THE UNIVERSITY OF CHICAGO

CHALLENGES IN MODERN MACHINE LEARNING: MULTIRESOLUTION STRUCTURE, MODEL UNDERSTANDING AND TRANSFER LEARNING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
PRAMOD KAUSHIK MUDRAKARTA

CHICAGO, ILLINOIS
DECEMBER 2019

Dedicated to my family – Rama Muktapuram (mother), Ramchander Mudrakarta (father), and Pratyusha Mudrakarta (younger sister).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This thesis means a lot to me. In 2008, while I was pursuing my undergraduate studies at IIT Bombay, I discovered the field of Machine Learning, started to understand what research meant, and soon realized that pursuing a doctoral program in Machine Learning is what I wanted to do. It took me nearly 11 years to achieve my goal. The journey has been long, embellished with exhilarating successes and excruciating failures. Today, as I am submitting this dissertation, I feel tremendous joy that cannot be described in words. I feel content, not just for having achieved my goal, but also for having discovered so much more along the way.

I have been inspired, supported and guided by many amazing people over the years, without whom I would not be where I am today. Strengthened by the support and wisdom that I have received, I have fought and learned to overcome various kinds of hurdles that people, places, social structures, circumstances, and internal systems of my own self placed before me. I am truly and immensely grateful to many many people who have gifted me their time, energy, resources, wisdom, knowledge and love. In this short section, I focus on mentioning only those whom I know personally, and who have had a direct impact on the successful completion of this dissertation.

First and foremost, I thank my advisor, Risi Kondor, for giving me an opportunity to pursue a doctoral degree with his guidance. Risi encouraged me to become an independent researcher, gave me plenty of space to pursue my (rather varied) research interests, and supported me in all the practical matters such as funding, attending conferences, applying for fellowships, etc. His deep understanding of mathematical concepts and creativity in problem solving continue to inspire me. Discussions with him almost always helped me clear my mind and work my way through difficult problems.

Thanks to Ankur Taly and Mukund Sundararajan for the nearly 2-year long collaboration, and for always making time to discuss each and every question or idea I had. Over countless

emails, phone calls and video-conferencing sessions, I learned several important aspects of doing research from them. I also thank Kedar Dhamdhere and Kevin McCurley for being involved in the collaboration and the copious amounts of valuable advice that they have given me. I also thank Mark Sandler with whose guidance I was able to do an extremely productive internship. All these people made my time at Google Research thoroughly enjoyable.

I thank Vijai Mohan for being an amazing mentor who boosted my confidence when I was doing an internship at Amazon. It was my first time working in an industry environment, working on Deep Learning, and it was also a time when I was strongly questioning my choice of pursuing a PhD.

Thanks to Kevin Gimpel for having provided feedback on my work at various points during my PhD, and for involving me in Midwest Speech and Language Days. His feedback meant a lot to me, as I had otherwise no experience in natural langauge processing. I also thank Shubhendu Trivedi for his feedback on my thesis.

My dissertation committee (Risi Kondor, Rebecca Willett, Kevin Gimpel and Ankur Taly) provided prompt and useful feedback as I was writing my thesis, and I thank them for that.

An important part of being a PhD student is attending conferences; an activity which has had significant impact in my research life. Many thanks to the Department of Computer Science at the University of Chicago and Google for funding my travel. Thanks to Brandon Anderson, Yi Ding, Horace Pan, Hy Truong Son and Shubhendu Trivedi for the very useful weekly reading group at the University of Chicago. Thanks also to the staff in the Department of Computer Science, especially Bob Bartlett, Margaret Jaffey and Nita Yack, whose commitment to helping students made my life smooth in all the day-to-day and bureaucratic affairs.

I take this opportunity to acknowledge the impact of the support I received from people even before I began my PhD. My advisor while I was working at the Max Planck Institute,

Gunnar Rätsch, has been a great positive influence in my life. The support he provided me during a rather troubled phase in my life kept up my motivation to do research and continue to pursue my goal. I thank him for believing in me.

Thanks to Sushant Sachdeva and Bharath K. Sriperumbudur for their invaluable advice and encouragement over the years.

Life is no fun without friends. I thank those whose presence made Chicago feel warm: Shan Abraham, Aswathy Ajith, Yadu Nand Babuji, Steven Basart, Sasha Belinkov, Marc Gössling, Aniket Joglekar, Krithika Mohan, Sangeetha Mugunthan, Savita Ramaprasad, Veeranjaneyulu Sadhanala, Sajid Shariff, Shalini Sivarajah, Tyler Skluzacek, Vishwas Srivastava, Shubham Toshniwal, Shubhendu Trivedi, Hing Yin (Joseph) Tsang, Srikant Veeraraghavan, Supraja Vella. I especially thank Yadu Nand Babuji and Aswathy Ajith for being very dependable friends, whom I am going to sorely miss when I leave Chicago. I thank Amod Jog for regularly keeping in touch ever since undergraduate days, and for all his support and encouragement throughout the pursuit of my PhD.

Finally, I thank the most important people in my life: Rama Muktapuram (my mother), Ramchander Mudrakarta (my father), and Pratyusha Mudrakarta (my younger sister). They are my role models. They inspire and instill in me an attitude of never giving up, of having a positive outlook and an affinity towards learning and growing, both professionally and as a human being. They have worked hard and made several sacrifices over the years to help me achieve my goals as well as to reach beyond myself and grow as a person. Words cannot express how much I owe them. This dissertation is dedicated to them.

ABSTRACT

Recent advances in Artificial Intelligence (AI) are characterized by ever-increasing sizes of datasets and reemergence of neural-network methods. The modern AI pipeline begins with building datasets, followed by designing and training machine-learning models, and concludes with deployment of trained models in the real world. We tackle three important challenges relevant to this era; one from each part of the pipeline: 1) efficiently manipulating large matrices arising in real-world datasets (e.g., graph Laplacians from social network datasets), 2) interpreting deep-neural-network models, and 3) efficiently deploying hundreds of deep-neural-network models on embedded devices.

Matrices arising in large, real-world datasets are oftentimes found to have high rank, rendering common matrix-manipulation approaches that are based on the low-rank assumption (e.g. SVD) ineffective. In the first part of this thesis, we build upon Multiresolution Matrix Factorization (MMF), a method originally proposed to perform multiresolution analysis on discrete spaces, and can consequently model hierarchical structure in symmetric matrices as a matrix factorization. We describe a parallel algorithm for computing the factorization that can scale up to matrices with a million rows and columns. We then showcase an application of MMF, wherein we demonstrate a preconditioner that accelerates iterative algorithms solving systems of linear equations. Among wavelet-based preconditioners, the MMF-preconditioner consistently results in faster convergence and is highly scalable. Finally, we propose approaches to extend MMF to asymmetric matrices and evaluate them in the context of matrix compression.

In the second part of the thesis, we address the black-box nature of deep-neural-network models. The goodness of a deep-neural-network model is typically measured by its test accuracy. We argue that it is an incomplete measure, and show that state-of-the-art question-answering models often ignore important question terms. We perform a case study of a question-answering model and expose various ways in which the network gets the right answer

for the wrong reasons. We propose a human-in-the-loop workflow based on the notion of attribution (word-importance) to understand the input-output behavior of neural network models, extract rules, identify weaknesses and construct adversarial attacks by leveraging the weaknesses. Our strongest attacks drop the accuracy of a visual question answering model from 61.1% to 19%, and that of a tabular question answering model from 33.5% to 3.3%. We propose a measure for overstability - the tendency of a model to rely on trigger logic and ignore semantics. We use a path-sensitive attribution method to extract contextual synonyms (rules) learned by a model. We discuss how attributions can augment standard measures of accuracy and empower investigation of model performance. We finish by identifying opportunities for research: abstraction tools that aid the debugging process, concepts and semantics of path-sensitive dataflow analysis, and formalizing the process of verifying natural-language-based specifications.

The third challenge pertains to real-world deployment of deep-neural-network models. With the proliferation of personal devices such as phones, smart assistants, etc., the grounds for much of the human-AI interactions has shifted away from the cloud. While this has critical advantages such as user privacy and faster response times, as the space of deep-learning-based applications expands, limited availability of memory on these devices makes deploying hundreds of models impractical. We tackle the problem of re-purposing trained deep-neural-network models to new tasks while keeping most of the learned weights intact. Our method introduces the concept of a "model patch" – a set of small, trainable layers – that can be applied to an existing trained model to adapt it to a new task. While keeping more than 98% of the weights intact, we show significantly higher transfer-learning performance from an object-detection task to an image-classification task, compared to traditional last-layer fine-tuning, among other results. We show how the model-patch idea can be used in multitask learning, where, despite using significantly fewer parameters, we incur zero accuracy loss compared to single-task performance for all the involved tasks.

PUBLICATIONS

This dissertation is based on the following published works co-authored by the dissertation author. Parts of the dissertation are directly excerpted from them.

Chapter 2: Multiresolution Matrix Factorization

- Pramod Kaushik Mudrakarta and Risi Kondor. A generic multiresolution preconditioner for sparse symmetric systems. arXiv preprint arXiv:1707.02054, 2017

- Nedelina Teneva, Pramod Kaushik Mudrakarta, and Risi Kondor. Multiresolution matrix compression. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, volume 51 of Proceedings of Machine Learning Research, pages 1441–1449. PMLR, 2016
  Winner of a notable student paper award (given to top 3 papers)

- Risi Kondor, Nedelina Teneva, and Pramod Kaushik Mudrakarta. Parallel MMF: a multiresolution approach to matrix computation. arXiv preprint arXiv:1507.04396, 2015

Chapter 3: Analyzing Deep Neural Network Models

- Pramod Kaushik Mudrakarta, Ankur Taly, Mukund Sundararajan, and Kedar Dhamdhere. It was the training data pruning too! arXiv preprint arXiv:1803.04579, 2018

- Pramod Kaushik Mudrakarta, Ankur Taly, Mukund Sundararajan, and Kedar Dhamdhere. Did the model understand the question? In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1896–1906. Association for Computational Linguistics, 2018
  Oral presentation

Chapter 4: Parameter-Efficient Transfer and Multitask Learning

- Pramod Kaushik Mudrakarta, Mark Sandler, Andrey Zhmoginov, and Andrew Howard. K for the price of 1: Parameter-efficient multi-task and transfer learning. In International Conference on Learning Representations, 2019

CREDIT ASSIGNMENT

Chapter 2 The pMMF algorithm (Section 2.2) for fast computation of multiresolution matrix factorizations and the associated C++ library[1] were developed by Risi Kondor, Nedelina Teneva and the dissertation author. The dissertation author is the primary contributor in work related to analyzing the library's performance, developing optimizations, adding third-party library support (e.g. MATLAB, Python, TensorFlow), and methods for automatically determining hyperparameters of the algorithm. Multiresolution preconditioning (Section 2.3) and asymmetric multiresolution matrix factorizations (Section 2.4) were primarily driven by the dissertation author in collaboration with Risi Kondor.

Chapter 3 Research presented in this chapter is the fruit of a long-term collaboration among the dissertation author, Ankur Taly, Mukund Sundararajan and Kedar Dhamdhere. The dissertation author (along with Ankur Taly) are primary contributors to performing the case study on Neural Programmer, and developing Path Integrated Gradients. Mukund Sundararajan and the dissertation author are primary developers of the overstability test and adversarial attacks, while Ankur Taly and Kedar Dhamdhere analyzed Jia and Liang [82]'s attacks on SQuAD. Ankur Taly is the primary contributor to analyzing the KDG model. Work related to Neural Machine Translation was primarily driven by the dissertation author while collaborating with Ankur Taly and Mukund Sundararajan.

Chapter 4 The dissertation author along with Mark Sandler are the primary developers of the "model patch" technique and associated numerical experiments. Andrey Zhmoginov developed the analysis with inputs from the dissertation author and Mark Sandler.

1. `http://people.cs.uchicago.edu/~risi/MMF/`

CHAPTER 1

INTRODUCTION

Artificial Intelligence (AI) and Machine Learning (ML) systems have pervaded our day-to-day lives. Thanks to our growing ability to collect massive amounts of data, advancements in computational resources such as Graphical Processing Units (GPUs) and reemergence of deep neural network technology, we find ourselves in an era that is moving towards an AI-first approach to solving problems. A renewed interest in Artificial General Intelligence (AGI) and rapid increase in human-AI-interactive systems have pulled together researchers from various fields of science, and have blurred the lines between academia and industry.

The process of solving a real-world task (Figure 1.1) using AI typically begins with the building of datasets, followed by designing and training a machine learning model, and completed by the deployment of the trained model on user devices.

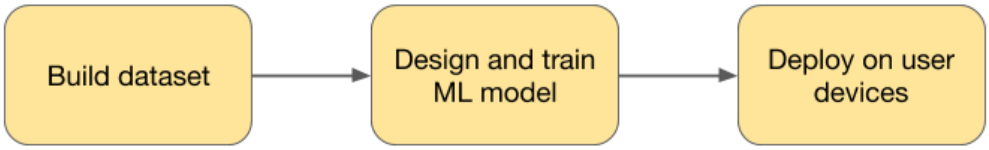

Figure 1.1: **Typical AI pipeline**: the process of solving a real-world task using Artificial Intelligence

For example, providing a German-English language translation service may involve a) collecting millions of German-English sentence pairs, b) designing and training a sequence-to-sequence deep neural network, and c) storing the trained model on a device (e.g., smartphone) and performing inference. A example of a non-neural-network problem would be performing community detection in a social network, that may involve constructing a large graph representing the population and designing a spectral clustering [181] algorithm.

In this thesis, we tackle three challenges, one from each part of the AI pipeline (Figure 1.1). The following sections build the background and describe our contributions.

1.1 Multiresolution Matrix Factorization

Data is often represented as matrices, for e.g., social network graphs are represented as adjacency matrices, user preferences are represented as document-user matrices, etc. As the sizes of modern datasets continue to grow, there is an ever-increasing need for developing techniques that efficiently manipulate matrices. Operations such as matrix-vector products, matrix inverse and determinant are used ubiquitously and repeatedly in machine learning algorithms and numerical a lgebra. A common approach to modeling matrices is to assume that they have low rank. This has led to the development of popular ideas such as Principal Component Analysis, Column Subspace Selection [83], Non-negative Matrix Factorization [111], Nyström methods for compression [195], etc.

Matrices that arise in real-world scenarios are however often found to violate the low-rank assumption. For instance, Laplacian matrices of graphs, similarity or distance matrices, and coefficient matrices in circuit simulations have high rank. These matrices are instead found to have a hierarchical or "cluster-of-clusters" structure. On such matrices, low-rank-based approaches would be ineffective.

One family of methods dedicated to modeling hierarchical structure in matrices include $\mathcal{H}$-matrices [65], $\mathcal{H}^2$-matrices [66], HODLR factorization [7], HSS matrices [186], etc. However, these methods physically divide the matrix using a hierarchical clustering of rows and columns, and can be sensitive to simple orthogonal transformations of the matrix.

Inspired by concepts from multiresolution analysis [122], Multiresolution Matrix Factorization (MMF) was first proposed in [95] to construct wavelets on graphs and extract implicit multiscale structure from matrices via matrix factorization. MMF approximately factorizes a given symmetric matrix $A \in \mathbb{R}^{n \times n}$ into a product

$$A \approx Q_1^T Q_2^T \dots Q_L^T H Q_L \dots Q_2 Q_1, \tag{1.1}$$

where $Q_i, i = 1, \ldots, n$ are carefully chosen sparse orthogonal matrices and H is a near-diagonal matrix with a small dense diagonal block. The factorization itself is computed via minimizing the Frobenius norm of the difference between the original matrix and the product of factors. The orthogonality and diagonality of factors make the MMF amenable to performing computationally efficient matrix operations. In this thesis, we make further contributions to the MMF factorization and its applications.

The algorithm proposed in [95] has $O(n^3)$ computational complexity where n is the number of rows of the symmetric matrix. In Section 2.2, we discuss MMF in more detail, and describe a parallel algorithm (pMMF) and C++ library that can compute the MMF of a large sparse symmetric matrix in linear time.

In Section 2.3, we present an application of MMF. Solving systems of linear equations is a fundamental problem in numerical algebra, and appears frequently in optimization algorithms in machine learning. Given a system of linear equations, $Ax = b$, when A is large and sparse, x is determined by iterative algorithms such as conjugate gradient (CG) [71] or generalized minimum residual (GMRES) [152]. The convergence of these iterative algorithms is slow when the condition number[1] of A is large, as is typically found to be the case for matrices arising in the real world. Preconditioning is a technique to speed up the convergence; the method involves constructing a linear operator M that imitates A^{-1} in some sense, so that we are essentially solving $MAx = Mb$. We survey preconditioners that exploit implicit hierarchical structure in matrices using classical wavelets. We construct a preconditioner based on MMF and show that it outperforms other wavelet-based preconditioners in both speedup of convergence and scalability.

In Section 2.4, we explore a broader version of MMF applicable to generic asymmetric matrices and report its performance for the task of compressing matrices.

1. The condition number of a symmetric matrix A is the ratio of the magnitude of its largest eigenvalue to its smallest.

1.2 Exploring Structure in Deep Neural Network Models

Increasingly, many tasks that were being solved using rule-based or statstical-machine-learning models are now being solved by deep-neural-network models, especially in natural language processing. Examples of such tasks include question answering [132], machine translation [9] and sentiment analysis [44]. The goodness of a trained deep-neural-network model is measured primarily in terms of test accuracy.

Figure 1.2: **VQA example**: an image from the Visual Question Answering 1.0 [4] dataset

While advancements in novel deep-neural-network architectures are increasing test accuracies on benchmark datasets, there are no automatic guarantees of the models' behavior when deployed in real-world scenarios. This can lead to embarrassing failures such as the following: consider a high-performing deep neural network [88] performing Visual Question Answering [4]. The task here is to answer a given natural language question based on the associated image. In Figure 1.2, the image contains a building, and the question is "How symmetric are the white bricks on either side of the building?". The model outputs "very", which is the correct answer. Inspect the following variations of the question and the corresponding model outputs:

- Question: How asymmetrical are the white bricks on either side of the building?
  Output: "very"
- Question: How big are the white bricks on either side of the building?
  Output: "very"
- Question: How spherical are the white bricks on either side of the building?
  Output: "very"

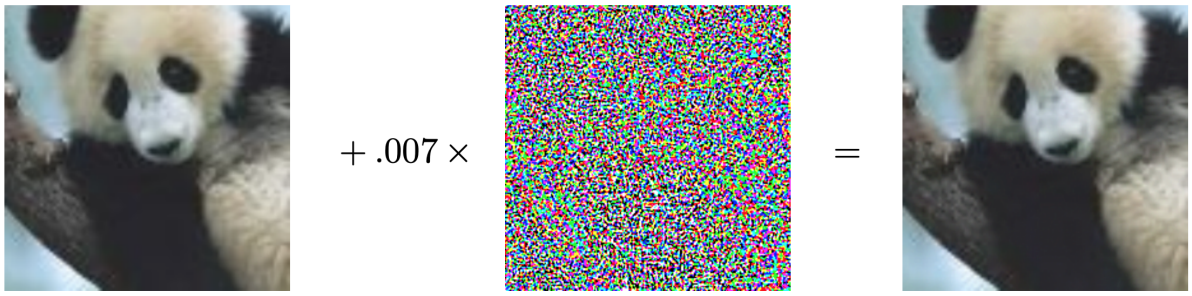

Figure 1.3: **Example of an adversarial attack**: a neural network misclassifying an image when imperceptible noise is added. The original image is classified as “panda” with 57.7% confidence, whereas the altered image is classified as “gibbon” with 99.3% confidence. Example sourced from [58].

- Question: How fast are the bricks speaking on either side of the building?

  Output: “very”

Even on completely irrelevant and absurd questions, the model's output is unchanged. This suggests that the model is not paying attention to the semantics of the question.

Another example of a deep neural network potentially failing in the real world is shown in Figure 1.3, where adding imperceptible noise to the input to an image-classification neural network results in a misclassification with high confidence.

High test accuracy is indicative of a good model only when the test set is representative of the real world. Most modern tasks have large training and test sets often built via crowdsourcing and may have inherent biases. It is impractical to manually ensure homogeneity in a large dataset. Therefore, it becomes important to understand how a deep neural network is making its predictions, and understand its failure modes. From both a transparency and debuggability standpoint, it is essential to develop tools and new measures to evaluate deep-neural-network models.

Unlike rule-based or statistical approaches, the exact logic employed by a neural network to make predictions is implicit and hidden in its parameters/weights. The basic unit

of functional abstraction in a neural network is a neuron, which by itself does not carry meaning. Instead, meaning is derived from sets of neurons acting in concert - this concept is referred to as distributed representation. For example, in a neural network performing image classification on animal types, individual neurons do not correspond to abstractions such as ear, eye, nose, etc. Due to the lack of reliable methods to extract abstractions from trained deep neural networks, there are currently no tools to analyze a given neural network and determine how it would perform on real-world inputs.

One way to get around the lack of strong abstractions is to understand the input-output behavior of neural networks. There has been recent work in attributing the output of a deep neural network to its input features in proportion to their importance [8, 159, 158, 19, 162, 164]. In this thesis, we utilize the notion of attribution to understand the logic, extract rules and construct adversarial attacks on deep neural networks performing question answering.

In Chapter 3, we present a case study of the question-answering neural network "Neural Programmer (NP)" [132], which was a milestone advancement in question-answering neural-network architectures. We demonstrate that NP often gets the correct answer for the wrong reasons. We present an analysis methodology that explains the input-output behavior of Neural Programmer by identifying words that influence the model's predictions. We observe that question-answering neural networks rely on erroneous trigger logic, in the sense that they rely on unimportant input words to make predictions. We leverage such weaknesses in the logic to construct adversarial inputs by making semantically content-free modifications to the input.

The phenomenon of relying on erroneous trigger logic is called overstability, and we present a test and adversarial attacks that measure the extent of this phenomenon. Jia and Liang [82] who first discovered overstability, also propose adversarial examples on reading comprehension models; we show how attributions can strengthen and increase the hit rate of their attacks.

We present a new attribution technique called "Path Integrated Gradients" to examine the influence of an input word via various dataflow paths in the neural network that can be used as a debugging tool for neural networks, and to peform rule extraction. We extract question intents and contextual synonyms from Neural Programmer, and demonstrate how to extract word alignments from a neural machine translation model [9].

By analyzing the state-of-the-art model [103] on WikiTableQuestions dataset, we demonstrate that the quality of the training dataset can have an equally critical impact on the model's performance as aspects of the neural network architecture.

We close by discussing avenues for research in developing debugging tools for neural networks.

1.3 Parameter-Efficient Transfer and Multitask Learning

As the space of deep-learning applications expands and starts to personalize to users, there is a growing need for the ability to quickly build and customize models. While model sizes have dropped dramatically from >50M parameters of the pioneering work of AlexNet [105] and VGG [160] to <5M of the recent Mobilenet [153, 73] and ShuffleNet [197, 119], the accuracy of models has been improving. However, delivering, maintaining and updating hundreds of models on the embedded device is still a significant expense in terms of bandwidth, energy and storage costs.

Compressing neural network models is an active area of research [148, 153, 73]. However, we explore a different angle: We would like to be able to build models that require only a few parameters to be trained in order to be re-purposed to a different task, with minimal loss in accuracy compared to a model trained from scratch. We employ transfer learning and multitask learning techniques to implement this idea.

Transferring pretrained neural networks to new tasks comes with a challenge: catastrophic forgetting [92, 59, 89, 112]. It is the phenomenon where neural networks that are trained

sequentially on multiple tasks lose their performance on earlier tasks as the network is optimized to the objectives of new tasks. For instance, we perform the following two experiments:

- Train a model for image classification ⇒ Fine-tune weights for object detection ⇒ Fine-tune weights for image classification
- Train a model for object detection ⇒ Fine-tune weights for image classification

We find that the accuracy on the final image classification task is the same in both scenarios; as the model weights were fine-tuned for object detection, it lost its performance from previous tasks. Catastrophic forgetting is problematic as it hinders the ability of a model pretrained on a large amount of data (such as BERT [38]) to be adapted to a task where very less data is available (such as question-answering in a particular domain).

In Chapter 4, we demonstrate a novel learning paradigm in which each task carries its own **model patch** – a small set of parameters – that, along with a shared set of parameters constitutes the model for that task (for a visual description of the idea, see Figure 4.1). We put this idea to use in two scenarios: a) in transfer learning, by fine-tuning only the model patch for new tasks, and b) in multi-task learning (ref. Figure 4.2), where each task performs gradient updates to both its own model patch, and the shared parameters. In our experiments (Section 4.4), the largest patch that we used is smaller than 10% of the size of the entire model.

Transfer learning. We demonstrate that by fine-tuning less than 35K parameters in MobilenetV2 [153] and InceptionV3 [167], our method leads to significant accuracy improvements over fine-tuning only the last layer (102K-1.2M parameters, depending on the number of classes) on multiple transfer learning tasks. When combined with fine-tuning the last layer, we train less than 10% of the model's parameters in total. We also show the effectiveness of our method over last-layer-based fine-tuning on transfer learning between completely different problems, namely COCO-trained SSD model [118] to classification over ImageNet [37].

Multitask learning. We explore a multitask learning paradigm wherein multiple models that share most of the parameters are trained simultaneously. Each model has a task-specific model patch. Training is done in a distributed manner; each task is assigned a subset of available workers that send independent gradient updates to both shared and task-specific parameters using standard optimization algorithms. Our results show that simultaneously training two such MobilenetV2 [153] models on ImageNet [37] and Places-365 [198] reach accuracies comparable to, and sometimes higher than individually trained models. **Domain adaptation.** We apply our multitask learning paradigm to domain adaptation. For ImageNet [37], we show that we can simultaneously train MobilenetV2 [153] models operating at 5 different resolution scales, 224, 192, 160, 128 and 96, while sharing more than 98% of the parameters and resulting in the same or higher accuracy as individually trained models. This has direct practical benefit in power-constrained operation, where an application can switch to a lower resolution to save on latency/power without needing to ship separate models and having to make that trade-off decision at the application-design time. The cascade algorithm from [163] can further be used to reduce the average running time by about 15% without loss in accuracy.

In the next chapter, we go in-depth into the first problem from the AI pipeline that we tackle in this dissertation: multiresolution matrix factorization

CHAPTER 2

MULTIRESOLUTION MATRIX FACTORIZATION

Our ability to collect large datasets has been critical to advances in various fields of science and engineering, including machine learning. A common structure used to represent data is the matrix. While the sizes of matrices generated from data has been on the rise, the efficiency of hardware is not increasing at a commensurate pace. Further, the complexity of many algorithms depends on the sizes of the matrices that they use. In many matrix operations, inversion, computing the determinant, etc. are common subroutines that are often the most time consuming parts. Accelerating computations can be achieved, for instance, by compressing the original matrix to a smaller size.

One way of approaching this problem is via explicit optimization of the space occupied by elements of the matrix. For instance, matrices that have few nonzero elements can be efficiently represented using the Column-Sparse-Row (CSR) or the banded matrix format that allows for reduced memory footprint as well as faster computation of several matrix-arithmetical operations [151]. With the advent of parallel processing and GPU computation, further speedups can be achieved by efficient usage of memory and processor caches [14, 130].

A complementary approach for accurately estimating and modeling matrices is via exploiting structure in the data. In this approach, a matrix is typically decomposed into several smaller matrices, which has the added benefit of potentially extracting human-interpretable insights about the data. The nature of the decomposition may allow for faster matrix arithmetic or more accurate compression. The most popular method in this regard is Singular Value Decomposition (SVD), wherein, a matrix $A \in \mathbb{R}^{m \times n}$ is decomposed into an additive form with rank-1 terms.

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

where $r \in \mathbb{N}$ is the rank of the matrix, $\sigma_i \in \mathbb{R}$ are the singular values, and $u_i \in \mathbb{R}^m, v_i \in \mathbb{R}^n, i = 1, \ldots, n$ are the left and right orthogonal singular vectors. The above decomposition can be written in matrix form as

$$A = USV^T$$

where $U \in \mathbb{R}^{m \times m}, V \in \mathbb{R}^{n \times n}$ are orthogonal matrices and $S \in \mathbb{R}^{m \times n}$ such that $S_{i,j} = 0$ if $i \neq j$. An example strategy for compressing A would involve ignoring the terms from the above decomposition for which $|\sigma_i|$ is small. As the computational complexity of performing SVD grows as $O(\max(m, n)^3)$, for large matrices, Nyström [106] and CUR [120] methods are used. These are similar to SVD in the sense that they approximate a given matrix to a lower dimensionality d such that the approximation error is as close to the best rank-d approximation error as possible. In the CUR decomposition [120], a matrix $A \in \mathbb{R}^{m \times n}$ is reduced to rank k by writing it as a product of three matrices

$$A = CUR$$

where $C \in \mathbb{R}^{m \times k}$ is a subset of the columns of A; $R \in \mathbb{R}^{k \times n}$ is a subset of the rows of A, and $U \in \mathbb{R}^{k \times k}$ is chosen such that $||A - CUR||_F$ is as close to the best rank-k approximation of A. The Nyström method has variants [183, 52, 57, 45] similarly defined on symmetric matrices. Both these methods are asymptotically faster than SVD and can be used when the matrices are large. They also offer the advantage that any properties of A that help efficiency can be reused in C and R, for example, sparsity.

The practical effectiveness of the above methods critically depends on the matrix having low-rank structure. While this has been found effective on matrices arising in recommender systems [56, 99], natural language processing [109], information retrieval [147], where the matrices are typically rectangular with one dimension significantly larger than the other, the low-rank assumption may not suit matrices arising in other fields of science and engineering.

For instance, matrices that are Laplacians of graphs[1], or those arising from discretization of differential or integral equations are typically square, and tend to be high rank. For such matrices, it may be beneficial to exploit other kinds of structure. Graphs, for example, those arising from social networks, have strong locality properties, i.e., they exhibit structure at various "scales of resolution". In other words, the linear transformations represented by the matrices may couple coordinates within a small cluster more strongly than with other coordinates. Such "local" interactions are not captured by the singular vectors of SVD, which are almost always dense. Sparse versions of an SVD-based technique for correlation matrices, namely sparse PCA [81], impose sparsity constraints on the singular vectors (while relaxing the orthogonality constraint) that can capture local interactions within coordinates, but fail to represent larger scale interactions between coordinates.

In this thesis, our focus is on modeling matrices by capturing both large-scale and local interactions between coordinates that is represented by the matrix. While there does not exist a formal and universal definition for hierarchical/multiscale/multiresolution structure in a similar vein as the rank, the following methods exploit the notion of such structure in modeling matrices.

The hierarchical matrix family of methods includes $\mathcal{H}$-matrices [65], $\mathcal{H}^2$-matrices [66], HODLR factorization [7], HSS matrices [186] that physically divide the matrix into a hierarchical cascade of block structures and place low-rank assumptions on some of the blocks. The idea is that, although a matrix is high rank, it can be composed of submatrices that have low rank. This is achieved, for instance, by first computing a hierarchical clustering of the rows and columns of A using a method such as METIS [86] and using the result to determine the appropriate submatrices. A visual illustration of such a decomposition is shown in Figure 2.1. Hierarchical matrix methods are closely related to the Fast Multipole Method [12] and Multigrid [23], which are techniques for solving partial differential equations; they are

1. The Laplacian $L \in \mathbb{R}^{n \times n}$ of an undirected graph G with n nodes is defined as $L = D - A$, where A is the adjacency matrix of the graph, and D is a diagonal matrix with the degrees of nodes on the diagonal.

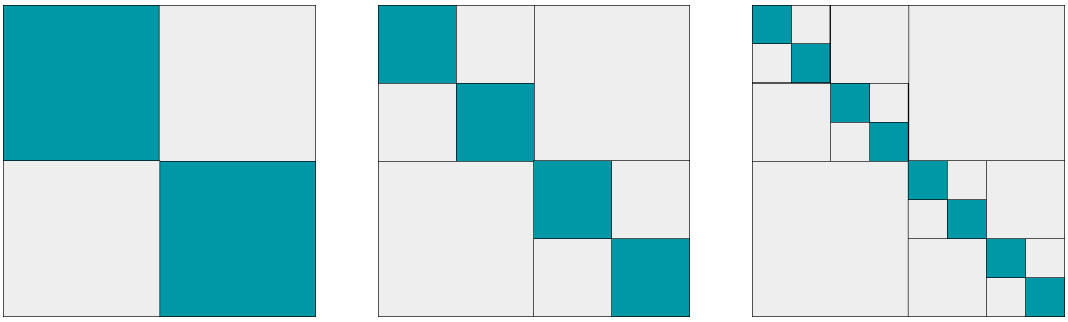

Figure 2.1: **Example of a hierarchical matrix**: The rows and columns of the matrix are assumed to be appropriately ordered to show the block structure visually. On the left most side, we have the level-1 approximation that represents the off-diagonal blocks in terms of a low-rank approximation. In the center is level-2 approximation that further divides the diagonal full-rank blocks as hierarchical matrix. The right image is a three-level hierarchical matrix, that has highest approximation error but leads to fastest matrix arithmetic.

effective on matrices that arise in this field. However, on matrices that arise from social network analysis, the matrix-generating process is unknown and these methods may not be the best suited. Further, hierarchical-matrix methods fail even on simple transformations of the matrix such as permutations of rows/columns, orthogonal transforms, etc.

Decomposing transformations/functions into hierarchically-arranged parts is an idea that has already been explored in the field of harmonic analysis, a branch of mathematics that deals with representations of functions in terms of "waves", or functions that are associated with the idea of frequency. While the most famous concept from this field, Fourier decomposition, decomposes a given function in terms of sinusoidal basis functions of varying frequencies, **wavelets** are basis functions that also have an associated frequency but, unlike Fourier basis functions, are localized in space as well as frequency. This allows for wavelet basis functions to exist in hierarchical forms, capturing both large-scale as well as local effects of a given function. Particular wavelet basis functions such as Haar wavelets [64], Daubechies wavelets [34], etc. have been extensively used in signal processing, image denoising and other applications. An example of the Haar wavelet basis is shown in Figure 2.2. The discrete version of a wavelet decomposition can be represented as an orthogonal matrix. For a brief introduction to discrete wavelet transforms, we refer the reader to Appendix A.

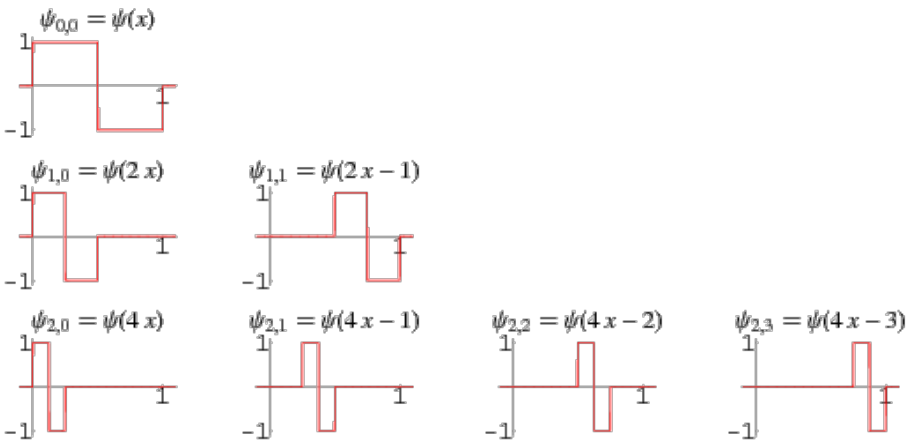

Image sourced from `http://mathworld.wolfram.com/HaarFunction.html`

Figure 2.2: **Haar wavelets**: basis functions on the real line at increasing levels (top to bottom). The functions are orthogonal to each other, and have compact support ($\{x \in \mathbb{R} | \phi_{i,j}(x) \neq 0\}$) that makes them localized in the function domain.

Wavelet-like bases are natural tools to model hierarchical structure in matrices. Hawkins et al. [70] express matrix inverses as sparse matrices in the Haar/Daubechies wavelet bases, and use it to construct approximate inverses that can be used for preconditioning linear solvers.

In this thesis, our aim is to make progress towards developing a general-purpose method for modeling hierarchical structure in matrices. In particular, we would like to work towards analogues of SVD or the CUR/Nyström methods to full-rank matrices that are just as powerful, general and efficient.

We start with the direction founded in **Multiresolution Matrix Factorization** [95], a method that was originally introduced to define multiresolution and wavelets on discrete spaces such as graphs, and express it via a matrix factorization. Unlike Diffusion Wavelets [30] that are restricted to symmetric positive-definite matrices, and Treelets [110] that are restricted to tree-Laplacians, MMF is applicable to any symmetric matrix.

In this chapter, we describe a fast matrix-factorization algorithm for MMF (Section 2.2) that is scalable to matrices with millions of dimensions, develop multiresolution decompo-

sition methods for asymmetric matrices (Section 2.4), and evaluate the algorithms' downstream performance in applications such as linear system preconditioning (Section 2.3) and matrix compression (Section 2.4.3).

We begin with an introduction to Multiresolution Matrix Factorization.

2.1 Preliminaries: Multiresolution Matrix Factorization

Any symmetric matrix $A \in \mathbb{R}^{n \times n}$ can be written as

$$A = \sum_{i=0}^{n} \lambda_i v_i v_i^T$$

where $v_i \in \mathbb{R}^n, i = 0, \ldots, n$ are eigenvectors and $\lambda_i \in \mathbb{R}$ are eigenvalues. In matrix form, applying an orthogonal transformation composed of the eigenvectors to A diagonalizes it and reveals the eigenvalues.

Matrices arising in large datasets such as graphs, circuit simulations or finite element matrices have strong locality properties. That is, coordinates belonging to one cluster of coordinates exhibit similar properties, and such behaviour may be present at multiple levels or scales. However, by virtue of the fact that eigenvectors are almost always dense, such localized behavior does not naturally reveal itself in the eigenvalue decomposition.

Multiresolution Matrix Factorization (MMF) [95] is an approximate multilevel factorization of a given symmetric matrix, intended to expose multiscale structure present in the matrix. It applies several carefully constructed sparse, orthogonal transformations to a given matrix in order to approximately diagonalize it. Each sparse, orthogonal transformation acts on a subset of the coordinates and is designed to reveal localized clusters at different "levels" or "scales of resolution".

MMF is closely related to classical wavelets and multiresolution analysis. Given a matrix $A \in \mathbb{R}^{n \times n}$, the domain of the linear transformation represented by it, i.e., $\mathbb{R}^n$ is split into

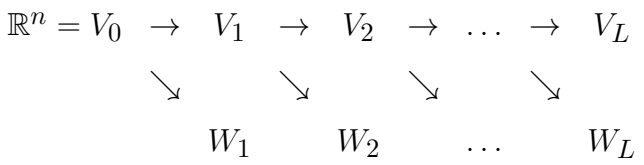

Figure 2.3: **Multiresolution analysis (MRA)**: A multiresolution decomposition of $\mathbb{R}^n$. Each space V_i is filtered to a smoother space V_{i+1}, and a "detail" space W_{i+1}, such that $V_i = V_{i+1} \oplus W_{i+1}$. Smoothness of a subpsace V with respect to a symmetric matrix A is defined, for instance by [95] as $\sup_{v \in V} \frac{\langle v, Av \rangle}{\langle v, v \rangle}$.

a telescoping sequence of spaces that are increasingly "smooth" according to a smoothness measure that depends on A (see Figure 2.3).

The MMF for a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is of the form

$$A \approx Q_1^T Q_2^T \ldots Q_L^T H Q_L \ldots Q_2 Q_1, \tag{2.1}$$

where the matrices $Q_1, \ldots, Q_L$ and H obey the following conditions:

1. Each $Q_\ell \in \mathbb{R}^{n \times n}$ is orthogonal and highly sparse. In the simplest case, each Q_ℓ is a Givens rotation, i.e., a matrix which differs from the identity in just the four matrix elements

$$[Q_\ell]_{i,i} = \cos\theta, \qquad [Q_\ell]_{i,j} = -\sin\theta,$$
$$[Q_\ell]_{j,i} = \sin\theta, \qquad [Q_\ell]_{j,j} = \cos\theta,$$

   for some pair of indices (i, j) and rotation angle θ. Multiplying a vector with such a matrix rotates it counter-clockwise by θ in the (i, j) plane. More generally, Q_ℓ is a so-called k-point rotation, which rotates not just two, but k coordinates.

2. Typically, in MMF factorizations $L = O(n)$, and the size of the active part of the Q_ℓ matrices decreases according to a set schedule $n = \delta_0 \geq \delta_1 \geq \ldots \geq \delta_L$. More precisely,

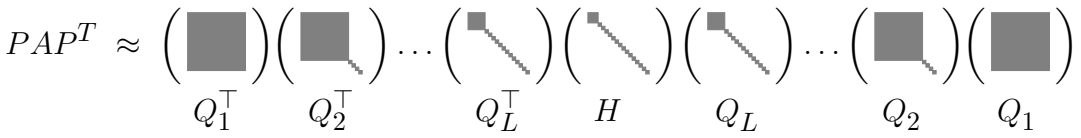

Figure 2.4: **Multiresolution Matrix Factorization**: A graphical representation of the structure of Multiresolution Matrix Factorization. Here, P is a permutation matrix which ensures that $S_\ell = \{1, \ldots, \delta_\ell\}$ for each ℓ. Note that P is introduced only for the sake of visualization, an actual MMF would not contain such an explicit permutation.

there is a nested sequence of sets $[n] = S_0 \supseteq S_1 \supseteq \ldots \supseteq S_L$ such that the $[Q_\ell]_{\overline{S_{\ell-1}},\overline{S_{\ell-1}}}$ part of each rotation is the $n - \delta_{\ell-1}$ dimensional identity. S_ℓ is called the **active set** at level ℓ. In the simplest case, $\delta_\ell = n - \ell$.

3. H is an S_L-core-diagonal matrix, which means that it is block diagonal with two blocks: H_{S_L,S_L}, called the core, which is dense, and $H_{\overline{S_L},\overline{S_L}}$ which is diagonal. In other words, $H_{i,j}=0$ unless $i, j \in S_L$ or $i=j$.

The structure implied by the above conditions is illustrated in Figure 2.4. MMF factorizations are, in general, only approximate, as there is no guarantee that $O(n)$ sparse orthogonal matrices can bring a symmetric matrix to core-diagonal form. Rather, the goal of MMF algorithms is to minimize the approximation error using the commonly used Frobenius norm of the difference between the original matrix and its MMF factorized form.

The columns of $Q^T = Q_1^T \ldots Q_{L-1}^T Q_L^T$ have a natural interpretation as wavelets, and the factorization itself is effectively a fast wavelet transform, mimicking the structure of classical orthogonal multiresolution analyses on the real line [122].

The MMF of a symmetric matrix A is usually computed by minimizing the Frobenius norm factorization error

$$\| A - Q_1^T \ldots Q_L^T H Q_L \ldots Q_1 \|_F \tag{2.2}$$

over all admissible choices of active sets $S_1, \ldots, S_L$ and rotation matrices $Q_1, \ldots, Q_L$.

The minimization is typically [95] carried out in a greedy manner, where the rotation matrices $Q_1, \ldots, Q_L$ are determined sequentially, as A is subjected to the sequence of transformations

$$A \mapsto \underbrace{Q_1 A Q_1^T}_{A_1} \mapsto \underbrace{Q_2 Q_1 A Q_1^T Q_2^T}_{A_2} \mapsto \ldots \mapsto \underbrace{Q_L \ldots Q_2 Q_1 A Q_1^T Q_2^T \ldots Q_L^T}_{H}.$$

In this process, at each level ℓ, the algorithm

1. Determines which subset of rows/columns $\{i_1, \ldots, i_k\} \subseteq S_{\ell-1}$ are to be involved in the next rotation, Q_ℓ.

2. Given $\{i_1, \ldots, i_k\}$, it optimizes the actual entries of Q_ℓ.

3. Selects a subset of the indices in $\{i_1, \ldots, i_k\}$ for removal from the active set (the corresponding rows/columns of the working matrix A_ℓ then become "wavelets").

4. Sets the off-diagonal parts of the resulting wavelet rows/columns to zero in H.

The final error is the sum of the squares of the zeroed out off-diagonal elements (see Proposition 1 in [95]). The objective therefore is to craft each Q_ℓ such that these off-diagonals are as small as possible.

For example, consider $k = 2$; each rotation Q_ℓ is a Jacobi rotation, i.e., an identity matrix with an additional two nonzero off-diagonal elements. The algorithms proposed in [95] pick, at each stage, two indices to rotate corresponding to the columns of the active part of matrix that are closest to each other measured by the absolute value of their inner product. Thus, it involves first computing a Gram matrix, i.e., $G = A_\ell^T A_\ell$, and picking indices i, j such that $G_{ij} = \max_{p \neq q} |G_{pq}|$. Then, a Jacobi rotation Q_ℓ is constructed such that the (i, j)-th element of $Q_\ell^T G Q_\ell$ is zero. This has the effect that, after applying the same rotation to A_ℓ, w.l.o.g, some of the magnitude of the j-th column vector of $Q_\ell^T A_\ell Q_\ell$ is transferred to the

i-th column. Finally, the off-diagonals of the j-th column can be zeroed out while incurring a small error.

Although performing only $O(n)$ fast rotations, the GreedyJacobi and GreedyParallel algorithms proposed in [95] have $O(n^3)$ running time complexity. In the next section, we describe a parallelized algorithm for computing MMF that has significantly reduced running time.

2.2 Parallel Multiresolution Matrix Factorization

The algorithms proposed in [95] scale poorly with n. For instance, the row/column subsets to rotate in Step 1 involve computing inner products, which means computing and storing a Gram matrix, that has complexity $O(n^3)$. This running time complexity is not practical for handling datasets with millions of rows and columns.

The pMMF algorithm [169] is a faster algorithm that employs two heuristics. First, the row/column selection process is accelerated by randomization: for each ℓ, the first index i_1 is chosen uniformly at random from the current active set $S_{\ell-1}$, and then $i_2, \ldots, i_k$ are chosen so as to ensure that Q_ℓ can produce $\delta_\ell - \delta_{\ell-1}$ rows/columns with suitably small off-diagonal norm. Second, exploiting the fundamentally local character of MMF pivoting, the entire algorithm is parallelized using a generalized blocking strategy first described in [169].

Notation 1. *Let $B_1 \dot\cup B_2 \dot\cup \ldots \dot\cup B_k = [n]$ be a partition of $[n]$ and $A \in \mathbb{R}^{n\times n}$. We use $[\![A]\!]_{i,j}$ to denote the $[A]_{B_i,B_j}$ block of A and say that A is $(B_1, \ldots, B_k)$-block-diagonal if $[\![A]\!]_{i,j} = 0$ if $i \neq j$.*

The pMMF algorithm uses a rough clustering algorithm to group the rows/columns of A into a predetermined number of blocks, and factors each block independently and in parallel. However, to avoid overcommitting to a specific clustering, each of these factorizations is only partial (typically the core size is on the order of 1/2 of the size of the block). The algorithm

Algorithm 1 pMMF (top level of the pMMF algorithm)

Input: a symmetric matrix $A \in \mathbb{R}^{n \times n}$
$A_0 \leftarrow A$
for $(p = 1$ to $P)$ {
  Cluster the active columns of A_{p-1} to $B_1^p \cup B_2^p \cup \ldots \cup B_m^p$
  Reblock A_{p-1} according to $(B_1^p, \ldots, B_m^p)$
  for $(u = 1$ **to** $m)$ $[\![\overline{Q}_p]\!]_{u,u} \leftarrow$ FindRotationsForCluster$([A_p]_{:,B_u})$
  for $(u = 1$ **to** $m)$ {
    for $(v = 1$ **to** $m)$ {
      $[\![A_p]\!]_{u,v} \leftarrow [\![\overline{Q}_p]\!]_{u,u} [\![A_{p-1}]\!]_{u,v} [\![\overline{Q}_p]\!]_{v,v}^\top$
  }}
}
$H \leftarrow$ the core of A_L plus its diagonal
Output: $(H, \overline{Q}_1, \ldots, \overline{Q}_p)$

proceeds in stages, where each stage consists of (re-)clustering the remaining active part of the matrix, performing partial MMF on each cluster in parallel, and then reassembling the active rows/columns from each cluster into a single matrix again (Algorithm 1).

Assuming that there are P stages in total, this process results in a two-level factorization. Abstracting Equation (2.1) to the stage level by collapsing multiple rotations, we have

$$A \approx \overline{Q}_1^T \overline{Q}_2^T \ldots \overline{Q}_P^T H \overline{Q}_P \ldots \overline{Q}_2 \overline{Q}_1, \tag{2.3}$$

where, assuming that the clustering in stage p is $B_1^p \cup B_2^p \cup \ldots \cup B_m^p$, each $\overline{Q}_p$ is a $(B_1^p, \ldots, B_m^p)$ block diagonal orthogonal matrix, which, in turn, factors into a product of a large number of elementary k-point rotations

$$\overline{Q}_p = Q_{l_p} \ldots Q_{l_{p-1}+2} Q_{l_{p-1}+1}. \tag{2.4}$$

The subroutine used to compute the rotations in each cluster is presented in Algorithm 2.

Thanks to the combination of these computational tricks, empirically, for sparse matrices, pMMF can achieve close to linear scaling behavior with n, both in memory and computation

Algorithm 2 FINDROTATIONSFORCLUSTER($\mathcal{U}$) (we assume $k=2$ and η is the compression ratio)

Input: a matrix $\mathcal{U}$ made up of the c columns of A_{p-1} forming cluster u in A_p
Compute the Gram matrix $G=\mathcal{U}^\top\mathcal{U}$
$S \leftarrow \{1,2,\ldots,c\}$ (the active set)
for $(s=1$ **to** $\lfloor \eta c \rfloor)\{$
  Select $i \in S$ uniformly at random
  Find $j = \text{argmax}_{S\setminus\{i\}} \left|\langle \mathcal{U}_{:,i}, \mathcal{U}_{:,j}\rangle\right| / \|\mathcal{U}_{:,j}\|$
  Find the optimal Givens rotation q_s of columns (i,j)
  $\mathcal{U} \leftarrow q_s \mathcal{U} q_s^\top$
  $G \leftarrow q_s G q_s^\top$
  if $\|\mathcal{U}_{i,:}\|_{\text{off-diag}} < \|\mathcal{U}_{j,:}\|_{\text{off-diag}}$ **then** $S \leftarrow S \setminus \{i\}$ **else** $S \leftarrow S \setminus \{j\}$
$\}$
Output: $[\![\overline{Q}_p]\!]_{u,u} = q_{\lfloor \eta c \rfloor} \cdots q_2 q_1$

time [169].

Once we are able to compute the MMF fast, we can then apply MMF to accelerate machine learning and numerical algebra algorithms. In the next section, we will see how to use MMF to accelerate solvers for systems of linear equations.

2.3 Multiresolution Preconditioning

Symmetric linear systems of the form

$$Ax = b \tag{2.5}$$

where $A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n$ are central to many numerical computations in science and engineering. In machine learning, linear systems are often solved as part of optimization algorithms [188, 189, 172]. Often, solving the linear system is the most time consuming part of large scale computations.

When A, the coefficient matrix, is large and sparse, usually iterative algorithms such as the minimum residual method (MINRES) [137] or the stabilized bi-conjugate gradient

method (BiCGStab) [173] are used to solve equation (2.5). However, if the condition number[2] $\kappa_2(A)$ is high (i.e., A is ill-conditioned), these methods tend to converge slowly. Many matrices arising from problems of interest are ill-conditioned.

Preconditioning is a technique to improve convergence, where, instead of equation (2.5), we solve

$$MAx = Mb, \tag{2.6}$$

where $M \in \mathbb{R}^{n \times n}$ is a rough approximation to A^{-1} [3]. While equation (2.6) is still a large linear system, it is generally easier to solve than equation (2.5), because MA is more favorably conditioned than A. Note that solving equation (2.6) with an iterative method involves computing many matrix-vector products with MA, but that does not necessarily mean that MA needs to be computed explicitly. This is an important point, because even if A is sparse, MA can be dense, and therefore expensive to compute.

There is no such thing as a "universal" preconditioner. Preconditioners are usually custom-made for different kinds of coefficient matrices and are evaluated differently based on what kind of problem they are used to solve (how accurate x needs to be, how easy the solver is to implement on parallel computers, storage requirements, etc.). Some of the most effective preconditioners exploit sparsity. The best case scenario is when both A and M are sparse, since in that case all matrix-vector products involved in solving equation (2.6) can be evaluated very fast. Starting in the 1970s, this lead to the development of so-called Sparse Approximate Inverse (SPAI) preconditioners [16, 63, 18, 70], which formuate finding M as

2. The condition number of a matrix is defined as the absolute value of the largest eigenvalue divided by the smallest. When the smallest eigenvalue is zero, the condition number is interpreted as infinity.

3. An alternate way to precondition is from the right, i.e., solve $AMx = b$, but, for simplicity, in this chapter we constrain ourselves to discussing left preconditioning.

a least squares problem

$$\min_{M \in \mathcal{S}} ||AM - I||_F, \tag{2.7}$$

where $\mathcal{S}$ is an appropriate class of sparse matrices. Note that since $||AM-I||_F^2 = \sum_{i=1}^{n} ||Am_i - e_i||_2^2$, where m_i is the i-th column of M and e_i is the i-th standard basis vector, equation (2.7) reduces to solving n independent least square problems, which can be done in parallel.

One step beyond generic SPAI preconditioners are methods that use prior knowledge about the system at hand to transform A to a basis where its inverse can be approximated in sparse form. For many problems, orthogonal wavelet bases are a natural choice. Recall that wavelets are similar to Fourier basis functions, but have the advantage of being localized in space. A brief introduction to wavelet bases for $\mathbb{R}^n$ is provided in Appendix A. Transforming equation (2.5) to a wavelet basis amounts to rewriting it as $\widetilde{A}\widetilde{x} = \widetilde{b}$, where

$$\widetilde{A} = W^T A W, \qquad \widetilde{x} = W^T x, \qquad \text{and} \qquad \widetilde{b} = W^T b.$$

Here, the wavelets appear as the columns of the orthogonal matrix W. This approach was first proposed by Chan, Tang and Wan [25].

Importantly, many wavelets admit fast transforms, meaning that W^T factors in the form

$$W^T = W_L^T W_{L-1}^T \dots W_1^T, \tag{2.8}$$

where each of the W_ℓ^T factors are sparse. While the wavelet transform itself is a dense transformation, in this case, transforming to the wavelet basis inside an iterative solver can be done by sparse matrix-vector arithmetic exclusively. Each W_ℓ matrix can be seen as being responsible for extracting information from x at a given scale, hence wavelet transforms constitute a form of multiresolution analysis.

Wavelet sparse preconditioners have proved to be effective primarily in the PDE domain, where the problem is typically 2D or 3D and the structure of the equations (together with the discretization) strongly suggest the form of the wavelet transform. However, multiscale data is much more broadly prevalent, e.g., in biological problems and social networks. For these kinds of data, the underlying generative process is unknown, rendering the classical wavelet-based preconditioners ineffective.

In this section, we propose and evaluate a preconditioner based on Multiresolution Matrix Factorization.

2.3.1 Overview of Wavelet-based Preconditioners

Chan, Tang and Wan [25] were the first to propose a wavelet sparse approximate inverse preconditioner. In their approach, the linear system equation (2.5) is first transformed into a standard wavelet basis such as the Daubechies [34] basis, and a sparse approximate inverse preconditioner is computed for the transformed coefficient matrix by solving

$$\min_{M \in \mathcal{S}_{\text{blockdiag}}} \left|\left| W^T A W M - I \right|\right|_F . \tag{2.9}$$

The preconditioner is constrained to be block diagonal in order to maintain its sparsity and simplify computation. They show the superiority of the wavelet preconditioner over an adaptive sparse approximate inverse preconditioner for elliptic PDEs with smooth coefficients over regular domains[4]. However, their method performs poorly for elliptic PDEs with discontinuous coeffcients. The block diagonal constraint does not fully capture the structure of the inverse in the wavelet basis.

Bridson and Tang [22] construct a multiresolution preconditioner similar to Chan, Tang and Wan [25], but determine the sparsity structure adaptively. Instead of using Daubechies

4. A regular domain refers to the use of a uniform, regular grid in the discretization of a PDE using the finite-element method

Algorithm 3 Solve $Ax = b$ using the implicit wavelet SPAI preconditioner [70]

1: Compute preconditioner $\widehat{M} = \arg\min_{M \in \mathcal{S}_W} ||AM - W||_F$
2: Solve $W^T A \widehat{M} y = W^T b$
3: **return** $x = \widehat{M} y$

Algorithm 4 Compute preconditioner $\widehat{M} = \arg\min_{M \in \mathcal{S}_W} ||AM - W||_F$

1: **for** $j = 1, \ldots, n$ **do**
2: S_j = indices of nonzero entries of w_j
3: T_j = indices of nonzero entries of $A(:, S_j)$
4: Solve $z^* = \arg\min ||A(T_j, S_j)z - w_j(T_j)||_2$ by reduced QR-factorization
5: Set $\widehat{m}_j(T_j) = z^*$
6: **end for**
7: **return** $\widehat{M}$

wavelets, they use second generation wavelets [166], which allows the preconditioner to be effective for PDEs over irregular domains. However, their algorithm requires the additional difficult step of finding a suitable ordering of the rows/columns of the coefficient matrix which limits the number of levels to which multiresolution structure can be exploited.

Hawkins and Chen [70] compute an implicit wavelet sparse approximate inverse preconditioner, which removes the computational overhead of transforming the coefficient matrix to a wavelet basis. Instead of equation (2.9), they solve

$$\min_{M \in \mathcal{S}_W} \left|\left| W^T A M - I \right|\right|_F, \tag{2.10}$$

where $\mathcal{S}_W$ is the class of matrices which have the same sparsity structure as W. They empirically show that this sparsity constraint is enough to construct a preconditioner superior to that of Chan, Tang and Wan [25]. The complete algorithm is described in Algorithms 3 and 4.

Hawkins and Chen [70] apply their preconditioner on Poisson and elliptic PDEs in 1D, 2D and 3D. We found, by experiment, that it is critical to use a wavelet transform of the same dimension as the underlying PDE of the linear system for success of their preconditioner. On

linear systems where the underlying data generator is unknown — this happens, for example, when we are dealing with Laplacians of graphs — their preconditioner is ineffective. Thus, there is a need for a wavelet sparse approximate inverse preconditioner which can mould itself to any kind of data, provided that it is reasonable to assume a multiresolution structure.

2.3.2 Related Work on Preconditioning

Constructing a good preconditioner hinges on two things: 1. being able to design an efficient algorithm to compute an approximate inverse to A, and 2. making the preconditioner as close to A^{-1} as possible. It is rare for both a matrix and its inverse to be sparse. For example, Duff et al. [46] show that the inverses of irreducible sparse matrices are generally dense. However, it is often the case that many entries of the inverse are small, making it possible to construct a good sparse approximate inverse. For example, [36] shows that when A is banded and symmetric positive definite, the distribution of the magnitudes of the matrix entries in A^{-1} decays exponentially away from the diagonal w.r.t. the eigenvalues of A. Benzi and Tuma [18] note that sparse approximate inverses have limited success because of the requirement that the actual inverse of the matrix has small entries.

A better way of computing approximate inverses is in factorized form using sparse factors. The dense nature of the inverse is still preserved in the approximation as the product of the factors (which is never explicitly computed) can be dense. Factorized approximate inverses have been proposed based on LU factorization. However, they are not easily parallelizable and are sensitive to reordering [18].

Multiscale variants of classic preconditioners have already been proposed and have often been found to be superior [17] to their one-level counterparts. The current frontiers of research on preconditioning also focus on designing algorithms for multi-core machines. Multilevel preconditioners assume that the coefficient matrix has a hierarchy in structure. These include the preconditioners that are based on rank structures, such as $\mathcal{H}$-matrices

[65], which represent a matrix in terms of a hierarchy of blocked submatrices where the off-diagonal blocks are low rank. This allows for fast inversion and LU factorization routines. Preconditioners based on $\mathcal{H}$-matrix approximations have been explored in [49, 101, 61]. Other multilevel preconditioners based on low rank have been proposed in [185].

Multigrid preconditioners [23, 143] are reduced tolerance multigrid solvers, which alternate between fine- and coarse-level representations to reduce the low and high frequency components of the error respectively. In contrast, hierarchical basis methods [192, 191] precondition the original linear system as in equation (2.6) by expressing A in a hierarchical representation. A hierarchical basis-multigrid preconditioner has been proposed in [11].

Hierarchical basis preconditioners can be thought of as a special kind of wavelet preconditioners as it is possible to interpret the piecewise linear functions of the hierarchical basis as wavelets. Connections between wavelets and hierarchical basis methods have also been explored in [176, 177] to improve the performance of hierarchical basis methods.

2.3.3 MMF Preconditioner

Recall that Multiresolution Matrix Factorization (Section 2.1) is an approximate factorization of a matrix A of the form

$$A \approx Q_1^T Q_2^T \dots Q_L^T H Q_L Q_{L-1} \dots Q_1, \tag{2.11}$$

where each of the Q_ℓ matrices are sparse and orthogonal, and H is close to diagonal. Similar to equation (2.8), MMF has a corresponding fast wavelet transform. However, in contrast to classical wavelet transforms, here the Q_ℓ matrices are not induced from any specific analytical form of wavelets, but rather "discovered" by the algorithm itself from the structure of A, somewhat similarly to algebraic multigrid methods [149]. This feature gives our preconditioner considerably more flexibility than existing wavelet sparse preconditioners, and

allows it to exploit latent multiresolution structure in a wide range of problem domains.

The key property of MMF we exploit is that equation (2.1) automatically gives rise to an approximation to A^{-1},

$$\widetilde{A^{-1}} = Q_1^T \dots Q_{L-1}^T Q_L^T H^{-1} Q_L Q_{L-1} \dots Q_1, \tag{2.12}$$

which is very fast to compute, since inverting H reduces to separately inverting its core (which is assumed to be small) and inverting its diagonal block (which is trivial). Assuming that the core is small enough, the overall cost of inversion becomes $O(n)$ as the matrices Q_i are sparse. When using equation (2.12) as a preconditioner, of course we never compute equation (2.12) explicitly, but rather (similarly to other wavelet sparse approximate inverse preconditioners) we apply it to vectors in factorized form as

$$\widetilde{A^{-1}} v = Q_1^T (\dots (Q_{L-1}^T (Q_L^T (H^{-1} (Q_L (Q_{L-1} \dots (Q_1 v)) \dots). \tag{2.13}$$

Since each of the factors here is sparse, the entire product can be computed in $O(n)$ time. We remark that computing the MMF preconditioner does not depend on the spatial dimension of the underlying problem. In fact, for matrices such as graph Laplacians, the underlying dimensionality is unknown. MMF is more widely applicable because of its dimension-free nature.

2.3.4 Numerical Results

We compare performance of the preconditioners on both model PDE problems and off-the-shelf sparse matrix datasets.

Implementation and parameters

We implemented parallelized versions of all preconditioners in both MATLAB and C++. In MATLAB, parallel code was written using the command `parfor` and in C++, we used the pthreads library.

For PDE problems which are relatively small, we use the MATLAB code. We used a block size of 8 for WSPAI as used in [25, 70]. For IWSPAI, we used 6 wavelet levels. To solve the minimization problem in Step 4 of Algorithm 4, we used the `lsqr` method in MATLAB, as we found it to be faster and accurate in reproducing the results in [70]. The maximum number of iterations in `lsqr` was set to 500. We used Daubechies wavelets [34] for both the wavelet preconditioners.

The pMMF library [97] was used to compute the MMF preconditioner. Default parameters supplied by the library were used. These include using second order rotations, i.e., Givens rotations, designating half of the active number of columns at each level as wavelets and compressing the matrix until the core is of size 100×100. The parameter which controls the extent of pMMF parallelization, namely the maximum size of blocks in blocked matrices, was set to 2000. The full set of pMMF parameters that were used are shown in Table 2.1.

We note that the C++ implementation of the wavelet preconditioners using the same matrix classes as pMMF was slower than its MATLAB counterpart. This suggests that specialized matrix libraries are needed to speed-up their code in C++.

The PDE experiments were run on 28-core Intel E5-2680v4 2.4GHz computers. We used GMRES (no restarts) with a maximum number of iterations of 1000.

For off-the-shelf matrices, we use the C++ implementations. The parameters used were the same, except pMMF on matrices larger than 65536 rows uses a maximum cluster size of 5000. We used GMRES (with restarts after 30 iterations) to a maximum of 500 iterations. The experiments were run on 32-core AMD Opteron 6386 SE.

Table 2.1: **pMMF parameters**: Full set of parameters used in pMMF for the preconditioning experiments

pMMF parameter	value
Order of rotations (k)	2
Max number of stages	30
Rotations per stage (fraction)	0.5
Number to eliminate after each rotation	1
Target core size	100
Prenormalize each channel	off
Rotation selection criterion	inner product
Inner products normalized	off
Grams based on diagonal blocks only	off
Clustering method	inner product
Number of clusters	1
Minimum cluster size	1
Maximum cluster size	2000
Maximum clustering depth	4
Maximum clustering iterations	8
Bypass option	on
Compute Frobenius error	off

Model PDE problems

We consider finite difference discretizations of PDE problems in 1D, 2D, and 3D. The problems used are

- *1D Laplacian.* One dimensional Poisson's equation

$$u_{xx} = (1 + x^2)^{-1} e^x, \quad x \in [0, 1],$$

  with a Dirichlet boundary condition discretized with central differences.

- *2D Laplacian.* Two dimensional Poisson's equation

$$u_{xx} + u_{yy} = -100x^2, \quad (x, y) \in [0, 1]^2,$$

with a Dirichlet boundary condition discretized with central differences.

- *3D Laplacian.* Three dimensional Poisson's equation

$$u_{xx} + u_{yy} + u_{zz} = -100x^2, \quad (x, y, z) \in [0, 1]^3.$$

- *2D Disc.* Two dimensional PDE with discontinuous coefficients

$$(a(x, y)u_x)_x + (b(x, y)u_y)_y = \sin(\pi xy), \quad (x, y) \in [0, 1]^2,$$

with

$$a(x, y) = b(x, y) = \begin{cases} 10^{-3}, & (x, y) \in [0, 0.5] \times [0.5, 1], \\ 10^3, & (x, y) \in [0.5, 1] \times [0, 0.5], \\ 1, & \text{otherwise}, \end{cases}$$

with a Dirichlet boundary condition discretized with central differences.

A regular mesh was assumed in constructing the finite difference matrices for these PDEs.

We used GMRES with a stopping tolerance of 10^{-8} in relative residual and a cap on the number of iterations at 1000. The iteration counts are shown in Table 2.2.

MMF preconditioning is consistently better on model problems in terms of iteration count. Higher dimensional finite difference Laplacian matrices are generally well conditioned, as the condition number depends more strongly on the mesh size. In fact, the condition number of d-dimensional finite difference Laplacian matrix grows as $n^{\frac{2}{h}}$, where h is the step size. Even on higher dimensional Laplacians, where the wavelet preconditioners fail to provide adequate speedup, MMF preconditioning is effective. On average, MMF preconditioning seems to converge in about half the number of iterations as that required by the best wavelet preconditioner.

Table 2.2: **Iteration counts on model PDEs for various preconditioners**: Iteration counts of GMRES until convergence to a relative residual of 10^{-8}. Here n is the number of rows of the finite difference matrix. WSPAI refers to the wavelet sparse preconditioner of Chan, Tang and Wan [25] and IWSPAI to the implicit sparse preconditioner of Hawkins and Chen [70]. It is clear that MMF preconditioner is consistently better. $\times$ indicates that the desired tolerance was not reached within 1000 iterations.

Dataset	n	no prec.	WSPAI	IWSPAI	MMF prec.
1D Laplacian	256	256	46	13	**10**
	512	512	64	13	**10**
	1024	1001	93	17	**13**
	2048	1001	131	17	**2**
2D Laplacian	256	45	33	28	**8**
	1024	91	41	28	**8**
	4096	180	59	30	**13**
3D Laplacian	512	28	26	28	**8**
	4096	55	41	30	**11**
2D Disc	256	240	256	37	**13**
	1024	868	$\times$	24	**13**

We observed that choosing the correct dimensionality of the wavelet transform is necessary for IWSPAI to be effective. Without it, IWSPAI and WSPAI on higher order Laplacians resulted in nearly no speed-up compared to no preconditioning.

Off-the-shelf matrices

We collected all symmetric sparse matrices of size larger than 8192 from the University of Florida Sparse Matrix Collection [35]. The matrices come from a variety of scientific problems: structural engineering, theoretical/quantum chemistry, heat flow, 3D vision, finite element approximations, networks, etc. The right hand sides were generated by multiplying the coefficient matrices with a random vector.

Wavelet preconditioners cannot be applied on matrices whose size is not divisible by 2. Moreover, the number of levels at which wavelet transforms is limited by the multiplicity of 2 as a factor of the matrix size. MMF preconditioner, on the other hand, can be applied to

arbitrary size matrices.

First, we evaluate the effectiveness of MMF preconditioning (MMFprec). The iteration counts when using GMRES(30) and a tolerance of 10^{-9} are shown in Table C.2. Of the 493 matrices, GMRES converges with no preconditioning in 53 cases, while with MMFprec converges in 120 cases. MMFprec results in fewer iterations than no preconditioning in 110 cases. Fifteen of the 31 cases where MMFprec is not effective belong to the "Nemeth" group, which has arbitrary matrices and may not have multiresolution structure.

MMFprec's performance by matrix group is shown in Table 2.3. For each group, we counted the number of matrices on which MMF achieves faster convergence than no preconditioning. MMFprec is able to accelerate linear systems arising from a variety of scientific problems, especially ones which are known to have multiresolution structure such as graphs and circuit simulations. Moreover, circuit simulation problems are known to have hierarchical structure different from those arising in PDEs [154]. We also note that MMFprec is effective on a range of sparsity levels and matrix sizes.

Comparison with wavelet preconditioners

We selected several off-the-shelf matrices of sizes between 8192 and 65536. To make the matrix size compatible with wavelet preconditioners, we discarded a random set of rows/-columns from each matrix such that its size is reduced to $p2^s$, where $s = \lfloor \log_2 n \rfloor$ and $p = \lfloor n/2^s \rfloor$. We limited ourselves to an upper bound of 65536 as the wavelet preconditioner does not scale well on wall-clock time (ref. Section 2.3.4).

Another challenge in applying the wavelet preconditioner on off-the-shelf matrices is that oftentimes the underlying generative process of the matrices is unknown and the dimensionality of the wavelet transform to use cannot be determined. For our experiments, we choose a dimensionality of one.

The iteration counts are tabled in Table C.1. We used GMRES (no restarts) for this

Table 2.3: **Performance of MMF preconditioning by matrix group**: %wins indicates the percentage of times MMFprec resulted in lower GMRES (30) iteration count compared to no preconditioning. $\times$ indicates a win percentage of zero. Sparsity is defined as f, where $fn = \text{nnz}$ where n is the size of the matrix and nnz is the number of nonzeros.

Group name	#prob.	Avg. size	Avg. sparsity	%wins	Description
Rajat	5	19962	4.31	100.0	Circuit simulation
JGD_Trefethen	2	19999	27.72	100.0	Combinatorial problems
Oberwolfach	8	50436	25.43	100.0	Model reduction
Um	1	101492	16.23	100.0	Electromagnetics
Boeing	7	25455	1.0	100.0	Structural engineering
QY	1	14454	10.24	100.0	Power network
Botonakis	2	153237	6.97	100.0	Thermal problem
Rothberg	1	53570	21.91	100.0	Structural engineering
TKK	1	13681	52.21	100.0	Structural engineering
Andrews	1	60000	12.67	100.0	Computer vision
DIMACS10	21	135252	4.73	100.0	Graphs
Lourakis	1	10581	72.85	100.0	Computer vision
Mulvey	2	74752	7.99	100.0	Economics
Cunningham	2	38617	25.11	100.0	Acoustics
TSOPF	6	39249	155.61	100.0	Power network
HB	5	26659	45.16	100.0	Assorted
Chen	4	26364	75.89	100.0	Structural engineering
Gupta	2	39423	68.45	100.0	Graphs
Pothen	5	45646	4.9	100.0	Structural engineering
IPSO	1	15435	9.17	100.0	Power network
AG-Monien	7	25197	4.0	85.7	Graphs
GHS_indef	15	55174	5.46	80.0	Structural engineering
Andrianov	4	13730	26.35	75.0	Optimization
GHS_psdef	13	39094	5.44	69.2	Structural engineering
Bindel	2	10605	13.63	50.0	Thermal problem
Nemeth	25	9506	41.53	4.0	Newton-Schultz iteration
Norris	2	9702	8.88	$\times$	Bioengineering
Schenk_IBMNA	1	23948	8.46	$\times$	Optimization
UTEP	1	16129	15.69	$\times$	PDEs
Okunbor	1	8205	15.3	$\times$	Acoustics
MaxPlanck	2	81920	4.0	$\times$	CFD

experiment. IWSPAI and no preconditioning are effective on 13 and 10 out of the 43 cases respectively. MMF preconditioning is better on 23 out of the 43 cases.

IWSPAI is effective on matrices arising in CFD and Statistical/Mathematical problems, and no preconditioning is sufficient on structural problems. MMF preconditioning outperforms the rest on all the graph problems.

Increasing the wavelet transform level increases the accuracy of the wavelet preconditioners. In this case, Hawkins and Chen [70] remark that a few iterations of GMRES can be used in place of reduced QR factorization in Step 4 of algorithm 4 to alleviate the increased setup time. However, using GMRES defeats the purpose of maintaining higher accuracy with a higher wavelet transform level. Further, we noted empirically that using GMRES drastically reduces the efficacy of the implicit wavelet preconditioner - with most of the cases being non-convergent.

Performance when only an approximate solution is desired

For a model PDE problem, we plot the relative residual as a function of iteration counts in Figure 2.5. We see that the curve corresponding to the MMF preconditioner is below the curves for the other preconditioners. This means that an approximate solution can be determined quickly by the MMF preconditioner.

In Table C.3 we tabulate iteration counts for reaching different values of GMRES tolerance on off-the-shelf matrices. Even here MMF preconditioning is successful on more number of matrices for all values of the tolerance. The starkest difference in performance is found at tolerance 10^{-5}.

Wall-clock time comparison

In Table 2.4, we present the wall clock running times for linear solves with the different preconditioners on the model PDE problems. In terms of the total time for the linear solve

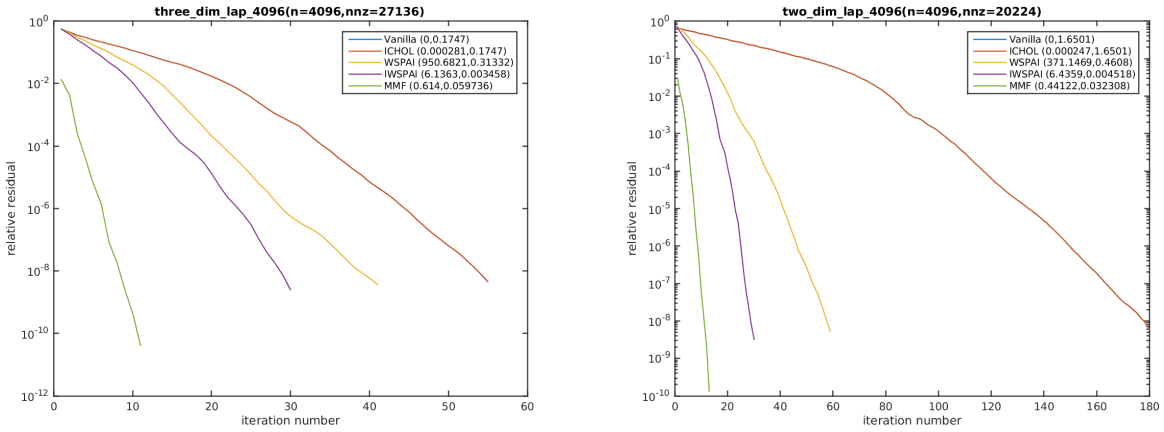

Figure 2.5: **Residual vs iteration in preconditioning**: Relative residual as a function of iteration number.

including preconditioner setup, MMF preconditioner is consistently better.

Table 2.4: **Timing results in preconditioning**: Wall clock running time of preconditioner setup and linear solve times in seconds. $\times$ indicates that the desired tolerance was not reached within 1000 iterations.

Dataset	n	no prec.	WSPAI		IWSPAI		MMF prec.	
		solve	setup	solve	setup	solve	setup	solve
1D Laplacian	256	0.3	0.77	0.01	0.8	**2e-05**	**0.01**	0.01
	512	1.35	1.70	0.03	1.73	**4.3e-05**	**0.03**	0.02
	1024	5.18	5.36	0.09	3.79	**8.2e-05**	**0.07**	0.02
	2048	7.80	24.2	0.24	9.9	**1.5e-04**	**0.15**	0.02
2D Laplacian	256	0.54	21	0.03	0.26	**3.4e-05**	**0.05**	0.04
	1024	0.08	3.87	0.03	0.32	**2.4e-04**	**0.10**	0.02
	4096	1.65	371	0.46	6.43	**4.5e-03**	**0.44**	0.03
3D Laplacian	512	0.01	0.16	0.01	0.16	**1.2e-05**	**0.04**	0.01
	4096	0.17	950	0.31	6.13	**3.4e-03**	**0.61**	0.05
2D Disc	256	0.23	0.18	0.30	0.20	**3.3e-05**	**0.01**	0.02
	1024	2.67	3.77	5.60	0.31	**2.9e-04**	**0.11**	0.03

On off-the-shelf matrices, the wall-clock time of the wavelet preconditioners is much worse. We took a large network matrix `SNAP/amazon0302` from the UFlorida repository and computed the wavelet and MMF preconditioners on submatrices of increasing size. The results are shown in Figure 2.6. For a matrix with just 80,000 nonzeros, the wavelet

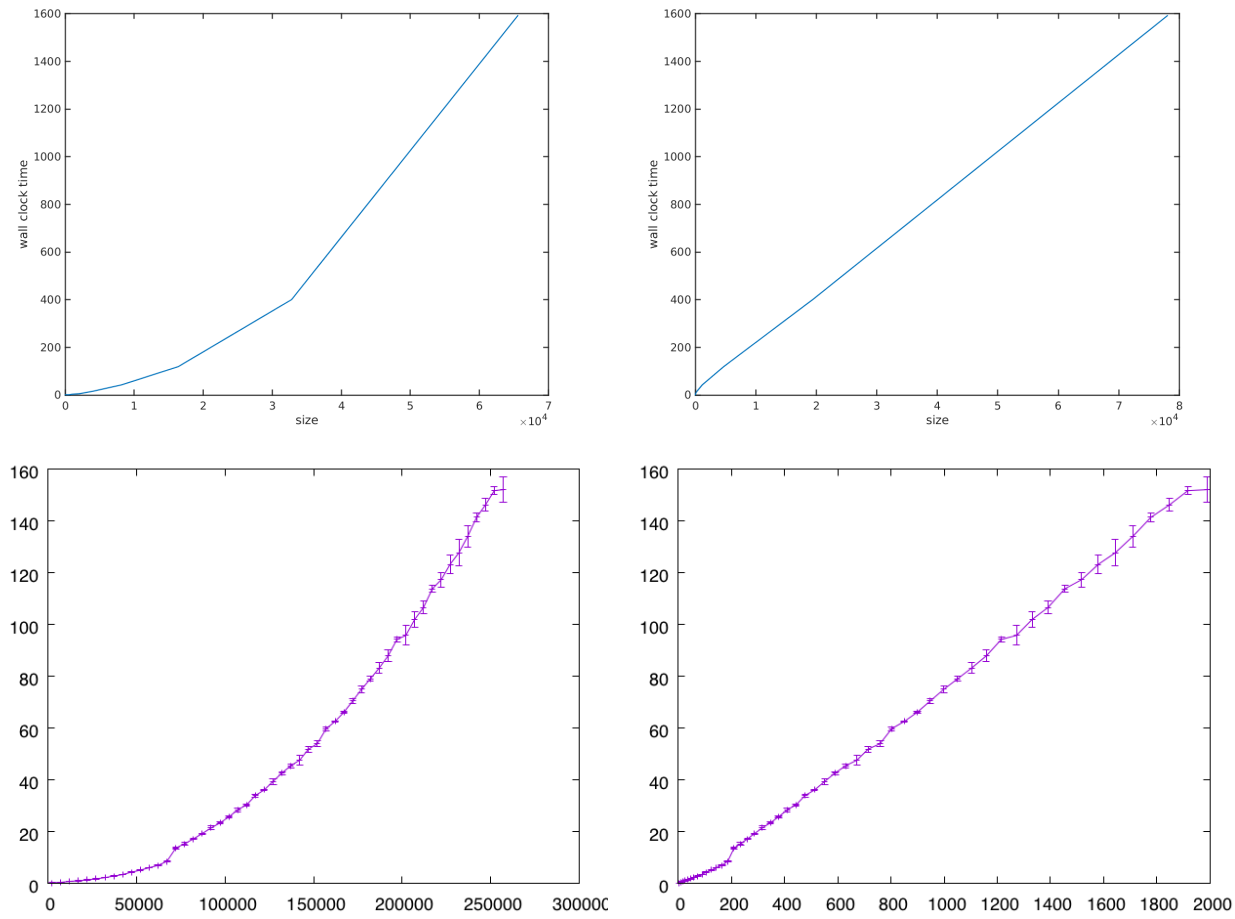

Figure 2.6: **Timing plots for preconditioning**: Wall-clock times as a function of matrix size (left) and number of nonzeros (right). Top row corresponds to IWSPAI [70] and bottom to MMF preconditioner.

preconditioner takes almost 20 minutes to setup, despite the parallelism in implementation.

Note that we used the most basic parameters while computing the MMF. With proper tuning, performance can be brought up, which would result in better performance. The other wavelet preconditioners have only one parameter, namely the level of the wavelet transform, which leaves little room for tuning.

2.3.5 Discussion

We observed that the MMFprec is, by and large, more effective than wavelet preconditioners. For a narrow set of problems, i.e., where the matrix size is a multiple of 2, and whose underlying process is known (e.g., low-order PDEs), the wavelet preconditioner is a reasonable choice of preconditioner when the matrix is not too big. We also empirically noted that wavelet preconditioner is more effective on CFD problems.

Our experiments also revealed that the wall-clock time of the wavelet preconditioner scales linear in number of nonzeros, which is the same as MMFprec. Perhaps a highly optimized implementation of the wavelet preconditioner would vastly expand its applicability. However, other limitations such as dependence on the underlying dimensionality of the problem and parity of matrix size still exist.

Then the question arises: when is MMF preconditioning useful? While we observed that MMF is effective on a range of problems, we also noted that is it especially applicable to graph problems and circuit simulations, which are known to have hierarchical structure. Further, when an approximate solution is desired, MMFprec is preferable to no preconditioning.

The "geometry free" nature of MMF preconditioner makes it more flexible than standard wavelet preconditioners. In particular, MMF can be applied to matrices of any size, not just $p2^s$. Furthermore, MMF preconditioning is completely invariant to the ordering of the rows/columns, in contrast to, for example, the multiresolution preconditioner of Bridson and Tang [22]. The adaptability of MMF makes it suitable to preconditioning a wide variety of linear systems.

In the next section, we expand the scope of MMF to general, asymmetric matrices.

2.4 Asymmetric Multiresolution Matrix Factorization

In Sections 2.2 and 2.3, we have seen how the Multiresolution Matrix Factorization can be powerful tool in estimating hierarchical structure in a wide variety of symmetric matrices.

Since originally proposed, MMF has also been used for symmetric-matrix compression [169], accelerating Gaussian Processes [42], exploring representations learned by deep neural networks, etc [77]. However, hierarchical structure may be present in data that is represented by asymmetric matrices, such as the directed graphs, or partial differential equations with variable coefficients (cref. examples in Section 7 of [70]).

The wavelet-based preconditioners we have seen in the previous section (Section 2.3, [25, 70]) essentially model hierarchical structure in the inverses of matrices using standard wavelet transforms. They show that under standard wavelet transforms, the inverses of sparse matrices are also sparse. This also means that the original matrices cannot be sparse under standard wavelet transforms.

MMF essentially constructs a custom wavelet-like basis for $\mathbb{R}^n$ given a symmetric matrix. However, MMF was originally only defined on symmetric matrices. To attack this problem, we explore two extensions to MMF for decomposing a given asymmetric matrix, inheriting many of its conceptual benefits.

2.4.1 MMF via an Additive Decomposition

Any square matrix $A \in \mathbb{R}^{n \times n}$ can be written as the sum of a symmetric matrix and a skew-symmetric matrix as follows

$$A = \underbrace{\frac{A + A^T}{2}}_{\text{symmetric}} + \underbrace{\frac{A - A^T}{2}}_{\text{skew-symmetric}}$$

We propose factorizing each of the two parts separately and writing A as a sum of two MMFs. In other words, we write

$$A \approx \left(P_1^T \dots P_L^T H_{\text{sym}} P_L \dots P_1\right) + \left(Q_1^T \dots Q_L^T H_{\text{skew}} Q_L \dots Q_1\right)$$

where

$$\frac{A + A^T}{2} \approx P_1^T \dots P_L^T H_{\text{sym}} P_L \dots P_1 \tag{2.14}$$

$$\frac{A - A^T}{2} \approx Q_1^T \dots Q_L^T H_{\text{skew}} Q_L \dots Q_1 \tag{2.15}$$

The factorization of $\frac{A+A^T}{2}$ can be determined via the usual MMF algorithms such as pMMF (Section 2.2). Thus, the problem of computing a multiresolution factorization of an asymmetric square matrix is now reduced to computing a multiresolution factorization for a skew-symmetric matrix.

MMF for skew-symmetric matrices

A matrix $A \in \mathbb{R}^{n \times n}$ is skew-symmetric if $A^T = -A$. Skew-symmetric matrices have complex eigenvalues and are therefore not diagonalizable in the real space. Thus, a skew-symmetric matrix can never be perfectly represented using the same form of MMF as for a symmetric matrix. However, it is known [129] that a skew-symmetric matrix, under a unitary transformation, assumes the so-called Murnaghan canonical form,

$$\begin{bmatrix} \Lambda_1 & 0 & \dots & 0 \\ 0 & \Lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \Lambda_L \end{bmatrix} \tag{2.16}$$

where $\Lambda_i, i = 1, \dots, L$ are 2×2 matrices of the form

$$\Lambda_i = \begin{pmatrix} 0 & \lambda_i \\ -\lambda_i & 0 \end{pmatrix}$$

where $\lambda_i \in \mathbb{R}$.

We propose a multiresolution factorization for skew-symmetric matrices identical to that for symmetric matrices (Section 2.1) except that the core-diagonal matrix is allowed to be in the following form:

$$H = \begin{bmatrix} H_{\text{core}} & 0 & \dots & \dots & 0 \\ 0 & \Lambda_1 & 0 & \dots & 0 \\ 0 & 0 & \Lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & \Lambda_L \end{bmatrix} \tag{2.17}$$

where $\Lambda_i, i = 1, \dots, L$ are 2×2 matrices as defined above.

This form preserves the skew-symmetricity of the matrix in its MMF, and allows for interaction between wavelets between adjacent levels. The computation is done similar to the GreedyJacobi algorithm from [95], except that in the last step, instead of zeroing out off-diagonal elements, we sparsify the matrix into the Murnaghan normal form as described above.

Remark. We note that this technique is designed to minimize the approximation error of an MMF-like factorization, as opposed to extending multiresolution analysis to skew-symmetric matrices. In fact, the notion of smoothness is rendered degenerate as $\sup_{v \in V \subseteq \mathbb{R}^n} \frac{\langle v, Av \rangle}{\langle v, v \rangle}$ is always zero when A is skew-symmetric. Our aim is to generate the least Frobenius norm error in approximating the skew-symmetric matrix while using only $O(n)$ rotations. Whether there exists a notion of multiresolution that can overcome the degeneracy of the smoothness for skew-symmetric matrices is an open question.

Remark. Note that

$$\sup_{v \in V \subseteq \mathbb{R}^n} \frac{\langle v, Av \rangle}{\langle v, v \rangle} = \sup_{v \in V \subseteq \mathbb{R}^n} \frac{\left\langle v, (A + A^T)v \right\rangle}{2 \langle v, v \rangle}$$

Thus, performing a multiresolution analysis (Figure 2.3) on $\mathbb{R}^n$ with respect to A is identical

as with $\frac{A+A^T}{2}$ which is a symmetric matrix. Suppose $U \in \mathbb{R}^{n\times n}$ represents the wavelet matrix from this multiresolution analysis. When A is symmetric, [95] defined the structure of U^TAU as core-diagonal. In future research, it may be worthwhile to understand what kind of structure makes sense for the asymmetric case.

2.4.2 MMF via Direct Factorization

In our second approach, we aim to perform multiresolution analysis using a direct extension of the methodology proposed in [95]. We aim to capture hierarchical structure directly in the matrix, as opposed to splitting it in an additive form. Recall that we would like to identify clusters of coordinates that a matrix A more closely couples than others, and determine an appropriate multiresolution analysis of $\mathbb{R}^n$. When the matrix is symmetric, the linear transformation that is represents is self-adjoint. In the asymmetric case, depending on whether we are transforming $x \to x^TA$ or $x \to Ax$, we may end up with different ways in which A might couple the coordinates.

Following the methodology in [95], we have two different ways of performing a multiresolution analysis of $\mathbb{R}^n$ (see Figure 2.3), one that is based on A, and another on A^T. Suppose $P \in \mathbb{R}^{n\times n}$ represents the wavelet transform in the former case, and $Q \in \mathbb{R}^{n\times n}$ of the latter. Thus, following a similiar methodology as [95] (Sec 3.2) in converting the multiresolution analysis into a matrix factorization, we have

$$H = P^TAQ$$

which is the matrix A in the bases P and Q. In the symmetric case, [95] argues that H can take a core-diagonal form, based on the fact that the eigenvalue decomposition converts a symmetric matrix to diagonal, and MMF uses orthonormal transformations that are designed to be as close to eigenvectors as possible. Using a similar argument, but with the singular value decomposition, it may be assumed that even in the asymmetric case H is core-diagonal.

However, this does not take into account interactions between wavelets at different levels, and we relax the structure to "core-sparse" as we define below.

Remark. When A represents the adjacency matrix of a directed graph, we are essentially identifying hierarchical cluster structures considering 1) only the outgoing edges, and 2) only the incoming edges. The matrix H gives a mapping between these two cluster structures.

We define asymmetric MMF to be a multiresolution factorization of the following form

$$A = P_1 P_2 \ldots, P_L H Q_L^T Q_{L-1}^T \ldots Q_1^T \tag{2.18}$$

, where

- Each $P_i, Q_i, i = 1, \ldots, L$ are highly sparse, orthogonal rotation matrices similar to the rotations in a symmetric MMF
- $H \in \mathbb{R}^{n \times n}$ is a core-sparse matrix, that is, under an appropriate permutation of row and columns, H would be a 2×2 block-diagonal matrix with the upper left block being dense, and the other 3 blocks would be highly sparse

When $P_i = Q_i$ and H is in core-diagonal form with a symmetric core, the above factorization would be a symmetric MMF.

We compute the asymmetric MMF using an approach similar to the symmetric MMF described in [95]. The algorithm we propose is designed to minimize the Frobenius norm error

$$A - P_1 P_2 \ldots, P_L H Q_L^T Q_{L-1}^T \ldots Q_1^T.$$

We determine the left and the right rotations greedily, that is, we compute a sequence of rotated versions of A

$$A \to P_1^T A Q_1 \to P_2^T P_1^T A Q_1 Q_2 \to \ldots \to P_L^T \ldots P_1^T A Q_1 \ldots Q_L.$$

Each of the rotations is computed by identifying sets of similar rows (for the left rotations) and columns (for the right rotations), and determining an appropriate k-point rotation. This requires us to compute gram matrices on both the rows as well as the columns. Finally, we sparsify the final rotated version of A to determine the core-sparse matrix H. We explore three ways of sparsification

1. CoreDiagonal: retain only the diagonal elements of H (provided $m = n$)
2. TopN: retain the top n elements in terms of magnitude
3. GreedyTopN: select n elements by magnitude greedily such that no two of the selected elements fall in the same row/column

The procedure is summarized in Algorithm 5.

The running time of this algorithm is $O(n^3)$ similar to GreedyJacobi proposed for the original symmetric MMF. Computation can be made faster by similarly extending the pMMF algorithm to maintain row and column gram matrices and computing separate left and right rotations (note that applying the left rotations to A does not affect the column gram matrix, and applying the right rotations does not affect the row gram matrix).

2.4.3 Numerical Results

We compare asymmetric MMF to the low-rank-based CUR decomposition [120] for the task of compressing square matrices. The CUR decomposition is a widely used alternative to the singular value decomposition to compress large matrices. To reduce a matrix $A \in \mathbb{R}^{n \times n}$ to rank-k, the method proceeds by carefully selecting k rows ($R \in \mathbb{R}^{n \times n}$) and columns ($C \in \mathbb{R}^{k \times n}$) and determines a matrix $U \in \mathbb{R}^{k \times k}$ such that the product CUR is as close to the best rank-k approximation of A as possible.

We sample 70 square matrices from the UFlorida sparse matrix repository. The matrices were sourced from various fields of science and engineering spanning circuit simulations,

Algorithm 5 Asymmetric MMF via direct factorization

Input: core size d, matrix $A_0 = A \in \mathbb{R}^{n \times n}$

Set of active rows $S_0^r \leftarrow [n]$

Set of active columns $S_0^c \leftarrow [n]$

for $\ell = 1$ **to** $n - d$ **do**

Select a random row i from the set S_ℓ^r

$G \leftarrow A_{S_\ell^r, S_\ell^c} A_{i, S_\ell^c}^T$

$j \leftarrow S_\ell^r[\arg\max(G)]$

$P_\ell \leftarrow$ Jacobi rotation on rows i, j

$A \leftarrow P_\ell^T A$

Remove $\arg\min_{t \in \{i,j\}} \left|\left| A_{t, S_\ell^c} \right|\right|_2$ from S_ℓ^r

Select a random column i' from the set S_ℓ^c

$G' \leftarrow A_{S_\ell^r, S_\ell^c}^T A_{S_\ell^r, i'}$

$j' \leftarrow S_\ell^c[\arg\max(G')]$

$Q_\ell \leftarrow$ Jacobi rotation on rows i', j'

$A \leftarrow Q_\ell$

Remove $\arg\min_{t \in \{i',j'\}} \left|\left| A_{S_\ell^r, t} \right|\right|_2$ from S_ℓ^c

end for

$H = \textsc{CoreDiagonal}(A)$ or $\textsc{TopN}(A)$ or $\textsc{GreedyTopN}(A)$

return $P_1, \ldots, P_{n-d}, H, Q_1, \ldots, Q_{n-d}$

social networks, chemical processes, 2D/3D problems, etc. All the sampled matrices were chosen such that the numerical symmetry was less than 25%, and a maximum dimensionality of $100,000$.

Results on MMF via an additive decomposition

Table 2.5 shows the percentage of times the asymmetric MMF had a better compression error over CUR over 65 matrices. As both the CUR and MMF algorithms are randomized, we report results averaged over 3 trials. CUR performs better than MMF in the majority of the cases.

Table 2.5: **Asymmetric-MMF-via-additive-decomposition results**: Percentage wins of asymmetric MMF (additive) over CUR in compressing 74 different asymmetric matrices to various compression ratios, as measured by Frobenius norm compression error.

Matrix kind	num. matrices	1%	10%	25%	50%	75%
problems with underlying 2D/3D geometry						
structural problem	4	50.0	50.0	50.0	0.0	0.0
computational fluid dynamics problem	3	0.0	0.0	0.0	0.0	0.0
2D/3D problem	5	80.0	80.0	60.0	60.0	60.0
materials problem	2	0.0	0.0	0.0	0.0	0.0
problems with no underlying geometry						
economic problem	11	18.2	9.1	9.1	9.1	9.1
directed graph	19	5.3	5.3	5.3	0.0	0.0
power network problem	3	0.0	0.0	0.0	0.0	0.0
circuit simulation problem	2	0.0	0.0	0.0	0.0	0.0
chemical process simulation problem	23	0.0	0.0	8.7	0.0	0.0
statistical/mathematical problem	1	100.0	0.0	0.0	0.0	0.0
counter-example problem	1	0.0	0.0	0.0	0.0	0.0
total wins:		13.5	10.8	12.2	5.4	5.4

Results on MMF via direct factorization

We report results on two variants of the directly factorized MMF based on what elements are retained in the core-sparse matrix H. In the first case, we retain the compressed $k \times k$

dense core, and $n - k$ elements chosen greedily according to magnitude and made sure that no two of these $n - k$ elements fall in the same row or column. Table 2.6 shows the percentage times this version of the asymmetric MMF wins over CUR in compression error. We observe that this version of MMF is significantly better than the additive version, and

Table 2.6: **Asymmetric-MMF-(GreedyTopN) results**: Percentage wins of asymmetric MMF (GreedyTopN) over CUR in compressing 63 different asymmetric matrices to various compression ratios, as measured by Frobenius norm compression error.

Matrix kind	num. matrices	1%	10%	25%	50%	75%
problems with underlying 2D/3D geometry						
structural problem	4	50.0	50.0	25.0	0.0	0.0
materials problem	1	0.0	0.0	0.0	0.0	0.0
2D/3D problem	4	100.0	75.0	75.0	75.0	0.0
computational fluid dynamics problem	3	100.0	100.0	0.0	0.0	0.0
problems with no underlying geometry						
statistical/mathematical problem	1	100.0	100.0	100.0	0.0	0.0
economic problem	9	77.8	22.2	11.1	33.3	11.1
power network problem	2	100.0	0.0	50.0	0.0	0.0
directed graph	15	80	20	6.7	0.0	0.0
chemical process simulation problem	21	80.9	23.8	14.3	0.0	0.0
circuit simulation problem	2	100.0	0.0	0.0	0.0	0.0
counter-example problem	1	0.0	0.0	0.0	0.0	0.0
total wins:		79.4	30.2	17.5	9.5	1.6

is able to capture structure in both problems with underlying known geometry and without underlying geometry.

Relaxing the structure of the matrix H further, we report the results for simply storing the top $n - k$ elements in the core-sparse matrix in Table 2.7. Here, we see that MMF is able to capture structure and compress nearly all kinds of matrices at all compression levels.

From the numerical results, it seems that allowing interaction between wavelet levels is critical to achieving good approximation errors. This indicates that hierarchical structure in asymmetric matrices is more complex than that uncovered by simply extending the symmetric MMF analogous to an SVD.

Table 2.7: **Asymmetric-MMF-(TopN)**: Percentage wins of asymmetric MMF (TopN) over CUR in compressing 69 different asymmetric matrices to various compression ratios, as measured by Frobenius norm compression error.

Matrix kind	num. matrices	1%	10%	25%	50%	75%
problems with underlying 2D/3D geometry						
structural problem	4	100.0	100.0	100.0	50.0	50.0
materials problem	1	100.0	100.0	100.0	100.0	100.0
2D/3D problem	5	100.0	100.0	100.0	80.0	60.0
computational fluid dynamics problem	3	100.0	100.0	100.0	0.0	0.0
problems with no underlying geometry						
directed graph	18	100.0	50.0	22.3	16.7	0.0
chemical process simulation problem	21	100.0	90.5	67.7	57.1	38.1
circuit simulation problem	2	100.0	100.0	100.0	50.0	0.0
power network problem	3	100.0	100.0	100.0	66.7	33.3
counter-example problem	1	100.0	100.0	100.0	100.0	100.0
economic problem	10	100.0	100.0	100.0	70.0	50.0
statistical/mathematical problem	1	100.0	100.0	100.0	100.0	0.0
total wins:		100.0	84.1	69.6	49.3	29.0

For smaller compression ratios, CUR seems to be preferable for some types of problems such as directed graphs. It is possible that some matrices may have global-scale behavior that is superposed over the hierarchical structure. This is possible, for instance, when the data is noisy. In the next part, we explore ways of combining MMF and low-rank approaches to achieve better compression errors.

Matrix compression via a combination of CUR and MMF

Consider Section 2.4.3 that shows the error of MMF approximation of a matrix A as a function of the decay rate of its singular values. We see that when the singular values decay faster than linear, MMF does not approximate the matrix well. In real-world matrices, the decay of singular values may not be uniform and may fluctuate to very high and very low values. An example of such a spectrum is shown in Figure 2.8. Although there are segments where the decay rate is sublinear, the large decay rate around singular value 1500

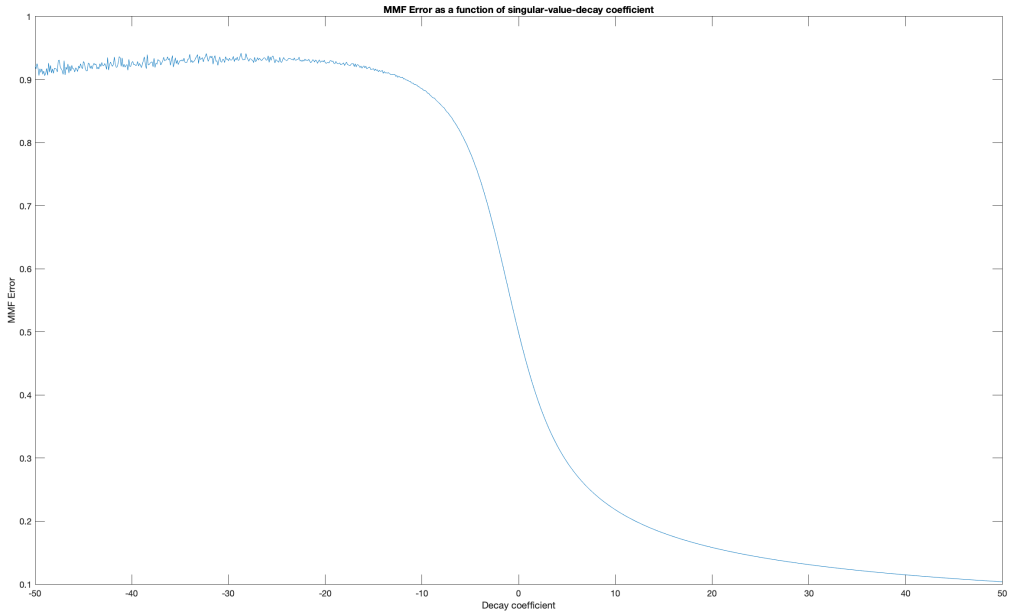

Figure 2.7: **MMF error vs rank**: Approximation error of symmetric MMF with different rates of decay in the eigenvalues. The 200×200 random matrices were generated by computing QDQ^T, where $Q \in \mathbb{R}^{200\times 200}$ is a fixed, randomly generated orthogonal matrix. D is a diagonal matrix whose entries are given by $\frac{1-e^{t(x-1)}}{1-e^t}$, where t is the decay coefficient and x takes 200 uniformly spaced values between 0 and 1. The X-axis corresponds to t.

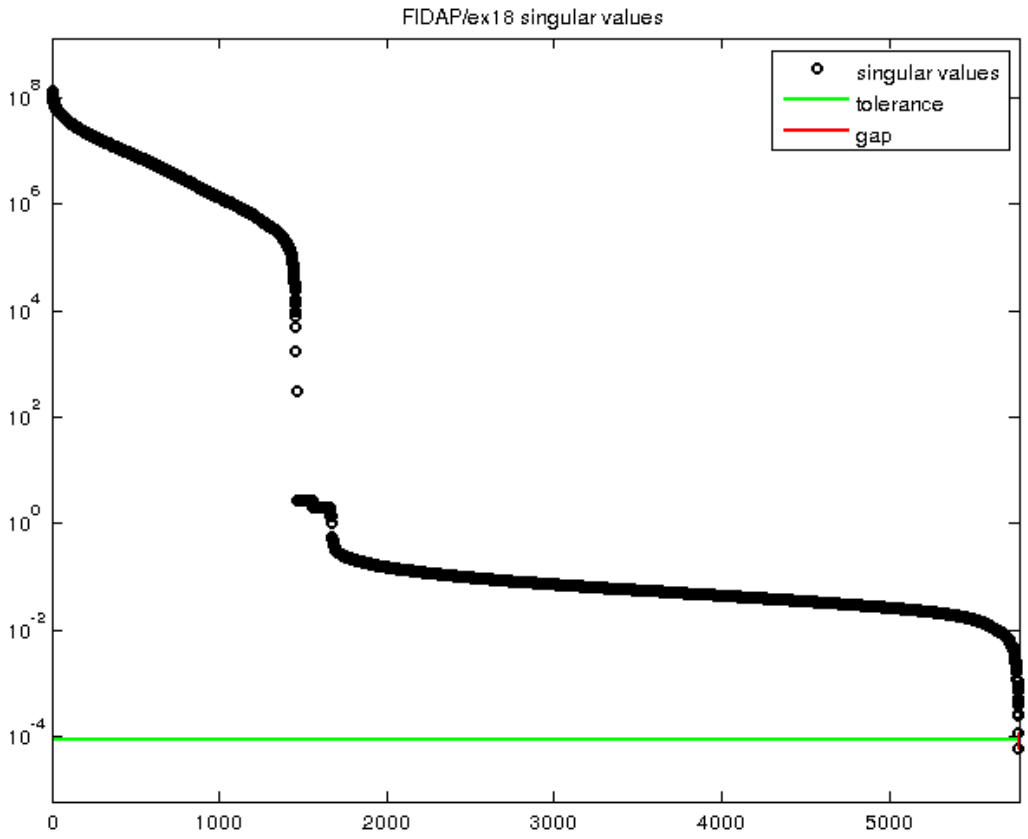

Figure 2.8: **Singular values of ex18 matrix**: Plot showing the singular values of the 'ex18' matrix from the UFlorida sparse matrix repository in descending order. Although the matrix is full-rank, the sharp drop in the singular values around index 1500 imparts a low-rank nature to the matrix. (Note that the y-axis is on log-scale). Image sourced from UFlorida sparse matrix repository [35]

may significantly impact the effectiveness of MMF on this matrix.

Thus, it may be beneficial to first compress the matrix using low-rank methods, and then use MMF on the residual part. The technique is described in Algorithm 6. For various values of k, we report the errors for compression various matrices to 5% of their original size in Figure 2.9.

Algorithm 6 Matrix compression using a combination of low-rank and multiresolution methods

Input: Square matrix A, compression size k, and r
[C,U,R] = CUR(A, r)
[P, H, Q] = MMF(CUR, k)
return $\left|\left|PHQ^T\right|\right|_F / \left|\left|A\right|\right|_F$

In most of the matrices, we observe that there exists at least one rank to which the matrix can be first compressed using CUR before compressing with MMF, that results in lower compression errors than each of CUR and MMF methods individually. In practice, there is no method to determine what this rank would be for a given matrix, however, it is an interesting and worthwhile problem to be able to remove a small global-scale component of the matrix, after which the hierarchical structure may be easily extracted.

2.5 Discussion

We have seen how multiresolution factorizations can be powerful tools in exploiting hierarchical structure in matrices for various applications. In addition to the results presented in this chapter, symmetric MMF has been used in accelerating Gaussian Processes [42] as well as compressing symmetric matrices [169]. On matrices where the underlying geometry is unknown, we have seen MMF successfully extract structure. However, hierarchical structure may be present even beyond obvious examples such as graphs. It can be understood as a complementary approach to understanding matrices based on rank. Given a generic matrix, it is an open question whether a low-rank based method would be more appropriate or a

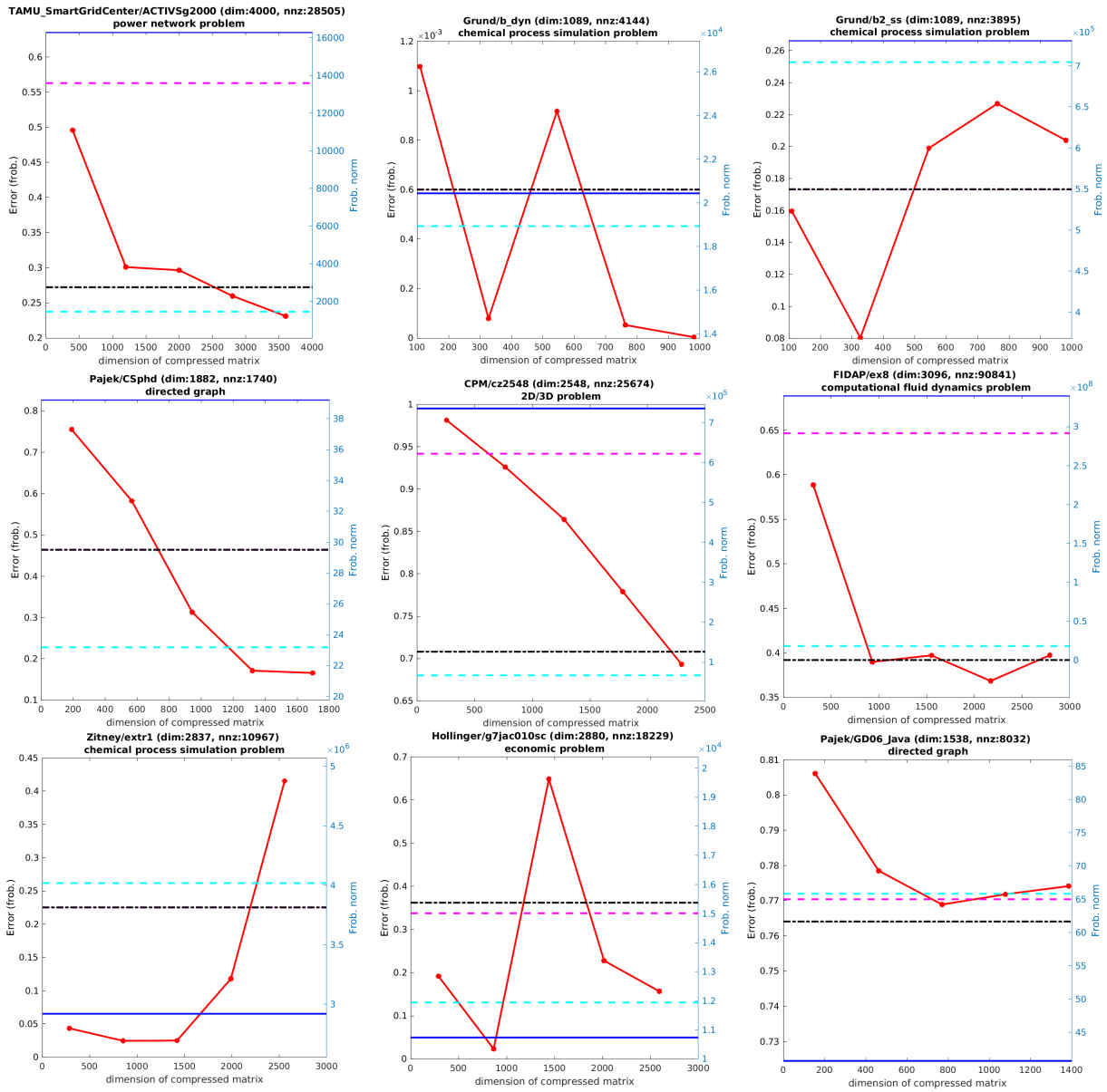

Figure 2.9: **Compression results when combining MMF and CUR**: Frobenius norm error (red in the plots) as a function of CUR compression rank for compressing the matrix to 5% of its original size using the combined low-rank + multiresolution method. The light blue dotted line is the error when compressing using MMF to 5%, while the blue solid line is the error with only using CUR to compress to 5%.

multiresolution-based one. It is also interesting to understand what kind of matrices multiresolution methods are better suited for, and also to be able to derive bounds on the error of MMF.

CHAPTER 3

ANALYZING DEEP NEURAL NETWORK MODELS

Many tasks that were earlier solved using rule-based, hand-tuned or statistical-ML systems are now increasingly being solved by Deep Neural Networks (DNNs). These models are the state of the art for many tasks of interest, especially in natural-language processing [178, 132, 38] and computer vision [117, 74, 133]. Due to the increasing preference for DNN models, there is a lot of human effort being invested in developing novel neural-network architectures and applying them for various tasks.

As systems based on DNNs are deployed in the real world, it is important to assess whether these systems generalize to usage scenarios that do not match the data on which the models were trained. The canonical way to do the assessment is to evaluate the models' performance on a test dataset that is designed to be representative of the real world [68], and is typically the method used in measuring the quality of a DNN model. However, as tasks solved by DNNs become more general and datasets bigger, it becomes impractical to manually verify if a given test set is sufficiently representative of the real world.

It has been repeatedly shown that DNN models are prone to adversarial attacks – inputs designed to force the model into making a wrong prediction. In computer vision, it has been shown that one can add human-imperceptible noise to input pixels of images and force state-of-the-art models to produce a wrong prediction for the original image [58]. In natural-language processing, it has been shown that neural networks performing reading comprehension can be fooled by adding adversarial sentences to the paragraph [82], or that classification and translation models are fooled by typos, misspellings and changes in punctuation [48, 13]. Existence of such attacks shows that test accuracy is not a good metric of model quality. While this is a hindrance for safe and reliable deployment of deep neural networks in mission-critical scenarios, it is also troublesome for researchers as they cannot rely on test accuracy as a good metric to measure progress in designing deep neural network

models.

Thus, it is worth understanding how DNN-based systems work, and develop techniques and tools to identify their failure modes, debug them and perhaps even compare them to equivalent rule-based models. However, DNNs are inherently black-box. The logic employed by the networks in making predictions is implicit in the parameters of the network; a single neuron or layer in the neural network does not indicate a meaningful abstraction of the logic, rather, predictions are made by neurons acting together – referred to as distributed representation.

Recently, there have been a number of techniques [108, 164, 158] that explain a deep neural network's predictions in terms of its input features – this is called attribution. These techniques rely on input perturbations and have been actively used in the computer vision community to understand input pixels relevant to models' predictions. As a result, there is recent development in developing adversarially robust models for computer vision tasks. However, in natural-language processing, there is no clear analogue, as there is no clear concept of "perturbing words" in the same way one can perturb pixel intensities. To a large extent, developers in NLP use the weights of the attention vector [9] to explain a neural network's predictions and extract rules learned by the neural network. While [82] do construct adversarial examples, their method is independent of the model and may not always result in exposing the weaknesses of a given model.

In the following sections, we utilize an attribution method called Integrated Gradients (IG) [164] to understand question-answering deep neural networks, extract rules and construct effective adversarial examples. In Section 3.2.2, we provide a comprehensive case-study of a question-answering neural network, Neural Programmer [132], wherein we systematically identify its weaknesses. Based on insights drawn from the analysis, we propose adversarial attacks in Section 3.4 that generalize to question- answering networks beyond Neural Programmer. We propose an extension to IG that takes into account the influence of words

via various dataflow paths in a neural network. We use this method to extract rules from neural networks (Section 3.3). Our contributions show that when model weaknesses exist, attributions can surface them and aid the developer in measuring the quality of the model and guide them towards making improvements.

3.1 Preliminaries: Integrated Gradients

DNNs are typically described using directed dataflow graphs [1], where the nodes (also called neurons) are primitive operators and the edges represent dataflow between operators. The values flowing along edges are tensors—arbitrary dimension arrays. Each operator applies to the tensors flowing on its in-edges and sends the result on all its out-edges. There are special nodes, called placeholders, through which inputs can be fed.

We study the influence of input variables (placeholders) on the output variables via attributions.

Definition (**Attribution**)**.** Formally, suppose we have a function $F : R^n \to [0, 1]$ that represents a deep network, and an input $x = (x_1, \ldots, x_n) \in R^n$. An attribution of the prediction at input x relative to a baseline input x' is a vector $A_F(x, x') = (a_1, \ldots, a_n) \in R^n$ where a_i is the contribution of x_i to the prediction $F(x)$.

In the definition above, one can think of F as the probability or a score of a specific output. The variables $(x_1, \ldots, x_n)$ are the inputs. The attributions $(a_1, \ldots, a_n)$ are the influences/blame-assignments to the variables $(x_1, \ldots, x_n)$ on the score F. Finally, notice that the attributions are relative to a reference input called the baseline. The baseline is a special, uninformative input, for instance the empty question. We discuss the precise choice of the baseline in Section 3.2.2. In our application of IG, the choice of baseline will be an important analysis knob; we will use it to control the type of explanation we desire.

We now give a brief description and some intuition for the IG Method. Intuitively, as we interpolate between the (uninformative) baseline and the true input, the score moves along

a trajectory, moving from uncertainty to certainty (the final probability/score). At each point along this trajectory, one can use the gradient of the function F with respect to the input to attribute the change in score back to the input variables. IG simply aggregates the gradients of the score with respect to the input along this trajectory. Specifically, Integrated Gradients are defined as the path integral of the gradients along the straightline path from the baseline x' to the input x.

Definition (**Integrated Gradients**)**.** Given an input x and baseline x', the integrated gradient along the i^{th} dimension is defined as follows.

$$\mathsf{IntegratedGrads}_i(x, x') ::= (x_i - x'_i) \times \int_{\alpha=0}^{1} \frac{\partial F(x'+\alpha\times(x-x'))}{\partial x_i}\, d\alpha$$

(here $\frac{\partial F(x)}{\partial x_i}$ is the gradient of F along the i^{th} dimension at x).

In [164], IG is justified by an axiomatic result. That is, it is essentially the unique method that satisfies certain desirable properties of an attribution method. We will rely on an empirical evaluation (Section 3.2.3) to justify our results so the axioms do not play a critical role in the presentation of the results in this chapter. For completeness, we informally mention some of the properties that it satisfies; see [164] for formal definitions and the axiomatic result.

IG satisfies the condition that the attributions sum to the difference between the score at the input and the score at the baseline. We call a variable uninfluential if all else fixed, varying it does not change the output score. IG satisfies the property that uninfluential variables do not get any attribution. Conversely, influential variables always get some attribution. IG satisfies the condition that symmetric variables get equal attributions.

3.2 Case Study: Neural Programmer

We begin with an analysis of Neural Programmer [132, 131] (NP)—it is a DNN trained to answer natural language questions on tables. The task is as follows: given a table t and a question x about the table, output a list of values y that answers the question according to the table. The only restriction on the question x is that a person must be able to answer it using just the table t. The training set for the question-answering system consists of tuples $(x_i, t_i, y_i)_{i=1}^N$ of questions, tables and answers.

It is like a program synthesis task where the specification is the question and a table, and the program to be synthesized is a logical form that answers the question. Specifically, the model takes as input a table ("medal tallies of various nations") and a question ("Who won more gold medals, France or Germany?"), and produces an answer ("France"). The model is trained on ⟨question, table, answer⟩ triples crowdsourced from Wikipedia [139]. It does not receive any other supervision. Yet, the model achieves accuracy that is close to the state of the art [131]. The model consists of a recurrent neural network, that at each step (probabilistically) picks a SQL-like operator (select rows, sort a column etc.), a column, and some rows to operate on. These operators are sequentially composed to (probabilistically) obtain the final answer.

What signals does NP rely on in the process of computing the answer? For instance, which question words are used to trigger which operators? How are columns selected? How are rows selected? Can we extract rules from NP that we could use in a hand-authored system? For instance, can we augment a hand-authored grammar with trigger words from NP? Further, how robust is all this reasoning? Are irrelevant words picked up as triggers?

To answer these questions, use Integrated Gradients [164] (IG) to understand the input-output behavior of this network.

```
def NPInference(qw, cnames, cm, tm, tbl):
# Question encoding
enc_hidden_states, enc_final_state = Encode(qw)

# Initialization
dec_state = initializeDecoderState()
op_selections = [None, None, None, None]
col_selections = [None, None, None, None]

# Operator and column selection phase
for k in range(4):
   # Operator selection
   ques_context = F1(dec_state, enc_hidden_states)
   op_controller = F2(ques_context, dec_state, enc_final_state)
   op_logits = F3(op_controller)
   op_probs = softmax(op_logits)
   op_selection[k] = argmax(op_probs)

   # Column selection
   col_contoller = F4(ques_context, dec_state, enc_final_state,
      op_probs[i])
   col_logits_soft = F5(col_controller, cnames)
   col_logits_tm = F6(col_controller, tm)
   col_logits_cm = F7(col_match)
   col_logits = col_logits_soft + col_logits_tm + col_logits_cm
   col_probs = softmax(col_logits)
   col_selection[k] = argmax(col_probs)

   # Update Controller RNN
   dec_input = DecoderInp(op_probs, col_probs, cnames)
   dec_state = DecoderStep(decoder_state, dec_input)

# Answer computation phase
ans = ComputeAns(tbl, op_selections, col_selections)
return ans
```

Listing 3.1: An abstraction of NP during inference

3.2.1 Abstracting Neural Programmer

Our objective is to study the input-output behavior of Neural Programmer. We are specifically interested in its behavior during inference, and therefore ignore details about its training process and parameters. We begin by carefully studying the various inputs and outputs of the network.

At a high level, the network operates on a question and a table, and produces an answer. There are two phases of operation, one continuous and one discrete. In the first phase, called operator and column selection, the network predicts a sequence of pairs of operators and table columns. In the second phase, called answer computation, the selected operators are discretely applied to a column of the table to obtain the final answer.

The inputs (question and table) feed into the two phases in very different ways. We identified these differences by carefully tracing data dependencies in the network. This tracing task is very similar to program slicing, a traditional program debugging technique, where given a variable of interest at a certain point in the program (in our case the final answer, or the intermediate operator and column selections), one wants to identify the minimal slice of the program that affects the variable (in our case a sub-network).

We abstract the architecture of NP and express it in the form of a program (written in Python) as shown in Listing 3.1. It describes how the input tensors feed into the operator and column selection phase, and the answer computation phase. The methods `F1`, ..., `F7`, `Encode`, `DecoderInp`, `DecoderStep`, and `ComputeAns` are purposefully left undefined as we choose to view those parts of the network as black boxes. The programmatic representation is mainly for ease of presentation; one could also describe the abstraction as a dataflow graph with the methods as operator nodes.

We now discuss the two phases in the context of this abstraction.

Operator and column selection. At the core of NP is a recurrent neural network called the "controller", which, in each time step, predicts a probability distribution over a (fixed)

set of primitive operators (Table 3.1), and another distribution over the table columns. The controller runs for exactly four time steps. The for-loop (lines 11—32) in the abstraction corresponds to the steps of the controller, and the variables `op_probs` and `col_probs` hold the probability distribution at each step. During inference, the operator and column with the highest probability are selected from these distributions; see `op_selection` and `col_selection`. Next, we discuss the inputs that matter to operator and column selection.

The first input is the question, which is tokenized and fed as a tensor of words $qw = \langle \mathbf{w}_1, \ldots, \mathbf{w}_Q \rangle$; dummy tokens are optionally appended to ensure that this tensor is of fixed shape $\langle Q \rangle$[1].

A table with shape $\langle m, n \rangle$ is fed in three parts via explicit featurization:

(1) column names (*cnames*), which are fed as a tensor of tokenized column names of shape $\langle n, C \rangle$ where n is the number of columns and C is the number of words in each column name; dummy tokens are optionally appended so that this tensor has a fixed shape.

(2) table-matches (*tm*): a boolean tensor of shape $\langle m, n \rangle$ with a 1 at position $\langle i, j \rangle$ whenever the corresponding table entry shares a word with the question.

(3) column-matches (*cm*): a boolean tensor of shape $\langle n \rangle$ with a 1 at the position i whenever the corresponding column name shares a word with the question.

Notice that the contents of the table (input variable `tbl`) are not used in the operator and column selection.

Instead, it uses the table-matches input (*tm*). This indicates that the network does not directly learn matches between the question and table; it only learns to prioritize a given set of matches.

The question-words tensor is extended with a special token tm_token whenever there is a table-match (i.e., $tm \neq 0$), and with the token cm_token whenever there is a column-match (i.e., $cm \neq 0$). This presumably makes the operator and column selection logic aware of the

1. The network has a fixed vocabulary of words, defined during training, and any question word outside this vocabulary is replaced with a special "unk" (meaning unknown) token.

Table 3.1: **Operators used by Neural Programmer**: NP uses a set of 15 operators to determine the logical form to execute on the table to produce the answer. Note that some of the operators such as first, last do not need to be associated with a particular column.

Operator	Description
count	report the number of selected rows
prev	rows above the ones selected
next	rows below the ones selected
first	first row among the ones selected
last	last row among the ones selected
mfe	rows with highest frequency value
gt	rows larger than given value
lt	rows smaller than given value
geq	gt with equality
leq	lt with equality
max	rows with largest value among selection
min	rows with smallest value among selection
select	rows whose entries appear in the question
reset	reset row selection (select all)
print	report the values in selected rows

presence of matches.

The column-matches (cm) input serves a slightly different purpose from the table-matches input. Notice, on line 24, that the final column logit distribution is obtained from adding three separate logit distributions: `col_logits_soft`, `col_logits_tm`, and `col_logits_cm`. While the first two distributions are affected by the question words, the last distribution (`col_logits_cm`) is simply a function of the column-matches (line 23). Essentially, the column-matches are used to upweight the logit distribution towards columns that have a match with the question. Consequently, we refer to `col_logits_cm` as hard logic, and `col_logits_soft`, `col_logits_tm` as soft logic.

We use O_k and C_k to refer to the functions determining the probability of the operator and column selected at time step k during inference. These functions have the signature: $\langle cnames, qw, tm, cm\rangle \rightarrow [0, 1]$.

Answer computation. In answer computation (line 35), the selected operators are eval-

uated on the selected columns of the table. Unlike the previous phase, this phase takes the entire table (input variable `tbl`) as input. Each operator (op_k) takes the selected column (col_k) along with a selection of rows and produces a selection of rows for the next step. The final operator is a print or count which either prints or counts the table entries along the selected rows and columns. Notice that answer computation is completely discrete, and does not involve a learnt component.

As an example, consider the question "who came right after Turkey?". It results in the operator sequence [select, next, last, print] and column sequence [nation, nation, total, nation]. Consequently, the answer computation involves selecting all rows that have a match with the question along the column "nation", shifting this row selection by one, selecting the last row amongst these, and printing the entry along the column "nation". Notice that the application of the operators next and last does not involve a column, and are thus column-independent operators. Other such operators include prev, first, reset, count. The operator reset always selects all rows of the table.

Differentiability of O_k and C_k. The first step of the network is to embed the question-words (qw) and column-names ($cnames$) into high-dimensional vectors. (This is quite standard for text models.) From this point onwards the network is fully differentiable. Henceforth, we abuse notation and use qw to refer to the embedded question words tensor with shape $\langle Q, d\rangle$, and $cnames$ to refer to the embedded column names tensors with shape $\langle n, C, d\rangle$; here d is the number of embedding dimensions. This ensures that the functions O_k and C_k are differentiable, and thus amenable to the IG method.

3.2.2 Understanding Neural Programmer

We wish to understand the behavior of NP during inference on a specific input (question and table). We do this by attributing operator (O_k), and column selections (C_k) made at each time step to the corresponding inputs.

Imposing a curried structure. So far we have treated column names (*cnames*), question words (*qw*), table-matches (*tm*), and column-matches (*cm*) as parallel inputs to the network. For analysis convenience, we would like to focus our effort on studying the influence of question words rather than the influence of the table. We expect operator selections to be triggered by question words, and for column selections, the influence of column name is relatively obvious. We henceforth think of the operator and column selections as curried functions that first take the table-only input, i.e. column names, and return a function that then takes the question-specific inputs (*qw*, *tm*, *cm*). Thus, one could express the signature of O_k and C_k as

$$cnames \rightarrow \langle qw, tm, cm \rangle \rightarrow [0, 1]$$

This splits our analysis into two phases. In the first, minor phase, we understand the effect of the table and in the second, major phase, we understand the effect of the question words (relative to a given table).

Understanding Table Influence

The most natural way of studying the influence of the table is to invoke NP on the table and an empty question. We call this the empty question input, and is defined as:

$$qw = [\mathsf{E}_{dummy}]^Q, tm = [0]^{m,n}, cm = [0]^n \tag{3.1}$$

where E_{dummy} is the embedding of the dummy token. The table-matches (*tm*) and column-matches (*cm*) are zero as the empty question has no matches with the table.

Default programs. For a given table, we expected the empty question to result in a uniform probability distributions across operators and columns at each time step. Instead, we found that for each table, the distributions are very skewed. We call the program obtained by picking the operators and columns with the highest probability at each time step, as the

default program for the table.

The default programs indicate that the column names alone influence the network to prefer certain operators and columns. We view this as a table-specific bias in the network. On further investigation, we found that the bias is more fundamental. Even if we set the input column names to empty strings, the network still has a preference for the operator selection [reset, prev, max, print] with probabilities [0.41, 0.37, 0.50, 0.97] respectively.

Table 3.2 lists various table-specific default programs along with the number of tables to which they apply. For each default program, we used the IG method to attribute operator and column selections to column names and show the 10 most frequently occurring ones across tables. NP uses reset, prev to exclude the last row from answer computation. Notice that the default program corresponding to [reset, prev, max, print] has attributions from column names such as "rank", "gold", "silver", "bronze", "nation", "year", which is indicative of tables with medal tallies which usually have a total row. The bias for such tables is to perform a max, which can be hurtful when questions do not ask for it. Another instance of table-specific bias is the operator sequence [reset, prev, last, print] whose column name attributions indicate tables pertaining to election results.

We speculate that such a bias exists in the network as it is ultimately beneficial during prediction. Perhaps when the question does not contain a trigger word, the network decides to fallback to the default program for the table. We verified this by supplying a question with out-of-vocabulary words to the network and confirming that the predicted program is the same as the default program. Additionally, for approximately 6% of questions in the dev dataset we found that the final predicted program is the same as the default program.

Understanding Question Influence

We apply the IG method to attribute the operator (O_k), and column selections (C_k) at each time step to each of the question words (qw), table-matches (tm), and column-matches

Table 3.2: **Table-specific default programs**: Column attributions in table-specific default programs, by operator sequence

Operator selections	Num. tables	Attributions to *cnames*
reset, reset, max, print	108	*unk*, year, date, name, points, position, competition, notes, team, no
reset, prev, max, print	67	*unk*, rank, total, gold, silver, bronze, nation, year, name, no
reset, reset, first, print	29	*unk*, name, notes, year, nationality, rank, date, location, previous, comments
reset, mfe, first, print	26	year, date, *unk*, notes, title, role, genre, opponent, score, surface
reset, reset, min, print	16	year, *unk*, name, height, location, jan, may, jun, notes, floors
reset, mfe, max, print	14	opponent, date, result, site, rank, year, attendance, location, notes, city
reset, next, first, print	10	*unk*, name, edition, year, death, time, type, men, birth, women
reset, reset, last, print	10	*unk*, year, date, location, album, winner, score, type, opponent, peak
reset, next, max, print	8	*unk*, ethnicity
reset, prev, first, print	7	name, intersecting, athlete, kilometers, location, rank, time, nationality, notes, serials
reset, mfe, last, print	6	*unk*, season, place, division, tier, date, cylinders, name, country, notes
reset, prev, last, print	5	date, votes, candidate, party, season, report, *unk*, city, west, east
reset, prev, min, print	4	movements, section, division, season, level, position, founded, gauge, year, enrollment
reset, select, last, print	3	finish, retired, start, car, laps, rank, qual, year, led, coordinates
reset, reset, first, count	2	owner, *unk*, programming, network
reset, mfe, select, print	1	length, *unk*, performer, producer, title
reset, next, last, print	1	comment, *unk*
reset, reset, last, count	1	year, *unk*, type, japanese, english
reset, reset, reset, print	1	district, candidates, result, incumbent, party, first
reset, select, select, print	1	lifetime, notes, notable, name, nationality

(cm). Recall from Section 3.1, that attributions are computed relative to a baseline. Given an input question and table, we use the input with the empty question as its baseline; see Equation 3.1. Thus the program selected at the baseline is the table-specific default program.

Attributions explain the difference between the prediction at a given input and that at the baseline. When an operator/column selection for an input is equal to that at its baseline, attributions do not convey meaningful information. Consequently, we examine attributions only for selections that are different from the table-specific default program.

Once the baseline is determined, attributions can be obtained by performing the computation shown in Definition 3.1 on the functions O_k and C_k.

Visualizing attributions. Good visualization of attributions is important as it allows us to effectively identify patterns. We visualize the attributions for NP as a heatmap for each question. Figure 3.1 contains visualizations for four questions that NP gets right.

In the first question, notice that "at" is a trigger for prev. This is unexpected, and we exploit this weakness to construct attacks in Section 3.4. In the second question, the word "below" is a trigger for the first three operators; this is evidence that words are possibly used to trigger an entire program sequence. We use this insight to extract rules in Section 3.3. In the fourth question, the superlative (most) does not trigger max, but instead triggers first. We use this to construct an attack in Section 3.4. Further, the operator is triggered by the word "gold". This is possibly a robust trigger for this dataset, but most likely not one a developer would author.

3.2.3 Evaluating Faithfulness of Attributions

We now present an empirical evaluation of the faithfulness of the attributions to NP's behavior. In particular, we wish to check that the attributions are (1) precise, in that, the terms with large attribution indeed affect the prediction, and (2) have good recall, in that, terms with tiny or no attribution do not affect the prediction. We perform these checks by

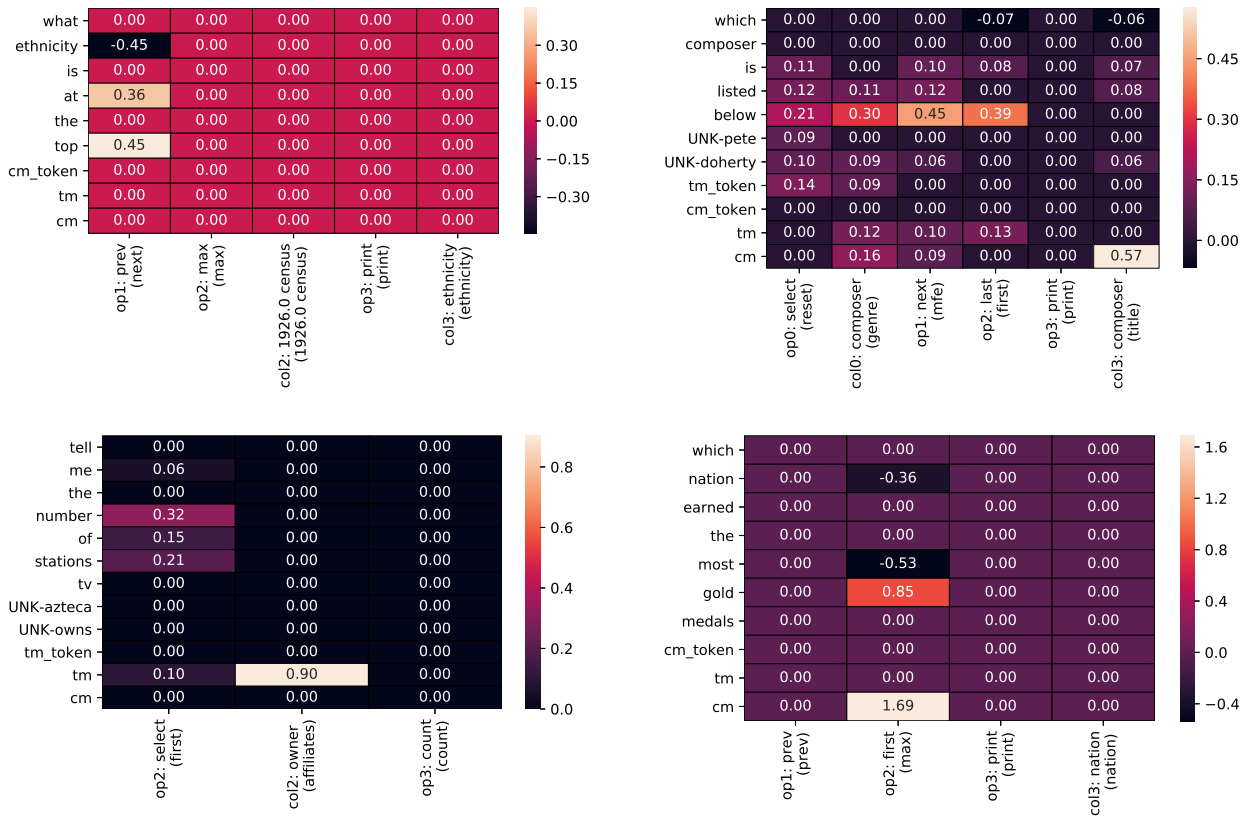

Figure 3.1: **Attributions to question words in NP**: Visualization of attributions for 4 questions that NP gets right. X-axis: the four operators (op) and columns (col) selected by NP. The table-specific default operator/columns are shown in parentheses alongside. Y-axis: the input to NP, i.e., question words with any tm_token and cm_token tokens, and table-match (*tm*) and column-match (*cm*) features (ref. Section 3.2.2). Values in the cells are the attribution values for the input (y-axis) to the selection (x-axis). When the selected operator/column is the same as its counterpart in the table-specific default program, we consider the input to have zero attribution. We also do not show operators and columns which have no effect on the answer computation (e.g. operator first is column-independent for which we exclude the corresponding column from the visualization)

ablating terms in the question and examining the change in the predicted program; the table is held fixed.

Precision. For each operator that NP selects for a question, we craft an ablated input by dropping the 3 terms with highest attribution value and check if the operator changes. We choose to ablate 3 terms as triggers can be phrases such as "how many" or "greater or equal". We ablate only when the selected operator is different from its counterpart in the table-specific default program. When the attribution stems from table-matches or column-matches, we zero out the corresponding input variables (tm, cm). For decoder time steps 2–4, we see that 82.9%, 98.3% and 98.9% respectively of the ablated questions result in an operator change. In decoder time step 1, the operator select is selected more than 95% of the time the table-specific default operator is not selected. select operator selects table rows which have entries matching the question and can be triggered by a range of words. As individual words do not have sufficiently higher attribution than others which trigger this operator, the notion of "top attributed word" does not make sense and we exclude decoder time step 1 from the analysis. Instead, we note that, by removing words amounting to 98% of the total attribution, we get a precision of 76.3%.

Recall. Per decoder time step, we remove words which contribute to less than 10% of the total attribution. For decoder time step 1–4 respectively, we see that 93.5%, 92.4%, 87.6% and 95.7% of the programs remain same.

3.3 Rule Extraction

Hand-authored systems are often preferred for their transparency, robustness, and precision. The main drawback of hand-authored approaches is the manual effort involved in making the rule-base comprehensive. One could leverage DNNs for extracting rules relieving developers of some of this burden. We demonstrate this by extracting rules from NP. Note that these

rules may have poor quality for a couple of reasons. First, they may overfit the training data. To deal with this, we can have the developer filter the output rules. Second, the extracted rules may not match the semantics of the hand-authored system. For this, we can have the developer define mappings between the output of the DNN and their system. We discuss both issues.

Rule extraction can also be useful as a mechanism to debug or make incremental improvements to DNNs. In Neural Machine Translation [9], attention weights are often used by developers to understand what the model has learnt and to make improvements. We argue that attention is not necessarily reflective of what the model has learnt and show that attributions can be a better alternative. Further, in networks where multiple attention heads are present, there is a priori no way to discern which or what combination of those attention weights would be useful.

3.3.1 Question Intents and Contextual Synonyms from Neural Programmer

Hand-authored approaches to question answering are based on semantic parsing wherein a hand-authored grammar is used to parse questions into a logical form (see Appendix B). We extract rules for two important steps in semantic parsing. First, intent classification of questions, i.e., identifying whether a question involves a lookup of table entries that satisfying one or more conditions, or a count, or a sort, etc. Second, mapping question words to column names based on synonymy. For instance, in the question "which country came in first?", "country" needs to be mapped to the column named "Nationality". We extract rules for learning such maps.

To improve the quality of the maps that we extract, we only run our method over questions in the dev dataset that are answered correctly by NP. Second, we exclude operator or column selections at a given step which are identical to those in the table-specific default program; recall that the underlying attributions do not have any meaning for such questions.

Third, some column selections are not used with answer computation. For instance, the column selected along with the operator first. Consequently, its trigger may not be meaningful. Thus, only columns co-selected with the operators [select, print, max, min, gt, lt, geq, leq, mfe] are considered for synonym extraction.

Extracting Question Intent Classification Rules

We extract rules that can be used in a hand-authored grammar based system from [41]. The important detail of this system is that it detects the intent of the question as an important substep. To ensure that our rules match the semantics of this grammar, we manually map operator sequences in NP to these intents. Table 3.3 maps operator sequences selected by NP to one of these intents[2] ; we do not consider sequences selected less than 10 times in the dev dataset. Sequences for which no intent made sense are mapped to `UNKNOWN`. As an example, notice that the NP program sequence [select, select, select, print] corresponds to the `LOOKUP` intent.

Given this mapping, we identify rules for question intent classification by identifying triggers for the operator sequence selected for each question. We call these operator sequence synonyms.

To extract operator sequence synonyms, we first consider the total attribution to a word from each of the operators selected for a question and table.

We find that the word "listed" is often the top attributed word for programs with the `BEFORE` and `AFTER` intents. This was surprising at first but on a closer examination we find that "listed" co-occurs with the words "before" and "after" which also receive a large attribution. Together the words {"listed", "before"} and {"listed", "after"} are reasonable triggers for `BEFORE` and `AFTER` questions. This motivates the use of tuples of words as operator sequence synonyms.

2. The mapping was defined by hand by examining the operator semantics and the intent descriptions in [41]

Table 3.3: **Mapping NP operator sequences to question intents**: We map commonly occurring operator sequences in Neural Programmer to question intents described in [41].

Operator Sequence	**Question Intent**
select, select, select, print	LOOKUP
select, select, select, count	COUNT
select, prev, first, print	BEFORE
reset, select, select, count	COUNT
select, next, first, print	AFTER
reset, reset, first, print	FIRST
reset, count, select, count	COUNT
reset, prev, max, print	SORT_MAX
reset, reset, reset, count	COUNT
select, next, last, print	AFTER
reset, reset, last, print	LAST
select, select, last, print	UNKNOWN
reset, reset, max, print	SORT_MAX
reset, reset, min, print	SORT_MIN
reset, mfe, first, print	MFE
count, reset, reset, count	COUNT
select, select, max, print	A_OR_B
count, select, select, count	COUNT
select, count, select, count	COUNT

Table 3.4: **Question intent triggers**: Top 3 triggers for each questions intent.

Question Intent	Triggers
COUNT	[tm_token, how, many], [how, many, times], [tm_token, number, of], [how, many, total], [number, of, total], [tm_token, number, total], [how, many, players], [how, many, years], [tm_token, number, times], [are, how, many], [as, how, many], [division, how, many], [listed, number, total], [how, many, there], [does, tm_token, many], [episodes, how, many], [how, many, played], [been, how, many], [tm_token, games, number], [games, how, many], [is, number, of], [number, times, total]
LOOKUP	[tm_token, how, many], [tm_token, number, of], [tm_token, how, long], [cm_token, tm_token, only], [tm_token, number, total], [by, tm_token, only], [tm_token, number, people], [tm_token, how, in], [tm_token, number, points]
A_OR_B	[cm_token, tm_token, or]
MFE	[cm_token, competitions, most]
LAST	[chart, last, the], [last, listed, the], [last]
FIRST	[chart, first, listed]
BEFORE	[before, tm_token, listed], [tm_token, listed, previous], [before, tm_token, nation], [above, tm_token, listed], [above, tm_token, is], [before, comes, tm_token]
SORT_MIN	[amount, least, the], [least]
AFTER	[after, tm_token, next], [after, tm_token, listed], [after, tm_token, who], [after, came, tm_token]

For each operator sequence, we consider the tuple of the top 3 words[3] by total attribution, and collect the tuples across questions and tables. Some tuples appear more frequently than others. We expect the triggers used frequently to be robust triggers while the infrequent ones to be a result of overfitting. Consequently, we discard tuples that appear with frequency less than 2.

Table 3.4 shows the tuples extracted for various intents, sorted by frequency. For intents absent on this list, we did not find any tuple with a frequency more than one. Notice the first few tuples for each intent are quite meaningful, and could be ported to other systems. For instance, the rule-based system can detect the A_OR_B type if it finds a question word that matches a column and another one that matches an entry in the table, and the question

3. The number 3 was picked by hand as it led to rules with high support in the dev dataset

contains the word "or"; this is essentially a grammar rule. Here is a question that would trigger this rule: "Which country won more medals, Austria or Russia", the column match would come from "country", the entry match from "Austria" or "Russia", and it has an "or". Likewise, the rule-based system can detect the `BEFORE` type if it finds a question word that matches a table entry, and the words "listed" and "before" in the question. The tuples for `FIRST` and `LAST` programs involve the word "chart" which seems surprising. On examining questions with the word "chart", we find that 15 out 20 questions have the `FIRST` or `LAST` intent. However, it is debatable if one should use this rule within a grammar.

Extracting Contextual Column Synonyms

We now extract synonyms for column headers. For instance, synonyms like "country" for the column header "Nationality". The synonyms we extract are not exact synonyms but are equivalent terms in the context of question-answering for a specific table.

The naive approach to extracting column synonyms is to collect the top attributed word for each column selection, and aggregate it across questions and tables. Words that frequently get picked for a column selection could then be extracted as its synonym.

The column synonyms found by this process are shown in Table 3.5, middle column. Some make sense, but there are several seemingly irrelevant triggers. For instance the question term "or" triggers the selection of the column "Nationality". To see why this happens, consider the question: "Did Austria or Russia score more points". There is no synonym for "Nationality" in the question. Intuitively, it is selected because the type of the question (`A_OR_B`) and because the filters "Austria" and "Russia" belong to this column. Perhaps this is why "or" shows up as a synonym for "Nationality".

For the synonym extraction task, we are specifically interested in words that influence column selection only by virtue of synonymy. If NP were expressed like a conventional program, we would try to find the set of control-flow paths that carry synonym triggers. In

a DNN, all such paths manifest as intertwined dataflow paths. While individual dataflow paths may not have meaningful semantics, we posit that groups of dataflow paths may carry cleaner semantics. We discuss this intuition further in Section 3.6.1.

We now identify dataflow paths in NP that may carry synonym triggers. In doing so, we leverage the abstraction of NP discussed in Section 3.2.1.

Abstraction into Dataflow Paths. Recall the abstraction in Listing 3.1. The variable of interest is `col_probs` which holds the column probabilities computed at each step. There are several dataflow paths from the input variable qw to this variable. Below, we cherry-pick a certain path that we believe captures synonym triggers. In describing the path, we use a standard trick of unrolling the for loop; any variable assigned in iteration k is suffixed with $\_k$.

The path from qw to `col_probs_k` can be divided into the following two groups:

A The path $qw \rightarrow$ `enc_hidden_states`
$\rightarrow$ `ques_context_k` $\rightarrow$ `col_controller_k`
$\rightarrow$ `col_logits_soft` $\rightarrow$ `col_probs_k`

B All other paths

All paths in group B either go through `col_logits_tbl_match`, or `col_probs_(k-1)`, or `op_probs_(k-1)`. Paths that go through `op_probs_(k-1)`, or `col_probs_(k-1)` influence column selection at step k indirectly by influencing the selections at the previous step. Therefore we do not expect synonym triggers specific to the column selection at step k to flow through these paths. Paths that go through `col_logits_tbl_match` influence column selection based on matches between the question words and the table contents, and not on the column names. Therefore, we do not expect direct synonyms triggers through these paths either.

For extracting column synonyms we attribute specifically through the path in Group A

Table 3.5: **Synonym extraction from Neural Programmer**: Synonyms for column names that appear with a frequency of more than 5 in the dev dataset. The top 5 synonyms found via each attribution method are shown.

Column name	Synonyms via all paths	Synonyms via question-context path
nation	[nation, number, many, or, difference, total, how, between, before]	[nation]
season	[year, season, tm_token, number]	[year, season]
date	[when, year, date, total, before, many, number, tm_token]	[when, date, year]
year	[when, or, year, before, most, after, popular, singles, locomotives, least]	[when, year, tm_token, higher, popular]
nationality	[country, or, nationality, each, many, which]	[country, nationality, which]
points	[top, number, many, total, how]	[top, number, many]
venue	[where, many, how, venue]	[where, venue]
competition	[event, long, competition, number]	[event, competition, number, long]
english	[song, english, how, after, which]	[song, english, after, which]
school	[team, no, school, after, tm_token]	[team, school]
country	[country, many, number]	[country]
time	[time, long, how]	[time, long]
album	[or, album, before]	[album]
opponent	[team, difference, which, number]	[team, which]
name	[before, or, difference, intersecting, number, after, total, mayor, rex, below]	[before, or, after, mayor]
title	[many, title, how, or, after, next, before, previous, when]	[title, after]
bronze	[country, was, many, medals, had, of, how]	[country, was, medals]
driver	[previous, number, next, before, consecutive, how, after, long, wins]	[how, previous]
position	[position, many, tm_token, how, which]	[position]
team	[after, or, before, below, team, number, times, next, and, many]	[previous, team, what]
city	[city, number, what, many]	[city, what]
population	[least, number, people]	[largest]

above. This is a direct path from the question words to the column selection via `ques_context` (functions `F1` and `F5` in the code). We call it the question context path.

Path-specific attributions. We now define path-specific attributions for a directed acyclic dataflow graph. Consider a dataflow graph with an output F and input $x ::= (x_1, \ldots, x_n)$. Let $p ::= (x_i, v_1, \ldots, v_n, F)$ be a dataflow path from the input variable x_i to an output F; here $v_1, \ldots, v_k$ represent outputs from intermediate nodes in the graph. The gradient of the output with respect to x_i through the path p is defined as

$$\frac{\partial F^p(x)}{\partial x_i} ::= \frac{\partial F(x)}{\partial v_k} \circ \ldots \circ \frac{\partial v_1(x)}{\partial x_i}$$

Definition (Path Integrated Gradients). Given an input x, a baseline x', and a dataflow path p from the input variable x_i to the output F, the integrated gradient of the output with respect to x_i through the path p is defined as follows.

$$\mathsf{IntegratedGrads}^p_{x_i}(x, x') ::= (x_i - x'_i) \times \int_{\alpha=0}^{1} \frac{\partial F^p(x' + \alpha \times (x - x'))}{\partial x_i} \, d\alpha$$

We now state a proposition that connects Path Integrated Gradients to Integrated Gradients.

Proposition. *Consider an input x and a baseline x'. Let $\{p_1, p_2 \ldots p_k\}$ be the set of all dataflow paths between an input variable x_i to an output F. Then, the following holds.*

$$\mathsf{IntegratedGrads}_{x_i}(x, x') ::= \sum_j \mathsf{IntegratedGrads}^{p_j}_{x_i}(x, x')$$

The main implication of Proposition 3.3.1 is that the path-specific attributions are a decomposition of the original attributions. We invoke Path Integrated Gradients on the column selections in NP along the question context path. This yields attributions for each of the question words through the question context path. Since the combination of this path

and the paths in group B together span the set of all the paths in the dataflow graph, the path specific attributions along the question context path, and the ones in group B will add up to the original attributions. In effect, we are ignoring the attributions from group B in the construction of column synonyms that follows.

Remark. In our computation of path-sensitive attributions, the gradients along a certan dataflow path are influenced by the values of the variables on other paths. The attributions forcefully break this connection. This does result in clean column synonyms, but it would be nice to identify formal semantics of the path-sensitive attributions.

Column synonyms. We consider the top 3 words by attribution for each column selection, and aggregate them across all questions in the dev dataset.

The synonyms obtained are shown in Table 3.5. In contrast to the column synonyms extracted based on attributions though all paths, the new column synonyms are much cleaner. For instance, "or" no longer shows up as a synonym for "Nationality". Moreover, the synonyms are contextual to specific questions and table. For instance, "country" is a synonym for "Nationality", and "where" is a synonym for "venue".

In the next subsection, we show that Path Integrated Gradients is useful beyond question answering systems in extracting synonyms learned by deep networks.

3.3.2 Word Alignments from Neural Machine Translation

Machine translation (MT) is the task of mapping sentences from a source language to a target language. Word alignment is a similar task, wherein the problem is to align words or phrases between one language and another. Given a source sentence with words $s_1, \ldots, s_m$ and target sentence with words $t_1, \ldots, t_n$, the task is to construct a matrix $M \in \mathbb{R}^{m \times n}$ whose $(i, j)^{\text{th}}$ entry indicates a "semantic equivalence score" of word s_i to t_j. Word alignments were earlier produced using statistical techniques such as GIZA++ [136], fast-align [47], which formed the basis for building statistical machine translation models [24].

The state of the art in machine translation are neural-network-based models [9, 178], which directly learn to translate from sentence pairs, bypassing the need for precomputed word alignments. However, extracting the alignments that neural models learn is a useful exercise for understanding what the models are learning and to make improvements.

The best performing neural MT models are based on two main kinds of architectures: sequence-to-sequence [165], and transformer [178]. Both these architectures contain so-called attention mechanisms, that provide the output-sentence-generating part of the neural network direct access to relevant input parts, as opposed to via highly encoded versions (more details later). As a result, the weights learned by the attention module in the neural network have frequently been used as a proxy for learned word alignments; visualizations of the attention weights are often found in literature [150, 187, 26, 29, 9, 178] spanning various tasks in language, speech and vision.

As noted by [93, 116, 6], although state-of-the-art MT models have very good translation performance, the word alignments extracted from attention weights do not compare well with word alignments computed via statistical techniques directly from sentence pairs. Keeping in mind the importance given to attention weights as a tool for analyzing neural MT models, it is important to analyze whether attention weights faithfully represent alignments learned by the neural network.

In this thesis, we propose Integrated-Gradient-based methods for analyzing word alignments learned by sequence-to-sequence networks. We compare our approach with word alignments extracted from attention weights. Although we focus only on sequence-to-sequence networks, the methods are also applicable to transformer-based networks; we discuss ideas for future research in Chapter 5.

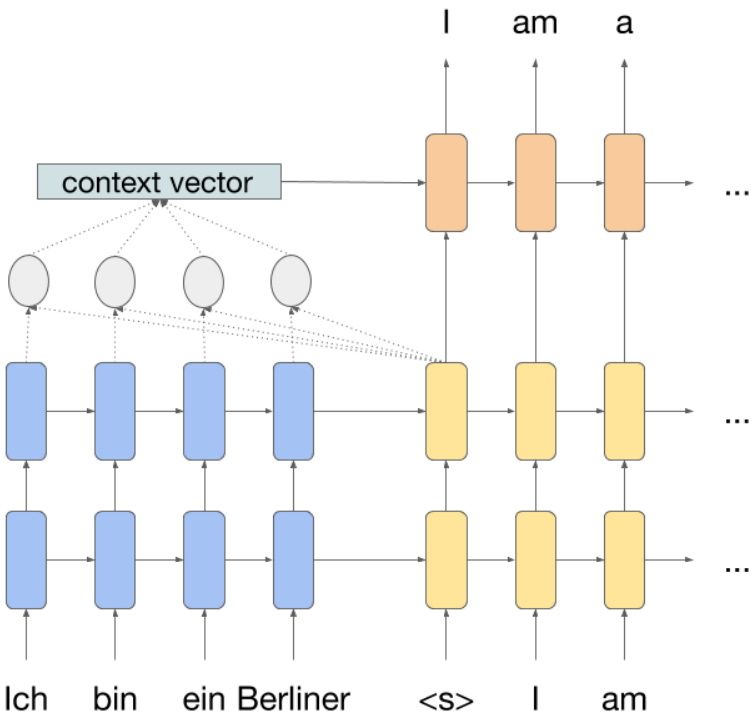

Figure 3.2: **Seq-to-seq network architecture**: Example sequence-to-sequence neural network with attention for neural machine translation. The encoder (blue) and decoder (yellow) are both stacked LSTMs with two layers. Attention is shown in the context of predicting the first token "I". The gray circles represent attention weights (scalars) assigned to each input word. The attention weights and the decoder determine the attention vector (orange), which then determines the output word.

Sequence-to-sequence learning and word alignments

We give a high-level overview of the network proposed in [9]. The neural network architecture is composed of three parts: an encoder, a decoder, and an attention mechanism. A visual illustration of the network is shown in Figure 3.2.

The network consists of an encoder recurrent neural network, which takes at each time step, a one-hot encoding of an input sentence token (word, or sub-word [155]). The hidden state of the encoder at time t can be written as $h_t^e = \texttt{enc}(x_t, h_{t-1}^e)$, where x_t is the embedding of the input token given by a learned embedding matrix. In the decoder recurrent neural network, the probability of an output word at time t' can be written as $\texttt{dec}(o_{t'-1}, h_{t'-1}^d, c)$

where $o_{t'-1}$ is the embedding of the output word at time $t'-1$, $h^d_{t'-1}$ is decoder hidden state and c is a context vector. In the simplest case, the context vector is simply h^e_m, or the hidden state of the encoder in the final step.

The attention mechanism essentially provides a different, learnable context at each time step to the decoder. That is, $P(o_i) = \texttt{dec}(o_{i-1}, h^d_{i-1}, c_i)$. The context vector itself is determined as follows from the encoder hidden states

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h^e_j$$

The weights α_{ij} are determined by an alignment model, which, typically is a feedforward neural network that is jointly trained with the entire machine translation model. The alignment model takes as input the decoder hidden state h^d_{i-1}, the encoder hidden states $h^e_j, j = 1, \ldots, n$ and produces $\alpha_{ij}, j = 1, \ldots, n$.

Attention-based word alignments are determined from the weights α_{ij} as follows

1. Construct a soft-alignment matrix M where element $M_{ij} = \alpha_{ij}$
2. Determine source-target alignment matrix S where $S_{ij} = 1$ only if S_{ij} is the maximum element column j, and zero otherwise
3. Determine target-source alignment matrix T where $T_{ij} = 1$ only if T_{ij} is the maximum element row i, and zero otherwise
4. Merge source-target and target-source alignments using the grow-diag-final-and heuristic [94] to get the final alignments A.

Note that attention maps hidden states of the encoder to the output word. This may result in multiple irrelevant input words that influence an encoder hidden state to be aligned to a single output word. We propose using Integrated Gradients to directly determine the

influence of input words in the model choosing the outputs. We consider two versions of alignments using Integrated Gradients

1. All-paths: for each output word, compute soft alignments as attributions to input word embeddings

2. Path-specific: for each output word, compute soft alignments as attributions to input word embeddings excluding the path via attention

The final word alignments are then determined as in the attention case.

Experimental results

We consider sequence-to-sequence NMT models for translating between German and English[4]. We use

- German → English model with stacked LSTM and Bahdanau [9] attention
- German → English model with stacked LSTM and GNMT [184] attention. Unlike in Bahdanau attention, in GNMT, the attention vector is copied to all cells of the stacked LSTM.
- English → German model with stacked LSTM and GNMT [184] attention.

Evaluation of the quality of extracted alignments is performed by comparing with human-annotated alignment data[5]. We use three metrics to measure the quality of alignments: match score [93], probability mass score [93], and alignment error rate [53].

The results are shown in Table 3.6. We observe the following

- path-specific attributions produce better word alignments than all-paths attributions in all cases

4. https://github.com/tensorflow/nmt/tree/0be864257a76c151eef20ea689755f08bc1faf4e

5. http://www-i6.informatik.rwth-aachen.de/goldAlignment/

- all-paths-based alignments are equal or worse than attention-based alignments
- attention-based alignments are better than attribution-based alignments in the case of GNMT attention

Table 3.6: **Quality of extracted word alignments**: Alignment quality metrics for various translation models and techniques for extracting alignments. Higher scores are better.

	Match Score	Prob. Mass Score	1-AER
NMT DE-EN with Bahdanau attention			
Path-specific	**0.842632**	**0.442453**	**0.769374**
All-paths	0.742962	0.307833	0.649282
Attention	0.754810	0.315105	0.646852
GNMT DE-EN			
Path-specific	0.776245	0.354733	0.694648
All-paths	0.684719	0.269675	0.588285
Attention	**0.787320**	**0.461235**	**0.709739**
GNMT EN-DE			
Path-specific	0.781308	0.389154	0.710498
All-paths	0.713376	0.293720	0.625008
Attention	**0.779599**	**0.427469**	**0.711420**

On inspecting the word alignments manually, we find that both all-paths alignments and attention-based alignments seem to highlight concepts related to co-occurrences of words or grammatical patterns. For instance, we found that oftentimes, a German noun is aligned to both its corresponding English noun and the preceding article. This may be a result of nouns being often preceded by an article. Another example is a punctuation mark such as comma in the English sentence is aligned to its corresponding punctuation in the German sentence as well as the succeeding word which is usually an "and" or "but".

We conjecture that the path through attention captures grammatical or co-occurrence concepts, while the path through decoder captures synonymy. The attention vector may determine the type of output word that is appropriate, and the decoder takes this information and produces an output word that is consistent with the type. One way to represent type

is by identifying the relevant input word – which was one of the original motivations for designing the attention mechanism – while another is via grammar (for e.g., part of speech), or it could be a mixture of both. We speculate that the all-paths-based alignments perform equal or worse than attention-based alignments due to the fact that the attention may be influenced by several input words when it conveys a part-of-speech or other grammatical information to the decoder. While the attention weights themselves may highlight the input word that is a synonym of the produced output word, they may be more sensitive to other input words that determine the grammatical pattern. This conjecture is also consistent with the observation that path-specific alignments are always better than all-paths alignments.

In NMT with Bahdanau attention, the attention vector is passed as input to the bottom cell of the stacked-LSTM decoder cell. In GNMT, the attention vector is passed as input to all cells of the stacked-LSTM, which may result in the attention weights sharing some of the functionality of the decoder, resulting in better word alignments than attribution-based methods.

For future research in this direction, we suggest the following experiments to shed more light on this conjecture

Experiment 1 Compute all-paths alignments on a model that does not contain attention, and test whether the alignment scores are worse than the path-specific results presented in this section. If yes, then it supports our conjecture.

Experiment 2 Compute all-paths alignments on a model whose decoder initial state is not copied from the encoder final state. This ensures that any path between output and input words goes via the attention weights. If the scores are comparable to the all-paths results presented in this section, then our conjecture is supported.

Experiment 3 Compute attention-based alignments on a model where the attention vector is copied only to two cells of the 4-layer stacked LSTM cell. If the alignment quality

metrics are worse than the attention-based results presented here, then it supports our conjecture.

3.3.3 Related Work

The idea of extracting symbolic rules from neural networks is not new [32, 171]. More recently, [128] developed techniques to extract rules from complex LSTM networks. Penkov et al. [141] induce LISP-like programs from the observed state transitions of a Deep Q-network. For a more thorough discussion, see the surveys by Jacobsson [79] and Hailesilassie [67].

Analogous to extracting rules, within the broader NLP community, as deep neural networks are replacing earlier models that were based on explicitly modeling linguistic features, there is an interest in understanding what kind of linguistic information is learned by neural networks. One group of methods [31, 20, 168, 2, 179] deals with predicting linguistic features (e.g., POS tags, subject-verb agreement, synonyms, anaphora, etc.) from neural network components (e.g., activations, embeddings, attention weights, etc.). These methods typically use statistical machine learning methods to predict the linguistic feature from neural network components. One of the conclusions of these methods is that lower layers of the neural network learn local features in the input sentence whereas the higher layers learn abstract concepts [157]. While these methods show that linguistic information is encoded in deep neural network models, [157] discuss a hypothesis that not all of this learned information is necessarily used by the neural network in making predictions. Results from [175] support the hypothesis and thus, pose interesting open questions about how useful are the extracted rules, and how one can enable a neural network model to make better use of the features that it encodes.

In the previous section, we have seen that attribution can be an effective alternative to attribution-based word alignments. Several papers debate the effectiveness of attention for interpreting NLP models. For instance, while [80] argues that attention weights are fre-

quently uncorrelated with gradient-based attribution methods and counterfactual attention distributions have little effect on the model's output, [182] questions the soundness of their analysis. Serrano & Smith [156] performs ablation-based analysis by zeroing out parts of the attention weights and measuring the change in the model's output. They conclude that while higher attention weights correlate with greater impact on model predictions, gradient-based attributions are often better at predicting their effects on model's output. Ghaeini [55] propose a combination of attention and attribution via identifying the parts of the attention map that most influence the model's output (computed via gradient-based attribution methods). They show that this results in more meaningful visualizations of alignment between input and output sentences in natural language inference (NLI) models. It would be worth exploring a similar approach for extracting word alignments from NMT models.

3.4 Measuring Comprehension of Semantics

In the previous sections, we have seen how Neural Programmer can rely on unimportant triggers from the question to produce a correct answer. We now describe ways to measure the extent of this phenomenon in two question-answering neural networks. First, we develop a test for overstability – the tendency of a model to rely on word matches and ignore semantics. This is based on systematically dropping words from questions. Second, we craft various adversarial attacks by exploiting the weaknesses as indicated by attributions.

We present results on two question-answering neural networks: 1) Neural Programmer, and 2) Visual Question Answering.

Visual Question Answering. : The Visual Question Answering Task [4, 170, 88, 15, 199] requires a system to answer questions about images (Figure 3.3). We analyze the deep network from [88]. It achieves 61.1% accuracy on the validation set (the state of the art [54] achieves 66.7%). We chose this model for its easy reproducibility.

The VQA 1.0 dataset [4] consists of 614,163 questions posed over 204,721 images (3

Question: how symmetrical are the white bricks on either side of the building
Prediction: very
Ground truth: very

Figure 3.3: **Visual QA [88] attributions**: Visualization of attributions (word importances) for a question that the network gets right. Red indicates high attribution, blue negative attribution, and gray near-zero attribution. The colors are determined by attributions normalized w.r.t the maximum magnitude of attributions among the question's words.

questions per image). The images were taken from COCO [115], and the questions and answers were crowdsourced.

The network in [88] treats question answering as a classification task wherein the classes are 3000 most frequent answers in the training data. The input question is tokenized, embedded and fed to a multi-layer LSTM. The states of the LSTM attend to a featurized version of the image, and ultimately produce a probability distribution over the answer classes.

We applied IG and attributed the top selected answer class to input question words. The baseline for a given input instance is the image and an empty question[6]. We omit instances where the top answer class predicted by the network remains the same even when the question is emptied (i.e., the baseline input). This is because IG attributions are not informative when the input and the baseline have the same prediction.

A visualization of the attributions is shown in Figure 3.3. Notice that very few words have high attribution. We verified that altering the low attribution words in the question does not change the network's answer. For instance, the following questions still return "*very*" as the answer: "*how spherical are the white bricks on either side of the building*", "*how soon are the bricks fading on either side of the building*", "*how fast are the bricks speaking on*

6. We do not black out the image in our baseline as our objective is to study the influence of just the question words for a given image

either side of the building".

On analyzing attributions across examples, we find that most of the highly attributed words are words such as "*there*", "*what*", "*how*", "*doing*" – they are usually the less important words in questions. In Section 3.4.1 we describe a test to measure the extent to which the network depends on such words. We also find that informative words in the question (e.g., nouns) often receive very low attribution, indicating a weakness on the part of the network.

3.4.1 Overstability Test

Here we propose a test to measure the extent of erroneous trigger logic present in a deep neural network model. To determine the set of question words that the network finds most important, we isolate words that most frequently occur as top attributed words in questions. We then drop all words except these and compute the accuracy. We plot the accuracy as a function of vocabulary size as the words are dropped.

For Visual QA, Figure 3.4 (left side) shows how the accuracy changes as the size of this isolated set is varied from 0 to 5305. We find that just one word is enough for the model to achieve more than 50% of its final accuracy. That word is "color".

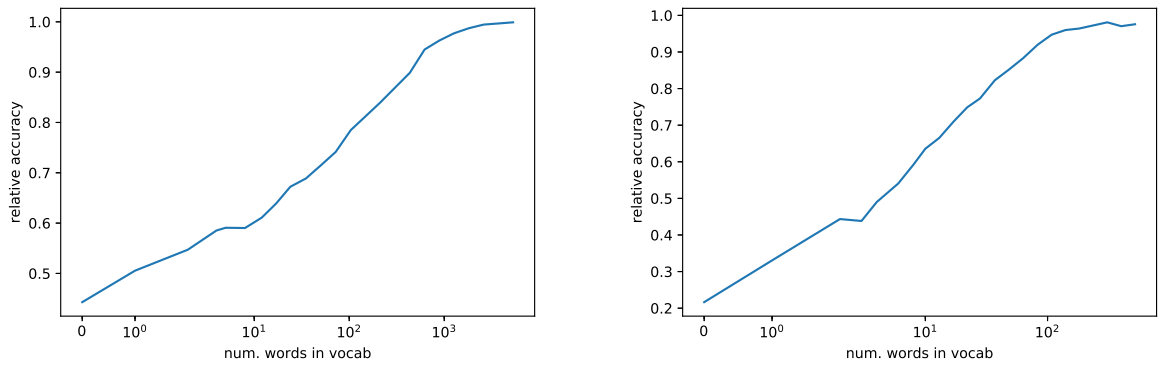

Figure 3.4: **Overstability test**: Accuracy as a function of vocabulary size, relative to its original accuracy. Words are chosen in the descending order of how frequently they appear as top attributions. The X-axis is on logscale, except near zero where it is linear. Left side is for Visual QA, right for Neural Programmer

Note that even when empty questions are passed as input to the network, its accuracy remains at about 44.3% of its original accuracy. This shows that the model is largely reliant on the image for producing the answer.

The accuracy increases (almost) monotonically with the size of the isolated set. The top 6 words in the isolated set are "color", "many", "what", "is", "there", and "how". We suspect that generic words like these are used to determine the type of the answer. The network then uses the type to choose between a few answers it can give for the image.

Similar to that for Visual QA, we check for overstability in NP by looking at accuracy as a function of the vocabulary size. We treat table match annotations tm_token, cm_token and the out-of-vocab token (*unk*) as part of the vocabulary. The results are in Figure 3.4 (right side). We see that the curve is similar to that of Visual QA (Figure 3.4 (left side). Just 5 words (along with the column selection priors) are sufficient for the model to reach more than 50% of its final accuracy on the validation set. These five words are: "many", "number", "tm_token", "after", and "total".

3.4.2 Adversarial Inputs

We are motivated by [82], and propose adversarial attacks to measure the extent of question comprehension. As they discuss, "*the extent to which [reading comprehension systems] truly understand language remains unclear*". The contrast between [82] and our work is instructive. Their main contribution is to fix the evaluation of reading comprehension systems by augmenting the test set with adversarially constructed examples. (As they point out in Section 4.6 of their paper, this does not necessarily fix the model; the model may simply learn to circumvent the specific attack underlying the adversarial examples.) Their method is independent of the specification of the model at hand. They use crowdsourcing to craft passage perturbations intended to fool the network, and then query the network to test their effectiveness.

In contrast, we propose improving the analysis of question answering systems. Our method peeks into the logic of a network to identify high-attribution question terms. Often there are several important question terms (e.g., nouns, adjectives) that receive tiny attribution. We leverage this weakness and perturb questions to craft targeted attacks. While [82] focus exclusively on systems for the reading comprehension task, we analyze one system each for two different tasks. In addition, we also propose ways to improve the hit rate of attacks from [82]. Our analysis technique is specific to deep-learning-based systems, whereas theirs is not.

Fluff word deletion attacks

On examining the attributions for operator and column selection to question words (see Section 3.2.2) in NP, we found that sometimes an operator selection is based on fluff words (also called stop words) like: "a", "at", "the", etc. For instance, in the question, "what ethnicity is at the top?", the operator next[7] is triggered on the word "at". Dropping the word "at" from the question changes the operator selection and causes NP to return the wrong answer.

Selecting operators based on fluff words is not robust. In real search usage logs, users often phrase their questions concisely by dropping fluff words. For instance, the user may simply say "top ethnicity". To measure the extent of this non-robustness, we drop fluff words from questions in the dev dataset that were originally answered correctly and test NP on them. The fluff words to be dropped were manually selected. They are *show, tell, did, me, my, our, are, is, were, this, on, would, and, for, should, be, do, I, have, had, the, there, look, give, has, was, we, get, does, a, an, 's, that, by, based, in, of, bring, with, to, from, whole, being, been, want, wanted, as, can, see, doing, got, sorted, draw, listed, chart, only.*

By dropping fluff words, the accuracy drops from 33.62% to 28.60%. It may be possible to defend against such examples by generating synthetic training data, and re-training the

7. The selection of the next operator in answering this question is surprising and non-robust in its own right; we leverage this kind of non-robustness in our table row reordering attacks

network on it.

Question concatenation attacks

In these attacks, we either append or prepend content-free phrases to questions. The phrases are crafted using irrelevant trigger words for operator selections (Table 3.7). We manually ensure that the phrases are content-free.

Table 3.7: **Neural Programmer [131] operator triggers**: Notice that there are several irrelevant triggers (highlighted in red). For instance, "many" is irrelevant to "prev". See Section 3.4 for attacks exploiting this weakness.

Operator	Triggers
select	[tm_token, many, how, number, or, total, after, before, only]
prev	[before, many, than, previous, above, how, at, most]
first	[tm_token, first, before, after, who, previous, or, peak]
reset	[many, total, how, number, last, least, the, first, of]
count	[many, how, number, total, of, difference, between, long, times]
next	[after, not, many, next, same, tm_token, how, below]
last	[last, or, after, tm_token, next, the, chart, not]
mfe	[most, cm_token, same]
min	[least, the, not]
max	[most, largest]
geq	[at, more, least, had, over, number, than, many]
print	[tm_token]

Table 3.8 describes our results. The first 4 phrases use irrelevant trigger words and result in a large drop in accuracy. For instance, the first phrase uses "not" which is a trigger for "next", "last", and "min", and the second uses "same" which is a trigger for "next" and "mfe". The four phrases combined result in the model's accuracy going down from 33.5% to 3.3%. The first two phrases alone drop the accuracy to 5.6%.

The next set of phrases use words that receive low attribution across questions, and are hence non-triggers for any operator. The resulting drop in accuracy on using these phrases is relatively low. Combined, they result in the model's accuracy dropping from 33.5% to 27.1%.

Table 3.8: **Question concatenation attacks on NP**: Validation accuracy when attack phrases are concatenated to the question. (Original: 33.5%)

Attack phrase	Prefix	Suffix
in not a lot of words	20.6%	10.0%
if its all the same	21.8%	18.7%
in not many words	15.6%	11.2%
one way or another	23.5%	20.0%
Union of above attacks	**3.3%**	
Baseline		
please answer	32.3%	30.7%
do you know	31.2%	29.5%
Union of baseline prefixes	**27.1%**	

These attacks demonstrate a different kind of weakness compared to the one exploited by the fluff word attacks.

For Visual QA, phrases are manually crafted using generic words that the network finds important (Section 3.4.1). Table 3.9 (top half) shows the resulting accuracy for three prefixes —"*in not a lot of words*", "*what is the answer to*", and "*in not many words*". All of these phrases nearly halve the model's accuracy. The union of the three attacks drops the model's accuracy from 61.1% to 19%.

We note that the attributions computed for the network were crucial in crafting the prefixes. For instance, we find that other prefixes like "*tell me*", "*answer this*" and "*answer this for me*" do not drop the accuracy by much; see Table 3.9 (bottom half). The union of these three ineffective prefixes drops the accuracy from 61.1% to only 46.9%. Per attributions, words present in these prefixes are not deemed important by the network.

Row reordering attacks

We found that NP often got the question right by leveraging artifacts of the table. For instance, the operators for the question "which nation earned the most gold medals" are

Table 3.9: **Question concatenation attacks on VQA**: Accuracy for prefix attacks; original accuracy is 61.1%.

Prefix	Accuracy
in not a lot of words	35.5%
in not many words	32.5%
what is the answer to	31.7%
Union of all three	**19%**
Baseline prefix	
tell me	51.3%
answer this	55.7%
answer this for me	49.8%
Union of baseline prefixes	**46.9%**

"reset", "prev", "first" and "print". The "prev" operator essentially excludes the last row from the answer computation. It gets the answer right for two reasons: (1) the answer is not in the last row, and (2) rows are sorted by the values in the column "gold".

In general, a question answering system should not rely on row ordering in tables. To quantify the extent of such biases, we used a perturbed version of WikiTableQuestions validation dataset as described in [140][8] and evaluated the existing NP model on it (there was no re-training involved here). We found that NP has only 23% accuracy on it, in contrast to an accuracy of 33.5% on the original validation dataset.

One approach to making the network robust to row-reordering attacks is to train against perturbed tables. This may also help the model generalize better. Indeed, [127] note that the state-of-the-art strongly supervised[9] model on WikiTableQuestions [103] enjoys a 7% gain in its final accuracy by leveraging perturbed tables during training.

Remark. To eliminate this kind of weakness, there are two steps one could potentially take during training of the DNN: (1) including perturbed versions of the tables in the training

8. based on data at `https://nlp.stanford.edu/software/sempre/wikitable/dpd/`

9. supervised on the structured program

set, and (2) constraining the DNN with rules. One example of a rule is, "Use operators first and last only when synonyms of those words appear in the question". Both approaches have the common purpose of forbidding operators such as first from being used when the question asks for a max.

Remark. We tested our attacks on the state-of-the-art model [Krishnamurthy et al., EMNLP-2017] and found it to significantly drop accuracy. For instance, the "table perturbation", "fluff word", and "question concatenation" attacks drop its dev accuracy by 5.2%, 5.7% and 24.5% respectively.

Subject ablation attack

In this attack, we replace the subject of a question with a specific noun that consistently receives low attribution across questions. We then determine, among the questions that the network originally answered correctly, what percentage result in the same answer after the ablation. We repeat this process for different nouns; specifically, "fits", "childhood", "copyrights", "mornings", "disorder", "importance", "topless", "critter", "jumper", "tweet", and average the result.

We find that, among the set of questions that the Visual QA network originally answered correctly, 75.6% of the questions return the same answer despite the subject replacement.

Predicting the effectiveness of model-independent attacks

Recall that [82] construct attacks on reading comprehension models by adding an irrelevant question to the paragraph designed to fool the model. Their attack AddSent appends sentences to the paragraph that resemble an answer to the question without changing the ground truth. See the second column of Table 3.10 for a few examples. Their method is model-independent, and therefore cannot predict whether a given sentence will fool the model. We show how attributions can help predict the effectiveness of their attacks. By

Table 3.10: **Analysis of Jia and Liang [82]'s attacks**: AddSent attacks that failed to fool the model. The first four rows show attacks, where, with modifications to preserve nouns with high attributions, they are successful in fooling the model. Question words that receive high attribution are colored red (intensity indicates magnitude).

Question	AddSent attack that does not work	Attack that works
Who was Count of Melfi	Jeff Dean was the mayor of Bracco.	Jeff Dean was the mayor of Melfi.
What country was Abhisit Vejjajiva prime minister of , despite having been born in Newcastle ?	Samak Samak was prime minister of the country of Chicago, despite having been born in Leeds.	Abhisit Vejjajiva was chief minister of the country of Chicago, despite having been born in Leeds.
Where according to gross state product does Victoria rank in Australia ?	According to net state product, Adelaide ranks 7 in New Zealand	According to net state product, Adelaide ranked 7 in Australia. (as a prefix)
When did the Methodist Protestant Church split from the Methodist Episcopal Church ?	The Presbyterian Catholics split from the Presbyterian Anglican in 1805.	The Methodist Protestant Church split from the Presbyterian Anglican in 1805. (as a prefix)

incorporating attributions into the crowdsourcing part of their algorithm, one can construct highly effective attacks.

The reading comprehension task involves identifying a span from a context paragraph as an answer to a question. The SQuAD dataset [144] for machine reading comprehension contains 107.7K query-answer pairs, with 87.5K for training, 10.1K for validation, and another 10.1K for testing. Deep learning methods are quite successful on this problem, with the state-of-the-art F1 score at 84.6 achieved by [193]; we analyze their model.

We investigate the effectiveness of their attacks using attributions. We analyze 100 examples generated by the AddSent method in [82], and find that an adversarial sentence is successful in fooling the model in two cases:

First, a contentful word in the question gets low/zero attribution and the adversarially added sentence modifies that word. E.g. in the question, "Who did Kubiak take the place of after Super Bowl XXIV?", the word "Super" gets low attribution. Adding "After Champ Bowl XXV, Crowton took the place of Jeff Dean" changes the prediction for the model. Second, a contentful word in the question that is not present in the context. E.g. in the question "Where hotel did the Panthers stay at?", "hotel", is not present in the context. Adding "The Vikings stayed at Chicago hotel." changes the prediction for the model.

On the flip side, an adversarial sentence is unsuccessful when a contentful word in the question having high attribution is not present in the added sentence. E.g. for "Where according to gross state product does Victoria rank in Australia?", "Australia" receives high attribution. Adding "According to net state product, Adelaide ranks 7 in New Zealand." does not fool the model. However, retaining "Australia" in the adversarial sentence does change the model's prediction.

Next we correlate attributions with efficacy of the AddSent attacks. We analyzed 1000 (question, attack phrase) instances[10] where [193] model has the correct baseline prediction.

10. data sourced from https://worksheets.codalab.org/worksheets/0xc86d3ebe69a3427d91f9aaa63f7d1e7d/

Of the 1000 cases, 508 are able to fool the model, while 492 are not. We split the examples into two groups. The first group has examples where a noun or adjective in the question has high attribution, but is missing from the adversarial sentence and the rest are in the second group. Our attribution analysis suggests that we should find more failed examples in the first group. That is indeed the case. The first group has 63% failed examples, while the second has only 40%.

Recall that the attack sentences were constructed by (a) generating a sentence that answers the question, (b) replacing all the adjectives and nouns with antonyms, and named entities by the nearest word in GloVe word vector space [142] and (c) crowdsourcing to check that the new sentence is grammatically correct. This suggests a use of attributions to improve the effectiveness of the attacks, namely ensuring that question words that the model thinks are important are left untouched in step (b) (we note that other changes in the construction of the sentence should be carried out). In Table 3.10, we show a few examples where an original attack did not fool the model, but preserving a noun with high attribution did.

3.4.3 Related Work

Recently, there have been a number of techniques for crafting adversarial attacks against DNNs. Most of these techniques are targeted at image networks, where, applying a tiny perturbation to the image drastically changes its prediction. Here the image does not visually change. There is no analog of such attacks for text. In our attacks, the question does perceptibly change, but there is no change in the semantics.

Adversarial attacks on reading comprehension systems were recently considered in [82]. Here, the system answers questions about a paragraph of text. These attacks involve extending a paragraph with an irrelevant sentence which causes the system to change its answer. Our attacks are different from these in two ways. First, some of our attacks modify the ques-

tion, whereas all their attacks modify the paragraph text (analogous to our tables). Second, their attacks are black box, while our attacks leverage the network's weaknesses (visible via the attributions) to handcraft an attack.

[3] analyze several VQA models. Among other attacks, they test the models on question fragments of telescopically increasing length. They observe that VQA models often arrive at the same answer by looking at a small fragment of the question. Our stability analysis in Section 3.4.1 explains, and intuitively subsumes this; indeed, several of the top attributed words appear in the prefix, while important words like "color" often occur in the middle of the question. Our analysis enables additional attacks, for instance, replacing question subject with low attribution nouns. [145] use a model explanation technique to illustrate overstability for two examples. They do not quantify their analysis at scale. [85, 196] examine the VQA data, identify deficiencies, and propose data augmentation to reduce over-representation of certain question/answer types. [60] propose the VQA 2.0 dataset, which has pairs of similar images that have different answers on the same question. We note that our method can be used to improve these datasets by identifying inputs where models ignore several words. [75] evaluate robustness of VQA models by appending questions with semantically similar questions. Our prefix attacks in Section 3.4 are in a similar vein and perhaps a more natural and targeted approach. Finally, [51] use saliency methods to produce image perturbations as adversarial examples; our attacks are on the question.

One of the main challenges in adversarial methods for NLP is generation of realistic inputs. Unlike images, the space of natural language inputs is discrete and consequently, small perturbations of a natural language input (at both word and character level) result in big changes in semantics. The aforementioned attacks as well as those in this thesis require the involvement of a human to be able to generate meaningful adversarial inputs. Since the work in this thesis was done, there have been a few approaches that attempt to generate meaningful inputs.For instance, [194] use a pretrained language model to regulate a sentence

perturbation process, while [78, 146] use backtranslation as a tool to maintain semantic equivalence between the input and its perturbation.

3.5 Influence of Training-Data Pruning

A natural question to ask is where model weaknesses come from: is it from the training data, or from the model architecture? Typically, whenever a novel architecture is used that results in improvement to benchmark accuracy, the gain is attributed to model architecture. In this section, we discuss how simple pruning of training data can be as critical as changes to model architecture in improving model performance.

Consider the current best model [103] (KDG) for question answering on tabular data evaluated over the WikiTableQuestions dataset. It achieves a single model[11] accuracy of only 43.3%[12]. This is nonetheless a significant improvement compared to the 34.8% accuracy achieved by the previous best single model [69]. We seek to analyze the source of the improvement achieved by the KDG model. The KDG paper claims that the improvement stems from certain aspects of the model architecture.

We find that a large part of the improvement also stems from a certain pruning of the data used to train the model. The KDG system generates its training data using an algorithm proposed by [140]. This algorithm applies a pruning step (discussed in Section 3.5.2) to eliminate spurious training data entries. We find that without this pruning of the training data, accuracy of the KDG model drops to 36.3%.

We consider this an important finding as the pruning step not only accounts for a large fraction of the improvement in the state-of-the-art KDG model but may also be relevant to training other models. In what follows, we briefly discuss the pruning algorithm, how we

11. A 5-model ensemble achieves a higher accuracy of 45.9%. We choose to work with a single model as it makes it easier for us to perform our analysis.

12. All quoted accuracy numbers are based on our training of the KDG model using the configuration and instructions provided at: `https://github.com/allenai/pnp/tree/wikitables2/experiments/wikitables`

identified its importance for the KDG model, and its relevance to further work.

3.5.1 KDG Model and Training Data

Similar to Neural Programmer, the KDG system operates by translating a natural language question and a table to a logical form in Lambda-DCS [114]. The logical form is then executed on the table to obtain the final answer.

Unlike Neural Programmer, the KDG model is supervised on the logical form. The WTQ dataset only contains the correct answer label for question-table instances. To obtain the desired training data, the KDG system enumerates consistent logical form candidates for each $\langle q, t, a \rangle$ triple in the WTQ dataset, i.e., it enumerates all logical forms that lead to the correct answer a on the given table t. For this, it relies on the dynamic programming algorithm of [140]. This algorithm is called dynamic programming on denotations (DPD).

3.5.2 Pruning Algorithm

A key challenge in generating consistent logical forms is that many of them are spurious, i.e., they do not represent the question's meaning. For instance, a spurious logical form for the question "which country won the highest number of gold medals" would be one which simply selects the country in the first row of the table. This logical form leads to the correct answer only because countries in the table happen to be sorted in descending order.

[140] propose a separate algorithm for pruning out spurious logical forms using fictitious tables. Specifically, for each question-table instance in the dataset, fictitious tables are generated[13], and answers are crowdsourced on them. A logical form that fails to obtain the correct answer on any fictitious table is filtered out. The paper presents an analysis over 300 questions revealing that the algorithm eliminated 92.1% of the spurious logical forms.

13. Fictitious tables are generated by applying perturbations, such as shuffling columns, to the original table. We refer to [140] for more details.

3.5.3 Importance of Pruning for KDG

The KDG system relies on the DPD algorithm for generating consistent logical form candidates for training. It does not explicitly prescribe pruning out spurious logical forms candidates before training. Since this training set contains spurious logical forms, we expected the model to also sometimes predict spurious logical forms. However, we were somewhat surprised to find that the logical forms predicted by the KDG model were largely non-spurious. We then examined the logical form candidates[14] that the KDG model was trained on.

Through personal communication with Panupong Pasupat, we learned that all of these candidates had been pruned using the algorithm mentioned in Section 3.5.2.

We trained the KDG model on unpruned logical form candidates[15] generated using the DPD algorithm, and found its accuracy to drop to 36.3% (from 43.3%); all configuring parameters were left unchanged[16]. This implies that pruning out spurious logical forms before training is necessary for the performance improvement achieved by the KDG model.

3.5.4 Commentary

[103] present several ablation studies to identify the sources of the performance improvement achieved by the KDG model. These studies comprehensively cover novel aspects of the model architecture. On the training side, the studies only vary the number of logical forms per question in the training dataset. Pruning of the logical forms was not considered. This may have happened inadvertently as the KDG system may have downloaded the logical forms dataset made available by Pasupat et al. without noticing that it had been pruned out[17].

14. Available at: `https://cs.stanford.edu/~ppasupat/research/h-strict-all-matching-lfs.tar.gz`

15. Available at: `https://nlp.stanford.edu/software/sempre/wikitable/dpd/h-strict-dump-combined.tar.gz`

16. It is possible that the model accuracy may slightly increase with some hyper-parameter tuning but we do not expect it to wipe out the large drop resulting from disabling the pruning.

17. This was initially the case for us until we learned from Panupong Pasupat that the dataset had been

We note that our finding implies that pruning out spurious logical forms before training is an important factor in the performance improvement achieved by the KDG model. It does not imply that pruning is the only important factor. We trained Neural Programmer with the same pruned dataset, and found no performance improvement. Thus, the architectural innovations are essential for the performance improvement too.

In light of our finding, we would like to emphasize that the performance of a machine learning system depends on several factors such as the model architecture, training algorithm, input pre-processing, hyper-parameter settings, etc. As [90] point out, attributing improvements in performance to the individual factors is a valuable exercise in understanding the system, and generating ideas for improving it and other systems. In performing these attributions, it is important to consider all factors that may be relevant to the system's performance.

[140] claimed "the pruned set of logical forms would provide a stronger supervision signal for training a semantic parser". This thesis provides empirical evidence in support of this claim. We further believe that the pruning algorithm may also be valuable to models that score logical forms. Such scoring models are typically used by grammar-based semantic parsers such as the one in [139]. Using the pruning algorithm, the scoring model can be trained to down-score spurious logical forms. Similarly, neural semantic parsers trained using reinforcement learning may use the pruning algorithm to only assign rewards to non-spurious logical forms.

The original WTQ dataset may also be extended with the fictitious tables used by the pruning algorithm. This means that for each $\langle q, t, a \rangle$ triple in the original dataset, we would add additional triples $\langle q, t_1, a_1 \rangle, \langle q, t_2, a_2 \rangle, \ldots$ where $t_1, t_2, \ldots$ are the fictitious tables and $a_1, a_2 \ldots$ are the corresponding answers to the question q on those tables. Such training data augmentation may improve the performance of neural networks that are directly trained over

pruned.

the WTQ dataset, such as [131]. The presence of fictitious tables in the training set may help these networks to generalize better, especially on tables that are outside the original WTQ training set.

3.6 Discussion

3.6.1 Related Work

Program analysis

Our analysis method has two different connections with classic program analysis. The first is a direct connection with program slicing which amounts to finding all statements in a program that directly or indirectly affect a certain target variable (also called slicing criterion) [180, 5]. We effectively use slicing to determine which inputs affect various output variables in NP; this is discussed in Section 3.1. For instance, through slicing, we discover that only matches between the table and the question (and not the raw table contents) affect the operator and column selection.

The second is a somewhat subtle connection with path-sensitive analysis of programs. Different control-flow paths in a program, presumably, carry different semantic influences on a target variable. Path-sensitive methods [21] examine data dependencies separately along individual control-flow paths. Unlike programs, DNNs typically do not contain explicit control-flow. However, if we think of a DNN as a continuous relaxation of an underlying (but unknown) discrete program then the control-flow logic may still be present softly in the dataflow paths. To illustrate this point, consider discrete control-flow of the form `if` $P(x)$ `then` $F(x)$ `else` $G(x)$. This may be relaxed as $(1 - P^c(x)).F^c(x) + P^c(x).G(x)$, where P^c, F^c and G^c are themselves continuous relaxations of the functions P, F, and G. The `if` branch now corresponds to the dataflow paths for the expression $(1 - P^c(x)).F^c(x)$.

Following this intuition, in Section 3.3, we posit that different groups of dataflow paths

in the network may capture different semantic influences. Analogous to path-sensitive analysis, we extended the Integrated Gradients method to compute attribution along separate dataflow paths. In general, it is not a priori clear how to group the paths, especially since the underlying discrete program is unknown. In our case, we manually abstracted NP (Listing 3.1) based on our understanding of its architecture, and speculated that a certain dataflow path may carry synonym influence.

Analysis techniques for DNNs

Similar to Integrated Gradients, there are many other methods that could be used to attribute a DNN's prediction to its input features [8, 159, 158, 19, 162]. We chose Integrated Gradients due to its ease of implementation and its axiomatic justification; we refer to the original Integrated Gradients paper [164] for a detailed comparison with other attribution methods. A novel contribution of this paper is the extension of the Integrated Gradients method to specifically attribute through certain dataflow paths in the network.

There has been work around verification of properties of DNNs, for instance [87].

In contrast to our work, the DNNs they analyze have a simpler topology, and the properties being verified are well-formulated safety properties. Consequently, their guarantees are cleaner. The complexity of our analysis stems from the hardness of stating analogous properties for our network and task. It is possible that our path-sensitive dataflow analysis can be combined with their method.

Neural program synthesis

Neural Programmer is part of a larger class of techniques that use deep learning for program synthesis. The idea is to train a DNN to predict a program from a given specification. The specification may be in natural language (as in NP), or input-output-examples. Important networks in this area include RobustFill [39], Neuro-Symbolic Program Synthe-

sis [138], DeepCoder [10], and many others; see [161] for a survey. Our analysis method applies to such program synthesis networks as well. For instance, we could use attributions to understand what parts of the input specification triggers the selection of various statements in the program. A related class of networks is those that perform neural program induction [107, 84, 62]. Here, conditioned on a specification, the network learns to predict the output for a given input; the program logic stays implicit in the network. Similar to NP, these networks typically have "controller" RNN that (softly) applies an operation on the input at each step. Our analysis method could help understand the decisions made by the controller by attributing them to features of the input.

3.6.2 Opportunities for Future Research

We analyzed a complex deep neural network that synthesizes programs that answer questions over semi-structured tables. In general, such models have the benefit of automation, and the risk of a lack of transparency. The latter stems from the fact that deep learning programs use distributed representations. Therefore, it is hard to identify strong semantics with the internal operations of a deep network. Consequently, we study the influence of inputs on outputs via attributions. We use this to understand the input-output behavior of the Neural Programmer (NP), to identify adversarial inputs, and to extract rules. Overall, we found that analyzing Neural Programmer is messier than the formal analysis of programs or protocols. It is possible that some of this mess is inherent to the nature of deep learning. Our analysis also highlights the following opportunities for further research.

Debugging tool user-experience

We sought to analyze the input-output behavior of NP. The first task was to identify the right set of inputs and outputs, and find the subset of inputs that affect each output (program slicing). While this sounds straightforward, it was painstaking due to lack of modularity. The

input is fed through several steps of preprocessing; we had to locate a step that was both interpretable and differentiable (to allow the computation of attributions). Similarly, the final output of the network is an answer, however from an analysis perspective analyzing the program (operator and column selections) was more useful in understanding how the network generalizes—an understanding we leverage both in rule extraction and attack computation. The result of this manual process was the program in Listing 3.1. How can we aid the developer through this activity?

Abstracting DNNs for path-sensitive analysis

The role played by control-flow path in a conventional program are played by dataflow paths in a deep-learning based program. However, these dataflow paths are a continuous, distributed version of discrete logic. Consequently, individual paths may not carry meaningful semantics. This does not imply that our analysis must be purely blackbox. One can posit groups of paths that do carry cleaner semantics. We leverage this in generating column synonyms in Section 3.3. We expect similar groupings of paths to occur in DNNs for other tasks; for instance in object recognition networks, often several neurons share parameters and hence can be grouped. (Such groups of neurons are called filters in the computer vision literature.) It is interesting to identify interesting grouping criteria that result in semantically meaningful analyses.

Verification properties for natural language tasks

How does one formally state correctness of tasks that use natural language specifications? Natural language does not have formal semantics; meaning is often contextual. The test/-validation set does specify correctness for a list of questions. Satisfying this spec does not guarantee good behavior on other inputs. One approach to establishing correctness in a more comprehensive way is to iteratively construct a grammar based system that can simulate the

network. Rules can be extracted from the network and added to the grammar until a tolerably close simulation of the operator selection is achieved. This is also a more thorough way of doing rule extraction. Contrast this with the rules we extract in Section 3.3 that do not form a holistic representation of NP despite being useful in a piecemeal way.

CHAPTER 4

PARAMETER-EFFICIENT TRANSFER AND MULTITASK LEARNING

The term "Internet of Things (IoT)" refers to the proliferation of internet services and connectivity into embedded devices and everyday objects. As a result of that and advancements in cloud technology, it has been possible for machine learning to be offered as a service to these IoT devices. An IoT device can train and run inference on a machine learning model that lives on the cloud. However, in more recent times, there is an emerging reversal of this trend.

Increasingly, much of the computation is being shifted away from the cloud and towards embedded devices such as phones, smartwatches, drones, etc. The main purpose and motivation behind this shift is to improve privacy of user data, create opportunities for personalization, and to reduce latency [98]. However, delivering and updating hundreds of neural network models on devices is challenged by limited memory, energy and bandwidth availability.

In this work, our focus is on deep-learning-based models. As the application space of deep learning continues to grow, there is a growing need to quickly build and customize models. While there still might be space for improvement in designing smaller deep-neural-network models, in this work we explore a different angle: we would like to be able to build models that require only a few parameters to be trained in order to be re-purposed to a different task, with minimal loss in accuracy compared to a model trained from scratch. While there is ample existing work on compressing models and learning as few weights as possible [148, 153, 73] to solve a single task, to the best of our awareness, there is no prior work that tries to minimize the number of model parameters when solving many tasks together.

Transfer learning is the term used for re-purposing models in machine learning. In the

context of deep neural networks, this involves taking a pretrained model for one task, and then finetuning the weights for the subsequent task. However, we cannot afford to change all the weights, as this would effectively result in a completely new model that needs to be stored on the embedded device.

In this chapter, we describe a novel learning paradigm that can help in greatly limiting the total number of parameters required for all the tasks. Our contribution is the concept of a "model patch" (described in the next section), and an evaluation of its performance for various image-based tasks (Section 4.4).

4.1 Method

The central concept in our method is that of a **model patch**. It is essentially a small set of per-channel transformations that are dispersed throughout the network resulting in only a tiny increase in the number of model parameters.

Suppose a deep neural network $\mathcal{M}$ is a sequence of layers represented by their parameters (weights, biases), $\mathbf{W}_1, \ldots, \mathbf{W}_n$. We ignore non-trainable layers (e.g., some kinds of activations) in this formulation. A model patch P is a set of parameters $\mathbf{W}'_{i_1}, \ldots, \mathbf{W}'_{i_k}$, $1 \leq i_1, \ldots, i_k \leq n$ that, when applied to $\mathcal{M}$, adds layers at positions $i_1, \ldots, i_n$. Thus, a patched model would be

$$\mathcal{M}' = \mathbf{W}_1, \ldots, \mathbf{W}_{i_1}, \mathbf{W}'_{i_1}, \ldots, \mathbf{W}_{i_n}, \mathbf{W}'_{i_n}, \ldots, \mathbf{W}_n$$

In this chapter, we introduce two kinds of patches. We will see below that they can be folded with the other layers in the network, eliminating the need to perform any explicit addition of layers. In Section 4.4, we shed some light on why the particular choice of these patches is important.

Scale-and-bias patch. This patch applies per-channel scale and bias to every layer in the

network. In practice these transformations can often be absorbed into normalization layer such as Batch Normalization [76]. Let $\mathbf{X}$ be an activation tensor. Then, the batch-normalized version of $\mathbf{X}$

$$BN(\mathbf{X}) = \gamma \frac{\mathbf{X} - \mu(\mathbf{X})}{\sigma(\mathbf{X})} + \beta$$

where $\mu(\mathbf{X}), \sigma(\mathbf{X})$ are mean and standard deviation computed per minibatch, and γ, β are learned via backpropagation. These statistics are computed as mini-batch average, while during inference they are computed using global averages.

The scale-and-bias patch corresponds to all the $\gamma, \beta, \mu, \sigma$ in the network. Using BN as the model patch also satisfies the criterion that the patch size should be small. For instance, the BN parameters in both MobilenetV2 [153] and InceptionV3 network performing classification on ImageNet amounts to less than 40K parameters, of about 1% for MobilenetV2 that has 3.5 million parameters, and less than 0.2% for Inception V3 that has 25 million parameters.

While we utilize batch normalization in this work, we note that this is merely an implementation detail and we can use explicit biases and scales with similar results.

Depthwise-convolution patch. The purpose of this patch is to re-learn spatial convolution filters in a network. Depth-wise separable convolutions were introduced in deep neural networks as way to reduce number of parameters without losing much accuracy [73, 28]. They were further developed in [153] by adding linear bottlenecks and expansions.

In depthwise separable convolutions, a standard convolution is decomposed into two layers: a depthwise convolution layer, that applies one convolutional filter per input channel, and a pointwise layer that computes the final convolutional features by linearly combining the depthwise convolutional layers' output across channels. We find that the set of depthwise convolution layers can be re-purposed as a model patch. They are also lightweight - for instance, they account for less than 3% of MobilenetV2's parameters when training on ImageNet.

Next, we describe how model patches can be used in transfer and multi-task learning.

Transfer learning. In transfer learning, the task is to adapt a pretrained model to a new task. Since the output space of the new task is different, it necessitates re-learning the last layer. Following our approach, we apply a model patch and train the patched parameters, optionally also the last layer. The rest of the parameters are left unchanged. In Section 4.4, we discuss the inclusion/exclusion of the last layer. When the last layer is not trained, it is fixed to its random initial value. See Figure 4.1 for an illustration.

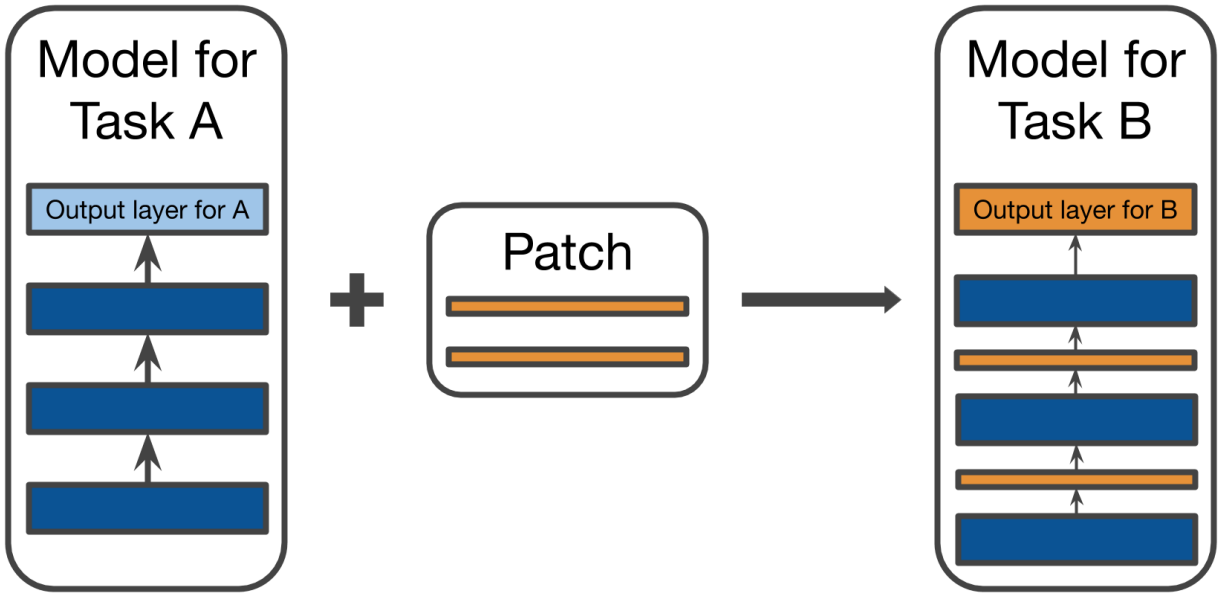

Figure 4.1: **Model patch**: An example illustrating the idea of a model patch. On task B, only the patched layers are trained, while the rest are fixed to their pretrained values from task A.

Multitask learning. We aim to simultaneously, but independently, train multiple neural networks that share most weights. Unlike in transfer learning, where a large fraction of the weights are kept frozen, here we learn all the weights. However, each task carries its own model patch, and trains a patched model. By training all the parameters, this setting offers more adaptability to tasks while not compromising on the total number of parameters.

To implement multi-task learning, we use the distributed TensorFlow paradigm[1]: a cen-

1. `https://www.tensorflow.org/guide/extend/architecture`

tral parameter server receives gradient updates from each of the workers and updates the weights. Each worker reads the input, computes the loss and sends gradients to the parameter server. We allow subsets of workers to train different tasks; workers thus may have different computational graphs, and task-specific input pipelines and loss functions. A visual depiction of this setting is shown in Figure 4.2.

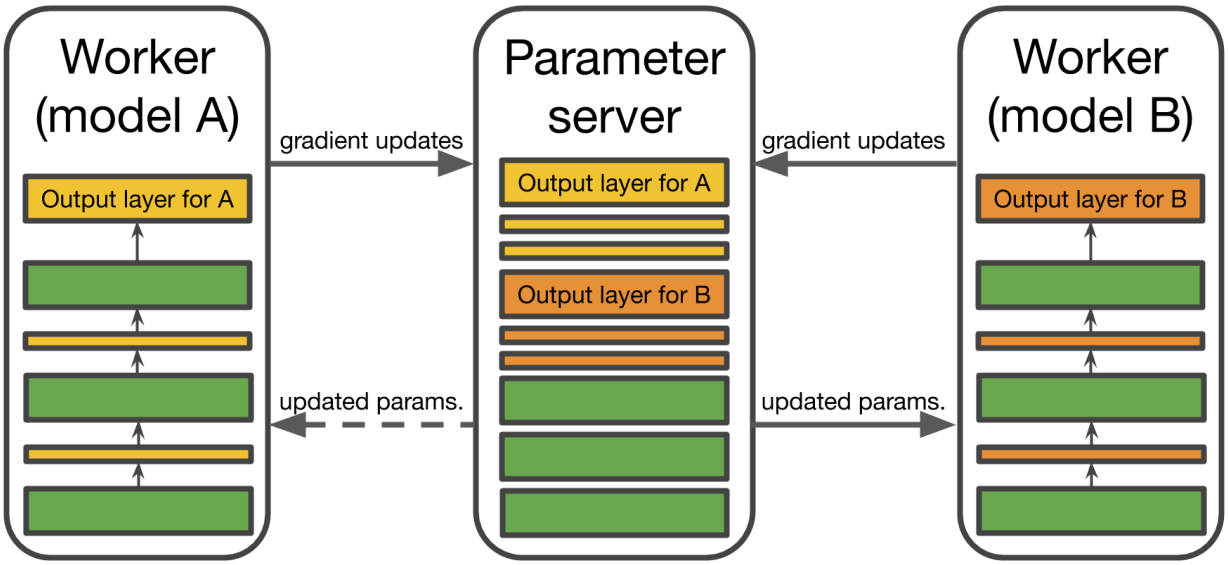

Figure 4.2: **Multitask learning**: An example of multi-task learning using model patches 4.1. Each model has its own input data pipeline and loss function, and are trained indepedently.

4.2 Related Work

One family of approaches [190, 43] widely used by practitioners for domain adaptation and transfer learning is based on fine-tuning only the last layer (or sometimes several last layers) of a neural network to solve a new task. Fine-tuning the last layer is equivalent to training a linear classifier on top of existing features. This is typically done by running SGD while keeping the rest of the network fixed, however other methods such as SVM has been explored as well [91]. It has been repeatedly shown that this approach often works best for similar tasks (for example, see [43]).

Another frequently used approach is to use full fine-tuning [33] where a pretrained model is simply used as a warm start for the training process. While this often leads to significantly improved accuracy over last-layer fine-tuning, downsides are that 1) it requires one to create and store a full model for each new task, and 2) it may lead to overfitting when there is limited data. In this work, we are primarily interested in approaches that allow one to produce highly accurate models while reusing a large fraction of the weights of the original model, which also addresses the overfitting issue.

While the core idea of our method is based on learning small model patches, we see significant boost in performance when we fine-tune the patch along with last layer (Section 4.4). This result is somewhat in contrast with [72], where the authors show that the linear classifier (last layer) does not matter when training full networks. Mapping out the conditions of when a linear classifier can be replaced with a random embedding is an important open question.

[113] show that re-computing batch normalization statistics for different domains helps to improve accuracy. In [148] it was suggested that learning batch normalization layers in an otherwise randomly initialized network is sufficient to build non-trivial models. Re-computing batch normalization statistics is also frequently used for model quantization where it prevents the model activation space from drifting [102]. In the present work, we significantly broaden and unify the scope of the idea and scale up the approach by performing transfer and multi-task learning across completely different tasks, providing a powerful tool for many practical applications.

Our work has interesting connections to meta-learning [134, 50, 27]. For instance, when training data is not small, one can allow each task to carry a small model patch in the Reptile algorithm of [134] in order to increase expressivity at low cost.

4.3 Analysis

Experiments (Section 4.4) show that model-patch based fine-tuning, especially with the scale-and-bias patch, is comparable and sometimes better than last-layer-based fine-tuning, despite utilizing a significantly smaller set of parameters. At a high level, our intuition is based on the observation that individual channels of hidden layers of neural network form an embedding space, rather than correspond to high-level features. Therefore, even simple transformations to the space could result in significant changes in the target classification of the network.

In this section, we attempt to gain some insight into this phenomenon by taking a closer look at the properties of the last layer and studying low-dimensional models.

A deep neural network performing classification can be understood as two parts:

1. a network base corresponding to a function $F : \mathbb{R}^d \to \mathbb{R}^n$ mapping d-dimensional input space $\mathbb{X}$ into an n-dimensional embedding space $\mathbb{G}$, and

2. a linear transformation $s : \mathbb{R}^n \to \mathbb{R}^k$ mapping embeddings to logits with each output component corresponding to an individual class.

An input $\boldsymbol{x} \in \mathbb{X}$ producing the output $\boldsymbol{o} := s(F(\boldsymbol{x})) \in \mathbb{R}^k$ is assigned class c iff $\forall i \neq c, o_i < o_c$.

We compare fine-tuning model patches with fine-tuning only the final layer s. Fine-tuning only the last layer has a severe limitation caused by the fact that linear transformations preserve convexity.

It is easy to see that, regardless of the details of s, the mapping from embeddings to logits is such that if both $\boldsymbol{\xi}^a, \boldsymbol{\xi}^b \in \mathbb{G}$ are assigned label c, the same label is assigned to every $\boldsymbol{\xi}^\tau := \tau\boldsymbol{\xi}^b + (1-\tau)\boldsymbol{\xi}^a$ for $0 \leq \tau \leq 1$. Indeed, $[s(\boldsymbol{\xi}^\tau)]_c = \tau o_c^b + (1-\tau)o_c^a > \tau o_i^b + (1-\tau)o_i^a = [s(\boldsymbol{\xi}^\tau)]_i$ for any $i \neq c$ and $0 \leq \tau \leq 1$, where $\boldsymbol{o}^a := s(\boldsymbol{\xi}^a)$ and $\boldsymbol{o}^b := s(\boldsymbol{\xi}^b)$. Thus, if the model assigns inputs $\{\boldsymbol{x}_i | i = 1, \ldots, n_c\}$ some class c, then the same class will also be assigned to any point in the preimage of the convex hull of $\{F(\boldsymbol{x}_i) | i = 1, \ldots, n_c\}$.

This property of the linear transformation s limits one's capability to tune the model given a new input space manifold. For instance, if the input space is "folded" by F and the neighborhoods of very different areas of the input space $\mathbb{X}$ are mapped to roughly the same neighborhood of the embedding space, the final layer cannot disentangle them while operating on the embedding space alone (should some new task require differentiating between such "folded" regions).

We illustrate the difference in expressivity between model-patch-based fine-tuning and last-layer-based fine-tuning in the cases of 1D and 2D inputs and outputs. Despite the simplicity, our analysis provides useful insights into how by simply adjusting biases and scales of a neural network, one can change which regions of the input space are folded and ultimately the learned classification function.

In what follows, we will work with a construct introduced by [123] that demonstrates how neural networks can "fold" the input space $\mathbb{X}$ a number of times that grows exponentially with the neural network depth[2]. We consider a simple neural network with one-dimensional inputs and outputs and demonstrate that a single bias can be sufficient to alter the number of "folds", the topology of the $\mathbb{X} \to \mathbb{G}$ mapping. More specifically, we illustrate how the number of connected components in the preimage of a one-dimensional segment $[\boldsymbol{\xi}^a, \boldsymbol{\xi}^b]$ can vary depending on a value of a single bias variable.

4.3.1 1D Case

As in [123], consider the following function:

$$q(x; b) \equiv 2\mathsf{ReLU}\left([1, -1, \ldots, (-1)^{p-1}] \cdot \boldsymbol{v}^T(x; \boldsymbol{b}) + b_p \right),$$

2. In other words, $F^{-1}(\boldsymbol{\xi})$ for some $\boldsymbol{\xi}$ contains an exponentially large number of disconnected components.

where

$$\boldsymbol{v}(x;\boldsymbol{b}) \;\equiv\; [\max(0, x + b_0), \max(0, 2x - 1 + b_1), \ldots, \max(0, 2x - (p-1) + b_{p-1})],$$

p is an even number, and $\boldsymbol{b} = (b_0, \ldots, b_p)$ is a $(p{+}1)$–dimensional vector of tunable parameters characterizing q. Function $q(x;\boldsymbol{b})$ can be represented as a two-layer neural network with ReLU activations.

Set $p = 2$. Then, this network has 2 hidden units and a single output value, and is capable of "folding" the input space twice. Defining F to be a composition of k such functions

$$F(x;\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(k)}) \equiv q(q(\ldots q(q(x;\boldsymbol{b}^{(1)});\boldsymbol{b}^{(2)});\ldots;\boldsymbol{b}^{(k-1)});\boldsymbol{b}^{(k)}), \tag{4.1}$$

we construct a neural network with $2k$ layers that can fold input domain $\mathbb{R}$ up to 2^k times. By plotting $F(x)$ for $k = 2$ and different values of $b_0^{(1)}$ while fixing all other biases to be zero, it is easy to observe that the preimage of a segment $[0.2, 0.4]$ transitions through several stages (see figure 4.3), in which it: (a) first contains 4 disconnected components for $b_0^{(1)} > -0.05$, (b) then 3 for $b_0^{(1)} \in (-0.1, -0.05]$, (c) 2 for $b_0^{(1)} \in (-0.4, -0.1]$, (d) becomes a simply connected segment for $b_0^{(1)} \in [-0.45, -0.4]$ and (e) finally becomes empty when $b_0^{(1)} < -0.45$. This result can also be extended to $k > 2$, where, by tuning $b_0^{(1)}$, the number of "folds" can vary from 2^k to 0.

4.3.2 2D Case

Here we show an example of a simple network that "folds" input space in the process of training and associates identical embeddings to different points of the input space. As a result, fine-tuning the final linear layer is shown to be insufficient to perform transfer learning to a new dataset. We also show that the same network can learn alternative embedding that avoids input space folding and permits transfer learning.

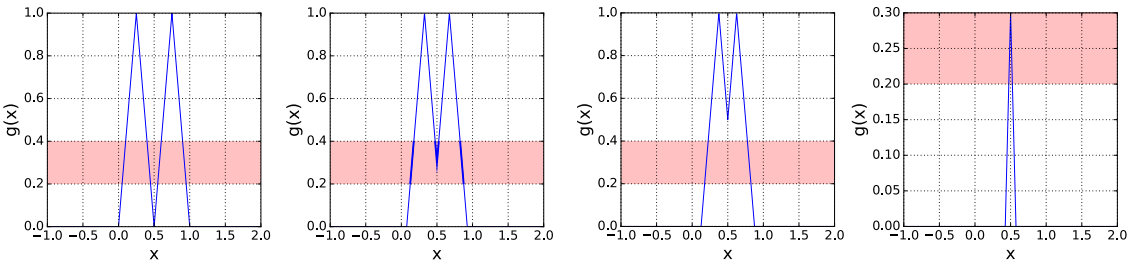

Figure 4.3: **Illustrative example for effectiveness of model patch**: Function plots $F(x; \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)})$ for a 4-layer network given by equation 4.1 with $k = 2$ and all biases except $b_0^{(1)}$ set to zero. From left to right: $b_0^{(1)} = 0$, $b_0^{(1)} = -0.075$, $b_0^{(1)} = -0.125$ and $b_0^{(1)} = -0.425$. The preimage of a segment $[0.2, 0.4]$ (shown as shaded region) contains 4, 3, 2 and 1 connected components respectively.

Consider a deep neural network mapping a 2D input into 2D logits via a set of 5 ReLU hidden layers: 2D input $\rightarrow$ 8D state $\rightarrow$ 16D state $\rightarrow$ 16D state $\rightarrow$ 8D state $\rightarrow$ m-D embedding (no ReLU) $\rightarrow$ 2D logits (no ReLU). Since the embedding dimension is typically smaller than the input space dimension, but larger than the number of categories, we first choose the embedding dimension m to be 2. This network is trained (applying sigmoid to the logits and using cross entropy loss function) to map (x, y) pairs to two classes according to the groundtruth dependence depicted in Figure 4.4(a). Learned function is shown in Figure 4.4(c). The model is then fine-tuned to approximate categories shown in Figure 4.4(b). Fine-tuning all variables, the model can perfectly fit this new data as shown in Figure 4.4(d).

Once the set of trainable parameters is restricted, model fine-tuning becomes less efficient. Figures 4.4(A) through 4.4(E) show output values obtained after fine-tuning different sets of parameters. In particular, it appears that training the last layer alone (see Figure 4.4(E; top)) is insufficient to adjust to new training data, while training biases and scales allows to approximate new class assignment (see Figure 4.4(C; top)). Notice that a combination of all three types of trainable parameters (biases, scales and logits) frequently results in the best function approximation even if the initial state is chosen to be random (see Figure 4.4(A)-(E); bottom row).

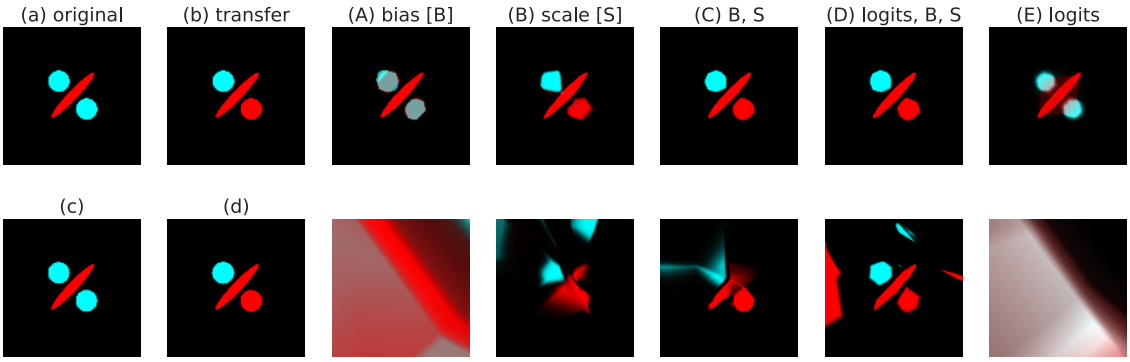

Figure 4.4: **Example comparing finetuning methods**: The neural network is first trained to approximate class assignment shown in (a) (with the corresponding learned outputs in (c)), network parameters are then fine-tuned to match new classes shown in (b). If all network parameters are trained, it is possible (d) to get a good approximation of the new class assignment. Outputs obtained by fine-tuning only a subset of parameters are shown in columns (A) through (E): functions fine-tuned from a pretrained state (c) are shown at the top row; functions trained from a random state (the same for all figures) are shown at the bottom. Each figure shows training with respect to a different parameter set: (A) biases; (B) scales; (C) biases and scales; (D) logits, biases and scales; (E) just logits.

4.4 Experiments

We evaluate the performance of our method in both transfer and multi-task learning using the image recognition networks MobilenetV2 [153] and InceptionV3 [167] and a variety of datasets: ImageNet [37], CIFAR-10/100 [104], Cars [100], Aircraft [121], Flowers-102 [135] and Places-365 [198]. An overview of these datasets can be found in Table 4.1. We also show preliminary results on transfer learning across completely different types of tasks using MobilenetV2 and Single-Shot Multibox Detector (SSD) [118] networks.

We use both scale-and-bias (S/B) and depthwise-convolution patches (DW) in our experiments. Both MobilenetV2 and InceptionV3 have batch normalization - we use those parameters as the S/B patch. MobilenetV2 has depthwise-convolutions from which we construct the DW patch. In our experiments, we also explore the effect of fine-tuning the patches along with the last layer of the network. We compare with two scenarios: 1) only fine-tuning the last layer, and 2) fine-tuning the entire network.

We use TensorFlow [1], and NVIDIA P100 and V100 GPUs for our experiments. Follow-

Table 4.1: **Datasets**: Datasets used in experiments (Section 4.4)

Name	CIFAR-100	Flowers-102	Cars	Aircraft	Places-365	ImageNet
#images	60,000	8,189	16,185	10,200	1.8 million	1.3 million
#classes	100	102	196	102	365	1000

ing the standard setup of Mobilenet and Inception we use 224×224 images for MobilenetV2 and 299×299 for InceptionV3. As a special-case, for Places-365 dataset, we use 256×256 images. We use RMSProp optimizer with a learning rate of 0.045 and decay factor 0.98 per 2.5 epochs.

4.4.1 Learning with Random Weights

To demonstrate the expressivity of the biases and scales, we perform an experiment on MobilenetV2, where we learn only the scale-and-bias patch while keeping the rest of the parameters frozen at their initial random state. The results are shown in Table 4.3 (right side). It is quite striking that simply adjusting biases and scales of random embeddings provides features powerful enough that even a linear classifier can achieve a non-trivial accuracy. Furthermore, the synergy exhibited by the combination of the last layer and the scale-and-bias patch is remarkable.

4.4.2 Transfer Learning

We take MobileNetV2 and InceptionV3 models pretrained on ImageNet (Top1 accuracies 71.8% and 76.6% respective), and fine-tune various model patches for other datasets.

Results on InceptionV3 are shown in Table 4.2. We see that fine-tuning only the scale-and-bias patch (using a fixed, random last layer) results in comparable accuracies as fine-tuning only the last layer while using fewer parameters. Compared to full fine-tuning [33], we use orders of magnitude fewer parameters while achieving nontrivial performance. Our

results using MobilenetV2 are similar (more on this later).

Table 4.2: **Accuracy of transfer learning**: Transfer-learning on Inception V3, against full-network fine-tuning.

Fine-tuned params.	Flowers		Cars		Aircraft	
	Acc.	#params	Acc.	#params	Acc.	#params
Last layer	84.5	208K	55	402K	45.9	205K
S/B + last layer	**90.4**	244K	**81**	437K	**70.7**	241K
S/B only (random last)	79.5	**36K**	33	**36K**	52.3	**36K**
All (ours)	93.3	25M	92.3	25M	87.3	25M
All [33]	96.3	25M	91.3	25M	82.6	25M

In the next experiment, we do transfer learning between completely different tasks. We take an 18-category object detection (SSD) model [118] pretrained on COCO images [115] and repurpose it for image classification on ImageNet. The SSD model uses MobilenetV2 (minus the last layer) as a featurizer for the input image. We extract it, append a linear layer and then fine-tune. The results are shown in Table 4.3. Again, we see the effectiveness of training the model patch along with the last layer - a 2% increase in the parameters translates to 19.4% increase in accuracy.

Table 4.3: **From SSD/Random to ImageNet**: Learning Imagenet from SSD feature extractor (left) and random filters (right)

Fine-tuned params.	#params	COCO→Imagenet, Top1	Random→ Imagenet, Top1
Last layer	1.31M	29.2%	0%
S/B + last layer	1.35M	47.8%	20%
S/B only	34K	6.4%	2.3%
All params	3.5M	71.8%	71.8%

Next, we discuss the effect of learning rate. It is common practice to use a small learning rate when fine-tuning the entire network. The intuition is that, when all parameters are trained, a large learning rate results in network essentially forgetting its initial starting point. Therefore, the choice of learning rate is a crucial factor in the performance of transfer

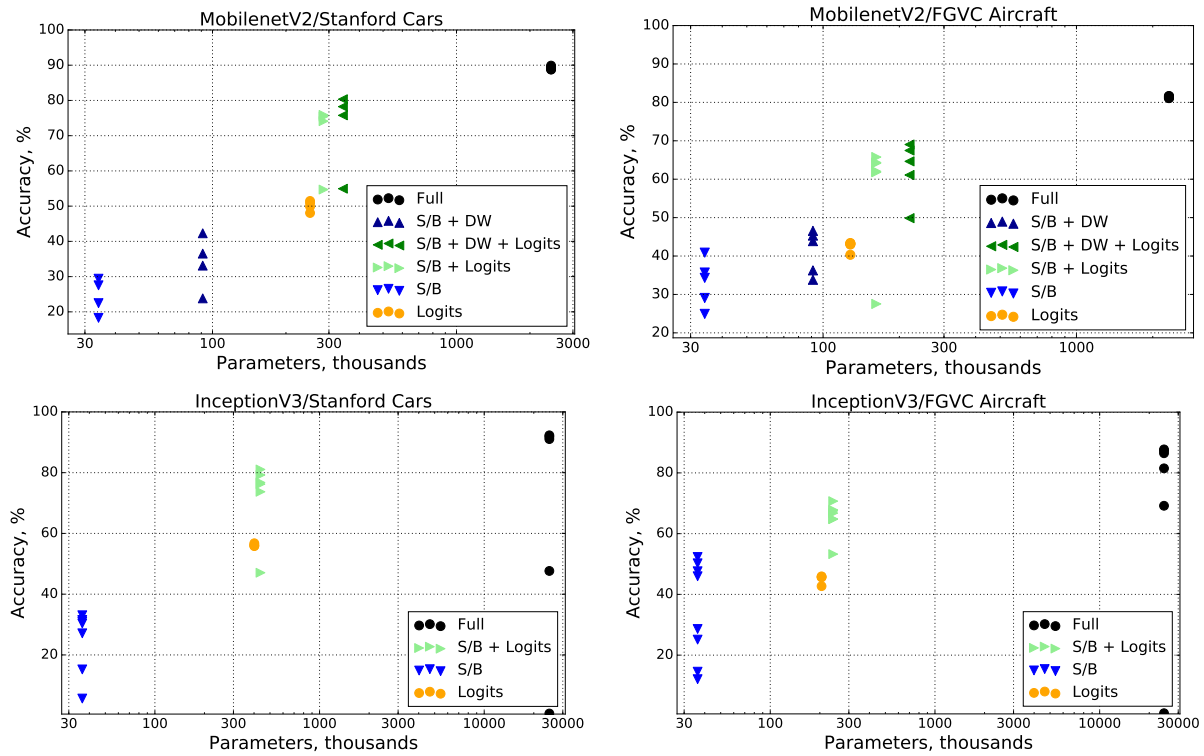

Figure 4.5: **Comparison of finetuning approaches on MobileNetV2**: Performance of different fine-tuning approaches for different datasets for MobileNetV2 and Inception. The same color points correspond to runs with different initial learning rates, starting from 0.0045 to 0.45 with factor 3. Best viewed in color.

learning. In our experiments (Figure 4.6) we observed the opposite behavior when fine-tuning only small model patches: the accuracy grows as learning rate increases. In practice, fine-tuning a patch that includes the last layer is more stable w.r.t. the learning rate than full fine-tuning or fine-tuning only the scale-and-bias patch. Combining model patches and fine-tuning results in a synergistic effect.

The results of [113] suggested that adjusting Batch Normalization statistics helps with domain adaption. Interestingly we found that it significantly worsens results for transfer learning, unless bias and scales are allows to learn. We find that fine-tuning on last layer with batch-norm statistics readjusted to keep activation space at mean 0/variance 1, makes the network to significantly under-perform compared to fine-tuning with frozen statistics. Even though adding *learned* bias/scales signifcanty outperforms logit-only based fine-tuning.

We summarize our experiments in Table 4.4

Table 4.4: **Effect of batch norm statistics**: The effect of batch-norm statistics on logit-based fine-tuning for MobileNetV2

Method	Flowers	Aircraft	Stanford Cars	Cifar100
Last layer (logits)	80.2	43.3	51.4	45.0
Same as above + batchnorm statistics	79.8	38.3	43.6	57.2
Same as above + scales and biases	**86.9**	**65.6**	**75.9**	**74.9**

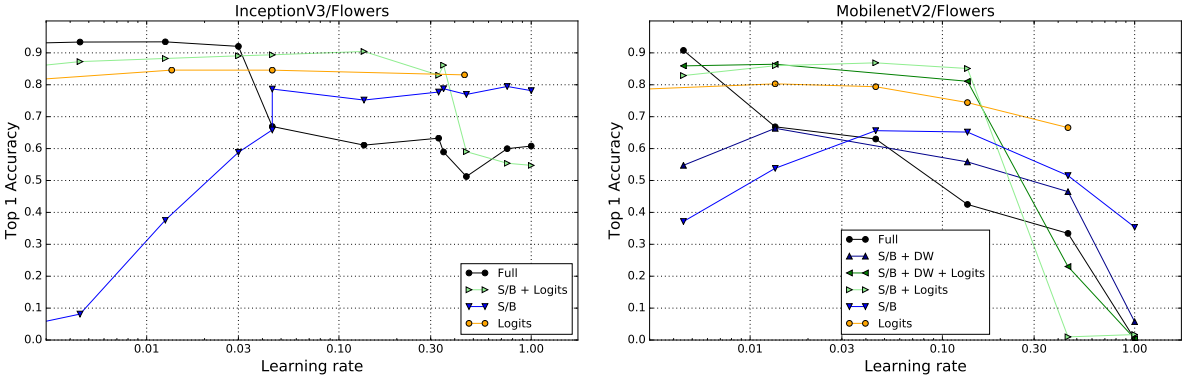

Figure 4.6: **Effect of learning rate**: Final accuracy as a function of learning rate. Note how full fine-tuning requires learning rate to be small, while bias/scale tuning requires learning rate to be large enough.

Generally, we did not see a large variation in training speed. All fine-tuning approaches needed 50-200K steps depending on the learning rate and the training method. While different approaches definitely differ in the number of steps necessary for convergence, we find these changes to be comparable to changes in other hyperparameters such as learning rate.

4.4.3 Multi-Task Learning

In this section we show that, when using model-specific patches during multi-task training, it leads to performance comparable to that of independently trained models, while essentially using a single model.

We simultaneously train MobilenetV2 [153] on two large datasets: ImageNet and Places365. Although the network architecture is the same for both datasets, each model has its own private patch that, along with the rest of the model weights constitutes the model for that dataset. We choose a combination of the scale-and-bias patch, and the last layer as the private model patch in this experiment. The rest of the weights are shared and receive gradient updates from all tasks.

In order to inhibit one task from dominating the learning of the weights, we ensure that the learning rates for different tasks are comparable at any given point in time. This is achieved by setting hyperparameters such that the number of batches processed for each learning rate decay step is the same for all tasks[3] We assign the same number of workers for each task in the distributed learning environment. The results are shown in Table 4.5.

Table 4.5: **Multitask learning results**: Multi-task learning with MobilenetV2 on ImageNet and Places-365.

Task	S/B patch + last layer	Last layer	Independently trained
Imagenet	70.2%	64.4%	71.8%
Places365	**54.3**%	51.4%	54.2%
# total parameters	3.97M	3.93M	6.05M

Multi-task validation accuracy using a separate S/B patch for each model, is comparable to single-task accuracy, while considerably better than the setup that only uses separate logit-layer for each task, while using only using 1% more parameters (and 50% less than the independently trained setup).

4.4.4 Domain Adaptation

In this experiment, each task corresponds to performing classification of ImageNet images at a different resolution. This problem is of great practical importance because it allows one

3. This is done by adjusting the number of epochs per decay for each task to match that of ImageNet.

to build very compact set of models that can operate at different speeds that can be chosen at inference time depending on power and latency requirements. Unlike in Section 4.4.3, we only have the scale-and-bias patch private to each task; the last layer weights are shared. We use bilinear interpolation to scale images before feeding them to the model. The learning rate schedule is the same as in Section 4.4.3.

The results are shown in Table 4.6. We compare our approach with S/B patch only against two baseline setups. *All shared* is where all parameters are shared across all models and *individually trained* is a much more expensive setup where each resolution has its own model. As can be seen from the table, scale-and-bias patch allows to close the accuracy gap between these two setups and even leads to a slight increase of accuracy for a couple of the models at the cost of 1% of extra parameters per each resolution.

Table 4.6: **Domain adaptation results**: Multi-task accuracies of 5 MobilenetV2 models acting at different resolutions on ImageNet.

Image resolution	S/B patch	All shared	Independently trained
96 x 96	60.3%	52.6%	60.3%
128 x 128	**66.3**%	62.4%	65.3%
160 x 160	**69.5**%	67.2%	68.8%
192 x 192	71%	69.4%	70.7%
224 x 224	71.8%	70.6%	71.8%
# total parameters	**3.7M**	3.5M	17.7M

An application of domain adaptation using model patches is cost-efficient model cascades. We employ the algorithm from [163] which takes several models (of varying costs) performing the same task, and determines a cascaded model with the same accuracy as the best task but lower average cost. Applying it to MobilenetV2 models on multiple resolutions that we trained via multi-task learning, we are able to lower the average cost of MobilenetV2 inference by 15.2%. Note that, in order to achieve this, we only need to store 5% more model parameters than for a single model.

CHAPTER 5

CONCLUSION

The new era in machine learning has brought new challenges. We have highlighted three such challenges in this thesis: 1) efficient manipulation of large sparse high-rank matrices, 2) analyzing and interpreting deep neural network models, and 3) parameter-efficient deployment of machine learning models on embedded devices.

In Chapter 2, we have presented results supporting the thesis that matrix factorization based on exploiting multiscale structure is a practical general-purpose alternative to low-rank-based methods in reducing the dimensionality of real-world matrices. The factorization methods presented in this thesis are approximations, and a natural direction forward would be to develop an error analysis. For this purpose, as well as for general research interest, it may be beneficial to develop the theory behind asymmetric MMF and establish a stronger connection to harmonic analysis. On the practical side, MMF can be used for matrices arising as weights in deep neural networks – for either compression, or to extract insights. We predict that multiscale methods would become more important in the future as datasets continue to grow.

Chapter 3 showcased various kinds of vulnerabilities of deep neural network models. We have discovered that question answering neural networks often ignore important input words, and that pruning the training dataset may be critical for better generalizability. Attributions are proposed as a powerful tool in being able to attack and understand deep neural network models. While most of our analysis has been human-in-the-loop, with advances in language models and neural language generation, it may be possible to automate this process, which would have a great impact on the advancement of research in natural language processing. However, developing tools and techniques that aid the researcher in improving deep neural networks would still be beneficial in both the short and long term. At the moment, we are seeing a transition in the field from focusing more on idea generation, to informed and

analytical improvement of the developed ideas. The cost of performing research in deep neural networks is high, as it requires a lot of computational power that costs both money as well as has a big impact on the environment. Being able to automate the incremental improvement of deep neural networks would become very important, if not necessary, in the future.

Chapter 4 proposed a novel learning paradigm that allows multiple deep neural network models to share parameters while maintaining high accuracy on their respective tasks. The space of deep learning applications is ever increasing, and there is still a great need for developing methods for deploying models on embedded devices. Exploring various types of model patches is a natural step forward in this research. From a theoretical point of view, it would be great to understand exactly how model patches work, which may also shed light on the capabilities and capacities of deep networks. The model-patch-based learning paradigm is also useful for multitask learning, which would be a critical component of artificial general intelligence.

Appendices

APPENDIX A

CLASSICAL ORTHOGONAL WAVELET TRANSFORMS

We give a brief introduction to classical orthogonal wavelet transforms. For a detailed introduction, see [34]. Assuming $n = 2^N$ for simplicity, the L-level wavelet transform of a signal $x \in \mathbb{R}^n$ can be written as a matrix vector product $W^T x$, where

$$W = W_1 W_2 \dots W_L \tag{A.1}$$

with $L \leq N$ and

$$W_k^T = \begin{pmatrix} U_k & 0 \\ V_k & 0 \\ 0 & I_{n-\frac{n}{2^{k-1}}} \end{pmatrix} \qquad k = 1, \dots, L, \tag{A.2}$$

where $U_k, V_k \in \mathbb{R}^{(n/2^k)\times(n/2^{k-1})}$ are of the form

$$U_k = \begin{pmatrix} h_0 & h_1 & h_2 & \cdots & h_{m-1} & & & \\ & & h_0 & h_1 & h_2 & \cdots & h_{m-1} & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ h_2 & \cdots & h_{m-1} & & & & h_0 & h_1 \end{pmatrix},$$

$$V_k = \begin{pmatrix} g_0 & g_1 & g_2 & \cdots & g_{m-1} & & & \\ & & g_0 & g_1 & g_2 & \cdots & g_{m-1} & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ g_2 & \cdots & g_{m-1} & & & & g_0 & g_1 \end{pmatrix}.$$

The scalars h_i, g_i for $i = 0, \ldots, m-1$ are the high-pass and low-pass filter coefficients of the wavelet transform, respectively. The above holds true even when $n = p2^s$ for some s and p. In that case, the maximum level of the wavelet transform applied is upper bounded by s.

On higher dimensional signals, wavelet transforms are applied dimension-wise. For example, let $x \in \mathbb{R}^{n^2}$ be a 2D signal (matrix) which has been vectorized by stacking the columns. The wavelet transform $\widetilde{x}$ is computed by first applying a 1D transform on the columns and then on the rows. If $W \in \mathbb{R}^{n \times n}$ is the 1D orthogonal wavelet transform matrix, then

$$\widetilde{x} = (I_n \otimes W^T)(W^T \otimes I_n)\, x = (W \otimes W)^T x,$$

where $\otimes$ is the Kronecker product [174] and I_n, the $n \times n$ identity matrix. Thus, $W \otimes W$ can be called the two dimensional wavelet transform matrix. For vectorized 3D signals (tensors), the wavelet transform matrix is $W \otimes W \otimes W$.

APPENDIX B

RULE-BASED SYSTEMS FOR QUESTION ANSWERING

We now describe the structure of rule-based systems based on semantic parsing. See [139] and [40] for detailed description of this method.

A semantic parsing system uses a manually constructed grammar to map a natural language question to a logical form. The logical form is then evaluated on the table data to get the final answer. It has following components:

- Matching words from question to columns in the table. This might be implemented using syntactic matches (string comparison or something more complicated based on edit distance or stemming) or semantic matches (via use of synonyms from a thesaurus. e.g. country $\rightarrow$ nation). These matches provide the set of operands in the logical form.
- A list of operators with corresponding trigger words (similar to Table 3.7). The trigger words that match question words imply the set of operators to be used in the logical form. Additionally, a list of stop-words is used to account for the fluff words in the question.
- A (context-free) grammar. This helps us decide which operands bind to which operators by using linguistic structure present in the question; e.g. a pattern like 'more than 10 `column`' should map to $geq(column, 10)$.
- Semantic predicates to check the validity of the logical form. An example predicate would check that `max` operator is applied only to numerical columns.

Qualitative Differences Between Neural Programmer and Rule-based Systems

We observed the following differences between the two systems.

- The rule-based system described above is limited in producing either an operand or an operator for a given question word; whereas NP can produce both from a single word. e.g. NP can map longest to operator MAX and column length. While the rule-based system has to pick one or the other.

- NP learns semantic word matches from the data; e.g. synonyms like country → nation. These need to be manually added to the rule-based system.

- Rule-based system is immune to addition or removal of fluff words from the question. It works based on the question words that match either the table entries or trigger words.

- Table shuffling doesn't change the logical form generated by the rule-based system.

- NP can produce programs that get the right answer using a shortcut. e.g. For "which nation earned the most gold medals", NP selects `first` instead of `max`, implicitly relying on the table being sorted. The rule-based system produces logical forms that include all the operators and operands mentioned in the question and hence does not rely on shortcuts.

APPENDIX C

ADDITIONAL RESULTS ON MULTIRESOLUTION PRECONDITIONING

Table C.1: **Iteration counts for preconditioning off-the-shelf matrices**: Iteration counts of GMRES solved to a relative error of 10^{-4}. $\times$ indicates that the method did not achieve the desired tolerance within 500 iterations.

Dataset	n	no prec.	IWSPAI	MMF prec.
nd3k	8192	455	**236**	323
nemeth03	9216	4	4	**2**
net25	9216	**460**	$\times$	$\times$
fv2	9216	**20**	**20**	27
fv3	9216	42	**38**	52
nemeth12	9216	13	10	**3**
nemeth11	9216	10	8	**3**
nemeth09	9216	7	6	**3**
nemeth14	9216	$\times$	$\times$	**8**
nemeth04	9216	5	4	**3**
nemeth23	9216	**211**	$\times$	$\times$
pf2177	9216	**174**	$\times$	$\times$
bloweybq	9216	$\times$	**8**	$\times$
nemeth10	9216	8	7	**3**
flowmeter0	9216	$\times$	$\times$	**9**
nemeth25	9216	**164**	$\times$	$\times$
nemeth24	9216	**179**	$\times$	$\times$
nemeth15	9216	282	$\times$	**70**
nopoly	10240	119	108	**105**

bcsstk17	10240	×	×	**266**
bundle1	10240	×	×	**30**
linverse	11264	×	**20**	×
t2dah	11264	×	×	**7**
crystm02	13312	**1**	**1**	30
Pres_Poisson	14336	436	**43**	114
bcsstm25	14336	×	×	**2**
gyro_m	16384	**1**	**1**	115
gyro_k	16384	×	×	**220**
nd6k	16384	×	**270**	330
bodyy4	16384	184	147	**91**
t3dl_a	18432	×	141	**6**
Si5H12	18432	103	**71**	89
Trefethen_20000b	18432	×	×	**8**
crystm03	24576	**1**	**1**	33
spmsrtls	28672	×	**150**	×
wathen100	28672	×	×	**33**
wathen120	32768	×	×	**33**
mario001	36864	269	×	×
torsion1	36864	41	**29**	50
bfly	49152	**59**	×	×
crankseg_2	57344	×	×	**246**
Ga3As3H12	57344	×	×	**104**
cant	57344	×	×	**83**
	# wins	10	13	23

Table C.2: **Preconditioning results on large off-the-shelf matrices**: Iteration counts of GMRES (30) with tolerance = 10^{-9} on off-the-shelf sparse matrices

Name	Group	N	NNZ	NoPrec	MMFPrec
delaunay_n13	DIMACS10	8192	49094	×	×
c-37	Schenk_IBMNA	8204	74676	×	×
ft01	Okunbor	8205	125567	2	×
benzene	PARSEC	8219	242669	×	×
hep-th	Newman	8361	32253	×	×
bcsstk33	HB	8738	591904	×	122
nd3k	ND	9000	3279690	×	×
G66	Gset	9000	36000	×	×
3elt_dual	AG-Monien	9000	26556	×	×
vsp_data_and_seymourl	DIMACS10	9167	111732	×	×
c-39	Schenk_IBMNA	9271	116587	×	×
nemeth09	Nemeth	9506	395506	11	11
nemeth05	Nemeth	9506	394808	7	7
nemeth13	Nemeth	9506	474472	29	×
nemeth24	Nemeth	9506	1506550	7	×
nemeth16	Nemeth	9506	587012	77	×
nemeth15	Nemeth	9506	539802	483	×
nemeth26	Nemeth	9506	1511760	5	×
nemeth12	Nemeth	9506	446818	20	20
nemeth23	Nemeth	9506	1506810	11	×
nemeth07	Nemeth	9506	394812	9	9
nemeth21	Nemeth	9506	1173746	26	×

nemeth08	Nemeth	9506	394816	10	10
nemeth22	Nemeth	9506	1358832	17	×
nemeth01	Nemeth	9506	725054	×	×
nemeth10	Nemeth	9506	401448	13	13
nemeth19	Nemeth	9506	818302	46	×
nemeth18	Nemeth	9506	695234	57	×
nemeth06	Nemeth	9506	394808	8	8
nemeth25	Nemeth	9506	1511758	5	×
nemeth03	Nemeth	9506	394808	7	6
nemeth11	Nemeth	9506	408264	16	16
nemeth14	Nemeth	9506	496144	319	×
nemeth04	Nemeth	9506	394808	7	7
nemeth20	Nemeth	9506	971870	37	×
nemeth17	Nemeth	9506	629620	62	×
nemeth02	Nemeth	9506	394808	6	6
net25	Andrianov	9520	401200	437	×
fv1	Norris	9604	85264	30	×
flowmeter0	Oberwolfach	9669	67391	×	×
pf2177	Andrianov	9728	725144	×	21
c-41	Schenk_IBMNA	9769	101635	×	×
TSC_OPF_300	IPSO	9774	820783	×	×
whitaker3	AG-Monien	9800	57978	×	11
fv3	Norris	9801	87025	×	×
fv2	Norris	9801	87025	30	×
ca-HepTh	SNAP	9877	51971	×	×
c-40	Schenk_IBMNA	9941	81501	×	×

G67	Gset	10000	40000	×	×
bloweybq	GHS_indef	10001	49999	×	×
crack	AG-Monien	10240	60760	×	6
hangGlider_3	VDOL	10260	92703	×	×
sit100	GHS_indef	10262	61046	×	×
shuttle_eddy	Pothen	10429	103599	×	×
c-42	Schenk_IBMNA	10471	110285	×	×
vsp_p0291_seymourl_iias	DIMACS10	10498	107736	×	×
bundle1	Lourakis	10581	770811	326	57
ed_B_unscaled	Bindel	10605	144579	12	×
ed_B	Bindel	10605	144579	×	33
PGPgiantcompo	Arenas	10680	48632	×	×
c-44	Schenk_IBMNA	10728	85000	×	×
nopoly	Gaertner	10774	70842	×	×
TSOPF_FS_b162_c1	TSOPF	10798	608540	×	14
pkustk02	Chen	10800	810000	×	15
sc10848	Boeing	10848	1229776	×	×
rajat06	Rajat	10922	46983	×	5
wing_nodal	DIMACS10	10937	150976	×	×
bcsstk17	HB	10974	428650	×	×
vsp_c-30_data_d	DIMACS10	11023	124368	×	×
CurlCurl_0	Bodendiek	11083	113343	×	×
3plates	Cunningham	11107	11105	33	2
c-43	Schenk_IBMNA	11125	123659	×	×
fe_4elt2	DIMACS10	11143	65636	×	×
2dah_	Oberwolfach	11445	176117	×	×

2dah_e	Oberwolfach	11445	176117	×	58
2dah	Oberwolfach	11445	176117	×	×
Oregon-1	SNAP	11492	47127	×	×
Oregon-2	SNAP	11806	65804	×	×
bcsstk18	HB	11948	149090	×	×
linverse	GHS_indef	11999	95977	×	×
ca-HepPh	SNAP	12008	237010	×	×
ncvxqp1	GHS_indef	12111	73963	×	×
vibrobox	Cote	12328	301700	×	×
stokes64s	GHS_indef	12546	140034	×	×
stokes64	GHS_indef	12546	140034	×	×
skir	Pothen	12598	196520	×	9
uma2	GHS_indef	12992	49365	×	9
c-45	Schenk_IBMNA	13206	174452	×	×
Reuters911	Pajek	13332	296094	×	×
lowThrust_4	VDOL	13562	160947	×	×
cbuckle	TKK	13681	676515	×	×
cyl6	TKK	13681	714241	×	3
pcrystk02	Boeing	13965	968583	×	92
crystk02	Boeing	13965	968583	×	×
crystm02	Boeing	13965	322905	137	47
bcsstk29	HB	13992	619488	×	×
vsp_befref_fxm_2_4_air02	DIMACS10	14109	196448	×	×
case9	QY	14454	147972	×	25
TSOPF_FS_b9_c6	TSOPF	14454	147972	×	24
Pres_Poisson	ACUSIM	14822	715804	×	×

rajat07	Rajat	14842	63913	×	11
c-46	Schenk_IBMNA	14913	130397	×	×
c-47	Schenk_IBMNA	15343	211401	×	×
OPF_3754	IPSO	15435	141478	×	122
bcsstm25	HB	15439	15439	×	2
bcsstk25	HB	15439	252241	×	×
opt1	GHS_psdef	15449	1930655	×	20
hangGlider_4	VDOL	15561	149532	×	×
barth5	Pothen	15606	107362	×	×
hangGlider_5	VDOL	16011	155246	×	×
Dubcova1	UTEP	16129	253009	320	×
olafu	Simon	16146	1015156	×	×
lowThrust_5	VDOL	16262	198369	×	×
net50	Andrianov	16320	945200	×	×
delaunay_n14	DIMACS10	16384	98244	×	×
fe_sphere	DIMACS10	16386	98304	×	×
ncvxqp9	GHS_indef	16554	54040	51	9
pds10	GHS_psdef	16558	149658	×	×
stro-ph	Newman	16706	243162	×	×
cond-	Newman	16726	95650	×	×
ex3sta1	Andrianov	16782	678998	×	5
gupta3	Gupta	16783	9323427	×	5
ramage02	GHS_psdef	16830	2866352	×	15
cti	DIMACS10	16840	96464	×	×
pkustk07	Chen	16860	2418804	×	29
lowThrust_6	VDOL	16928	207349	×	×

Si10H16	PARSEC	17077	875923	×	×
copter1	GHS_psdef	17222	211064	×	×
gyro	Oberwolfach	17361	1021159	×	×
gyro_k	Oberwolfach	17361	1021159	×	×
gyro_	Oberwolfach	17361	340431	×	×
lowThrust_7	VDOL	17378	211561	×	×
cvxqp3	GHS_indef	17500	114962	×	×
bodyy4	Pothen	17546	121550	279	213
lowThrust_8	VDOL	17702	216445	×	×
L-9	AG-Monien	17983	71192	×	×
nd6k	ND	18000	6897316	×	×
crplat2	DNVS	18010	960946	×	×
lowThrust_9	VDOL	18044	219589	×	×
lowThrust_10	VDOL	18260	222005	×	×
c-48	Schenk_IBMNA	18354	166080	×	×
lowThrust_11	VDOL	18368	223801	×	×
ndem_vtx	Pothen	18454	253350	×	6
lowThrust_12	VDOL	18458	224593	×	×
lowThrust_13	VDOL	18476	224897	×	×
bodyy5	Pothen	18589	128853	×	×
ford1	GHS_psdef	18728	101576	×	5
ca-AstroPh	SNAP	18772	396160	×	×
fxm4_6	Andrianov	18892	497844	×	153
whitaker3_dual	AG-Monien	19190	57162	×	×
pattern1	Andrianov	19242	9323432	×	×
rajat08	Rajat	19362	83443	×	17

bodyy6	Pothen	19366	134208	×	×
raefsky4	Simon	19779	1316789	×	×
Si5H12	PARSEC	19896	738598	×	×
LFAT5000	Oberwolfach	19994	79966	×	13
LF10000	Oberwolfach	19998	99982	×	13
Trefethen_20000b	JGD_Trefethen	19999	554435	×	11
qpband	GHS_indef	20000	45000	9	×
Trefethen_20000	JGD_Trefethen	20000	554466	×	12
crack_dual	AG-Monien	20141	60086	×	2
rail_20209	Oberwolfach	20209	139233	×	×
3dl_e	Oberwolfach	20360	20360	×	2
3dl	Oberwolfach	20360	509866	×	×
3dl_	Oberwolfach	20360	509866	×	×
syl201	DNVS	20685	2454957	×	×
c-49	Schenk_IBMNA	21132	157040	×	×
ube1	TKK	21498	897056	×	×
ube2	TKK	21498	897056	×	×
biplane-9	AG-Monien	21701	84076	×	×
pkustk01	Chen	22044	979380	×	×
rdhei	DNVS	22098	1935324	×	×
pkustk08	Chen	22209	3226671	×	145
c-50	Schenk_IBMNA	22401	180245	×	×
cs4	DIMACS10	22499	87716	×	×
pli	Li	22695	1350309	×	×
s-22july06	Newman	22963	96872	×	×
uma1	GHS_indef	22967	87760	×	28

bcsstk36	Boeing	23052	1143140	×	×
sc23052	Boeing	23052	1142686	×	×
bcsstm36	Boeing	23052	331484	×	169
net75	Andrianov	23120	1489200	×	×
ca-CondM	SNAP	23133	186936	×	×
c-51	Schenk_IBMNA	23196	203048	×	×
c-52	Schenk_IBMNA	23948	202708	5	×
stufe-10	AG-Monien	24010	92828	×	6
de2010	DIMACS10	24115	116056	×	×
ug3d	GHS_indef	24300	69984	267	×
rajat09	Rajat	24482	105573	×	23
pcrystk03	Boeing	24696	1751178	×	×
crystk03	Boeing	24696	1751178	×	×
crystm03	Boeing	24696	583770	138	50
dtoc	GHS_indef	24993	69972	×	×
hi2010	DIMACS10	25016	124126	×	460
ri2010	DIMACS10	25181	125750	×	×
bcsstm37	Boeing	25503	27021	×	128
bcsstk37	Boeing	25503	1140977	×	×
s	TKK	25710	3749582	×	×
brainpc2	GHS_indef	27607	179395	×	11
bratu3d	GHS_indef	27792	173796	×	×
TSOPF_FS_b39_c7	TSOPF	28216	730080	×	13
bcsstk30	HB	28924	2043492	×	341
ug2d	GHS_indef	29008	76832	×	×
TSOPF_FS_b300_c1	TSOPF	29214	4400122	×	×

TSOPF_FS_b300	TSOPF	29214	4400122	×	×
hread	DNVS	29736	4444880	×	×
OPF_6000	IPSO	29902	274697	×	×
net100	Andrianov	29920	2033200	×	×
spmsrtls	GHS_indef	29995	229947	×	×
bloweybl	GHS_indef	30003	110000	×	×
blowey	GHS_indef	30004	150009	×	17
ug2dc	GHS_indef	30200	80000	×	×
rajat10	Rajat	30202	130303	×	31
c-53	Schenk_IBMNA	30235	355139	×	×
bcsstk35	Boeing	30237	1450163	×	×
bcsstm35	Boeing	30237	35050	×	21
big_dual	AG-Monien	30269	89858	×	10
wathen100	GHS_psdef	30401	471601	×	51
TSOPF_FS_b162_c3	TSOPF	30798	1801300	×	×
cond-mat-2003	Newman	31163	240761	×	×
c-54	Schenk_IBMNA	31793	385987	×	×
gupta1	Gupta	31802	2164210	×	×
helm3d01	GHS_indef	32226	428444	×	×
lpl1	Andrianov	32460	328036	×	×
vt2010	DIMACS10	32580	155598	×	×
rgg_n_2_15_s0	DIMACS10	32768	320482	×	4
delaunay_n15	DIMACS10	32768	196548	×	16
se	AG-Monien	32768	98300	×	14
c-55	GHS_indef	32780	403450	×	×
SiO	PARSEC	33401	1317655	×	×

pkustk09	Chen	33960	1583640	×	×
ship_001	DNVS	34920	3896496	×	×
ug3dcqp	GHS_indef	35543	128115	×	×
bcsstk31	HB	35588	1181416	×	77
c-56	Schenk_IBMNA	35910	380240	×	×
nd12k	ND	36000	14220946	×	×
pdb1HYS	Williams	36417	4344765	×	×
wathen120	GHS_psdef	36441	565761	×	50
shock-9	AG-Monien	36476	142580	×	×
pw	GHS_psdef	36519	326107	×	5
email-Enron	SNAP	36692	367662	×	×
net125	Andrianov	36720	2577200	×	×
pkustk05	Chen	37164	2205144	×	×
finance256	GHS_psdef	37376	298496	×	12
c-58	GHS_indef	37595	552551	×	×
c-57	Schenk_IBMNA	37833	403373	×	×
rio001	GHS_indef	38434	204912	×	8
vsp_south31_slptsk	DIMACS10	39668	379828	×	×
jnlbrng1	GHS_psdef	40000	199200	137	×
obstclae	GHS_psdef	40000	197608	60	×
orsion1	GHS_psdef	40000	197608	60	×
case39	QY	40216	1042160	×	×
cond-mat-2005	Newman	40421	352226	×	×
TSOPF_FS_b162_c4	TSOPF	40798	2398220	×	122
insurfo	GHS_psdef	40806	203622	91	×
c-59	GHS_indef	41282	480536	×	×

c-62	Schenk_IBMNA	41731	559341	×	×
c-62ghs	GHS_indef	41731	559339	×	×
pkustk06	Chen	43164	2571768	×	×
net150	Andrianov	43520	3121200	×	×
c-61	Schenk_IBMNA	43618	310016	×	×
c-60	Schenk_IBMNA	43640	298570	×	×
OPF_10000	IPSO	43887	426898	×	×
c-63	GHS_indef	44234	434704	×	×
bcsstk32	HB	44609	2014701	×	3
fe_body	DIMACS10	45087	327780	×	5
k2010	DIMACS10	45292	217114	×	×
3dtube	Rothberg	45330	3213618	×	×
bcsstk39	Boeing	46772	2060662	×	×
bcsstm39	Boeing	46772	46772	×	2
vanbody	GHS_psdef	47072	2329056	×	×
c-65	Schenk_IBMNA	48066	360428	×	×
nh2010	DIMACS10	48837	234550	×	×
gridgen	GHS_psdef	48962	512084	×	×
cc	AG-Monien	49152	139264	×	×
ccc	AG-Monien	49152	147456	×	×
bfly	AG-Monien	49152	196608	152	×
stokes128	GHS_indef	49666	558594	×	×
c-66b	Schenk_IBMNA	49989	444851	×	×
c-66	Schenk_IBMNA	49989	444853	×	×
sparsine	GHS_indef	50000	1548988	×	×
cvxbqp1	GHS_psdef	50000	349968	×	×

ncvxbqp1	GHS_indef	50000	349968	×	×
c-64	Schenk_IBMNA	51035	707985	×	×
c-64b	Schenk_IBMNA	51035	707601	×	×
dawson5	GHS_indef	51537	1010777	×	×
pct20stif	Boeing	52329	2698463	×	×
ct20stif	Boeing	52329	2600295	×	×
dictionary28	Pajek	52652	191400	×	×
crankseg_1	GHS_psdef	52804	10614210	×	×
struct3	Rothberg	53570	1173694	×	58
nasasrb	Nasa	54870	2677324	×	×
srb1	GHS_psdef	54924	2962152	×	×
copter2	GHS_psdef	55476	759952	×	18
pkustk04	Chen	55590	4218660	×	390
TSOPF_FS_b300_c2	TSOPF	56814	8767466	×	244
c-67b	Schenk_IBMNA	57975	530583	×	×
c-67	Schenk_IBMNA	57975	530229	×	×
dixmaanl	GHS_indef	60000	299998	×	×
Andrews	Andrews	60000	760154	181	176
60k	DIMACS10	60005	178880	×	×
5esindl	GHS_indef	60008	255004	×	×
blockqp1	GHS_indef	60012	640033	10	×
Ga3As3H12	PARSEC	61349	5970947	×	×
GaAsH6	PARSEC	61349	3381809	×	×
wing	DIMACS10	62032	243088	×	5
gupta2	Gupta	62064	4248286	×	184
can	Williams	62451	4007383	×	×

ncvxqp5	GHS_indef	62500	424966	×	3
brack2	AG-Monien	62631	733118	×	×
pkustk03	Chen	63336	3130416	×	×
crankseg_2	GHS_psdef	63838	14148858	×	×
c-68	GHS_indef	64810	565996	×	×
Dubcova2	UTEP	65025	1030225	×	×
rgg_n_2_16_s0	DIMACS10	65536	684258	×	×
delaunay_n16	DIMACS10	65536	393150	×	16
qa8f	Cunningham	66127	1660579	75	60
qa8fk	Cunningham	66127	1660579	×	×
gas_sensor	Oberwolfach	66917	1703365	×	9
H2O	PARSEC	67024	2216736	×	×
c-69	GHS_indef	67458	623914	×	×
ct2010	DIMACS10	67578	336352	×	28
k1_san	GHS_indef	67759	559775	×	×
c-70	GHS_indef	68924	658986	×	×
e2010	DIMACS10	69518	335476	×	×
cfd1	Rothberg	70656	1825580	×	×
F2	Koutsovasilis	71505	5294285	×	×
oilpan	GHS_psdef	73752	2148558	×	×
finan512	Mulvey	74752	596992	54	27
pfinan512	Mulvey	74752	596992	×	13
ncvxqp3	GHS_indef	75000	499964	×	7
TSOPF_FS_b39_c19	TSOPF	76216	1977600	×	×
c-71	GHS_indef	76638	859520	×	×
fe_tooth	DIMACS10	78136	905182	×	×

3dh_e	Oberwolfach	79171	4352105	×	×
3dh	Oberwolfach	79171	4352105	×	9
3dh_	Oberwolfach	79171	4352105	×	7
rail_79841	Oberwolfach	79841	553921	×	×
0nsdsil	GHS_indef	80016	355034	×	×
2nnsnsl	GHS_indef	80016	347222	×	×
cont-201	GHS_indef	80595	438795	×	14
pkustk10	Chen	80676	4308984	×	×
pache1	GHS_psdef	80800	542184	×	×
shallow_water2	MaxPlanck	81920	327680	35	×
shallow_water1	MaxPlanck	81920	327680	19	×
hermal1	Schmid	82654	574458	×	×
consph	Williams	83334	6010480	×	×
c-72	GHS_indef	84064	707546	×	×
TSOPF_FS_b300_c3	TSOPF	84414	13135930	×	2
nv2010	DIMACS10	84538	416998	×	×
onera_dual	Pothen	85567	419201	×	3
wy2010	DIMACS10	86204	427586	×	28
ncvxqp7	GHS_indef	87500	574962	×	9
pkustk11	Chen	87804	5217912	×	×
olesnik0	GHS_indef	88263	744216	×	11
net4-1	Andrianov	88343	2441727	×	×
sd2010	DIMACS10	88360	410722	×	×
denormal	Castrillon	89400	1156224	×	×
s3dkt3m2	GHS_psdef	90449	3686223	×	×
s3dkq4m2	GHS_psdef	90449	4427725	×	×

s4dkt3m2	TKK	90449	3753461	×	×
boyd1	GHS_indef	93279	1211231	×	×
ndem_dual	Pothen	94069	460493	×	28
pkustk12	Chen	94653	7512317	×	×
pkustk13	Chen	94893	6616827	×	×
Si34H36	PARSEC	97569	5156379	×	×
_t1	DNVS	97578	9753570	×	×
fe_rotor	DIMACS10	99617	1324862	×	×
G_n_pin_pou	DIMACS10	100000	1002401	×	×
preferentialAttachmen	DIMACS10	100000	999970	×	×
smallworld	DIMACS10	100000	999996	×	×
ford2	GHS_psdef	100196	544688	×	24
2cubes_sphere	Um	101492	1647264	×	36
hermomech_TK	Botonakis	102158	711558	×	×
hermomech_TC	Botonakis	102158	711558	76	23
filter3D	Oberwolfach	106437	2707179	×	13
x104	DNVS	108384	8713602	×	×
598	DIMACS10	110971	1483868	×	×
Ge87H76	PARSEC	112985	7892195	×	×
Ge99H100	PARSEC	112985	8451395	×	×
Ga10As10H30	PARSEC	113081	6115633	×	×
luxembourg_os	DIMACS10	114599	239332	×	×
shipsec8	DNVS	114919	3303553	×	×
ut2010	DIMACS10	115406	572066	×	59
TSOPF_FS_b39_c30	TSOPF	120216	3121160	×	×
cop20k_A	Williams	121192	2645680	×	×

ship_003	DNVS	121728	3777036	×	×
cfd2	Rothberg	123440	3085406	×	×
usroads-48	Gleich	126146	323900	×	×
boneS01	Oberwolfach	127224	5516602	×	×
usroads	Gleich	129164	330870	×	×
rgg_n_2_17_s0	DIMACS10	131072	1457508	×	×
delaunay_n17	DIMACS10	131072	786352	×	24
2010	DIMACS10	132288	638668	×	×
Ga19As19H42	PARSEC	133123	8884839	×	×
nd2010	DIMACS10	133769	625946	×	×
wv2010	DIMACS10	135218	662922	×	14
shipsec1	DNVS	140874	3568176	×	×
bmw7st_1	GHS_psdef	141347	7318399	×	×
fe_ocean	DIMACS10	143437	819186	×	×
engine	TKK	143571	4706073	×	×
144	DIMACS10	144649	2148786	×	×
d2010	DIMACS10	145247	700378	×	×
Dubcova3	UTEP	146689	3636643	×	×
bmwcra_1	GHS_psdef	148770	10641602	×	×
id2010	DIMACS10	149842	728264	×	×
G2_circui	AMD	150102	726674	×	×
pkustk14	Chen	151926	14836504	×	×
gearbox	Rothberg	153746	9080404	×	×
SiO2	PARSEC	155331	11283503	×	×
wave	AG-Monien	156317	2118662	×	×
2010	DIMACS10	157508	776610	×	26

ky2010	DIMACS10	161672	787778	×	20
nm2010	DIMACS10	168609	830970	×	30
c-73	Schenk_IBMNA	169422	1279274	×	×
nj2010	DIMACS10	169588	829912	×	×
s2010	DIMACS10	171778	839980	×	×
shipsec5	DNVS	179860	4598604	×	×
cont-300	GHS_indef	180895	988195	×	15
sc2010	DIMACS10	181908	893160	×	×
d_pretok	GHS_indef	182730	1641672	×	×
Si41Ge41H72	PARSEC	185639	15011265	×	×
r2010	DIMACS10	186211	904310	×	×
uron_	GHS_indef	189924	1690876	×	×
caidaRouterLevel	DIMACS10	192244	1218132	×	×
ne2010	DIMACS10	193352	913854	×	21
wa2010	DIMACS10	195574	947432	×	15
or2010	DIMACS10	196621	979512	×	×
fullb	DNVS	199187	11708077	×	×
co2010	DIMACS10	201062	974574	×	18
fcondp2	DNVS	201822	11294316	×	×
hermomech_dM	Botonakis	204316	1423116	76	22
la2010	DIMACS10	204447	980634	×	×
roll	DNVS	213453	11985111	×	×
14b	DIMACS10	214765	3358036	×	×
ia2010	DIMACS10	216007	1021170	×	20
pwtk	Boeing	217918	11524432	×	×
hood	GHS_psdef	220542	9895422	×	×

CO	PARSEC	221119	7666057	×	×
halfb	DNVS	224617	12387821	×	×
HTC_336_4438	IPSO	226340	783496	×	×
HTC_336_9129	IPSO	226340	762969	×	×
CurlCurl_1	Bodendiek	226451	2472071	×	×
coAuthorsCiteseer	DIMACS10	227320	1628268	×	×
bmw3_2	GHS_indef	227362	11288630	×	×
ks2010	DIMACS10	238600	1121798	×	24
n2010	DIMACS10	240116	1193966	×	18
Si87H76	PARSEC	240369	10661631	×	×
z2010	DIMACS10	241666	1196094	×	×
BenElechi1	BenElechi	245874	13150496	×	×
l2010	DIMACS10	252266	1230482	×	×
wi2010	DIMACS10	253096	1209404	×	×
Lin	Lin	256000	1766400	×	×
n2010	DIMACS10	259777	1227102	×	×
offshore	Um	259789	4242673	×	×
rgg_n_2_18_s0	DIMACS10	262144	3094569	×	×
delaunay_n18	DIMACS10	262144	1572792	×	×
in2010	DIMACS10	267071	1281716	×	×
citationCiteseer	DIMACS10	268495	2313294	×	×
ok2010	DIMACS10	269118	1274148	×	15
va2010	DIMACS10	285762	1402128	×	×
nc2010	DIMACS10	288987	1416620	×	×
ga2010	DIMACS10	291086	1418056	×	×
3Dspectralwave2	Sinclair	292008	12859992	×	×

coAuthorsDBLP	DIMACS10	299067	1955352	×	×
dblp-2010	LAW	326186	1640936	×	×
i2010	DIMACS10	329885	1578090	×	×
o2010	DIMACS10	343565	1656568	×	×
ny2010	DIMACS10	350169	1709546	×	×
oh2010	DIMACS10	365344	1768240	×	×
darcy003	GHS_indef	389874	2097566	×	×
rio002	GHS_indef	389874	2097566	×	×
helm2d03	GHS_indef	392257	2741935	×	×
pa2010	DIMACS10	421545	2058462	×	×
uto	DIMACS10	448695	6629222	×	×
il2010	DIMACS10	451554	2164464	×	×
fl2010	DIMACS10	484481	2346294	×	×
f_2_k101	Schenk_AFE	503625	17550675	×	×
f_3_k101	Schenk_AFE	503625	17550675	×	×
f_1_k101	Schenk_AFE	503625	17550675	×	×
rgg_n_2_19_s0	DIMACS10	524288	6539536	×	×
delaunay_n19	DIMACS10	524288	3145646	×	×
parabolic_fe	Wissgott	525825	3674625	×	×
			Wins	31	110
			Converged	53	120

Table C.3: **Preconditioning performance by tolerance**: Comparison of no preconditioning and MMFprec for various levels of GMRES(30) tolerance.

Name	N	10^{-3}	10^{-5}	10^{-7}	10^{-9}
delaunay_n13	8192	MMFPrec	MMFPrec	Both	Both
c-37	8204	Both	Both	Both	Both
ft01	8205	NoPrec	NoPrec	NoPrec	NoPrec
benzene	8219	NoPrec	Both	Both	Both
hep-th	8361	MMFPrec	MMFPrec	Both	Both
bcsstk33	8738	MMFPrec	MMFPrec	MMFPrec	MMFPrec
nd3k	9000	NoPrec	NoPrec	Both	Both
G66	9000	MMFPrec	Both	Both	Both
3elt_dual	9000	MMFPrec	MMFPrec	MMFPrec	Both
c-39	9271	NoPrec	Both	Both	Both
nemeth09	9506	Both	Both	Both	Both
nemeth05	9506	MMFPrec	Both	Both	Both
nemeth13	9506	MMFPrec	Both	NoPrec	NoPrec
nemeth24	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth16	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth15	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth26	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth12	9506	MMFPrec	Both	Both	Both
nemeth23	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth07	9506	Both	Both	Both	Both
nemeth21	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth08	9506	Both	Both	Both	Both
nemeth22	9506	NoPrec	NoPrec	NoPrec	NoPrec

nemeth01	9506	NoPrec	NoPrec	NoPrec	Both
nemeth10	9506	Both	Both	MMFPrec	Both
nemeth19	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth18	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth06	9506	Both	Both	Both	Both
nemeth25	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth03	9506	Both	Both	Both	MMFPrec
nemeth11	9506	Both	MMFPrec	MMFPrec	Both
nemeth14	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth04	9506	Both	Both	Both	Both
nemeth20	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth17	9506	NoPrec	NoPrec	NoPrec	NoPrec
nemeth02	9506	Both	Both	Both	Both
net25	9520	NoPrec	NoPrec	NoPrec	NoPrec
fv1	9604	NoPrec	NoPrec	NoPrec	NoPrec
flowmeter0	9669	NoPrec	Both	Both	Both
pf2177	9728	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-41	9769	Both	Both	Both	Both
TSC_OPF_300	9774	MMFPrec	MMFPrec	MMFPrec	Both
whitaker3	9800	MMFPrec	MMFPrec	MMFPrec	MMFPrec
fv3	9801	NoPrec	NoPrec	NoPrec	Both
fv2	9801	NoPrec	NoPrec	NoPrec	NoPrec
ca-HepTh	9877	MMFPrec	MMFPrec	MMFPrec	Both
c-40	9941	Both	Both	Both	Both
G67	10000	MMFPrec	MMFPrec	Both	Both
bloweybq	10001	NoPrec	NoPrec	Both	Both

crack	10240	MMFPrec	MMFPrec	MMFPrec	MMFPrec
hangGlider_3	10260	MMFPrec	Both	Both	Both
sit100	10262	Both	Both	Both	Both
shuttle_eddy	10429	MMFPrec	MMFPrec	Both	Both
c-42	10471	Both	Both	Both	Both
bundle1	10581	MMFPrec	MMFPrec	MMFPrec	MMFPrec
ed_B_unscaled	10605	NoPrec	NoPrec	NoPrec	NoPrec
ed_B	10605	NoPrec	NoPrec	MMFPrec	MMFPrec
PGPgiantcompo	10680	MMFPrec	MMFPrec	Both	Both
c-44	10728	Both	Both	Both	Both
nopoly	10774	NoPrec	NoPrec	Both	Both
TSOPF_FS_b162_c1	10798	MMFPrec	MMFPrec	MMFPrec	MMFPrec
pkustk02	10800	MMFPrec	MMFPrec	MMFPrec	MMFPrec
sc10848	10848	NoPrec	NoPrec	Both	Both
rajat06	10922	MMFPrec	MMFPrec	MMFPrec	MMFPrec
wing_nodal	10937	NoPrec	Both	Both	Both
bcsstk17	10974	MMFPrec	MMFPrec	Both	Both
vsp_c-30_data_d	11023	NoPrec	Both	Both	Both
CurlCurl_0	11083	MMFPrec	MMFPrec	Both	Both
3plates	11107	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-43	11125	Both	Both	Both	Both
fe_4elt2	11143	NoPrec	Both	Both	Both
2dah_	11445	MMFPrec	MMFPrec	Both	Both
2dah_e	11445	MMFPrec	MMFPrec	MMFPrec	MMFPrec
2dah	11445	MMFPrec	MMFPrec	Both	Both
Oregon-1	11492	NoPrec	Both	Both	Both

Oregon-2	11806	NoPrec	Both	Both	Both
bcsstk18	11948	MMFPrec	MMFPrec	Both	Both
linverse	11999	NoPrec	Both	Both	Both
ca-HepPh	12008	Both	Both	Both	Both
ncvxqp1	12111	MMFPrec	MMFPrec	MMFPrec	Both
vibrobox	12328	NoPrec	MMFPrec	Both	Both
stokes64s	12546	MMFPrec	Both	Both	Both
stokes64	12546	MMFPrec	MMFPrec	Both	Both
skir	12598	MMFPrec	MMFPrec	MMFPrec	MMFPrec
uma2	12992	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-45	13206	NoPrec	Both	Both	Both
Reuters911	13332	Both	Both	Both	Both
lowThrust_4	13562	MMFPrec	Both	Both	Both
cbuckle	13681	NoPrec	MMFPrec	Both	Both
cyl6	13681	MMFPrec	MMFPrec	MMFPrec	MMFPrec
pcrystk02	13965	MMFPrec	MMFPrec	MMFPrec	MMFPrec
crystk02	13965	MMFPrec	MMFPrec	Both	Both
crystm02	13965	MMFPrec	MMFPrec	MMFPrec	MMFPrec
bcsstk29	13992	MMFPrec	MMFPrec	MMFPrec	Both
case9	14454	MMFPrec	MMFPrec	MMFPrec	MMFPrec
TSOPF_FS_b9_c6	14454	MMFPrec	MMFPrec	MMFPrec	MMFPrec
Pres_Poisson	14822	MMFPrec	Both	Both	Both
rajat07	14842	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-46	14913	Both	Both	Both	Both
c-47	15343	Both	Both	Both	Both
OPF_3754	15435	MMFPrec	MMFPrec	MMFPrec	MMFPrec

bcsstm25	15439	MMFPrec	MMFPrec	MMFPrec	MMFPrec
bcsstk25	15439	NoPrec	MMFPrec	Both	Both
opt1	15449	MMFPrec	MMFPrec	MMFPrec	MMFPrec
hangGlider_4	15561	MMFPrec	Both	Both	Both
barth5	15606	NoPrec	MMFPrec	Both	Both
hangGlider_5	16011	MMFPrec	MMFPrec	MMFPrec	Both
Dubcova1	16129	NoPrec	NoPrec	NoPrec	NoPrec
olafu	16146	NoPrec	Both	Both	Both
lowThrust_5	16262	MMFPrec	MMFPrec	MMFPrec	Both
net50	16320	NoPrec	NoPrec	NoPrec	Both
delaunay_n14	16384	MMFPrec	Both	Both	Both
fe_sphere	16386	MMFPrec	Both	Both	Both
ncvxqp9	16554	MMFPrec	MMFPrec	NoPrec	MMFPrec
pds10	16558	MMFPrec	MMFPrec	Both	Both
stro-ph	16706	Both	Both	Both	Both
cond-	16726	Both	Both	Both	Both
ex3sta1	16782	MMFPrec	MMFPrec	MMFPrec	MMFPrec
gupta3	16783	MMFPrec	MMFPrec	MMFPrec	MMFPrec
ramage02	16830	MMFPrec	MMFPrec	MMFPrec	MMFPrec
cti	16840	NoPrec	Both	Both	Both
pkustk07	16860	MMFPrec	MMFPrec	MMFPrec	MMFPrec
lowThrust_6	16928	MMFPrec	Both	Both	Both
Si10H16	17077	NoPrec	Both	Both	Both
copter1	17222	NoPrec	Both	Both	Both
gyro	17361	MMFPrec	Both	Both	Both
gyro_k	17361	MMFPrec	Both	Both	Both

gyro_	17361	MMFPrec	MMFPrec	Both	Both
lowThrust_7	17378	Both	Both	Both	Both
cvxqp3	17500	MMFPrec	MMFPrec	Both	Both
bodyy4	17546	MMFPrec	MMFPrec	MMFPrec	MMFPrec
lowThrust_8	17702	Both	Both	Both	Both
L-9	17983	MMFPrec	MMFPrec	MMFPrec	Both
nd6k	18000	NoPrec	NoPrec	Both	Both
crplat2	18010	NoPrec	MMFPrec	Both	Both
lowThrust_9	18044	MMFPrec	Both	Both	Both
lowThrust_10	18260	MMFPrec	Both	Both	Both
c-48	18354	NoPrec	Both	Both	Both
lowThrust_11	18368	MMFPrec	Both	Both	Both
ndem_vtx	18454	MMFPrec	MMFPrec	MMFPrec	MMFPrec
lowThrust_12	18458	MMFPrec	MMFPrec	MMFPrec	Both
lowThrust_13	18476	MMFPrec	Both	Both	Both
bodyy5	18589	MMFPrec	MMFPrec	MMFPrec	Both
ford1	18728	MMFPrec	MMFPrec	MMFPrec	MMFPrec
ca-AstroPh	18772	MMFPrec	MMFPrec	Both	Both
fxm4_6	18892	MMFPrec	MMFPrec	MMFPrec	MMFPrec
whitaker3_dual	19190	MMFPrec	Both	Both	Both
pattern1	19242	NoPrec	Both	Both	Both
rajat08	19362	MMFPrec	MMFPrec	MMFPrec	MMFPrec
bodyy6	19366	MMFPrec	MMFPrec	Both	Both
raefsky4	19779	NoPrec	NoPrec	Both	Both
Si5H12	19896	NoPrec	Both	Both	Both
LFAT5000	19994	MMFPrec	MMFPrec	MMFPrec	MMFPrec

LF10000	19998	MMFPrec	MMFPrec	MMFPrec	MMFPrec
Trefethen_20000b	19999	MMFPrec	MMFPrec	MMFPrec	MMFPrec
qpband	20000	NoPrec	NoPrec	NoPrec	NoPrec
Trefethen_20000	20000	MMFPrec	MMFPrec	MMFPrec	MMFPrec
crack_dual	20141	MMFPrec	MMFPrec	MMFPrec	MMFPrec
rail_20209	20209	NoPrec	Both	Both	Both
3dl_e	20360	MMFPrec	MMFPrec	MMFPrec	MMFPrec
3dl	20360	MMFPrec	MMFPrec	Both	Both
3dl_	20360	MMFPrec	MMFPrec	Both	Both
syl201	20685	NoPrec	Both	Both	Both
c-49	21132	Both	Both	Both	Both
ube1	21498	NoPrec	Both	Both	Both
ube2	21498	NoPrec	Both	Both	Both
biplane-9	21701	NoPrec	Both	Both	Both
pkustk01	22044	NoPrec	Both	Both	Both
rdhei	22098	NoPrec	Both	Both	Both
pkustk08	22209	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-50	22401	NoPrec	Both	Both	Both
cs4	22499	MMFPrec	MMFPrec	MMFPrec	Both
pli	22695	MMFPrec	Both	Both	Both
s-22july06	22963	NoPrec	Both	Both	Both
uma1	22967	MMFPrec	MMFPrec	MMFPrec	MMFPrec
bcsstk36	23052	MMFPrec	MMFPrec	Both	Both
sc23052	23052	MMFPrec	MMFPrec	Both	Both
bcsstm36	23052	MMFPrec	MMFPrec	MMFPrec	MMFPrec
net75	23120	MMFPrec	NoPrec	Both	Both

ca-CondM	23133	Both	Both	Both	Both
c-51	23196	Both	Both	Both	Both
c-52	23948	NoPrec	NoPrec	NoPrec	NoPrec
stufe-10	24010	MMFPrec	MMFPrec	MMFPrec	MMFPrec
de2010	24115	MMFPrec	MMFPrec	Both	Both
ug3d	24300	NoPrec	NoPrec	NoPrec	NoPrec
rajat09	24482	MMFPrec	MMFPrec	MMFPrec	MMFPrec
pcrystk03	24696	NoPrec	Both	Both	Both
crystk03	24696	MMFPrec	MMFPrec	Both	Both
crystm03	24696	MMFPrec	MMFPrec	MMFPrec	MMFPrec
dtoc	24993	Both	Both	Both	Both
hi2010	25016	MMFPrec	MMFPrec	MMFPrec	MMFPrec
ri2010	25181	MMFPrec	MMFPrec	Both	Both
bcsstm37	25503	MMFPrec	MMFPrec	MMFPrec	MMFPrec
bcsstk37	25503	MMFPrec	MMFPrec	Both	Both
s	25710	MMFPrec	Both	Both	Both
brainpc2	27607	MMFPrec	MMFPrec	MMFPrec	MMFPrec
bratu3d	27792	MMFPrec	Both	Both	Both
TSOPF_FS_b39_c7	28216	MMFPrec	MMFPrec	MMFPrec	MMFPrec
bcsstk30	28924	MMFPrec	MMFPrec	MMFPrec	MMFPrec
ug2d	29008	Both	Both	Both	Both
TSOPF_FS_b300_c1	29214	MMFPrec	MMFPrec	Both	Both
TSOPF_FS_b300	29214	MMFPrec	Both	Both	Both
hread	29736	NoPrec	Both	Both	Both
OPF_6000	29902	MMFPrec	MMFPrec	Both	Both
net100	29920	MMFPrec	NoPrec	Both	Both

spmsrtls	29995	Both	Both	Both	Both
bloweybl	30003	MMFPrec	MMFPrec	Both	Both
blowey	30004	MMFPrec	MMFPrec	MMFPrec	MMFPrec
ug2dc	30200	Both	Both	Both	Both
rajat10	30202	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-53	30235	Both	Both	Both	Both
bcsstk35	30237	MMFPrec	MMFPrec	Both	Both
bcsstm35	30237	MMFPrec	MMFPrec	MMFPrec	MMFPrec
big_dual	30269	MMFPrec	MMFPrec	MMFPrec	MMFPrec
wathen100	30401	MMFPrec	MMFPrec	MMFPrec	MMFPrec
TSOPF_FS_b162_c3	30798	NoPrec	Both	Both	Both
cond-mat-2003	31163	Both	Both	Both	Both
c-54	31793	NoPrec	NoPrec	Both	Both
gupta1	31802	MMFPrec	Both	Both	Both
helm3d01	32226	NoPrec	Both	Both	Both
lpl1	32460	MMFPrec	Both	Both	Both
vt2010	32580	Both	Both	Both	Both
rgg_n_2_15_s0	32768	MMFPrec	MMFPrec	MMFPrec	MMFPrec
delaunay_n15	32768	MMFPrec	MMFPrec	MMFPrec	MMFPrec
se	32768	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-55	32780	Both	Both	Both	Both
SiO	33401	NoPrec	Both	Both	Both
pkustk09	33960	MMFPrec	MMFPrec	Both	Both
ship_001	34920	MMFPrec	MMFPrec	Both	Both
ug3dcqp	35543	MMFPrec	MMFPrec	Both	Both
bcsstk31	35588	MMFPrec	MMFPrec	MMFPrec	MMFPrec

c-56	35910	NoPrec	Both	Both	Both
nd12k	36000	NoPrec	NoPrec	Both	Both
pdb1HYS	36417	MMFPrec	NoPrec	Both	Both
wathen120	36441	MMFPrec	MMFPrec	MMFPrec	MMFPrec
shock-9	36476	MMFPrec	MMFPrec	Both	Both
pw	36519	MMFPrec	MMFPrec	MMFPrec	MMFPrec
email-Enron	36692	Both	Both	Both	Both
net125	36720	MMFPrec	NoPrec	Both	Both
pkustk05	37164	MMFPrec	MMFPrec	MMFPrec	Both
finance256	37376	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-58	37595	Both	Both	Both	Both
c-57	37833	Both	Both	Both	Both
rio001	38434	MMFPrec	MMFPrec	MMFPrec	MMFPrec
vsp_south31_slptsk	39668	NoPrec	Both	Both	Both
jnlbrng1	40000	NoPrec	NoPrec	NoPrec	NoPrec
obstclae	40000	NoPrec	NoPrec	NoPrec	NoPrec
orsion1	40000	NoPrec	NoPrec	NoPrec	NoPrec
case39	40216	MMFPrec	MMFPrec	MMFPrec	Both
cond-mat-2005	40421	Both	Both	Both	Both
TSOPF_FS_b162_c4	40798	MMFPrec	MMFPrec	MMFPrec	MMFPrec
insurfo	40806	NoPrec	NoPrec	NoPrec	NoPrec
c-59	41282	Both	Both	Both	Both
c-62	41731	MMFPrec	MMFPrec	Both	Both
c-62ghs	41731	Both	Both	Both	Both
pkustk06	43164	MMFPrec	MMFPrec	Both	Both
net150	43520	NoPrec	NoPrec	Both	Both

c-61	43618	Both	Both	Both	Both
c-60	43640	Both	Both	Both	Both
OPF_10000	43887	MMFPrec	MMFPrec	Both	Both
c-63	44234	Both	Both	Both	Both
bcsstk32	44609	MMFPrec	MMFPrec	MMFPrec	MMFPrec
fe_body	45087	MMFPrec	MMFPrec	MMFPrec	MMFPrec
k2010	45292	MMFPrec	MMFPrec	MMFPrec	Both
3dtube	45330	MMFPrec	MMFPrec	Both	Both
bcsstk39	46772	NoPrec	MMFPrec	Both	Both
bcsstm39	46772	MMFPrec	MMFPrec	MMFPrec	MMFPrec
vanbody	47072	NoPrec	MMFPrec	Both	Both
c-65	48066	Both	Both	Both	Both
nh2010	48837	MMFPrec	MMFPrec	Both	Both
gridgen	48962	NoPrec	Both	Both	Both
cc	49152	NoPrec	Both	Both	Both
ccc	49152	NoPrec	Both	Both	Both
bfly	49152	NoPrec	NoPrec	NoPrec	NoPrec
stokes128	49666	MMFPrec	MMFPrec	MMFPrec	Both
c-66b	49989	Both	Both	Both	Both
c-66	49989	Both	Both	Both	Both
sparsine	50000	NoPrec	Both	Both	Both
cvxbqp1	50000	MMFPrec	MMFPrec	Both	Both
ncvxbqp1	50000	NoPrec	Both	Both	Both
c-64	51035	Both	Both	Both	Both
c-64b	51035	Both	Both	Both	Both
dawson5	51537	Both	Both	Both	Both

pct20stif	52329	MMFPrec	Both	Both	Both
ct20stif	52329	MMFPrec	MMFPrec	Both	Both
dictionary28	52652	Both	Both	Both	Both
crankseg_1	52804	NoPrec	NoPrec	Both	Both
struct3	53570	MMFPrec	MMFPrec	MMFPrec	MMFPrec
nasasrb	54870	MMFPrec	MMFPrec	Both	Both
srb1	54924	MMFPrec	MMFPrec	Both	Both
copter2	55476	MMFPrec	MMFPrec	MMFPrec	MMFPrec
pkustk04	55590	MMFPrec	MMFPrec	MMFPrec	MMFPrec
TSOPF_FS_b300_c2	56814	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-67b	57975	Both	Both	Both	Both
c-67	57975	MMFPrec	MMFPrec	MMFPrec	Both
dixmaanl	60000	NoPrec	Both	Both	Both
Andrews	60000	NoPrec	NoPrec	NoPrec	MMFPrec
60k	60005	NoPrec	Both	Both	Both
5esindl	60008	Both	Both	Both	Both
blockqp1	60012	NoPrec	NoPrec	NoPrec	NoPrec
Ga3As3H12	61349	Both	Both	Both	Both
GaAsH6	61349	Both	Both	Both	Both
wing	62032	MMFPrec	MMFPrec	MMFPrec	MMFPrec
gupta2	62064	MMFPrec	MMFPrec	MMFPrec	MMFPrec
can	62451	NoPrec	Both	Both	Both
ncvxqp5	62500	MMFPrec	MMFPrec	MMFPrec	MMFPrec
brack2	62631	MMFPrec	MMFPrec	MMFPrec	Both
pkustk03	63336	MMFPrec	Both	Both	Both
crankseg_2	63838	NoPrec	Both	Both	Both

c-68	64810	NoPrec	NoPrec	Both	Both
Dubcova2	65025	NoPrec	NoPrec	Both	Both
rgg_n_2_16_s0	65536	NoPrec	Both	Both	Both
delaunay_n16	65536	MMFPrec	MMFPrec	MMFPrec	MMFPrec
qa8f	66127	NoPrec	MMFPrec	MMFPrec	MMFPrec
qa8fk	66127	NoPrec	NoPrec	NoPrec	Both
gas_sensor	66917	MMFPrec	MMFPrec	MMFPrec	MMFPrec
H2O	67024	NoPrec	Both	Both	Both
c-69	67458	Both	Both	Both	Both
ct2010	67578	MMFPrec	MMFPrec	MMFPrec	MMFPrec
k1_san	67759	MMFPrec	MMFPrec	Both	Both
c-70	68924	Both	Both	Both	Both
e2010	69518	Both	Both	Both	Both
cfd1	70656	NoPrec	Both	Both	Both
F2	71505	MMFPrec	MMFPrec	Both	Both
oilpan	73752	MMFPrec	Both	Both	Both
finan512	74752	MMFPrec	MMFPrec	MMFPrec	MMFPrec
pfinan512	74752	MMFPrec	MMFPrec	MMFPrec	MMFPrec
ncvxqp3	75000	MMFPrec	MMFPrec	MMFPrec	MMFPrec
TSOPF_FS_b39_c19	76216	Both	Both	Both	Both
c-71	76638	Both	Both	Both	Both
fe_tooth	78136	NoPrec	Both	Both	Both
3dh_e	79171	MMFPrec	MMFPrec	Both	Both
3dh	79171	MMFPrec	MMFPrec	MMFPrec	MMFPrec
3dh_	79171	MMFPrec	MMFPrec	MMFPrec	MMFPrec
rail_79841	79841	NoPrec	NoPrec	Both	Both

0nsdsil	80016	Both	Both	Both	Both
2nnsnsl	80016	Both	Both	Both	Both
cont-201	80595	MMFPrec	MMFPrec	MMFPrec	MMFPrec
pkustk10	80676	NoPrec	Both	Both	Both
pache1	80800	NoPrec	NoPrec	Both	Both
shallow_water2	81920	NoPrec	NoPrec	NoPrec	NoPrec
shallow_water1	81920	NoPrec	NoPrec	NoPrec	NoPrec
hermal1	82654	NoPrec	Both	Both	Both
consph	83334	MMFPrec	MMFPrec	Both	Both
c-72	84064	Both	Both	Both	Both
TSOPF_FS_b300_c3	84414	MMFPrec	MMFPrec	MMFPrec	MMFPrec
nv2010	84538	Both	Both	Both	Both
onera_dual	85567	MMFPrec	MMFPrec	MMFPrec	MMFPrec
wy2010	86204	MMFPrec	MMFPrec	MMFPrec	MMFPrec
ncvxqp7	87500	MMFPrec	MMFPrec	MMFPrec	MMFPrec
pkustk11	87804	MMFPrec	MMFPrec	Both	Both
olesnik0	88263	MMFPrec	MMFPrec	MMFPrec	MMFPrec
net4-1	88343	MMFPrec	MMFPrec	Both	Both
sd2010	88360	Both	Both	Both	Both
denormal	89400	NoPrec	NoPrec	Both	Both
s3dkt3m2	90449	NoPrec	Both	Both	Both
s3dkq4m2	90449	NoPrec	Both	Both	Both
s4dkt3m2	90449	MMFPrec	Both	Both	Both
boyd1	93279	NoPrec	NoPrec	NoPrec	Both
ndem_dual	94069	MMFPrec	MMFPrec	MMFPrec	MMFPrec
pkustk12	94653	MMFPrec	MMFPrec	MMFPrec	Both

pkustk13	94893	MMFPrec	Both	Both	Both
Si34H36	97569	Both	Both	Both	Both
_t1	97578	MMFPrec	MMFPrec	Both	Both
fe_rotor	99617	NoPrec	Both	Both	Both
G_n_pin_pou	100000	NoPrec	Both	Both	Both
preferentialAttachmen	100000	Both	Both	Both	Both
smallworld	100000	NoPrec	Both	Both	Both
ford2	100196	MMFPrec	MMFPrec	MMFPrec	MMFPrec
vsp_mod2_pgp2_slptsk	101364	Both	Both	Both	Both
2cubes_sphere	101492	MMFPrec	MMFPrec	MMFPrec	MMFPrec
hermomech_TK	102158	NoPrec	NoPrec	Both	Both
hermomech_TC	102158	MMFPrec	MMFPrec	MMFPrec	MMFPrec
filter3D	106437	MMFPrec	MMFPrec	MMFPrec	MMFPrec
x104	108384	MMFPrec	MMFPrec	Both	Both
598	110971	NoPrec	Both	Both	Both
Ge87H76	112985	Both	Both	Both	Both
Ge99H100	112985	Both	Both	Both	Both
Ga10As10H30	113081	Both	Both	Both	Both
luxembourg_os	114599	NoPrec	Both	Both	Both
shipsec8	114919	MMFPrec	MMFPrec	Both	Both
ut2010	115406	MMFPrec	MMFPrec	MMFPrec	MMFPrec
TSOPF_FS_b39_c30	120216	Both	Both	Both	Both
cop20k_A	121192	Both	Both	Both	Both
ship_003	121728	NoPrec	Both	Both	Both
cfd2	123440	NoPrec	Both	Both	Both
usroads-48	126146	NoPrec	Both	Both	Both

boneS01	127224	MMFPrec	Both	Both	Both
usroads	129164	NoPrec	Both	Both	Both
rgg_n_2_17_s0	131072	NoPrec	Both	Both	Both
delaunay_n17	131072	MMFPrec	MMFPrec	MMFPrec	MMFPrec
2010	132288	Both	Both	Both	Both
Ga19As19H42	133123	Both	Both	Both	Both
nd2010	133769	Both	Both	Both	Both
wv2010	135218	MMFPrec	MMFPrec	MMFPrec	MMFPrec
shipsec1	140874	NoPrec	Both	Both	Both
bmw7st_1	141347	NoPrec	NoPrec	NoPrec	Both
fe_ocean	143437	NoPrec	Both	Both	Both
engine	143571	MMFPrec	Both	Both	Both
144	144649	NoPrec	Both	Both	Both
d2010	145247	Both	Both	Both	Both
Dubcova3	146689	Both	NoPrec	Both	Both
bmwcra_1	148770	MMFPrec	Both	Both	Both
id2010	149842	Both	Both	Both	Both
G2_circui	150102	NoPrec	NoPrec	Both	Both
pkustk14	151926	NoPrec	Both	Both	Both
gearbox	153746	MMFPrec	Both	Both	Both
SiO2	155331	NoPrec	Both	Both	Both
wave	156317	MMFPrec	MMFPrec	Both	Both
2010	157508	MMFPrec	MMFPrec	MMFPrec	MMFPrec
ky2010	161672	MMFPrec	MMFPrec	MMFPrec	MMFPrec
nm2010	168609	MMFPrec	MMFPrec	MMFPrec	MMFPrec
c-73	169422	MMFPrec	MMFPrec	Both	Both

nj2010	169588	Both	Both	Both	Both
s2010	171778	MMFPrec	MMFPrec	MMFPrec	Both
shipsec5	179860	MMFPrec	MMFPrec	Both	Both
cont-300	180895	MMFPrec	MMFPrec	MMFPrec	MMFPrec
sc2010	181908	Both	Both	Both	Both
d_pretok	182730	MMFPrec	MMFPrec	Both	Both
Si41Ge41H72	185639	Both	Both	Both	Both
r2010	186211	Both	Both	Both	Both
uron_	189924	Both	Both	Both	Both
caidaRouterLevel	192244	Both	Both	Both	Both
ne2010	193352	MMFPrec	MMFPrec	MMFPrec	MMFPrec
wa2010	195574	MMFPrec	MMFPrec	MMFPrec	MMFPrec
or2010	196621	Both	Both	Both	Both
fullb	199187	MMFPrec	Both	Both	Both
co2010	201062	MMFPrec	MMFPrec	MMFPrec	MMFPrec
fcondp2	201822	MMFPrec	MMFPrec	Both	Both
hermomech_dM	204316	MMFPrec	MMFPrec	MMFPrec	MMFPrec
la2010	204447	Both	Both	Both	Both
roll	213453	NoPrec	Both	Both	Both
14b	214765	NoPrec	Both	Both	Both
ia2010	216007	MMFPrec	MMFPrec	MMFPrec	MMFPrec
pwtk	217918	MMFPrec	MMFPrec	Both	Both
hood	220542	MMFPrec	Both	Both	Both
CO	221119	Both	Both	Both	Both
halfb	224617	MMFPrec	MMFPrec	Both	Both
HTC_336_4438	226340	MMFPrec	MMFPrec	Both	Both

HTC_336_9129	226340	MMFPrec	MMFPrec	MMFPrec	Both
CurlCurl_1	226451	MMFPrec	Both	Both	Both
coAuthorsCiteseer	227320	Both	Both	Both	Both
bmw3_2	227362	NoPrec	Both	Both	Both
ks2010	238600	MMFPrec	MMFPrec	MMFPrec	MMFPrec
n2010	240116	MMFPrec	MMFPrec	MMFPrec	MMFPrec
Si87H76	240369	Both	Both	Both	Both
z2010	241666	Both	Both	Both	Both
BenElechi1	245874	MMFPrec	MMFPrec	Both	Both
l2010	252266	Both	Both	Both	Both
wi2010	253096	Both	Both	Both	Both
Lin	256000	NoPrec	Both	Both	Both
n2010	259777	Both	Both	Both	Both
offshore	259789	MMFPrec	MMFPrec	Both	Both
rgg_n_2_18_s0	262144	NoPrec	Both	Both	Both
delaunay_n18	262144	NoPrec	Both	Both	Both
in2010	267071	Both	Both	Both	Both
citationCiteseer	268495	Both	Both	Both	Both
ok2010	269118	MMFPrec	MMFPrec	MMFPrec	MMFPrec
va2010	285762	Both	Both	Both	Both
nc2010	288987	Both	Both	Both	Both
ga2010	291086	Both	Both	Both	Both
3Dspectralwave2	292008	Both	Both	Both	Both
coAuthorsDBLP	299067	Both	Both	Both	Both
dblp-2010	326186	Both	Both	Both	Both
i2010	329885	Both	Both	Both	Both

o2010	343565	Both	Both	Both	Both
ny2010	350169	Both	Both	Both	Both
oh2010	365344	Both	Both	Both	Both
darcy003	389874	MMFPrec	MMFPrec	Both	Both
rio002	389874	NoPrec	Both	Both	Both
helm2d03	392257	NoPrec	Both	Both	Both
pa2010	421545	Both	Both	Both	Both
uto	448695	NoPrec	Both	Both	Both
il2010	451554	Both	Both	Both	Both
fl2010	484481	Both	Both	Both	Both
f_2_k101	503625	NoPrec	MMFPrec	Both	Both
f_3_k101	503625	MMFPrec	Both	Both	Both
f_1_k101	503625	MMFPrec	MMFPrec	Both	Both
rgg_n_2_19_s0	524288	NoPrec	Both	Both	Both
delaunay_n19	524288	NoPrec	Both	Both	Both
parabolic_fe	525825	NoPrec	Both	Both	Both
Wins	NoPrec	148	60	39	31
	MMFPrec	231	199	129	110

APPENDIX D

ADDITIONAL RESULTS ON COMPRESSING ASYMMETRIC MATRICES

Table D.1: **Asymmetric MMF (additive) results**: Compression results on MMF (additive) vs CUR

Compressed size→	1%		10%		25%		50%		75%	
Matrix ↓	MMF	CUR	MMF	CUR	MMF	CUR	MMF	CUR	MMF	CUR
bayer06	1.00	**0.89**	0.78	**0.00**	**0.71**	0.89	0.71	**0.00**	0.71	**0.00**
bayer02	1.00	**0.00**	1.00	**0.00**	0.71	**0.00**	0.71	**0.00**	0.71	**0.00**
extr1b	1.00	**0.75**	1.00	**0.03**	0.95	**0.03**	0.80	**0.02**	0.73	**0.01**
p2p-Gnutella04	1.00	**0.99**	1.00	**0.89**	1.00	**0.58**	0.96	**0.05**	0.88	**0.01**
gemat11	0.86	**0.40**	0.98	**0.31**	0.79	**0.27**	0.77	**0.19**	0.75	**0.13**
gre_1107	**0.86**	0.99	**0.86**	0.98	**0.84**	0.91	0.79	**0.72**	0.71	**0.44**
g7jac020sc	0.97	**0.07**	0.93	**0.02**	0.80	**0.02**	0.78	**0.01**	0.77	**0.00**
bayer05	1.00	**0.89**	0.92	**0.00**	0.78	**0.00**	0.71	**0.00**	0.71	**0.00**
rw5151	**0.99**	1.00	0.98	**0.97**	0.95	**0.90**	0.88	**0.72**	0.78	**0.46**
Californi	1.00	**0.90**	0.99	**0.37**	0.99	**0.11**	0.98	**0.08**	0.94	**0.04**
fd18	1.00	**0.07**	1.00	**0.08**	0.90	**0.00**	0.90	**0.00**	0.88	**0.00**
hydr1c	1.00	**0.87**	1.00	**0.40**	0.98	**0.08**	0.93	**0.17**	0.86	**0.00**
polblogs	0.94	**0.90**	0.94	**0.62**	0.94	**0.30**	0.94	**0.03**	0.89	**0.01**
Kohonen	0.99	**0.89**	0.99	**0.56**	0.99	**0.32**	0.97	**0.07**	0.91	**0.03**
Hamrle2	0.99	**0.66**	1.00	**0.50**	0.94	**0.44**	0.90	**0.16**	0.84	**0.05**
lhr01	1.00	**0.99**	0.99	**0.87**	0.98	**0.73**	0.93	**0.56**	0.84	**0.31**
orani678	0.99	**0.92**	0.98	**0.80**	0.97	**0.66**	0.91	**0.58**	0.81	**0.43**
jan99jac040sc	**0.41**	0.48	0.41	**0.27**	0.42	**0.19**	0.41	**0.16**	0.40	**0.08**
FA	0.98	**0.91**	0.97	**0.54**	0.97	**0.26**	0.94	**0.08**	0.84	**0.05**
poli	**0.53**	0.98	**0.53**	0.92	**0.53**	0.85	**0.52**	0.73	**0.50**	0.51
bayer09	1.00	**0.00**	0.97	**0.00**	0.96	**0.00**	0.89	**0.00**	0.86	**0.00**
cz2548	**0.42**	1.00	**0.42**	0.98	**0.41**	0.92	**0.36**	0.75	**0.27**	0.46
barth	**0.83**	1.00	**0.83**	0.96	**0.81**	0.82	0.76	**0.56**	0.69	**0.29**

barth4	**0.83**	1.00	**0.83**	0.97	**0.81**	0.86	0.76	**0.58**	0.69	**0.30**
lhr04c	0.99	**0.97**	1.00	**0.83**	0.99	**0.68**	0.95	**0.46**	0.85	**0.29**
lhr04	1.00	**0.96**	1.00	**0.84**	0.98	**0.68**	0.95	**0.46**	0.85	**0.28**
extr1	1.00	**0.75**	0.99	**0.04**	0.95	**0.02**	0.78	**0.01**	0.74	**0.01**
p2p-Gnutella06	1.00	**0.98**	1.00	**0.87**	0.99	**0.56**	0.96	**0.04**	0.87	**0.02**
bayer07	1.00	**0.00**	0.78	**0.00**	0.78	**0.00**	0.71	**0.00**	0.71	**0.00**
bayer03	1.00	**0.39**	1.00	**0.01**	0.95	**0.00**	0.87	**0.00**	0.81	**0.00**
SciMe	0.99	**0.97**	0.99	**0.75**	0.99	**0.47**	0.96	**0.10**	0.89	**0.04**
b_dyn	1.00	**0.00**	0.71	**0.00**	1.00	**0.00**	0.71	**0.00**	0.71	**0.00**
GD96_	1.00	**0.91**	0.99	**0.81**	0.98	**0.68**	0.94	**0.60**	0.84	**0.27**
SmaGri	0.98	**0.94**	0.98	**0.67**	0.98	**0.38**	0.96	**0.06**	0.90	**0.03**
rdist2	0.81	**0.14**	0.80	**0.04**	0.81	**0.02**	0.81	**0.01**	0.80	**0.00**
Pd	0.86	**0.00**	0.86	**0.00**	0.86	**0.03**	0.86	**0.00**	0.85	**0.02**
shermanAC	**0.24**	0.25	**0.24**	0.96	0.23	**0.12**	0.23	**0.00**	0.21	**0.00**
Chebyshev2	0.86	**0.00**	0.88	**0.00**	0.72	**0.00**	0.87	**0.00**	0.72	**0.00**
fd12	1.00	**0.11**	1.00	**0.06**	0.99	**0.03**	0.89	**0.00**	0.88	**0.00**
EPA	1.00	**0.86**	1.00	**0.45**	0.99	**0.09**	0.98	**0.05**	0.92	**0.03**
g7jac010	0.86	**0.62**	0.83	**0.44**	0.83	**0.17**	0.83	**0.01**	0.78	**0.00**
rk3jac020sc	0.77	**0.40**	0.76	**0.33**	0.75	**0.29**	0.74	**0.52**	0.70	**0.43**
bayer08	1.00	**0.46**	0.78	**0.00**	**0.71**	0.89	0.71	**0.00**	0.71	**0.00**
jan99jac020	0.91	**0.54**	0.89	**0.13**	0.90	**0.12**	0.83	**0.08**	0.78	**0.10**
Zewail	0.99	**0.97**	0.99	**0.74**	0.98	**0.47**	0.96	**0.18**	0.89	**0.02**
west2021	0.99	**0.59**	0.97	**0.07**	0.99	**0.00**	0.93	**0.00**	0.82	**0.00**
radfr1	0.73	**0.24**	0.72	**0.08**	0.73	**0.02**	0.72	**0.01**	0.73	**0.00**
p2p-Gnutella09	1.00	**0.93**	1.00	**0.81**	0.99	**0.42**	0.96	**0.05**	0.88	**0.01**
g7jac010sc	0.84	**0.10**	0.79	**0.04**	0.80	**0.08**	0.87	**0.01**	0.77	**0.01**
ols2000	0.99	**0.88**	1.00	**0.19**	0.99	**0.00**	1.00	**0.00**	0.84	**0.00**
cz5108	**0.42**	1.00	**0.42**	0.99	**0.41**	0.92	**0.36**	0.75	**0.27**	0.47
eg1	0.70	**0.18**	0.70	**0.03**	0.70	**0.01**	0.66	**0.00**	0.69	**0.00**
cz1268	**0.42**	1.00	**0.42**	0.98	**0.41**	0.93	**0.36**	0.74	**0.27**	0.46
shermanACd	0.83	**0.41**	0.86	**0.29**	0.83	**0.30**	0.82	**0.11**	0.80	**0.02**
ols1090	0.99	**0.88**	0.99	**0.19**	0.99	**0.00**	0.99	**0.00**	0.86	**0.00**

ols4000	1.00	**0.88**	1.00	**0.18**	1.00	**0.00**	1.00	**0.00**	0.85	**0.00**
rdist3	0.97	**0.82**	0.97	**0.01**	0.97	**0.00**	0.97	**0.00**	0.97	**0.00**
Chebyshev3	0.86	**0.00**	0.87	**0.00**	0.87	**0.00**	0.87	**0.00**	0.87	**0.00**
gemat12	1.00	**0.38**	0.85	**0.26**	0.85	**0.21**	0.76	**0.16**	0.74	**0.10**
west1505	0.98	**0.58**	0.98	**0.04**	0.97	**0.00**	0.91	**0.00**	0.85	**0.00**
CSphd	1.00	**0.95**	0.99	**0.77**	0.98	**0.65**	0.93	**0.21**	0.83	**0.10**
hydr1	1.00	**0.82**	1.00	**0.53**	0.98	**0.09**	0.93	**0.27**	0.87	**0.00**
TSOPF_RS_b9_c6	0.90	**0.84**	0.90	**0.82**	0.88	**0.73**	0.84	**0.48**	0.80	**0.20**
EVA	1.00	**0.80**	0.99	**0.56**	0.97	**0.21**	0.93	**0.14**	0.88	**0.08**
lhr02	1.00	**0.99**	0.99	**0.88**	0.98	**0.76**	0.93	**0.55**	0.84	**0.32**
ODLIS	0.98	**0.92**	0.98	**0.65**	0.98	**0.46**	0.96	**0.25**	0.89	**0.04**
g7jac020	0.86	**0.65**	0.84	**0.35**	0.85	**0.15**	0.81	**0.08**	0.79	**0.00**
GD06_Jav	0.98	**0.88**	0.97	**0.60**	0.97	**0.34**	0.95	**0.07**	0.88	**0.02**
p2p-Gnutella08	1.00	**0.94**	1.00	**0.81**	0.99	**0.47**	0.96	**0.05**	0.88	**0.01**
b2_ss	0.99	**0.81**	1.00	**0.09**	0.96	**0.02**	0.89	**0.00**	0.78	**0.00**
rk3jac020	0.94	**0.18**	0.94	**0.20**	0.97	**0.00**	0.95	**0.00**	0.93	**0.00**
hindas	1.00	**1.00**	1.00	**0.00**	1.00	**0.00**	0.71	**0.00**	0.71	**0.00**
rdist1	0.77	**0.07**	0.77	**0.03**	0.77	**0.02**	0.77	**0.01**	0.78	**0.00**
p2p-Gnutella05	1.00	**0.97**	1.00	**0.86**	0.99	**0.55**	0.96	**0.05**	0.87	**0.02**
total %wins:	13.5	86.5	10.8	89.2	12.2	87.8	5.4	94.6	5.4	94.6

Table D.2: **Asymmetric MMF (GreedyTopN) results**: Compression results on MMF (GreedyTopN) vs CUR

Compressed size→	1%		10%		25%		50%		75%	
Matrix ↓	MMF	CUR	MMF	CUR	MMF	CUR	MMF	CUR	MMF	CUR
bayer06	0.29	**0.00**	0.03	**0.00**	**0.46**	0.89	0.46	**0.00**	0.01	**0.00**
extr1b	**0.12**	0.75	0.34	**0.05**	0.66	**0.04**	0.85	**0.01**	0.71	**0.00**
gemat11	**0.31**	0.96	0.33	**0.32**	0.33	**0.26**	0.33	**0.24**	0.32	**0.13**
gre_1107	**0.75**	1.00	**0.78**	0.97	**0.79**	0.91	0.74	**0.70**	0.59	**0.45**
g7jac020sc	0.20	**0.07**	0.21	**0.14**	0.26	**0.16**	0.43	**0.01**	0.36	**0.00**
bayer05	**0.00**	0.00	**0.01**	0.89	**0.46**	1.00	0.46	**0.00**	0.46	**0.00**

rw5151	**0.77**	1.00	**0.79**	0.97	**0.80**	0.90	0.75	**0.72**	0.59	**0.46**
hydr1c	**0.46**	0.81	0.53	**0.43**	0.59	**0.34**	0.62	**0.00**	0.61	**0.00**
polblogs	**0.90**	0.90	0.90	**0.62**	0.90	**0.31**	0.89	**0.03**	0.81	**0.01**
Hamrle2	**0.66**	0.68	0.74	**0.64**	0.75	**0.45**	0.63	**0.33**	0.39	**0.05**
lhr01	**0.81**	0.98	**0.83**	0.88	0.83	**0.74**	0.78	**0.57**	0.64	**0.33**
orani678	**0.68**	0.90	**0.70**	0.79	0.70	**0.66**	0.70	**0.58**	0.59	**0.41**
poli	**0.57**	0.98	**0.58**	0.91	**0.59**	0.84	**0.60**	0.74	0.57	**0.50**
bayer09	0.10	**0.00**	0.01	**0.00**	0.33	**0.00**	0.42	**0.00**	0.52	**0.00**
cz2548	**0.69**	1.00	**0.72**	0.99	**0.73**	0.92	**0.68**	0.75	0.54	**0.46**
barth	**0.86**	1.00	**0.88**	0.96	0.86	**0.83**	0.77	**0.55**	0.59	**0.29**
barth4	**0.85**	1.00	**0.86**	0.97	**0.85**	0.86	0.75	**0.60**	0.58	**0.29**
lhr04c	**0.80**	0.96	**0.82**	0.85	0.83	**0.68**	0.81	**0.47**	0.65	**0.27**
lhr04	**0.80**	0.98	**0.82**	0.84	0.83	**0.68**	0.81	**0.46**	0.67	**0.29**
extr1	**0.12**	0.75	0.37	**0.04**	0.60	**0.03**	0.86	**0.01**	0.82	**0.00**
p2p-Gnutella06	**0.92**	0.98	0.92	**0.87**	0.93	**0.55**	0.89	**0.04**	0.72	**0.01**
bayer07	**0.00**	0.89	0.46	**0.46**	**0.46**	0.89	0.01	**0.00**	0.01	**0.00**
SciMe	**0.92**	0.98	0.92	**0.75**	0.92	**0.48**	0.91	**0.09**	0.79	**0.03**
b_dyn	**0.00**	0.00	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
GD96_	**0.66**	0.96	**0.70**	0.81	0.76	**0.68**	0.75	**0.59**	0.60	**0.32**
SmaGri	**0.91**	0.94	0.91	**0.67**	0.92	**0.38**	0.89	**0.06**	0.77	**0.02**
rdist2	0.44	**0.15**	0.44	**0.04**	0.35	**0.04**	0.45	**0.01**	0.36	**0.00**
Pd	0.03	**0.02**	0.03	**0.00**	0.03	**0.00**	0.03	**0.00**	0.02	**0.00**
shermanAC	**0.16**	0.24	0.17	**0.11**	0.19	**0.08**	0.17	**0.02**	0.16	**0.00**
Chebyshev2	0.05	**0.00**	0.04	**0.00**	0.04	**0.00**	0.06	**0.00**	0.04	**0.00**
fd12	0.52	**0.04**	0.60	**0.02**	0.51	**0.03**	0.43	**0.00**	0.70	**0.00**
EPA	0.91	**0.90**	0.91	**0.45**	0.92	**0.11**	0.92	**0.05**	0.86	**0.01**
g7jac010	**0.49**	0.67	0.50	**0.28**	0.51	**0.14**	0.50	**0.04**	0.50	**0.00**
rk3jac020sc	0.50	**0.40**	0.50	**0.37**	0.54	**0.26**	**0.31**	0.38	**0.26**	0.38
bayer08	**0.01**	1.00	0.29	**0.00**	0.03	**0.00**	0.46	**0.00**	0.01	**0.00**
jan99jac020	**0.44**	0.60	0.48	**0.11**	0.58	**0.12**	0.59	**0.10**	0.48	**0.07**
Zewail	**0.94**	0.97	0.95	**0.73**	0.95	**0.47**	0.93	**0.18**	0.76	**0.02**
west2021	**0.19**	0.59	0.47	**0.31**	0.44	**0.00**	0.64	**0.00**	0.35	**0.00**

radfr1	**0.21**	0.25	0.20	**0.08**	0.19	**0.04**	0.20	**0.01**	0.21	**0.00**
p2p-Gnutella09	**0.92**	0.95	0.92	**0.83**	0.92	**0.40**	0.89	**0.04**	0.73	**0.02**
g7jac010sc	**0.12**	0.20	0.36	**0.14**	0.37	**0.01**	**0.20**	0.65	0.08	**0.01**
ols2000	**0.00**	0.88	**0.00**	0.19	0.00	**0.00**	0.87	**0.00**	0.80	**0.00**
cz5108	**0.69**	1.00	**0.72**	0.98	**0.73**	0.92	**0.68**	0.74	0.54	**0.47**
eg1	**0.11**	0.14	0.12	**0.04**	0.21	**0.01**	0.17	**0.01**	0.14	**0.00**
cz1268	**0.69**	1.00	**0.71**	0.99	**0.73**	0.92	**0.67**	0.74	0.55	**0.45**
ols1090	**0.00**	0.88	**0.00**	0.19	0.00	**0.00**	0.88	**0.00**	0.78	**0.00**
ols4000	**0.00**	0.88	**0.00**	0.18	0.00	**0.00**	0.88	**0.00**	0.87	**0.00**
rdist3	**0.76**	0.84	0.74	**0.01**	0.75	**0.00**	0.75	**0.00**	0.75	**0.00**
Chebyshev3	0.04	**0.00**	0.04	**0.00**	0.04	**0.00**	0.69	**0.00**	0.00	**0.00**
gemat12	**0.36**	0.62	0.30	**0.28**	**0.36**	0.57	0.27	**0.14**	0.27	**0.12**
west1505	**0.09**	0.58	0.46	**0.03**	0.46	**0.00**	0.61	**0.00**	0.42	**0.00**
CSphd	**0.71**	0.94	**0.73**	0.75	0.76	**0.62**	0.78	**0.26**	0.72	**0.10**
hydr1	**0.50**	0.81	0.50	**0.44**	0.55	**0.08**	0.64	**0.00**	0.60	**0.00**
EVA	0.83	**0.76**	0.80	**0.61**	0.84	**0.20**	0.75	**0.14**	0.71	**0.10**
lhr02	**0.81**	0.99	**0.82**	0.89	0.84	**0.75**	0.79	**0.54**	0.63	**0.32**
ODLIS	0.92	**0.91**	0.92	**0.67**	0.92	**0.45**	0.91	**0.24**	0.81	**0.04**
g7jac020	**0.52**	0.72	0.53	**0.36**	0.50	**0.15**	0.58	**0.08**	0.59	**0.00**
GD06_Jav	**0.87**	0.91	0.88	**0.58**	0.88	**0.32**	0.86	**0.09**	0.80	**0.06**
p2p-Gnutella08	**0.92**	0.94	0.92	**0.82**	0.92	**0.42**	0.89	**0.03**	0.75	**0.02**
b2_ss	**0.68**	0.76	0.75	**0.10**	0.79	**0.02**	0.77	**0.00**	0.52	**0.00**
hindas	**0.00**	0.00	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
rdist1	0.47	**0.07**	0.42	**0.03**	0.47	**0.02**	0.55	**0.01**	0.48	**0.00**
p2p-Gnutella05	**0.92**	0.98	0.92	**0.87**	0.93	**0.56**	0.89	**0.05**	0.72	**0.02**
total %wins:	79.4	20.6	30.2	69.8	17.5	82.5	9.5	90.5	1.6	98.4

Table D.3: **Asymmetric MMF (TopN) results**: Compression results on MMF (TopN) vs CUR

Compressed size→	1%		10%		25%		50%		75%	
Matrix ↓	MMF	CUR	MMF	CUR	MMF	CUR	MMF	CUR	MMF	CUR

bayer06	**0.00**	0.89	**0.00**	0.00	**0.00**	0.89	**0.00**	0.00	**0.00**	0.00
extr1b	**0.00**	0.75	**0.00**	0.03	**0.00**	0.03	**0.00**	0.02	**0.00**	0.01
p2p-Gnutella04	**0.82**	0.98	**0.83**	0.88	0.85	**0.58**	0.84	**0.04**	0.68	**0.02**
gemat11	**0.16**	0.40	**0.17**	0.31	**0.17**	0.27	**0.16**	0.19	**0.13**	0.13
gre_1107	**0.71**	0.99	**0.71**	0.98	**0.71**	0.91	**0.65**	0.72	0.52	**0.44**
g7jac020sc	**0.01**	0.07	**0.01**	0.02	**0.01**	0.02	0.01	**0.01**	0.01	**0.00**
bayer05	**0.00**	0.89	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
rw5151	**0.67**	1.00	**0.68**	0.97	**0.68**	0.90	**0.63**	0.72	0.50	**0.46**
hydr1c	**0.00**	0.87	**0.00**	0.40	**0.00**	0.08	**0.00**	0.17	0.00	**0.00**
polblogs	**0.84**	0.90	0.83	**0.62**	0.84	**0.30**	0.83	**0.03**	0.78	**0.01**
Kohonen	**0.72**	0.89	0.74	**0.58**	0.77	**0.32**	0.80	**0.08**	0.77	**0.02**
Hamrle2	**0.09**	0.66	**0.09**	0.50	**0.09**	0.44	**0.09**	0.16	0.07	**0.05**
lhr01	**0.71**	0.99	**0.73**	0.87	0.73	**0.73**	0.70	**0.56**	0.57	**0.31**
orani678	**0.49**	0.92	**0.50**	0.80	**0.48**	0.66	**0.50**	0.58	**0.42**	0.43
poli	**0.51**	0.98	**0.52**	0.92	**0.52**	0.85	**0.53**	0.73	**0.51**	0.51
bayer09	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
cz2548	**0.66**	1.00	**0.66**	0.98	**0.65**	0.92	**0.58**	0.75	0.47	**0.46**
barth	**0.79**	1.00	**0.80**	0.96	**0.79**	0.82	0.71	**0.56**	0.55	**0.29**
barth4	**0.80**	1.00	**0.80**	0.97	**0.78**	0.86	0.70	**0.58**	0.54	**0.30**
Lederberg	**0.81**	0.94	0.82	**0.65**	0.84	**0.37**	0.84	**0.07**	0.78	**0.03**
lhr04c	**0.68**	0.97	**0.70**	0.83	0.71	**0.68**	0.70	**0.46**	0.57	**0.29**
lhr04	**0.68**	0.96	**0.70**	0.84	0.72	**0.68**	0.70	**0.46**	0.58	**0.28**
extr1	**0.00**	0.75	**0.00**	0.04	**0.00**	0.02	**0.00**	0.01	**0.00**	0.01
p2p-Gnutella06	**0.80**	0.98	**0.82**	0.86	0.84	**0.56**	0.82	**0.04**	0.68	**0.01**
bayer07	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
SciMe	**0.79**	0.97	0.80	**0.75**	0.82	**0.47**	0.82	**0.10**	0.72	**0.04**
b_dyn	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
GD96_	**0.47**	0.91	**0.51**	0.81	**0.55**	0.68	**0.56**	0.60	0.49	**0.27**
SmaGri	**0.80**	0.94	0.81	**0.67**	0.82	**0.38**	0.83	**0.06**	0.75	**0.03**
rdist2	**0.04**	0.14	**0.04**	0.04	0.05	**0.02**	0.05	**0.01**	0.07	**0.00**
Pd	**0.00**	0.00	**0.00**	0.00	**0.00**	0.03	**0.00**	0.00	**0.00**	0.02
shermanAC	**0.01**	0.25	**0.01**	0.96	**0.01**	0.12	0.01	**0.00**	0.02	**0.00**

Chebyshev2	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
fd12	**0.00**	0.11	**0.00**	0.06	**0.00**	0.03	**0.00**	0.00	**0.00**	0.00
EPA	**0.55**	0.86	0.57	**0.45**	0.64	**0.09**	0.75	**0.05**	0.77	**0.03**
g7jac010	**0.00**	0.62	**0.01**	0.44	**0.01**	0.17	0.03	**0.01**	0.05	**0.00**
rk3jac020sc	**0.00**	0.43	**0.00**	0.42	**0.00**	0.42	**0.00**	0.41	**0.00**	0.25
bayer08	**0.00**	0.46	**0.00**	0.00	**0.00**	0.89	**0.00**	0.00	**0.00**	0.00
jan99jac020	**0.00**	0.54	**0.00**	0.13	**0.00**	0.12	**0.00**	0.08	**0.00**	0.10
Zewail	**0.86**	0.97	0.86	**0.73**	0.88	**0.48**	0.86	**0.18**	0.73	**0.02**
west2021	**0.00**	0.59	**0.00**	0.02	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
radfr1	**0.09**	0.24	0.08	**0.08**	0.10	**0.02**	0.11	**0.01**	0.13	**0.00**
p2p-Gnutella09	**0.78**	0.94	**0.79**	0.83	0.82	**0.39**	0.82	**0.05**	0.69	**0.01**
g7jac010sc	**0.01**	0.10	**0.01**	0.04	**0.01**	0.08	**0.01**	0.01	**0.01**	0.01
ols2000	**0.00**	0.88	**0.00**	0.19	**0.00**	0.00	0.00	**0.00**	0.00	**0.00**
cz5108	**0.66**	1.00	**0.66**	0.99	**0.66**	0.92	**0.59**	0.75	**0.46**	0.47
eg1	**0.01**	0.18	**0.01**	0.03	**0.01**	0.01	0.01	**0.00**	0.01	**0.00**
cz1268	**0.66**	1.00	**0.66**	0.98	**0.65**	0.93	**0.59**	0.74	**0.46**	0.46
shermanACd	**0.00**	0.57	**0.00**	0.32	**0.01**	0.21	**0.01**	0.12	**0.01**	0.02
ols1090	**0.00**	0.88	**0.00**	0.19	**0.00**	0.00	0.00	**0.00**	0.00	**0.00**
ols4000	**0.00**	0.88	**0.00**	0.18	**0.00**	0.00	0.00	**0.00**	0.00	**0.00**
rdist3	**0.06**	0.82	0.07	**0.01**	0.08	**0.00**	0.11	**0.00**	0.24	**0.00**
Chebyshev3	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
gemat12	**0.14**	0.38	**0.14**	0.26	**0.14**	0.21	**0.13**	0.16	0.11	**0.10**
west1505	**0.00**	0.58	**0.00**	0.04	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
CSphd	**0.00**	0.95	**0.00**	0.77	**0.24**	0.65	0.50	**0.21**	0.55	**0.10**
hydr1	**0.00**	0.82	**0.00**	0.53	**0.00**	0.09	**0.00**	0.27	0.00	**0.00**
TSOPF_RS_b9_c6	**0.51**	0.84	**0.52**	0.82	**0.53**	0.73	0.53	**0.50**	0.49	**0.19**
EVA	**0.00**	0.80	**0.00**	0.56	**0.00**	0.21	**0.01**	0.14	0.41	**0.08**
lhr02	**0.72**	0.99	**0.72**	0.88	**0.74**	0.76	0.70	**0.55**	0.58	**0.32**
ODLIS	**0.85**	0.92	0.86	**0.65**	0.86	**0.46**	0.85	**0.25**	0.77	**0.04**
g7jac020	**0.01**	0.65	**0.02**	0.35	**0.02**	0.15	**0.04**	0.08	0.08	**0.00**
GD06_Jav	**0.76**	0.88	0.77	**0.60**	0.78	**0.34**	0.80	**0.07**	0.77	**0.02**
p2p-Gnutella08	**0.78**	0.94	**0.80**	0.81	0.82	**0.47**	0.82	**0.05**	0.70	**0.01**

b2_ss	**0.00**	0.89	**0.00**	0.07	**0.00**	0.05	0.00	**0.00**	0.00	**0.00**
rk3jac020	**0.00**	0.17	**0.00**	0.28	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
hindas	**0.00**	1.00	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**	0.00	**0.00**
rdist1	**0.02**	0.07	**0.02**	0.03	0.03	**0.02**	0.03	**0.01**	0.04	**0.00**
p2p-Gnutella05	**0.81**	0.97	**0.82**	0.86	0.84	**0.57**	0.83	**0.04**	0.68	**0.02**
total %wins:	100.0	0.0	84.1	15.9	69.6	30.4	49.3	50.7	29.0	71.0

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. arXiv preprint arXiv:1608.04207, 2016.

[3] Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. Analyzing the behavior of visual question answering models. arXiv preprint arXiv:1606.07356, 2016.

[4] Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C Lawrence Zitnick, Dhruv Batra, and Devi Parikh. Vqa: Visual question answering. arXiv preprint arXiv:1505.00468, 2015.

[5] Hiralal Agrawal and Joseph R. Horgan. Dynamic program slicing. In Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation, PLDI '90, pages 246–256, 1990.

[6] Tamer Alkhouli, Gabriel Bretschner, Jan-Thorsten Peter, Mohammed Hethnawi, Andreas Guta, and Hermann Ney. Alignment-based neural machine translation. In Proceedings of the First Conference on Machine Translation: Volume 1, Research Papers, pages 54–65, 2016.

[7] Amirhossein Aminfar, Sivaram Ambikasaran, and Eric Darve. A fast block low-rank dense solver with applications to finite-element matrices. Journal of Computational Physics, 304:170–188, 2016.

[8] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. Journal of Machine Learning Research, pages 1803–1831, 2010.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

[10] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. In International Conference on Learning Representations ICLR, 2017.

[11] Randolph E Bank, Todd F Dupont, and Harry Yserentant. The hierarchical basis multigrid method. Numerische Mathematik, 52(4):427–458, 1988.

[12] Rick Beatson and Leslie Greengard. A short course on fast multipole methods.

[13] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. arXiv preprint arXiv:1711.02173, 2017.

[14] Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication on cuda. Technical report, Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008.

[15] Hedi Ben-younes, Rémi Cadene, Matthieu Cord, and Nicolas Thome. Mutan: Multimodal tucker fusion for visual question answering. arXiv preprint arXiv:1705.06676, 2017.

[16] M.W. Benson. Iterative solution of large scale linear systems. Mathematics report. Thesis (M.Sc.)–Lakehead University, 1973.

[17] Michele Benzi. Preconditioning techniques for large linear systems: a survey. Journal of Computational Physics, 182(2):418–477, 2002.

[18] Michele Benzi and Miroslav Tuma. A comparative study of sparse approximate inverse preconditioners. Applied Numerical Mathematics, 30(2-3):305–340, 1999.

[19] Alexander Binder, Grégoire Montavon, Sebastian Bach, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. CoRR, 2016.

[20] Terra Blevins, Omer Levy, and Luke Zettlemoyer. Deep rnns encode soft hierarchical syntax. arXiv preprint arXiv:1805.04218, 2018.

[21] Rastisalv Bodík and Sadun Anik. Path-sensitive value-flow analysis. In Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '98, pages 237–251, 1998.

[22] Robert Bridson and Wei-Pai Tang. Multiresolution approximate inverse preconditioners. SIAM Journal on Scientific Computing, 23(2):463–479, 2001.

[23] William L Briggs, Steve F McCormick, et al. A multigrid tutorial, volume 72. Siam, 2000.

[24] Chris Callison-Burch, David Talbot, and Miles Osborne. Statistical machine translation with word-and sentence-aligned parallel corpora. In Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, page 175. Association for Computational Linguistics, 2004.

[25] Tony F Chan, Wei Pai Tang, and Wing Lok Wan. Wavelet sparse approximate inverse preconditioners. BIT Numerical Mathematics, 37(3):644–660, 1997.

[26] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4960–4964. IEEE, 2016.

[27] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning for recommendation. arXiv preprint arXiv:1802.07876, 2018.

[28] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[29] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In Advances in neural information processing systems, pages 577–585, 2015.

[30] Ronald R Coifman, Stephane Lafon, Ann B Lee, Mauro Maggioni, Boaz Nadler, Frederick Warner, and Steven W Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. Proceedings of the national academy of sciences, 102(21):7426–7431, 2005.

[31] Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. arXiv preprint arXiv:1805.01070, 2018.

[32] M.W. Craven and J.W. Shavlik. Learning symbolic rules using articial neural networks. Proceedings of the Tenth International Conference on Machine Learning, pages 73–80, 1993.

[33] Y. Cui, Y. Song, C. Sun, A. Howard, and S. Belongie. Large Scale Fine-Grained Categorization and Domain-Specific Transfer Learning. ArXiv e-prints, June 2018.

[34] Ingrid Daubechies. Ten lectures on wavelets, volume 61. Siam, 1992.

[35] Timothy A Davis and Yifan Hu. The University of Florida sparse matrix collection. ACM Transactions on Mathematical Software (TOMS), 38(1):1, 2011.

[36] Stephen Demko, William F Moss, and Philip W Smith. Decay rates for inverses of band matrices. Mathematics of Computation, 43(168):491–499, 1984.

[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.

[38] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[39] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy I/O. In International Conference on Machine Learning, ICML, 2017.

[40] Kedar Dhamdhere, Kevin S McCurley, Ralfi Nahmias, Mukund Sundararajan, and Qiqi Yan. Analyza: Exploring data with conversation. In Proceedings of the 22nd International Conference on Intelligent User Interfaces, pages 493–504. ACM, 2017.

[41] Kedar Dhamdhere, Kevin S. McCurley, Mukund Sundararajan, and Ankur Taly. Abductive matching in question answering. CoRR, abs/1709.03036, 2017.

[42] Yi Ding, Risi Kondor, and Jonathan Eskreis-Winkler. Multiresolution kernel approximation for gaussian process regression. In Advances in Neural Information Processing Systems, pages 3740–3748, 2017.

[43] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In Eric P. Xing and Tony Jebara, editors, Proceedings of the 31st International Conference on Machine Learning, volume 32 of Proceedings of Machine Learning Research, pages 647–655, Bejing, China, 22–24 Jun 2014. PMLR.

[44] Cicero Dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 69–78, 2014.

[45] Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. journal of machine learning research, 6(Dec):2153–2175, 2005.

[46] Iain S Duff, AM Erisman, CW Gear, and John K Reid. Sparsity structure and Gaussian elimination. ACM SIGNUM Newsletter, 23(2):2–8, 1988.

[47] Chris Dyer, Victor Chahuneau, and Noah A Smith. A simple, fast, and effective reparameterization of ibm model 2. In Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 644–648, 2013.

[48] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 31–36, 2018.

[49] Markus Faustmann, Jens Markus Melenk, and Dirk Praetorius. $\mathcal{H}$-matrix approximability of the inverses of FEM matrices. Numerische Mathematik, 131(4):615–642, 2015.

[50] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400, 2017.

[51] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3429–3437, 2017.

[52] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nystrom method. IEEE transactions on pattern analysis and machine intelligence, 26(2):214–225, 2004.

[53] Alexander Fraser and Daniel Marcu. Measuring word alignment quality for statistical machine translation. Computational Linguistics, 33(3):293–303, 2007.

[54] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. arXiv preprint arXiv:1606.01847, 2016.

[55] Reza Ghaeini, Xiaoli Z Fern, and Prasad Tadepalli. Interpreting recurrent and attention-based neural models: a case study on natural language inference. arXiv preprint arXiv:1808.03894, 2018.

[56] Nicolas Gillis. The why and how of nonnegative matrix factorization. Regularization, Optimization, Kernels, and Support Vector Machines, 12(257):257–291, 2014.

[57] Alex Gittens and Michael W Mahoney. Revisiting the nyström method for improved large-scale machine learning. The Journal of Machine Learning Research, 17(1):3977–4041, 2016.

[58] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In International Conference on Learning Representations, 2015.

[59] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211, 2013.

[60] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. arXiv preprint arXiv:1612.00837, 2016.

[61] Lars Grasedyck, Ronald Kriemann, and Sabine Le Borne. Domain decomposition based LU preconditioning. Numerische Mathematik, 112(4):565–600, 2009.

[62] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. CoRR, 2014.

[63] Marcus J Grote and Thomas Huckle. Parallel preconditioning with sparse approximate inverses. SIAM Journal on Scientific Computing, 18(3):838–853, 1997.

[64] Alfred Haar. Zur theorie der orthogonalen funktionensysteme. Mathematische Annalen, 69(3):331–371, 1910.

[65] Wolfgang Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part I: introduction to $\mathcal{H}$-matrices. Computing, 62(2):89–108, 1999.

[66] Wolfgang Hackbusch, Boris Khoromskij, and Stefan A Sauter. On h 2-matrices. In Lectures on applied mathematics, pages 9–29. Springer, 2000.

[67] Tameru Hailesilassie. Rule extraction algorithm for deep neural networks: A review. CoRR, abs/1610.05267, 2016.

[68] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. The Mathematical Intelligencer, 27(2):83–85, 2005.

[69] Till Haug, Octavian-Eugen Ganea, and Paulina Grnarova. Neural multi-step reasoning for question answering on semi-structured tables. CoRR, abs/1702.06589, 2017.

[70] Stuart C Hawkins and Ke Chen. An implicit wavelet sparse approximate inverse preconditioner. SIAM Journal on Scientific Computing, 27(2):667–686, 2005.

[71] Magnus Rudolph Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems, volume 49. NBS Washington, DC, 1952.

[72] Elad Hoffer, Itay Hubara, and Daniel Soudry. Fix your classifier: the marginal value of training the last weight layer. CoRR, abs/1801.04540, 2018.

[73] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017.

[74] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 7132–7141, 2018.

[75] Jia-Hong Huang, Cuong Duc Dao, Modar Alfadly, and Bernard Ghanem. A novel framework for robustness analysis of visual qa models. arXiv preprint arXiv:1711.06232, 2017.

[76] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, pages 448–456. JMLR.org, July 2015.

[77] Vamsi K Ithapu. Decoding the deep: Exploring class hierarchies of deep representations using multiresolution matrix factorization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 45–54, 2017.

[78] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. arXiv preprint arXiv:1804.06059, 2018.

[79] Henrik Jacobsson. Rule extraction from recurrent neural networks: A taxonomy and review. Neural Computation, 17:1223–1263, 2005.

[80] Sarthak Jain and Byron C Wallace. Attention is not explanation. arXiv preprint arXiv:1902.10186, 2019.

[81] Rodolphe Jenatton, Guillaume Obozinski, and Francis Bach. Structured sparse principal component analysis. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pages 366–373, 2010.

[82] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017, 2017.

[83] Ian Jolliffe. Principal component analysis. Springer, 2011.

[84] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In Advances in Neural Information Processing Systems NIPS, pages 190–198, 2015.

[85] Kushal Kafle and Christopher Kanan. An analysis of visual question answering algorithms. In 2017 IEEE International Conference on Computer Vision (ICCV), pages 1983–1991. IEEE, 2017.

[86] George Karypis and Vipin Kumar. A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1998.

[87] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I, 2017.

[88] Vahid Kazemi and Ali Elqursh. Show, ask, attend, and answer: A strong baseline for visual question answering. arXiv preprint arXiv:1704.03162, 2017.

[89] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In Thirty-second AAAI conference on artificial intelligence, 2018.

[90] Dennis F. Kibler and Pat Langley. Machine learning as an experimental science. In Proceedings of the Third European Working Session on Learning, EWSL 1988, pages 81–92, 1988.

[91] Sangwook Kim, Swathi Kavuri, and Minho Lee. Deep network with support vector machines. In Minho Lee, Akira Hirose, Zeng-Guang Hou, and Rhee Man Kil, editors, Neural Information Processing, pages 458–465, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[92] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences, 114(13):3521–3526, 2017.

[93] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. arXiv preprint arXiv:1706.03872, 2017.

[94] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, pages 48–54. Association for Computational Linguistics, 2003.

[95] Risi Kondor, Nedelina Teneva, and Vikas Garg. Multiresolution matrix factorization. In Proceedings of the 31st International Conference on Machine Learning (ICML-14), pages 1620–1628, 2014.

[96] Risi Kondor, Nedelina Teneva, and Pramod Kaushik Mudrakarta. Parallel MMF: a multiresolution approach to matrix computation. arXiv preprint arXiv:1507.04396, 2015.

[97] Risi Kondor, Nedelina Teneva, and Pramod Kaushik Mudrakarta. Parallel MMF: a multiresolution approach to matrix computation. CoRR, abs/1507.04396, 2015.

[98] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492, 2016.

[99] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Computer, (8):30–37, 2009.

[100] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In Proceedings of the IEEE International Conference on Computer Vision Workshops, pages 554–561, 2013.

[101] Ronald Kriemann and Sabine Le Borne. $\mathcal{H}$-FAINV: hierarchically factored approximate inverse preconditioners. Computing and Visualization in Science, 17(3):135–150, 2015.

[102] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. ArXiv e-prints, June 2018.

[103] Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. Neural semantic parsing with type constraints for semi-structured tables. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 1516–1526, 2017.

[104] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[105] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States., pages 1106–1114, 2012.

[106] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling methods for the nyström method. Journal of Machine Learning Research, 13(Apr):981–1006, 2012.

[107] Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random access machines. In International Conference on Learning Representations ICLR, 2016.

[108] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Jure Leskovec. Interpretable & explorable approximations of black box models. arXiv preprint arXiv:1707.01154, 2017.

[109] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. Discourse processes, 25(2-3):259–284, 1998.

[110] Ann B Lee, Boaz Nadler, and Larry Wasserman. Treelets–an adaptive multi-scale basis for sparse unordered data. arXiv preprint arXiv:0707.0481, 2007.

[111] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In Advances in neural information processing systems, pages 556–562, 2001.

[112] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In Advances in neural information processing systems, pages 4652–4662, 2017.

[113] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. CoRR, abs/1603.04779, 2016.

[114] Percy Liang. Lambda dependency-based compositional semantics. CoRR, abs/1309.4408, 2013.

[115] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.

[116] Lemao Liu, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. Neural machine translation with supervised attention. arXiv preprint arXiv:1609.04186, 2016.

[117] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 8759–8768, 2018.

[118] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016.

[119] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. arXiv preprint arXiv:1807.11164, 2018.

[120] Michael W Mahoney and Petros Drineas. Cur matrix decompositions for improved data analysis. Proceedings of the National Academy of Sciences, 106(3):697–702, 2009.

[121] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. arXiv preprint arXiv:1306.5151, 2013.

[122] Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(7):674–693, 1989.

[123] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pages 2924–2932. Curran Associates, Inc., 2014.

[124] Pramod Kaushik Mudrakarta and Risi Kondor. A generic multiresolution preconditioner for sparse symmetric systems. arXiv preprint arXiv:1707.02054, 2017.

[125] Pramod Kaushik Mudrakarta, Mark Sandler, Andrey Zhmoginov, and Andrew Howard. K for the price of 1: Parameter-efficient multi-task and transfer learning. In International Conference on Learning Representations, 2019.

[126] Pramod Kaushik Mudrakarta, Ankur Taly, Mukund Sundararajan, and Kedar Dhamdhere. Did the model understand the question? In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1896–1906. Association for Computational Linguistics, 2018.

[127] Pramod Kaushik Mudrakarta, Ankur Taly, Mukund Sundararajan, and Kedar Dhamdhere. It was the training data pruning too! arXiv preprint arXiv:1803.04579, 2018.

[128] W. James Murdoch and Arthur Szlam. Automatic rule extraction from long short term memory networks. CoRR, abs/1702.02540, 2017.

[129] FD Murnaghan and Aurel Wintner. A canonical form for real matrices under orthogonal transformations. Proceedings of the National Academy of Sciences of the United States of America, 17(7):417, 1931.

[130] M Naumov, LS Chien, P Vandermersch, and U Kapasi. Cusparse library. In GPU Technology Conference, 2010.

[131] Arvind Neelakantan, Quoc V. Le, Martín Abadi, Andrew McCallum, and Dario Amodei. Learning a natural language interface with neural programmer. 2017.

[132] Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. In International Conference on Learning Representations ICLR, 2016.

[133] Duy-Kien Nguyen and Takayuki Okatani. Improved fusion of visual and language representations by dense symmetric co-attention for visual question answering. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[134] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. arXiv preprint arXiv:1803.02999, 2018.

[135] M-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing, Dec 2008.

[136] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. Computational Linguistics, 29(1):19–51, 2003.

[137] Christopher C Paige and Michael A Saunders. Solution of sparse indefinite systems of linear equations. SIAM Journal on Numerical Analysis, 12(4):617–629, 1975.

[138] Emilio Parisotto, Abdelrahman Mohamed, Rishabh Singh, Lihong Li, Denny Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. In International Conference on Learning Representations ICLR, 2017.

[139] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In In Proceedings of the Annual Meeting of the Association for Computational Linguistics, 2015.

[140] Panupong Pasupat and Percy Liang. Inferring logical forms from denotations. In In Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL 2016, 2016.

[141] Svetlin Penkov and Subramanian Ramamoorthy. Explaining transition systems through program induction. CoRR, abs/1705.08320, 2017.

[142] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 1532–1543, 2014.

[143] Fábio Henrique Pereira, Sérgio Luís Lopes Verardi, and Silvio Ikuyo Nabeta. A fast algebraic multigrid preconditioned conjugate gradient solver. Applied Mathematics and Computation, 179(1):344–351, 2006.

[144] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016, pages 2383–2392, 2016.

[145] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Nothing else matters: model-agnostic explanations by identifying prediction invariance. arXiv preprint arXiv:1611.05817, 2016.

[146] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging nlp models. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 856–865, 2018.

[147] Barbara Rosario. Latent semantic indexing: An overview.

[148] Amir Rosenfeld and John K. Tsotsos. Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing. CoRR, abs/1802.00844, 2018.

[149] John W Ruge and Klaus Stüben. Algebraic multigrid. Multigrid methods, 3(13):73–130, 1987.

[150] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. arXiv preprint arXiv:1509.00685, 2015.

[151] Youcef Saad. Sparskit: A basic tool kit for sparse matrix computations. 1990.

[152] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on scientific and statistical computing, 7(3):856–869, 1986.

[153] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. arXiv preprint arXiv:1801.04381, 2018.

[154] Wilhelmus HA Schilders. Iterative solution of linear systems in circuit simulation. In Progress in Industrial Mathematics at ECMI 2000, pages 272–277. Springer, 2002.

[155] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909, 2015.

[156] Sofia Serrano and Noah A Smith. Is attention interpretable? arXiv preprint arXiv:1906.03731, 2019.

[157] Xing Shi, Inkit Padhi, and Kevin Knight. Does string-based neural mt learn source syntax? In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1526–1534, 2016.

[158] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. CoRR, 2016.

[159] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. CoRR, 2013.

[160] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[161] Rishabh Singh and Pushmeet Kohli. AP: Artificial Programming. In Benjamin S. Lerner, Rastislav Bodík, and Shriram Krishnamurthi, editors, 2nd Summit on Advances in Programming Languages (SNAPL 2017), volume 71 of Leibniz International Proceedings in Informatics (LIPIcs), pages 16:1–16:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[162] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. CoRR, 2014.

[163] Matthew Streeter. Approximation algorithms for cascading prediction models. In ICML, 2018.

[164] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, pages 3319–3328, 2017.

[165] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 3104–3112, 2014.

[166] Wim Sweldens. The lifting scheme: a construction of second generation wavelets. SIAM Journal on Mathematical Analysis, 29(2):511–546, 1998.

[167] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2818–2826, 2016.

[168] Gongbo Tang, Rico Sennrich, and Joakim Nivre. An analysis of attention mechanisms: The case of word sense disambiguation in neural machine translation. arXiv preprint arXiv:1810.07595, 2018.

[169] Nedelina Teneva, Pramod Kaushik Mudrakarta, and Risi Kondor. Multiresolution matrix compression. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, volume 51 of Proceedings of Machine Learning Research, pages 1441–1449. PMLR, 2016.

[170] Damien Teney, Peter Anderson, Xiaodong He, and Anton van den Hengel. Tips and tricks for visual question answering: Learnings from the 2017 challenge. arXiv preprint arXiv:1708.02711, 2017.

[171] Sebastian Thrun. Extracting rules from artificial neural networks with distributed representations. In Proceedings of the 7th International Conference on Neural Information Processing Systems, NIPS'94, pages 505–512. MIT Press, 1994.

[172] Michael J Todd and Yinyu Ye. A centered projective algorithm for linear programming. Mathematics of Operations Research, 15(3):508–529, 1990.

[173] Henk A Van der Vorst. Bi-CGStab: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM Journal on Scientific and Statistical Computing, 13(2):631–644, 1992.

[174] Charles F Van Loan. The ubiquitous Kronecker product. Journal of Computational and Applied Mathematics, 123(1):85–100, 2000.

[175] Eva Vanmassenhove, Jinhua Du, and Andy Way. Investigating 'aspect'in nmt and smt: Translating the english simple past and present perfect. Computational Linguistics in the Netherlands Journal, 7:109–128, 2017.

[176] Panayot S Vassilevski and Junping Wang. Stabilizing the hierarchical basis by approximate wavelets, I: theory. Numerical Linear Algebra with Applications, 4(2):103–126, 1997.

[177] Panayot S Vassilevski and Junping Wang. Stabilizing the hierarchical basis by approximate wavelets II: implementation and numerical results. SIAM Journal on Scientific Computing, 20(2):490–514, 1998.

[178] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008, 2017.

[179] Elena Voita, Pavel Serdyukov, Rico Sennrich, and Ivan Titov. Context-aware neural machine translation learns anaphora resolution. arXiv preprint arXiv:1805.10163, 2018.

[180] Mark Weiser. Program slicing. In Proceedings of the 5th International Conference on Software Engineering, ICSE '81, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.

[181] Scott White and Padhraic Smyth. A spectral clustering approach to finding communities in graphs. In Proceedings of the 2005 SIAM international conference on data mining, pages 274–285. SIAM, 2005.

[182] Sarah Wiegreffe and Yuval Pinter. Attention is not not explanation. arXiv preprint arXiv:1908.04626, 2019.

[183] Christopher KI Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In Advances in neural information processing systems, pages 682–688, 2001.

[184] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144, 2016.

[185] Yuanzhe Xi, Ruipeng Li, and Yousef Saad. An algebraic multilevel preconditioner with low-rank corrections for sparse symmetric matrices. SIAM Journal on Matrix Analysis and Applications, 37(1):235–259, 2016.

[186] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S Li. Fast algorithms for hierarchically semiseparable matrices. Numerical Linear Algebra with Applications, 17(6):953–976, 2010.

[187] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In International conference on machine learning, pages 2048–2057, 2015.

[188] YINYU Ye. Interior-point algorithms for quadratic programming. Recent Developments in Mathematical Programming, pages 237–261, 1991.

[189] Yinyu Ye. On the finite convergence of interior-point algorithms for linear programming. Mathematical Programming, 57(1):325–335, 1992.

[190] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, pages 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[191] Harry Yserentant. Hierarchical bases give conjugate gradient type methods a multigrid speed of convergence. Applied Mathematics and Computation, 19(1-4):347–358, 1986.

[192] Harry Yserentant. On the multi-level splitting of finite element spaces. Numerische Mathematik, 49(4):379–412, 1986.

[193] Adams Wei Yu, David Dohan, Quoc Le, Thang Luong, Rui Zhao, and Kai Chen. Fast and accurate reading comprehension by combining self-attention and convolution. In International Conference on Learning Representations, 2018.

[194] Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. Generating fluent adversarial examples for natural languages. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 5564–5569, Florence, Italy, July 2019. Association for Computational Linguistics.

[195] Kai Zhang, Ivor W Tsang, and James T Kwok. Improved nyström low-rank approximation and error analysis. In Proceedings of the 25th international conference on Machine learning, pages 1232–1239. ACM, 2008.

[196] Peng Zhang, Yash Goyal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Yin and yang: Balancing and answering binary visual questions. In Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on, pages 5014–5022. IEEE, 2016.

[197] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. CoRR, abs/1707.01083, 2017.

[198] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. IEEE transactions on pattern analysis and machine intelligence, 2017.

[199] Yuke Zhu, Oliver Groth, Michael Bernstein, and Li Fei-Fei. Visual7w: Grounded question answering in images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4995–5004, 2016.