the most computational time in the BERT-base models. The detailed average inference time of each module is summarized in Table A.2 for CPU and Table A.3 for GPU. There are two important points to note.

Firstly, we could see that attention module is not the bottleneck at all under the normal size of context (128). Therefore, many works on accelerating attention module alone would not improve the overall inference time of the module except a very long sequence appears. The necessity of the long sequence is out of the domain of this paper and what we want to show is that the proposed DRONE would work on both attention and feed-forward layer, which collectively could accelerate the real(overall) inference time.

Secondly, we could observe that the FF2 layer could be accelerated most. A plausible reason could be that the input dimension to the FF2 layer is in a larger dimension (3072) than all the other layers (64 or 768). When the input distribution actually lies in a lower-dimensional space, there is much more room for FF2 layer to be compressed and accelerated by the data-aware low-rank method.

Table A.1: The average inference time of each component in the model of 2-layer LSTM model. Both proposed methods and SVD use same ranks so the inference time is approximately the same. The unit is in millisecond and the number in parenthesis shows the ratio respective to the overall inference time.

Device	Models	LSTM-1	LSTM-2	Softmax	Others	Total Time	Perplexity
CPU	PTB-Large	1.27ms	1.30ms	1.09ms	0.13ms	3.79ms	78.32
	PTB-Large-SVD	-	-	-	-	-	81.09
	PTB-Large-SVD-Retrain	-	-	-	-	-	80.89
	PTB-Large-FINGER	-	-	-	-	-	80.87
	PTB-Large-FINGER-Retrain	0.24ms	0.34ms	0.42ms	0.11ms	1.11ms(3.4x)	79.01
GPU	PTB-Large	0.019ms	0.018ms	0.015ms	0.32ms	0.11ms	78.32
	PTB-Large-SVD	-	-	-	-	-	81.09
	PTB-Large-SVD-Retrain	-	-	-	-	-	80.89
	PTB-Large-FINGER	-	-	-	-	-	80.87
	PTB-Large-FINGER-Retrain	0.01ms	0.01ms	0.015ms	0.055ms	0.09ms(1.2x)	79.01

Table A.2: The detailed average inference time (in milliseconds) on CPU of each component in the model by retrained DRONE.

Tasks	Self-Attention	Feed-Forward 0	Feed-Forward 1	Feed-Forward 2	Others	Total Time
MNLI	122.7	19.5	78.5	46.1	4.2	271.0
QQP	131.5	29.9	99.2	66.5	5.8	333.0
SST-2	100.5	24.7	79.3	54.5	4.5	263.5
QNLI	128.3	28.4	111.0	79.0	5.9	352.6
MRPC	82.6	12.8	89.4	38.2	2.4	225.4
RTE	116.0	25.6	85.4	62.3	3.4	292.7
CoLA	108.2	22.7	93.1	70.8	3.4	298.2
STS-B	109.1	19.3	90.8	53.0	4.0	276.2

Table A.3: The detailed average inference time (in milliseconds) on GPU of each component in the model by retrained DRONE.

Tasks	Self-Attention	Feed-Forward 0	Feed-Forward 1	Feed-Forward 2	Others	Total Time
MNLI	0.94	0.26	0.76	0.60	0.003	2.56
QQP	0.92	0.24	0.64	0.52	0.001	2.32
SST-2	0.85	0.25	0.59	0.52	0.005	2.22
QNLI	0.91	0.25	0.72	0.60	0.001	2.48
MRPC	0.89	0.22	0.57	0.43	0.001	2.11
RTE	1.02	0.29	0.60	0.59	0.008	2.51
CoLA	0.93	0.25	0.68	0.61	0.002	2.47
STS-B	0.83	0.2	0.57	0.42	0.002	2.02

A.4 Combination with Quantization Methods

Distillation in practice achieves the STOA without extra hardware accelerator, so it serves as a good target to show how DRONE can be combined with other methods. The benefit of quantization/pruning can only be shown when a ASIC/FPGA accelerator is provided. Since we don't have one, we can't only use the software to simulate. We can apply any Quantization scheme and empirically show combined method can achieve a competitive accuracy with lower bit bandwidth. Algorithmically, we combine DRONE with vanilla quantization with fixed precision which gets 87.5 (vs 89.5 on MRPC) and 51.0 (vs 53.4 on CoLA) with 12 bits(vs 32 bits). Thus we can hypothesize that with the hardware accelerator, there could be at least further 3x speedup when DRONE is combined with Quantization methods.

A.5 An example of ranks used in SST-2

Below are the ranks obtained by performing DRONE in SST-2 dataset. The order is from the bottom layer to the top layer. Full rank of all the matrices are 768.

Attention layer: [192, 384, 192, 768, 768, 192, 768, 768, 192, 192, 768, 192],

Feed-Forward 0 : [288, 768, 96, 192, 288, 192, 768, 768, 96, 96, 288, 96],

Feed-Forward 1: [96, 96, 768, 768, 288, 288, 768, 768, 96, 96, 288, 96], Feed-Forward 2 : [192, 192, 768, 288, 768, 768, 768, 192, 96, 192, 96, 96].

APPENDIX B

Appendix in Fast Graph-base Approximate Nearest Neighbor Search by Local Data Information

B.1 Formulation of Inner-product

In the main text, we presented derivation of $L2$ distance, and in this section we will derive the approximation for inner-product distance measure. Notice that angle measure can be obtained by firstly normalizing data vectors and then apply inner-product distance and thus the derivation is the same. For a query q and data point d, inner-product distance measure is $Dist = q^T d$. Similar to $L2$ distance, we can apply the same decomposition to write $q = q_{proj} + q_{res}$ and $d = d_{proj} + d_{res}$. substituting the decomposition into distance definition, we have

$$Dist = q_{proj}^T d_{proj} + q_{res}^T d_{res}.$$

As in $L2$ case q_{proj} and d_{proj} can be obtained by simple operations and the remaining uncertainy term is again $q_{res}^T d_{res}$. Therefore, in inner-product case, angle between neighboring residual vectors is still the target to approximate.

B.2 Approximate Greedy Search Algorithm

Algorithm B.1: Approximate Greedy Graph Search

```
Input: graph $G$, query $q$, starting point $p$, distance function dist(), appxoaimate
       distance function appx(), number of nearest points to return $efs$
Output: top candidate set $T$
candidate set $C = \{p\}$
dynamic list of currently best candidates $T = \{p\}$
visited $V = \{p\}$
while $C$ is not empty do
    cur ← nearest element from $C$ to
    ub ← distance of the furthest element from $T$ to $q$ (i.e., upper bound of the
      candidate search)
    if dist(cur, q) $>$ ub then
        return T
    for point $n \in$ neighbour of cur in $G$ do
        if $n \in V$ then
            continue
        V.add($n$)
        if #updates of cur ¿ 5 times then
            e = appx(n, q)
        else
            e = dist(n, q)
        if $e \leq ub$ or $|T| \leq efs$ then
            update distance to be dist(n,q)
            C.add(n)
            T.add(n)
            if $|T| > efs$ then
                remove furthest point to $q$ from T
            ub ← distance of the furthest element from $T$ to $q$ (i.e., update ub)
return T
```

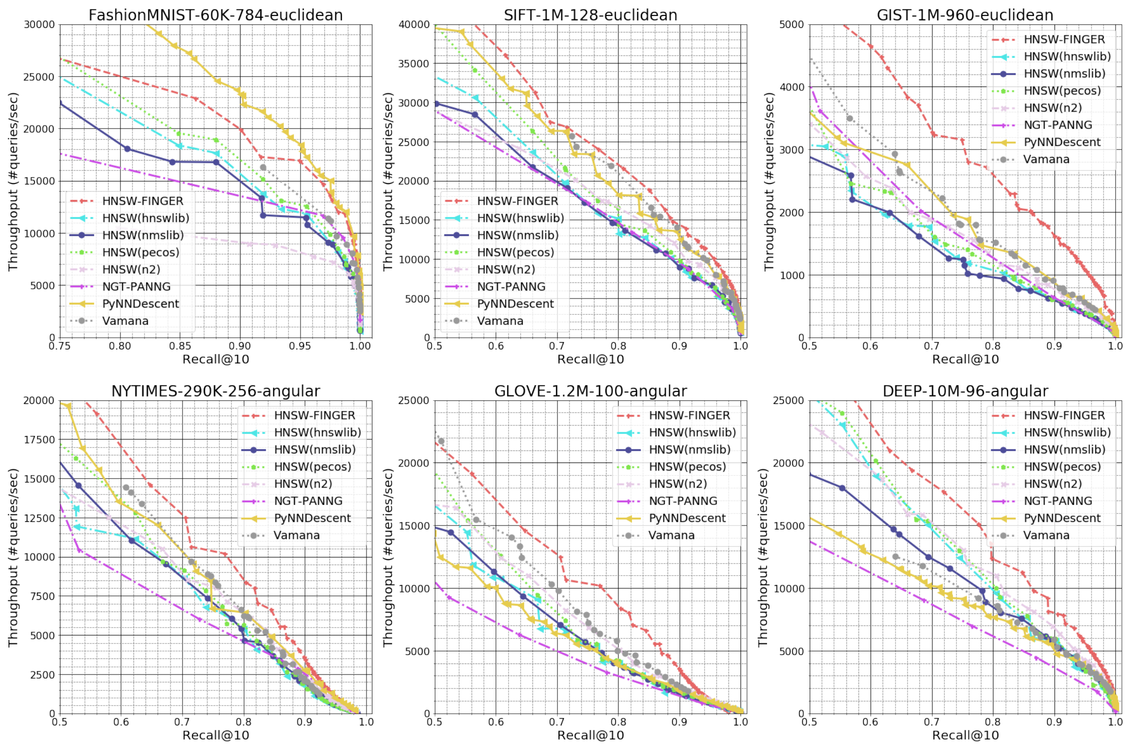


Figure B.1: Experimental results of graph-based methods. Throughput versus Recall@10 chart is plotted for all datasets. Top row presents datasets with $L2$ distance measure and bottom row presents datasets with angular distance measure. We can observe a significant performance gain of HNSW-FINGER over existing graph-based methods.

B.3 Complete Comparison of Graph-based Methods

Complete results of all graph-based methods are shown in Figure B.1. HNSW-FINGER basically outperforms all existing graph-based methods except on FashionMNIST-60K-784 where PyNNDescent performs extremely well. In principle, FINGER could also be applied on PyNNDescent to further improve the result. Results show that currently no graph-based methods completely exploits the training data distribution. This reflects the importance of the inference acceleration methods as FINGER that can create consistently faster inference on all underlying search graph. Making a search graph maximally suitable for applying FINGER is also an interesting future direction.

B.4 Selection of Rank Parameter r

Under ANN-benchmark protocol, we could have made the selection of rank r in FINGER as a hyper-parameter to search in order to achieve best performance. But this might be time-consuming for real applications. Instead, here we provide a practical rule of thumb for choosing r by calculating the correlation coefficient of X, Y in Algorithm 7.2. X stores true angles between neighboring pairs and Y stores approximated angles. We start r to be 8 in order to maximally leverage SIMD. Specifically, AVX2 SIMD allows a single instruction with 8 parallel floating point computation. Increase the rank in a multiple of 8 will maximally leverage the capability of SIMD instructions. Now, if the correlation is smaller than 0.7, we enlarge r by 8 and redo Algorithm 7.2 again with increased r until correlation between X and y is larger than 0.7. In this work, to show the effectiveness of applying FINGER in read world applications, we use this search scheme and ranks learned in FashionMNIST-60K-784: 16, SIFT-1M-128: 16, GIST-1M-960: 16, NYTIMES-290K-256: 48, GLOVE-1.2M-100: 32 and DEEP-10M-96: 24.

Table B.1: Construction statistics of HNSW-FINGER and HNSW. Pre-processing time in second is shown in the table. Numbers in parentheses represent the memory footprint in GB.

Dataset	M	HNSW-FINGER	HNSW(PECOS)
SIFT-1M-128	12	291.5s (2.8G)	215.4s (1G)
	48	521.4s (9.2G)	433.9s (2.4G)
GLOVE-1.2M-100	12	386.2s(4.8G)	300.1s (1.1G)
	48	1409.8s (18G)	1317.3s (2.7G)

B.5 Pre-processing Time and Memory footprint of HNSW-FINGER and HNSW

Examples of pre-processing time and memory footprint of HNSW-FINGER and HNSW is shown in Table B.1. FINGER requires additional linear scan of training data, so it will add some additional processing time to the base method. The difference is around 90 seconds which is not significant compared to the pre-processing time of base HNSW method. Memory usage of HNSW is approximately memory of data plus number of edges $|E| \times$ sizeof(int). For a selected low-rank dimension r, FINGER requires additional $(r + 2) \times$ —E— $\times$ sizeof(float) to store the pre-computed values.

BIBLIOGRAPHY

[1] "The president's agenda to build back better will reduce emissions and keep energy costs low," *https://www.whitehouse.gov/cea/written-materials/2021/09/16/the-presidents-agenda-to-build-back-better-will-reduce-emissions-and-keep-energy-costs-low/*, Online 2021. 3

[2] O. Y. Al-Jarrah, P. D. Yoo, S. Muhaidat, G. K. Karagiannidis, and K. Taha, "Efficient machine learning for big data: A review," *Big Data Research*, vol. 2, no. 3, pp. 87–93, 2015. 3

[3] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya, "Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces," *arXiv preprint arXiv:1804.06829*, 2018. 1

[4] S. Arya and D. M. Mount, "Approximate nearest neighbor queries in fixed dimensions." in *SODA*, vol. 93. Citeseer, 1993, pp. 271–280. 103

[5] M. Aumüller, E. Bernhardsson, and A. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," *Information Systems*, vol. 87, p. 101374, 2020. 100, 101, 103

[6] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991. 103

[7] Y. Bachrach, Y. Finkelstein, R. Gilad-Bachrach, L. Katzir, N. Koenigstein, N. Nice, and U. Paquet, "Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces," in *Proceedings of the 8th ACM Conference on Recommender systems*. ACM, 2014, pp. 257–264. 40, 42, 50, 74

[8] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *Proceedings of the fourth ACM international conference on Web search and data mining*, 2011, pp. 635–644. 1, 58

[9] G. Ballard, T. G. Kolda, A. Pinar, and C. Seshadhri, "Diamond sampling for approximate maximum all-pairs dot-product (mad) search," in *2015 IEEE International Conference on Data Mining.* IEEE, 2015, pp. 11–20. 74

[10] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," in *SIGMOD*, 1990, pp. 322–331. 103

[11] C. M. Bishop, "Pattern recognition," *Machine learning*, vol. 128, no. 9, 2006. 99

[12] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry *et al.*, "An updated set of basic linear algebra subprograms (blas)," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, 2002. 75

[13] P. Bork, L. J. Jensen, C. Von Mering, A. K. Ramani, I. Lee, and E. M. Marcotte, "Protein interaction networks from yeast to human," *Current opinion in structural biology*, vol. 14, no. 3, pp. 292–299, 2004. 70

[14] L. Boytsov and B. Naidan, "Engineering efficient and effective non-metric space library," in *Similarity Search and Applications - 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings*, 2013, pp. 280–293. 50

[15] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020. 1

[16] D. Cai, "A revisit of hashing algorithms for approximate nearest neighbor search," *IEEE Transactions on Knowledge and Data Engineering*, 2019. 100, 103, 108

[17] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision.* Springer, 2020, pp. 213–229. 1

[18] O. Celma, *Music Recommendation and Discovery in the Long Tail.* Springer, 2010. 73

[19] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, and M. Federico, "Report on the 11th iwslt evaluation campaign, iwslt 2014," in *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, 2014. 20, 32, 48

[20] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC*, 2002, pp. 380–388. 102, 103, 108

[21] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," *arXiv preprint arXiv:1312.3005*, 2013. 20

[22] D. Chen, Y. Li, M. Qiu, Z. Wang, B. Li, B. Ding, H. Deng, J. Huang, W. Lin, and J. Zhou, "AdaBERT: Task-adaptive BERT compression with differentiable neural architecture search," *arXiv preprint arXiv:2001.04246*, 2020. 79, 82

[23] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 491–502. 1

[24] P. Chen, S. Si, Y. Li, C. Chelba, and C.-J. Hsieh, "Groupreduce: Block-wise low-rank approximation for neural language model shrinking," *Advances in Neural Information Processing Systems*, vol. 31, 2018. 1

[25] ——, "Groupreduce: Block-wise low-rank approximation for neural language model shrinking," in *Advances in Neural Information Processing Systems 31.* Curran Associates, Inc., 2018, pp. 10 988–10 998. 32, 33

[26] Q. Chen, B. Zhao, H. Wang, M. Li, C. Liu, Z. Zheng, M. Yang, and J. Wang, "Spann: Highly-efficient billion-scale approximate nearest neighborhood search," *NeurIPS*, vol. 34, 2021. 104

[27] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, and M. Carbin, "The Lottery Ticket Hypothesis for Pre-trained BERT Networks," *arXiv preprint arXiv:2007.12223*, 2020. 82

[28] T. Chen, M. R. Min, and Y. Sun, "Learning k-way d-dimensional discrete codes for compact embedding representations," in *International Conference on Machine Learning*, 2018, pp. 853–862. 25, 26, 33, 59, 60, 74, 80, 92, 99

[29] Y. Chen, L. Mou, Y. Xu, G. Li, and Z. Jin, "Compressing neural language models by sparse word representations," *arXiv preprint arXiv:1610.03950*, 2016. 23, 26

[30] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019. 81, 93

[31] W.-S. Chin, B.-W. Yuan, M.-Y. Yang, Y. Zhuang, Y.-C. Juan, and C.-J. Lin, "Libmf: a library for parallel matrix factorization in shared-memory systems," *JMLR*, vol. 17, no. 1, pp. 2971–2975, 2016. 75

[32] Y. Choi, M. El-Khamy, and J. Lee, "Universal deep neural network compression," *CoRR*, vol. abs/1802.02271, 2018. [Online]. Available: http://arxiv.org/abs/1802.02271 11

[33] D. Dearholt, N. Gonzales, and G. Kurup, "Monotonic search networks for computer vision databases," in *Twenty-Second Asilomar Conference on Signals, Systems and Computers*, vol. 2. IEEE, 1988, pp. 548–553. 103

[34] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, 2014, pp. 1269–1277. 10, 11

[35] M.-B. D. Design, "Vivado design suite user guide," 2012. 6

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. 1, 81

[37] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *WWW*, 2011, pp. 577–586. 101, 115

[38] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 293–302. 82

[39] M. Douze, A. Sablayrolles, and H. Jégou, "Link and code: Fast indexing with graphs and compact regression codes," in *CVPR*, 2018, pp. 3646–3654. 102, 104

[40] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer, "The yahoo! music dataset and kdd-cup'11," in *Proceedings of the 2011 International Conference on KDD Cup 2011-Volume 18*. JMLR. org, 2011, pp. 3–18. 58

[41] L. Duan, C. Aggarwal, S. Ma, R. Hu, and J. Huai, "Scaling up link prediction with ensembles," in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 2016, pp. 367–376. 59, 60, 73, 74

[42] P. W. Foltz, W. Kintsch, and T. K. Landauer, "The measurement of textual coherence with latent semantic analysis," *Discourse processes*, vol. 25, no. 2-3, pp. 285–307, 1998. 26

[43] C. B. Freksen, "An introduction to johnson-lindenstrauss transforms," *arXiv preprint arXiv:2103.00564*, 2021. 110

[44] C. Fu and D. Cai, "Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph," *arXiv preprint arXiv:1609.07228*, 2016. 103

[45] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," Jul. 2017. 103, 104

[46] ——, “Fast approximate nearest neighbor search with the navigating spreading-out graph,” *arXiv preprint arXiv:1707.00143*, 2017. 2

[47] C. Fu, C. Wang, and D. Cai, “High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PP, Mar. 2021. 100, 101, 103

[48] T. Ge, K. He, Q. Ke, and J. Sun, “Optimized product quantization,” *IEEE TPAMI*, vol. 36, no. 4, pp. 744–755, 2013. 102

[49] C. Gentile, S. Li, and G. Zappella, “Online clustering of bandits,” in *International Conference on Machine Learning*, 2014, pp. 757–765. 75

[50] A. Gionis, P. Indyk, R. Motwani *et al.*, “Similarity search in high dimensions via hashing,” in *VLDB*, vol. 99, no. 6, 1999, pp. 518–529. 103

[51] A. Gittens, D. Achlioptas, and M. W. Mahoney, “Skip-gram- zipf+ uniform= vector additivity,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 69–76. 26

[52] M. A. Gordon, K. Duh, and N. Andrews, “Compressing BERT: Studying the effects of weight pruning on transfer learning,” *arXiv preprint arXiv:2002.08307*, 2020. 82

[53] S. Goyal, A. R. Choudhury, S. M. Raje, V. T. Chakaravarthy, Y. Sabharwal, and A. Verma, “PoWER-BERT: Accelerating BERT Inference via Progressive Word-vector Elimination.” 81

[54] E. Grave, A. Joulin, M. Cissé, D. Grangier, and H. Jégou, “Efficient softmax approximation for gpus,” in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 1302–1310. 42, 44, 50, 64

[55] R. Guo, S. Kumar, K. Choromanski, and D. Simcha, “Quantization based fast inner product search,” in *Artificial Intelligence and Statistics*, 2016, pp. 482–490. 40, 43

[56] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar, "Accelerating large-scale inference with anisotropic vector quantization," in *ICML*. PMLR, 2020, pp. 3887–3896. 102, 103, 115

[57] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011. 103

[58] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149 10, 11, 82

[59] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015. [Online]. Available: http://arxiv.org/abs/1506.02626 4, 10, 11, 82

[60] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, p. 19, 2016. 73

[61] B. Harwood and T. Drummond, "Fanng: Fast approximate nearest neighbour graphs," in *CVPR*, 2016, pp. 5713–5722. 103

[62] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *CVPR*, 2013. 103

[63] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017. 1, 11

[64] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *arXiv preprint arXiv:1609.07061*, 2016. 10, 11, 12, 23, 82

[65] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing.* ACM, 1998, pp. 604–613. 42, 43, 59, 100

[66] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014. 11

[67] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016. 26, 29

[68] ——, "Categorical reparametrization with gumble-softmax," in *International Conference on Learning Representations 2017.* OpenReviews. net, 2017. 41, 45, 46

[69] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnawamy, and R. Kadekodi, "Diskann: Fast accurate billion-point nearest neighbor search on a single node," *NeurIPS*, vol. 32, 2019. 100, 101, 103, 104, 115

[70] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," *arXiv preprint arXiv:1412.2007*, 2014. 42

[71] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE TPAMI*, vol. 33, no. 1, pp. 117–128, 2010. 102

[72] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "TinyBERT: Distilling BERT for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019. 79, 82, 92

[73] Z. Jin, D. Zhang, Y. Hu, S. Lin, D. Cai, and X. He, "Fast and accurate hashing via iterative nearest neighbors expansion," *IEEE transactions on cybernetics*, vol. 44, no. 11, pp. 2167–2177, 2014. 103

[74] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019. 103, 115

[75] W. B. Johnson and J. Lindenstrauss, “Extensions of lipschitz mappings into a hilbert space 26,” *Contemporary mathematics*, vol. 26, 1984. 108

[76] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, “Exploring the limits of language modeling,” *arXiv preprint arXiv:1602.02410*, 2016. 20, 32

[77] M. Khodak, N. Tenenholtz, L. Mackey, and N. Fusi, “Initialization and regularization of factorized neural layers,” *arXiv preprint arXiv:2105.01029*, 2021. 83

[78] V. Khrulkov, O. Hrinchuk, L. Mirvakhabova, and I. Oseledets, “Tensorized embedding layers for efficient model compression,” *arXiv preprint arXiv:1901.10787*, 2019. 33

[79] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, “Compression of deep convolutional neural networks for fast and low power mobile applications,” in *ICLR*, 2016. 11

[80] N. Kitaev, Ł. Kaiser, and A. Levskaya, “Reformer: The Efficient Transformer,” *arXiv preprint arXiv:2001.04451*, 2020. 79, 81, 93

[81] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. Rush, “OpenNMT: Open-source toolkit for neural machine translation,” in *Proceedings of ACL 2017, System Demonstrations*. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 67–72. 32

[82] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, “Opennmt: Open-source toolkit for neural machine translation,” *arXiv preprint arXiv:1701.02810*, 2017. 20, 48

[83] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, no. 8, pp. 30–37, 2009. 2, 58

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012. 1

[85] J. Kunegis, "Konect: the koblenz network collection," in *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 2013, pp. 1343–1350. 73

[86] M. Lam, "Word2bits - quantized word vectors," *arXiv preprint arXiv:1803.05651*, 2018. 12

[87] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "AlBERT: A lite BERT for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019. 79, 82

[88] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605. 11

[89] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980. 103

[90] C. Li, M. Zhang, D. G. Andersen, and Y. He, "Improving approximate nearest neighbor search through learned adaptive early termination," in *SIGMOD*, 2020, pp. 2539–2554. 104

[91] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky, "Deep reinforcement learning for dialogue generation," *arXiv preprint arXiv:1606.01541*, 2016. 39

[92] S. Li, A. Karatzoglou, and C. Gentile, "Collaborative filtering bandits," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2016, pp. 539–548. 75

[93] S. Li, W. Chen, S. Li, and K.-S. Leung, "Improved algorithm on online clustering of bandits," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2019, pp. 2923–2929. 75

[94] X. Li and P. Li, "Random projections with asymmetric quantization," *NeurIPS*, vol. 32, 2019. 102

[95] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858. 11, 82

[96] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017. 31

[97] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *IEEE Internet computing*, no. 1, pp. 76–80, 2003. 58

[98] W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju, "FastBERT: a Self-distilling BERT with Adaptive Inference Time," *arXiv preprint arXiv:2004.02178*, 2020. 79, 82

[99] E. Lobacheva, N. Chirkova, and D. Vetrov, "Bayesian sparsification of recurrent neural networks," *arXiv preprint arXiv:1708.00077*, 2017. 12, 21

[100] Z. Lu, V. Sindhwani, and T. N. Sainath, "Learning compact recurrent neural networks," *CoRR*, vol. abs/1604.02594, 2016. [Online]. Available: http://arxiv.org/abs/1604.02594 21

[101] M.-T. Luong and C. D. Manning, "Stanford neural machine translation systems for spoken language domain," in *International Workshop on Spoken Language Translation*, Da Nang, Vietnam, 2015. 48

[102] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *arXiv preprint arXiv:1603.09320*, 2016. 42

[103] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014. 42, 71

[104] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *CoRR*, vol. abs/1603.09320, 2016. 50

[105] ——, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, 2018. 60, 70, 71, 101, 103, 114

[106] Y. Mao, Y. Wang, C. Wu, C. Zhang, Y. Wang, Y. Yang, Q. Zhang, Y. Tong, and J. Bai, "LadaBERT: Lightweight Adaptation of BERT through Hybrid Model Compression," *arXiv preprint arXiv:2004.04124*, 2020. 82, 92

[107] E. Marcheret, V. Goel, and P. A. Olsen, "Optimal quantization and bit allocation for compressing large discriminative feature space transforms," in *2009 IEEE Workshop on ASRU*. IEEE, 2009, pp. 64–69. 102

[108] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: the penn treebank," *Comput. Linguist.*, vol. 19, no. 2, pp. 313–330, 1993. 48

[109] J. Martinez, S. Zakhmi, H. H. Hoos, and J. J. Little, "Lsq++: Lower running time and higher recall in multi-codebook quantization," in *ECCV*, 2018. 102

[110] Y. Matsui, Y. Uchida, H. Jégou, and S. Satoh, "A survey of product quantization," *ITE Transactions on Media Technology and Applications*, vol. 6, no. 1, pp. 2–10, 2018. 99, 100

[111] R. Memisevic, "Learning to relate images: Mapping units, complex cells and simultaneous eigenspaces," *arXiv preprint arXiv:1110.0107*, 2011. 25

[112] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119. 26

[113] J. Mitchell and M. Lapata, "Vector-based models of semantic composition," *proceedings of ACL-08: HLT*, pp. 236–244, 2008. 27

[114] ——, "Language models based on semantic composition," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*. Association for Computational Linguistics, 2009, pp. 430–439. 27, 80

[115] A. Mnih and Y. W. Teh, "A fast and simple algorithm for training neural probabilistic language models," in *ICML*, 2012. 42

[116] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model." in *AISTATS*, vol. 5, 2005, pp. 246–252. 42

[117] S. Morozov and A. Babenko, "Unsupervised neural quantization for compressed-domain similarity search," in *ICCV*, 2019, pp. 3036–3045. 102

[118] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," in *ICLR*. 12, 21

[119] B. Neyshabur and N. Srebro, "On symmetric and asymmetric lshs for inner product search," in *ICML*, 2015. 40, 42, 50, 59, 74

[120] E. Ning, "Microsoft open sources breakthrough optimizations for transformer inference on GPU and CPU ," tinyurl.com/y26jdsn9, 2020, accessed: 2010-09-30. 81

[121] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in neural information processing systems*, 2015, pp. 442–450. 25

[122] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4055–4064. 1

[123] T. Plötz and S. Roth, "Neural nearest neighbors networks," *arXiv preprint arXiv:1810.12575*, 2018. 1, 99

[124] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 379–389. 39

[125] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on.* IEEE, 2013, pp. 6655–6659. 5, 10, 11, 84

[126] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520. 1

[127] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019. 79, 82

[128] V. Sanh, T. Wolf, and A. M. Rush, "Movement Pruning: Adaptive Sparsity by Fine-Tuning," *arXiv preprint arXiv:2005.07683*, 2020. 82

[129] A. Shah and A. Majumdar, "Accelerating low-rank matrix completion on gpus," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI).* IEEE, 2014, pp. 182–187. 94

[130] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2018. 81

[131] K. Shim, M. Lee, I. Choi, Y. Boo, and W. Sung, "Svd-softmax: Fast softmax approximation on large vocabulary neural networks," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 5463–5473. 43, 44, 47, 50, 55, 64, 74

[132] ——, “SVD-softmax: Fast softmax approximation on large vocabulary neural networks,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5463–5473. 79, 80, 84, 92

[133] A. Shrivastava and P. Li, “Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips),” in *Advances in Neural Information Processing Systems*, 2014, pp. 2321–2329. 40, 42, 59, 74

[134] R. Shu and H. Nakayama, “Compressing word embeddings via deep compositional code learning,” in *ICLR*, 2018. 12, 22, 23, 25, 26, 33, 34, 35

[135] S. Si, C.-J. Hsieh, and I. S. Dhillon, “Memory efficient kernel approximation,” *J. Mach. Learn. Res*, vol. 32, pp. 701–9. 16

[136] C. Silpa-Anan and R. Hartley, “Optimised kd-trees for fast image descriptor matching,” in *CVPR*. IEEE, 2008, pp. 1–8. 102, 103

[137] V. Sindhwani, T. Sainath, and S. Kumar, “Structured transforms for small-footprint deep learning,” in *Advances in Neural Information Processing Systems*, 2015, pp. 3088–3096. 79

[138] R. F. Sproull, “Refinements to nearest-neighbor searching ink-dimensional trees,” *Algorithmica*, vol. 6, no. 1-6, pp. 579–589, 1991. 42, 43, 59

[139] N. Srebro and T. Jaakkola, “Weighted low-rank approximations,” in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 720–727. 16

[140] K. Sugawara, H. Kobayashi, and M. Iwasaki, “On approximately searching for similar word embeddings,” in *ACL*, 2016. 101, 115

[141] S. Sun, Y. Cheng, Z. Gan, and J. Liu, “Patient knowledge distillation for BERT model compression,” *arXiv preprint arXiv:1908.09355*, 2019. 79, 82

[142] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, “MobileBERT: a compact task-agnostic BERT for resource-limited devices,” *arXiv preprint arXiv:2004.02984*, 2020. 79, 82

[143] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, 2014, pp. 3104–3112. 39

[144] J. Tang, S. Chang, C. Aggarwal, and H. Liu, “Negative link prediction in social media,” in *Proceedings of the eighth ACM international conference on web search and data mining*. ACM, 2015, pp. 87–96. 58

[145] A. Tjandra, S. Sakti, and S. Nakamura, “Compressing recurrent neural network with tensor train,” in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 4451–4458. 12

[146] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008. 79, 81

[147] R. Waleffe and T. Rekatsinas, “Principal component networks: Parameter reduction early in training,” *arXiv preprint arXiv:2006.13347*, 2020. 83

[148] H. Wang, Z. Wang, W. Wang, Y. Xiao, Z. Zhao, and K. Yang, “A note on graph-based nearest neighbor search,” *arXiv preprint arXiv:2012.11083*, 2020. 100

[149] J. Wang, S. Kumar, and S.-F. Chang, “Sequential projection learning for hashing with compact codes,” 2010. 103

[150] S. Wang, B. Li, M. Khabsa, H. Fang, and H. Ma, “Linformer: Self-Attention with Linear Complexity,” *arXiv preprint arXiv:2006.04768*, 2020. 82, 93

[151] Z. Wang, T. Luo, R. S. M. Goh, and J. T. Zhou, “Edcompress: Energy-aware model

compression for dataflows," *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 6

[152] Z. Wang, J. Wohlwend, and T. Lei, "Structured pruning of large language models," *arXiv preprint arXiv:1910.04732*, 2019. 83, 96

[153] W. Wen, Y. He, S. Rajbhandari, W. Wang, F. Liu, B. Hu, Y. Chen, and H. Li, "Learning intrinsic sparse structures within long short-term memory," *CoRR*, vol. abs/1709.05027, 2017. 33

[154] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv, abs/1910.03771*, 2019. 91, 97

[155] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," *CoRR*, vol. abs/1512.06473, 2015. [Online]. Available: http://arxiv.org/abs/1512.06473 10

[156] X. Wu, R. Guo, A. T. Suresh, S. Kumar, D. N. Holtmann-Rice, D. Simcha, and F. Yu, "Multiscale quantization for fast similarity search," in *NIPS*, 2017, pp. 5745–5755. 43

[157] ——, "Multiscale quantization for fast similarity search," *NeurIPS*, vol. 30, pp. 5745–5755, 2017. 102

[158] C. Xu, W. Zhou, T. Ge, F. Wei, and M. Zhou, "BERT-of-theseus: Compressing BERT by progressive module replacing," *arXiv preprint arXiv:2002.02925*, 2020. 79, 82

[159] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong, "Deep neural network compression with single and multiple level quantization," *CoRR*, vol. abs/1803.03289, 2018. [Online]. Available: http://arxiv.org/abs/1803.03289 11

[160] X. Yang, M. Gao, J. Pu, A. Nayak, Q. Liu, S. E. Bell, J. O. Setter, K. Cao, H. Ha, C. Kozyrakis *et al.*, "Dnn dataflow choice is overrated," *arXiv preprint arXiv:1809.04070*, vol. 6, 2018. 5

[161] H.-F. Yu, C.-J. Hsieh, Q. Lei, and I. Dhillon, "A greedy approach for budgeted maximum inner product search," in *NIPS*, 2017. 40, 43, 50

[162] H.-F. Yu, C.-J. Hsieh, Q. Lei, and I. S. Dhillon, "A greedy approach for budgeted maximum inner product search," in *Advances in Neural Information Processing Systems*, 2017, pp. 5453–5462. 59, 60, 74

[163] H.-F. Yu, K. Zhong, and I. S. Dhillon, "Pecos: Prediction for enormous and correlated output spaces," *arXiv preprint arXiv:2010.05878*, 2020. 114

[164] H.-F. Yu, K. Zhong, J. Zhang, W.-C. Chang, and I. S. Dhillon, "Pecos: Prediction for enormous and correlated output spaces," *Journal of Machine Learning Research*, vol. 23, no. 98, pp. 1–32, 2022. 3

[165] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 67–76, 2017. 10, 11, 79

[166] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8BERT: Quantized 8bit BERT," *arXiv preprint arXiv:1910.06188*, 2019. 82

[167] B. Zhang, D. Xiong, and J. Su, "Accelerating neural transformer via an average attention network," *arXiv preprint arXiv:1805.00631*, 2018. 79, 81

[168] M. Zhang, X. Liu, W. Wang, J. Gao, and Y. He, "Navigating with graph representations for fast and scalable decoding of neural language models," in *NIPS*, 2018. 43, 44, 47, 49, 60, 64, 70, 71, 74

[169] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019. 99

[170] S. Zhao, R. Gupta, Y. Song, and D. Zhou, "Extreme language model compression with

optimal subwords and shared projections," *arXiv preprint arXiv:1909.11687*, 2019. 79, 82