Nonparametric High-dimensional Models: Sparsity, Efficiency, Interpretability

By

Shibal Ibrahim

B.S., Lahore University of Management Sciences (2014)
S.M., Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by:	Shibal Ibrahim Department of Electrical Engineering and Computer Science May 17, 2024
Certified by:	Rahul Mazumder Associate Professor of Operation Research and Statistics Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Students

Nonparametric High-dimensional Models: Sparsity, Efficiency, Interpretability

by

Shibal Ibrahim

Submitted to the Department of Electrical Engineering and Computer Science on May 17, 2024 in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

ABSTRACT

This thesis explores ensemble methods in machine learning, a technique that builds a predictive model by jointly training simpler base models. It examines three types of ensemble methods: additive models, tree ensembles, and mixtures of experts. Each ensemble method is characterized by a specific structure: additive models can involve base learners with single or pairwise covariates, tree ensembles use a decision tree as a base learner, and mixtures of experts typically employ neural networks. The focus of this thesis is on considering various sparsity and structural constraints within these methods and develop optimization based approaches to enhance training efficiency, inference, and/or interpretability.

In the first part, we consider additive models with interactions under component selection constraints and additional structural constraints e.g., hierarchical interactions. We consider different optimization based formulations and propose efficient algorithms to learn a good subset of components. We develop two toolkits that are scalable to large number of samples and large set of pairwise interactions.

In the second part, we consider tree ensemble learning. In this setting, we consider flexible and efficient formulation of differentiable tree ensemble learning. We study flexible loss functions, multitask learning etc. We also consider end-to-end feature selection in tree ensembles, i.e., we perform feature selection while training of tree ensembles. This is in contrast to popular tree ensemble learning toolkits, which perform post-training feature selection based on feature importances. Our toolkit provides substantial improvements in predictive performance for a desired feature budget.

In the third part, we consider sparse gating in mixture of experts. Sparse Mixture of Experts is a paradigm where a subset of experts (typically neural networks) are activated for each input sample. This is used to scale training as well as inference of large-scale vision and language models. We consider multiple approaches to improve sparse gating in mixture of expert models. Our new approaches show improvements in large-scale experiments on machine translation as well as distillation of pre-trained models on natural language processing tasks.

Thesis supervisor: Rahul Mazumder
Title: Associate Professor of Operation Research and Statistics

Acknowledgments

I extend my deepest and heartfelt appreciation to my advisor, Rahul Mazumder. His steadfast support, inspiring enthusiasm, and relentless encouragement have been the bedrock of my PhD journey. His enthusiasm for our research area has not only invigorated my own passion but also spurred me to explore creative avenues of thought. The countless hours we have invested in discussions and brainstorming sessions have been both intellectually enriching and profoundly influential in honing my research skills. His guidance in navigating the complexities of research, his lessons in approaching problems from a strategic vantage point, and his unwavering belief in my potential have been pivotal in molding my academic and personal growth. I consider myself extraordinarily lucky to have been under the mentorship of such a dedicated and inspiring mentor.

I am also profoundly thankful to my esteemed thesis committee members, Rahul Mazumder, Patrick Jaillet, and Stephen Bates. Their invaluable support, perceptive advice, and commitment to the excellence of my dissertation have been instrumental in my academic progress. My heartfelt gratitude also goes to Patrick Jaillet and Ashia Wilson for their insightful comments during my research qualifying exam.

My journey at MIT has been greatly enriched by the intellectual stimulation provided by my colleagues. Special thanks to Hussein Hazimeh, Wenyu Chen, Kayhan Behdin, and Haoyue Wang, whose respective expertise in various machine learning, optimization and statistics domains have been immensely beneficial. My experience in supervising Gabriel I. Afriat and Xiang Meng have been particularly rewarding, offering me fresh perspective and learning. I am deeply appreciative of their friendship and collegiality. A sincere acknowledgement goes to Brian Hsu, Mathieu Sibue, Paul Theron, Max R. Tell, Denis Sai, Danying Xiao, Xiaming Jin, and Peijun Xu.

The opportunity to interact with esteemed researchers beyond MIT has been a privilege. In this regard, I am particularly grateful to Professor Peter Radchenko for his critical insights into statistics and constructive feedback. My time at Google Research as an intern was significantly enhanced by the mentorship of Natalia Ponomareva, to whom I am immensely thankful. I also enjoyed working with Abhishek Nandy, Yada Zhu, and Emmanuel Ben-David.

I am equally grateful to the faculty and staff at MIT, especially Una-May O'Reilly, Janet E. Fischer, and Leslie A. Kolodziejski, who were extremely supportive during my research area transition from Electrical Engineering to Artificial Intelligence and Decision Making. I would also like to thank Peter Szolovits, Jacob White, Luca Daniel, and Alexandre Megretski, who enriched my teaching experience, when I TA'ed their classes.

My undergraduate experiences have also been crucial in shaping my academic path. I owe a debt of gratitude to Nauman Zaffar, my advisor at Lahore University of Management

Sciences, for his guidance in engineering research. My appreciation also extends to Abubakr Muhammad, Zartash Afzal, and Nauman Zaffar for their pivotal advice during my decision to embark on this PhD journey.

My friends at MIT, Syed Imaduddin, Ibrahim Wasiq Khan, Zaid Saeed Khan, and Ahmed Bilal have enriched this journey immeasurably. My heartful thanks to my LUMS friends, including Rehan, Soban, Musa, Umair, Furqan, Farwa, Yasar, Usama, Talha, Muzammil, among others, for their friendship and encouragement.

Finally, I reserve my most profound gratitude for my family: my parents Muhammad Ibrahim and Tehseen Ibrahim, my siblings Sadaf, Adeel, and their partners Omair and Maheen. Their unconditional love, unwavering support, and constant encouragement have been my anchor throughout this endeavor. To my nieces and nephews, Inaya, Wali, and Hashir, whose smiles and joy have been a source of immense happiness in challenging times, I am eternally grateful. This thesis is a tribute to their unwavering faith and love.

Biographical Sketch

Shibal Ibrahim was born in Lahore, Pakistan on June 9, 1991. He received his B.S. degree in Electrical Engineering from Lahore University of Management Sciences (LUMS) in 2014. He worked for a year as a research assistant to develop power management solutions before joining MIT Department of Electrical Engineering and Computer Science as a graduate student. He completed his S.M. degree in 2018 in Electrical Engineering and Computer Science, where he worked on development of power electronics for ship micro-grids. During this time at MIT, he developed interest in Machine Learning and Artificial Intelligence and decided to pursue a PhD in Machine Learning. He joined Dr. Rahul Mazumder's research group in 2019, where he worked on various topics in nonparametric additive models, trees and sparse mixture of experts etc. He graduated with a PhD in Machine Learning from MIT in 2024.

Contents

Title page **1**

Abstract **3**

Acknowledgments **5**

Biographical Sketch **7**

List of Figures **15**

List of Tables **19**

1 Introduction **23**

2 Generalized Additive Models with Interactions under Sparsity and Structural Constraints **29**

2.1 Introduction 29
2.2 Related Work 33
2.3 Problem Formulation 35
2.3.1 An optimization formulation with binary variables 35
2.3.2 A Smooth Reformulation 36
2.3.2.1 Relaxing Binary Variables with Smooth Gates 37
2.3.2.2 Soft trees 38
2.3.2.3 Statistical Theory 38
2.4 Efficient Implementation 39
2.5 Experiments 41
2.5.1 Prediction Performance 42
2.5.2 Variable Selection 44
2.5.3 Computational Scalability 44
2.5.4 Variance Reduction with Structural Constraints 45
2.6 Conclusion 46
2.7 Appendix 47
2.7.1 Definition of Soft Trees 47
2.7.2 Extension to third-order interactions 47
2.7.3 Datasets, Computing Setup and Tuning 48
2.7.4 Additional Results 50

2.7.4.1 Comparison with full complexity models e.g., DNN 50
2.7.4.2 GRAND-SLAMIN versus GAMs with all pairwise interactions . 51
2.7.4.3 Comparison with Group Lasso 52
2.7.4.4 GRAND-SLAMIN with different shape functions (Soft Trees vs MLPs) 53
2.7.4.5 Effect of entropy regularization 54
2.7.4.6 Effect of screening on performance 55
2.7.4.7 Timing comparison 56
2.7.5 Additional visualizations for variance reduction in estimation of main effects under structural constraints 56

3 Additive Models and Structured Interactions: Alternative Optimization Approaches 59
3.1 Introduction 59
3.2 Statistical Models and Methodology 63
3.2.1 Smooth additive models with pairwise interactions 63
3.2.2 Parsimonious models via ℓ_0-penalization 64
3.2.2.1 Sparse pairwise interactions 64
3.2.3 Efficient Computations at Scale 67
3.2.4 Block Coordinate Descent for solving (3.2.4) 69
3.2.5 Scalability considerations for solving (3.2.4) 71
3.2.5.1 Active set updates 71
3.2.5.2 Cached matrix factorizations 72
3.2.5.3 Warm-starts 72
3.3 Incorporating strong hierarchy constraints 73
3.3.1 Algorithms for ELAAN-H with sparse hierarchical interactions: Problem (3.3.2) 75
3.4 Simulations 76
3.4.1 Sparse additive model with interactions 76
3.4.2 Large-scale setting with correlated features 79
3.4.3 Conclusion 80
3.5 Appendix 80
3.5.1 Computational details 80
3.5.1.1 Univariate and bivariate smooth function estimation with splines 80
3.5.1.2 A finite dimensional quadratic program 83
3.5.2 Additional Details for Simulations 84
3.5.2.1 Definition of F1-score 84
3.5.2.2 Effect of number of basis functions 84
3.5.2.3 Tuning Details 85

4 Additive Models and Structured Interactions: A Large-Scale Case Study for Census Survey Response Prediction 87
4.1 Introduction 87
4.2 Additional Motivation for Additive Models with Interactions 91

4.2.0.1 Sparse pairwise interactions 92
4.3 Predicting the Census Survey Self-Response Rate 94
4.3.1 Experimental setup 95
4.3.2 Comparing methods: prediction, sparsity and model structure 97
4.3.3 Interpreting important features 103
4.3.4 Conclusion 106
4.4 Appendix 107
4.4.1 Definition of important Census/American Community Survey variables 107
4.4.1.1 Definition of the variables in Figure 4.2.2 107
4.4.1.2 Definition of the variables in Figure 4.3.4(b) 111

5 Tree Ensembles: Flexible Modeling and Efficient Training 115
5.1 Introduction 115
5.2 Related Work 117
5.3 Optimizing Tree Ensembles 120
5.3.1 Preliminaries and Setup 120
5.3.2 Efficient Tensor Formulation 123
5.3.3 Toolkit 124
5.4 Flexible loss functions 125
5.5 Multi-task Learning with Tree Ensembles 128
5.6 Experiments 130
5.6.1 Studying a Single Tree 131
5.6.2 Zero-inflation 132
5.6.3 Multi-task Regression 134
5.6.4 Large-scale multi-task data from a multinational financial services company 136
5.7 Conclusion 138
5.8 Appendix 138
5.8.1 Tuning parameters and optimal depths comparison for a single tree in Section 5.6.1 139
5.8.2 Tuning parameters for Sections 5.6.2 140
5.8.3 Tuning parameters for Sections 5.6.3 141
5.8.4 Tuning parameters for Sections 5.6.4 141
5.8.5 Exploration of different loss and training strategies for tree ensemble learning 142

6 Tree Ensembles: End-to-end Feature Selection Approach for Learning Skinny Trees 145
6.1 Introduction 145
6.2 Related Work 147
6.3 Problem Formulation 149
6.4 End-to-end Optimization Approach 151
6.4.1 Proximal mini-batch gradient descent 152
6.4.2 Convergence Analysis of Algorithm 1 153
6.4.3 Dense-to-Sparse Learning (DSL) 153

6.5 Synthetic Experiments 154
6.6 Real Data Experiments 156
6.6.1 Studying a single tree 157
6.6.2 `Skinny Trees` vs Dense Soft Trees 158
6.6.3 `Skinny Trees` vs Classical Trees 159
6.6.4 `Skinny Trees` vs Neural Networks 160
6.6.5 Dense-to-Sparse Learning 161
6.6.6 Discussion on training times. 161
6.7 Conclusion 161
6.8 Appendix 162
6.8.1 Experimental Details for section 6.5 164
6.8.2 Experimental Details for section 6.6 165
6.8.2.1 Datasets, Computing Setup and Tuning Setup 165
6.8.2.2 Experimental Details for Sec. 6.6.1 166
6.8.2.3 Tuning Details for Sec. 6.6.2 and 6.6.3 169
6.8.2.4 Experimental Details for Sec. 6.6.4 170
6.8.2.5 Tuning Details for Sec. 6.6.5 171
6.8.2.6 Comparison with ControlBurn 172
6.8.3 Measuring Statistical Significance 173

7 Sparse Mixture of Experts: A Cardinality Constrained Routing Approach with Trees and Local Search 175
7.1 Introduction 175
7.2 Related work 178
7.3 Learning Sparse Mixture of Experts with decision trees 179
7.3.1 Sparse-MoE with k decision trees 181
7.3.2 Preliminaries: Differential Decision Tree with Conditional Computation 181
7.3.3 Cardinality constraint with k trees 184
7.3.4 Non-powers of 2 185
7.3.5 Stable numerical implementation 186
7.4 Local search 186
7.4.1 Permutation-based Local Search 187
7.4.2 Preliminaries: Permutation and a differentiable relaxation 188
7.4.3 Practical considerations for optimization 190
7.4.4 Ablation study for local search 192
7.5 Experiments 193
7.5.1 Experiments on Recommender Systems and Image Datasets 194
7.5.1.1 Performance of *COMET* and *COMET+* 196
7.5.1.2 Reducing Hyperparameter Search with *COMET* 197
7.5.1.3 Effect of Local Search on Hyperparameter Tuning 198
7.5.2 Experiments on NLP Tasks 200
7.6 Conclusion 201
7.7 Appendix 202
7.7.1 Proof for Proposition 7.3.1 202
7.7.2 *COMET* for non-powers of 2 202

7.7.3 Timing Discussion 203
7.7.3.1 Cost Complexity 203
7.7.3.2 FLOPs Comparison 204
7.7.3.3 Effect of local search on computation 205
7.7.4 Task-specific metrics corresponding to Tables 7.5.1 and 7.5.2 207
7.7.5 Bootstrapping Procedure for studying hyperparameter tuning 207
7.7.6 Additional Details for Section 5.1 207
7.7.6.1 Datasets 207
7.7.6.2 Architectures 209
7.7.6.3 Hyperparameters and Tuning 210
7.7.7 Additional Details for Section 5.2 215
7.7.7.1 Datasets 215
7.7.7.2 Tuning Procedure for MoEBERT and *COMET-BERT* . . . 216

8 Sparse Mixture of Experts: An Effective Sampling-based Routing Approach 219
8.1 Introduction 219
8.2 Related work 221
8.3 Routing in Mixture of Experts 223
8.3.1 *MOESART* for Conditional Computing 225
8.3.1.1 Computing $\tilde{g}$ during training and inference 228
8.3.2 Additional Trimmed Lasso regularization 230
8.3.3 *MOESART* versus S-MoE(S). 231
8.4 Experiments 231
8.4.1 Experiments on Recommender Systems and Images 231
8.4.2 Experiments on NLP Tasks 234
8.4.2.1 Distillation on Language Understanding and Question Answering Tasks 234
8.4.2.2 Machine Translation Tasks 235
8.4.3 Ablation Studies for *MOESART* 236
8.5 Conclusion 238
8.6 Appendix 239
8.6.1 Proof for $\tilde{g}_i$ in (iii) being an unbiased estimator of g_i 239
8.6.2 Additional Ablation Studies 240
8.6.2.1 Bias of different adjustment strategies in (i)-(ii) and proposed strategy in (8.3.3) 241
8.6.2.2 Performance comparison of our proposed adjustment in (8.3.3) versus (i)-(ii) on Image datasets. 242
8.6.2.3 Effect of varying $\boldsymbol{k}$ 243
8.6.2.4 Effect of Trimmed Lasso regularization on performance on Recommender Systems 243
8.6.2.5 Performance under Load balancing 245
8.6.2.6 Sensitivity of *MOESART* to hyperparameter tuning 245
8.6.2.7 Comparison of *MOESART* with Differentiable Gates 246
8.6.2.8 Comparison of *MOESART* with S-MoE(S) 248
8.6.2.9 Architectures 249

8.6.2.10 Hyperparameters and Tuning 250
8.6.3 Additional Details for Section 8.4.2 250
8.6.3.1 Tuning Procedure for MoEBERT 250

9 Pruning Large Vision and Language Models 253
9.1 Structured Pruning in Vision and Language Models with Combinatorial Optimization 253
9.2 Multi-Objective Pruning with Non-Uniform Layer-wise Sparsity Allocation in Vision Transformers 254

10 Conclusion 257

References 296

List of Figures

2.1 *Modeling main and interaction effects with soft trees. ϕ denotes the sigmoid activation function. We omit biases in split nodes for brevity. For interaction effect, $W^{j,k}_{i,1} \in \mathbb{R}$ and $W^{j,k}_{i,2} \in \mathbb{R}$ denote the weights in i-th node of the (j,k)-th tree.* 38
2.2 *`GRAND-SLAMIN` with standard (dense) backpropagation vs sparse backpropagation on Activity dataset. (a) shows the number of nonzero effects: $\sum_j z_j + \sum_{(j,k)\in\mathcal{I}} q(z_j, z_k, z_{j,k})$ and (b) shows the time for each epoch during the course of training.* 45
2.3 *Estimated main effects in the presence of interaction effects on bikesharing dataset [Left] without hierarchy, [Middle] weak hierarchy and [Right] strong hierarchy. Strong hierarchy has the smallest error bars.* 46
2.7.1 *Performance of `GRAND-SLAMIN` with various pre-training screening levels on Madelon.* 55
2.7.2 *Estimated main effects in the presence of interaction effects on American Community Survey dataset [Left] without hierarchy, [Middle] with weak hierarchy and [Right] with strong hierarchy. Strong hierarchy has the smallest error bars. We show visualization for 5 features: (a) Row 1: Average household income, (b) Row 2: Percentage of households with internet access, (c) Row 3: Percentage of limited English speaking households, (d) Row 4: Percentage of people below poverty level, (e) Row 5: Percentage of single-family homes.* 58

4.1.1 *The 2013-2017 American Community Survey self-response rates for all tracts in the continental United States. The North in general, and the Upper Midwest and the Northeast in particular, have higher self-response rates than the rest of the country. Tracts with lower self-response rates are visible in many states – in particular, in the South and in the Mountain region.* 88
4.2.1 *Panels [Left]-[Right] illustrate marginal nonparametric fits for the self-response rate output variable versus three covariates. Each marginal fit, displayed on a scatter plot with a solid blue line, clearly suggests a nonlinear relationship of the output vs the individual covariate (we note that the covariates are standardized.) The x-axis corresponds to: [Left] "Persons of Hispanic origin in the ACS"; [Middle] "Number of households that have only a smartphone and no other computing device"; [Right] "Persons 25 years and over with college degree or higher in the ACS".* 91

4.2.2 *Sparsity pattern of the main and interaction effects presented in a $p \times p$ matrix: a black square on the diagonal indicates the presence of a main effect, and an off-diagonal black square indicates the presence of an interaction effect in the joint model. [Left] Panel illustrates the sparsity pattern of a Lasso model with main and interaction effects. There are 37 main and 555 interaction effects in the optimal model. [Right] Panel illustrates the sparsity pattern of a nonlinear AM with main and interaction effects, i.e., model* (3.2.3)*. There are 8 main and 92 interaction effects in the optimal model. Model* (3.2.3) *has prediction performance similar to the Lasso model, with only 3 main and 33 interaction effects. Nonlinear models lead to much more compact and hence, easier to interpret models compared to linear models. Both models were trained on a 2019 US Census Bureau Planning Database dataset (predicting the tract-level self-response rate) with $p = 40$ covariates and $74,000$ observations.* 93
4.3.1 *Predicted ACS self-response rates for all tracts in the United States.* 101
4.3.2 *ACS self-response rates for all tracts in the District of Columbia. a) Actual ACS self-response rates. b) Predicted ACS self-response rates for AMs with interactions* (3.2.4)*. c) Difference between the actual and predicted self-response rate: difference = actual - predicted.* 101
4.3.3 *Comparison of quintile groups based on Lasso (linear model) [142, 250] and our proposed* ***ELAAN-I****.* 102
4.3.4 *Estimates available from our model* ***ELAAN-I*** *i.e., model* (3.2.4)*: ℓ_0-sparse AMs with main effects and interactions. [Left] number of nonzero effects vs the corresponding RMSE. [Right] Variables corresponding to main effects selected by the model along the regularization path. For visualization purposes, we focus on the top 19 main effects.* 104

5.3.1 *Timing comparison of classical formulation against our tensor-based formulation of a tree ensemble. Classical formulation models trees in the ensemble individually as pointed out in Section 5.3.2. Tensor-based formulation with CPU training is up to* $10\times$ *faster than classical formulation. Tensor-based formulation with GPU training leads to an additional 40% improvement, leading to an effective* $20\times$ *gain over classical formulation.* 123

6.3.1 *Illustration of* ***Skinny Trees****. Each horizontal slice $\boldsymbol{\mathcal{W}}_{k,:,:}$ depicts a single feature. White slices indicate features filtered out by the ensemble while training. Each vertical slice (along the depth of the page), $\boldsymbol{\mathcal{W}}_{:,:,j} = \boldsymbol{W}_j$ corresponds to parameters in j-th (splitting) supernode (blue circles) in the ensemble, eventually producing the routing decisions. The red squares depict leaf nodes. $S(\cdot)$ denotes an activation function, which can be Sigmoid [131] or Smooth-Step [96].* 150
6.4.1 *Trajectory of validation loss and feature sparsity during training with dense-to-sparse learning.* 154
6.5.1 *Features selected by Random Forests, XGBoost and* ***Skinny Trees*** *for different sample sizes* 155

6.6.1 *Performance without/with Dense-to-sparse learning for different feature selection budgets.* . . . 161

7.3.1 *COMET for 8 experts. Note $z_q(w_q{\cdot}x)$ denotes the binary state $\{0,1\}$ for $h(w_q{\cdot}x)$ achieved due to Smooth-step activation function and entropic regularization.* . . 182
7.5.1 *Sensitivity of COMET to hyperparameter tuning. COMET can achieve the same level of performance as popular gates (e.g., Hash-r and Top-k) with significantly lesser number of hyperparameter trials. We see tuning reduction by $5\times-100\times$ for COMET over Top-k and Hash routing.* . . . 198
7.5.2 *Effect of local search on hyperparameter tuning.* Comparison of *Hash-r+* vs Hash-r, *Top-k+* vs Top-k and *COMET+* vs *COMET* on MovieLens with two different task weight settings. Local search appears to achieve the same level of performance with much lesser number of hyperparameter trials. We see tuning reduction by a factor of $3\times-100\times$ for *Hash-r+*, $20\times-100\times$ for *Top-k+* and $2\times-5\times$ for *COMET+*. . . . 199
7.7.1 *COMET for 5 (non-powers of 2) experts.* . . . 203

8.3.1 *Sparse-MoE with MOESART gate.* . . . 225
8.4.1 *Out-of-sample loss for MOESART for different sampling strategies on SVHN.* 237
8.6.2 *Out-of-sample loss achieved by MOESART and Top-k for different k on SVHN.* 243
8.6.3 *Sensitivity of MOESART to hyperparameter tuning. MOESART* can achieve the same level of performance as Top-k with significantly lesser number of hyperparameter trials. We see tuning reduction by $100\times$ for *MOESART* over Top-k. . . . 246

List of Tables

2.1 *Relevant work on sparse GAMs with Interactions. Models in rows 1-2 have some variable selection but no hierarchy; models in row 3-5 have screening-based approaches for hierarchy.* . . . 33

2.2 *Test ROC AUC of* `GRAND-SLAMIN`*, EB^{2}M and NODE-GA^{2}M. We report median along with mean absolute deviation across 10 runs.* . . . 42

2.3 *Test ROC AUC for* `GRAND-SLAMIN` *with structural constraints i.e., (2.6) or (2.7), GAMI-Net and SIAN. We report median across 10 runs along with mean absolute deviation.* . . . 43

2.4 *Number of features used by* `GRAND-SLAMIN` *without/with additional structural constraints and competing approaches. Hyphen (-) indicates multiclass classification is not supported by GAMI-Net and SIAN.* . . . 44

2.7.1 *Summary of Datasets* . . . 49

2.7.2 *Test ROC AUC of* `GRAND-SLAMIN`*, EB^{2}M and NODE-GA^{2}M. We report median across 10 runs along with the mean absolute deviation (MAD).* . . . 51

2.7.3 *Test ROC AUC for* `GRAND-SLAMIN` *and GAMs with all pairwise interaction models, e.g., NA^{2}M, NB^{2}M and SPAM.* . . . 52

2.7.4 *Comparison of Test ROC AUC of* `GRAND-SLAMIN` *with Group Lasso.* . . . 53

2.7.5 *Soft tree shape functions vs MLP shape functions.* . . . 53

2.7.6 *Training times for different choices of MLP shape functions.* . . . 54

2.7.7 *AUC performance with MLP shape functions.* . . . 54

2.7.8 *Effect of entropy regularization τ on test AUC and component selection on Spambase. We report median and mean absolute deviation across 50 runs.* . . . 55

2.7.9 *Training time in seconds of* `GRAND-SLAMIN`*, EB^{2}M, NODE-GA^{2}M, GAMI-Net and SIAN. We report median across 10 runs. Hyphen (-) indicates either the toolkit does not support multiclass e.g., GAMI-Net, SIAN or does not fit interaction effects for multiclass e.g., EB^{2}M.* . . . 57

3.4.1 *Integrated Squared Errors for* `ELAAN-I`*,* `ELAAN-H`*, MARS, COSSO, EBM and GAMI-Net.* . . . 78

3.4.2 *Support recovery metric (F1-score) for the main effects and the interaction effects.* . . . 78

3.4.3 *Relative frequency of the true main and interaction effects appearing in the* `ELAAN-I` *and* `ELAAN-H` *models.* . . . 78

3.4.4 *Integrated Squared Errors and support recovery metrics for EBM, GAMI-Net, `ELAAN-I` and `ELAAN-H` on large-scale synthetic data with different correlation strengths (σ).* 79
3.5.1 *Average integrated squared error for different configurations of basis elements. "No" in Smoothing column indicates $\lambda_1 = 0$. "Yes" in Smoothing column indicates λ_1 is tuned via cross-validation. Note that for all cases λ_2 is tuned via cross-validation.* 84

4.3.1 *Comparisons of our methods with several benchmark models as discussed in the text. We display the average test RMSE for the different models, along with the corresponding number of covariates, and number of effects (main and interactions). Best metrics are highlighted in bold. Numbers after ± provide the standard errors. Asterisk(*) indicates statistical significance (p-value<0.05) over the best existing model, using a one-sided paired t-test. Models (ix), (x) are black-box models using higher order interactions, hence #Main and #Interactions are left as '-' (dash).* 97

5.6.1 *Test mean squared error of single axis-aligned and oblique decision trees on various regression datasets.* 132
5.6.2 *Test poisson deviance across models on various datasets. Flexible modeling via zero-inflated Poisson for Soft Tree Ensembles leads to better poisson deviance.* 133
5.6.3 *Tree ensemble compactness (# trees and depth) for GBDT and Soft Tree Ensembles for different datasets. Flexible modeling via zero-inflated Poisson for Soft Tree Ensembles can lead to more compact tree ensembles, which can improve interpretability* 134
5.6.4 *Test MSE of RF, GRF and multi-task Differentiable Tree Ensembles on 11 multi-task regression datasets with fully observed responses across tasks.* 135
5.6.5 *Tree ensemble sizes for soft trees, RF, and GRF.* 136
5.6.6 *Test MSE of GBDT, single-task and multi-task Soft Tree Ensembles on 11 multi-task regression datasets, with 50% missing responses per task.* 137
5.6.7 *Out-of-sample performance of single-task and multi-task tree ensembles with flexible loss functions for zero-inflation/overdispersion. We evaluate performance with weighted Poisson deviance and AUC across tasks.* 138
5.8.1 *Optimal depth of a single axis aligned and oblique decision tree on various regression datasets.* 140
5.8.2 Test AUC on 9 classification tasks for different ensemble learning techniques. 144

6.5.1 *Test MSE, feature sparsity and support recovery metrics (F1-score) for a linear setting with correlated design matrix. `Skinny Trees` outperforms feature-importance-based methods across all metrics.* 155
6.6.1 *Test AUC for TAO with ℓ_1 regularization, single soft tree with Group Lasso and `Skinny Tree` (a single soft tree with Group $\ell_0 - \ell_2$).* 158
6.6.2 *Test AUC for `Skinny Trees` vs* dense *Soft Trees. We also report feature compression.* 158

6.6.3 *Test AUC (%) performance of* **`Skinny Trees`** *and feature-importance-based toolkits for* **trees** *for 25% feature budget ($K = 0.25p$). Bold and italics indicates best and runner-up models respectively.* . . . 159
6.6.4 *Test AUC (%) performance of* **`Skinny Trees`** *and embedded feature selection methods from* ***neural networks*** *(LassoNet, AlgNet, DFS) for 25% feature budget.* . . . 160
6.8.1 *List of notation used.* . . . 163
6.8.2 *Classification datasets* . . . 165
6.8.3 *Comparison of test AUC (%) performance of* **`Skinny Trees`** *against Control-Burn for 25% feature selection budget on binary classification tasks.* . . . 172

7.4.1 *Test loss ($\times 10^{-2}$) achieved for different gates without and with (marked with +) local search in early stages of optimization. Asterisk(*) indicates statistical significance (p-value<0.05) over the corresponding gate without permutation with a one-sided unpaired t-test.* . . . 193
7.5.1 *Tess Loss ($\times 10^{-2}$) and actual no. of experts per sample (k_a) at inference for COMET, COMET+ and benchmark gates across various recommender system datasets. Asterisk(*) indicates statistical significance (p-value<0.05) over the best existing gate, using a one-sided unpaired t-test.* . . . 195
7.5.2 *Tess Loss ($\times 10^{-2}$) and actual no. of experts per sample (k_a) at inference for COMET, COMET+ and benchmark gates across various image datasets. Asterisk(*) indicates statistical significance (p-value<0.05) over the best existing gate, using a one-sided unpaired t-test.* . . . 196
7.5.3 *Performance metrics on the GLUE and SQuAD development sets. Models are trained without data augmentation. Both models have 66M (effective) parameters for inference.* . . . 201
7.7.1 *Time complexity of Top-k and COMET.* . . . 203
7.7.2 *#FLOPs per-sample at inference time for COMET against benchmarks (Softmax, Top-k, DSelect-k) across various datasets.* . . . 205
7.7.3 *Test AUC/Accuracy/MSE for COMET+ and benchmark gates on recommender systems and image datasets.* . . . 206
7.7.4 *Summary of GLUE benchmark.* . . . 215
7.7.5 *Summary of SQuAD benchmark.* . . . 216

8.4.1 *Test loss, task-specific metrics and number of experts activated per sample (k/s)* while *training for MOESART and existing purely sparse gating methods: (1) Top-k [233], (2) V-MoE [227], (3) S-MoE [72], (4) Expert Choice [290] and (5) X-MoE [48] across various datasets. Bold indicates statistical significance (p-value<0.05) over the best existing sparse gate, using a one-sided unpaired t-test.* . . . 233
8.4.2 *Performance metrics for distillation of BERT into MoEBERT with different gates on the GLUE and SQuAD development sets.* . . . 235
8.4.3 *Test BLEU scores for 1 billion MoE-Transformer with different gates on machine translation tasks.* . . . 235

8.6.1 *Comparison of test loss and task-specific metrics for MOESART with the proposed adjustment strategy in (8.3.3) against other strategies in (i)-(ii). We also include Top-k for reference.* 242
8.6.2 *Comparison of test loss for best trial for MOESART without and with Trimmed Lasso Regularization on Recommender system datasets.* 244
8.6.3 *Comparison of test loss for best trial for Top-k without/with Trimmed Lasso Regularization on Books dataset.* 244
8.6.4 *Test loss with load balancing on MovieLens.* 245
8.6.5 *Comparison of test loss and task-specific metrics for MOESART against differentiable gating methods: (1) DSelect-k [101], (2) COMET [117], across various datasets. Note that these differentiable gates can only support conditional training partially with customized implementations, hence they are more expensive during training.* 247
8.6.6 *Comparison of test loss and task-specific metrics for MOESART against sampling-base variant S-MoE(S) across various datasets.* 249

Chapter 1

Introduction

Ensemble methods are an important class of machine learning methods. The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models. Ensemble learning can be broken down into two tasks: developing a population of base learners as well as combining them to form the composite predictor. Ensemble learning appears in different forms in statistical literature. In this thesis, we study three different classes of ensembling methods: (i) additive models [93, 94], (ii) trees [30, 46, 100, 139] and (iii) mixture of experts [131, 233]. All these models fit a model of the following form:

$$f(x) = \sum_{i \in [m]} w_i(x) f_i(x). \tag{1.1}$$

For additive models, each of $w_i(x) = 1$, and each of the base learner f_i depends nonlinearly on a single covariate. This can be extended to pairwise interactions, where each of the base learner f_i can depend at most on two covariates. For tree ensembles, $w_i(x) = 1/m$ and each base learner f_i is a decision tree. In mixture of experts, $w(x)$ is input-dependent and lies on a simplex, and each of the base learner f_i can be a neural network.

In particular, we study ensemble methods under various types of sparsity and structural constraints. First, in additive models with interactions, we consider component selection and other structural constraints such as hierarchy. We discuss these in Chapters 2, 3, and

4. Second, in tree ensembles, we consider structural constraints e.g., sharing information across tasks in multi-task learning, global feature selection across all trees in the ensemble etc. We discuss these in Chapters 5 and 6. Third, in mixture of experts, we consider per-input sparsity constraints such that only a subset of base learners (or experts) can be activated for each input. We discuss these in Chapters 7 and 8. In all three cases above, we consider ensemble learning with the perspective of efficient training and inference. Additionally many of these constraints can also aid interpretability, engendering trust and may provide tools to identify hidden issues and biases in data. We provide a summary of each chapter next.

Chapter 2: Generalized Additive Models with Interactions under Sparsity and Structural Constraints. Generalized Additive Models (GAMs) are a family of flexible and interpretable models with old roots in statistics. GAMs are often used with pairwise interactions to improve model accuracy while still retaining flexibility and interpretability but lead to computational challenges as we are dealing with order of p^2 terms. It is desirable to restrict the number of components (i.e., encourage sparsity) for easier interpretability, and better computational and statistical properties. Earlier approaches, considering sparse pairwise interactions, have limited scalability, especially when imposing additional structural interpretability constraints. We propose a flexible `GRAND-SLAMIN` framework that can learn GAMs with interactions under sparsity and additional structural constraints in a differentiable end-to-end fashion. We customize first-order gradient-based optimization to perform sparse backpropagation to exploit sparsity in additive effects for any differentiable loss function in a GPU-compatible manner. Numerical experiments on real-world datasets show that our toolkit performs favorably in terms of performance, variable selection and scalability when compared with popular toolkits to fit GAMs with interactions. Our work expands the landscape of interpretable modeling while maintaining prediction accuracy competitive with non-interpretable black-box models.

Chapter 3: Additive Models and Structured Interactions: Alternative Optimization Approaches. We consider nonparametric additive models with small number of main and pairwise interaction effects using ℓ_0-based penalization. From a methodological viewpoint, we propose efficient algorithms to approximately solve the optimization problem. We also discuss variants that incorporate strong hierarchical interactions. Our algorithms extend the computational frontiers of existing algorithms for sparse additive models, to be able to handle datasets relevant for the application we consider in the next chapter.

Chapter 4: Additive Models and Structured Interactions: A Large-Scale Case Study for Census Survey Response Prediction. In this chapter, we consider a large-scale case study for Census Survey Response Prediction. In particular, we consider the problem of predicting survey response rates using a family of flexible and interpretable nonparametric models we develop in previous chapter. The study is motivated by the US Census Bureau's well-known ROAM application which uses a linear regression model trained on the US Census Planning Database data to identify hard-to-survey areas. A crowdsourcing competition [68] organized around ten years ago revealed that machine learning methods based on ensembles of regression trees led to the best performance in predicting survey response rates; however, the corresponding models could not be adopted for the intended application due to their black-box nature. We consider nonparametric additive models with interaction through our methods. We discuss and interpret findings from our model on the US Census Planning Database. In addition to being useful from an interpretability standpoint, our models lead to predictions that appear to be better than popular black-box machine learning methods based on gradient boosting and feedforward neural networks – suggesting that it is possible to have models that have the best of both worlds: good model accuracy and interpretability.

Chapter 5: Tree Ensembles: Flexible Modeling and Efficient Training. In this chapter, we consider flexible and scalable tree ensemble learning. Decision tree ensembles are widely used and competitive learning models. Despite their success, popular toolkits for learning tree ensembles have limited modeling capabilities. For instance, these toolkits support a limited number of loss functions and are restricted to single task learning. We propose a flexible framework for learning tree ensembles, which goes beyond existing toolkits to support arbitrary loss functions, missing responses, and multi-task learning. Our framework builds on differentiable (a.k.a. soft) tree ensembles, which can be trained using first-order methods. However, unlike classical trees, differentiable trees are difficult to scale. We therefore propose a novel tensor-based formulation of differentiable trees that allows for efficient vectorization on GPUs. We introduce `FASTEL`: a new toolkit (based on Tensorflow 2) for learning differentiable tree ensembles. We perform experiments on many real open-source and proprietary datasets, which demonstrate that our framework can lead to 100x more compact and 23% more expressive tree ensembles than those obtained by popular toolkits.

Chapter 6: Tree Ensembles: End-to-end Feature Selection Approach for Learning Skinny Trees. In this chapter, we propose a new optimization-based approach for feature selection in tree ensembles, an important problem in statistics and machine learning. Popular tree ensemble toolkits e.g., Gradient Boosted Trees and Random Forests support feature selection post-training based on feature importance scores, while very popular, they are known to have drawbacks. We propose `Skinny Trees`: an end-to-end toolkit for feature selection in tree ensembles where we train a tree ensemble while controlling the number of selected features. Our optimization-based approach learns an ensemble of differentiable trees, and simultaneously performs feature selection using a grouped ℓ_0-regularizer. We use first-order methods for optimization and present convergence guarantees for our approach. We use a dense-to-sparse regularization scheduling scheme that can lead to more expressive and sparser tree ensembles. On 15 synthetic and real-world datasets, `Skinny Trees` can achieve $1.5\times - 620\times$ feature compression rates, leading up to $10\times$ faster inference over dense trees, without any loss in performance. `Skinny Trees` lead to superior feature selection than many existing toolkits e.g., in terms of AUC performance for a target feature budget.

Chapter 7: Sparse Mixture of Experts: A Cardinality Constrained Routing Approach with Trees and Local Search. The sparse Mixture-of-Experts (Sparse-MoE) framework efficiently scales up model capacity in various domains, such as natural language processing and vision. Sparse-MoEs select a subset of the "experts" (thus, only a portion of the overall network) for each input sample using a sparse, trainable gate. Existing sparse gates are prone to convergence and performance issues when training with first-order optimization methods. In this paper, we introduce two improvements to current MoE approaches. First, we propose a new sparse gate: *COMET*, which relies on a novel tree-based mechanism. *COMET* is differentiable, can exploit sparsity to speed up computation, and outperforms state-of-the-art gates. Second, due to the challenging combinatorial nature of sparse expert selection, first-order methods are typically prone to low-quality solutions. To deal with this challenge, we propose a novel, permutation-based local search method that can complement first-order methods in training *any* sparse gate, e.g., Hash routing, Top-k, DSelect-k, and *COMET*. We show that local search can help networks escape bad initializations or solutions. We performed large-scale experiments on various domains, including recommender systems, vision, and natural language processing. On standard vision and recommender systems benchmarks, *COMET+* (*COMET* with local search) achieves up to 13% improvement in ROC AUC over popular gates, e.g., Hash routing and Top-k, and up to 9% over prior differentiable gates e.g., DSelect-k. When Top-k and Hash gates are combined with local search, we see up to 100$\times$ reduction in the budget needed for hyperparameter tuning. Moreover, for language modeling, our approach improves over the state-of-the-art MoEBERT model for distilling BERT on 5/7 GLUE benchmarks as well as SQuAD dataset.

Chapter 8: Sparse Mixture of Experts: An Effective Sampling-based Routing Approach. The Sparse Mixture-of-Experts (Sparse-MoE) framework, an efficient paradigm for increasing model capacity, employs a network of experts and a gate. The gate selectively activates a few experts for each data example. Top-k style gates, common in this framework, activate a fixed number of experts per example but face performance challenges due to the non-continuous nature of gate. Although differentiable gates, as the one proposed in previous chapter, overcome these issues, they do not maintain k-sparsity during training, limiting their utility for efficient training. We introduce *MOESART*, a new k-sparse gating method effective in both training and inference stages. *MOESART* innovatively learns an effective k-sparse approximation of the traditional softmax gate, using sampling and a carefully designed expert reweighting technique. We compare *MOESART* with state-of-the-art MoE gates on 16 datasets from various domains, including recommender systems, vision, and natural language processing. *MOESART* achieves up to 16% (relative) reduction in out-of-sample loss on standard image datasets, and up to 15% (relative) improvement in AUC on standard recommender systems, over many k-sparse gates. For a 1 billion MoE-Transformer, *MOESART* also improves performance over Top-k gate by 2.1 BLEU points in German $\rightarrow$ English translation task.

Chapter 9: Pruning Large Vision and Language Models. In a departure from the previous chapters on ensemble learning, I also advised two junior PhD students (Xiang Meng, Gabriel I. Afriat) on post-training pruning of large vision and language models for improving inference efficiency of these models. These settings bring up interesting structural constraints. We propose optimization based approaches to address these problems.

Funding. The work presented in this theses was funded in part by grants from the Office of Naval Research: ONR-N000141812298, N000142112841, the National Science Foundation: NSF-IIS-1718258, Liberty Mutual Insurance, MIT IBM Watson AI Lab and Google Research.

Computing resources. The authors acknowledge MIT SuperCloud [222], Google Cloud and Liberty Mutual Insurance for providing HPC resources that have contributed to the research results reported within this thesis.

Chapter 2

Generalized Additive Models with Interactions under Sparsity and Structural Constraints

2.1 Introduction

Many state-of-the-art learners e.g., tree ensembles, neural networks, kernel support vector machines, can be difficult to interpret. There have been various efforts to derive some post-training explainability from these models —see [34] for a survey. Post-hoc explainability attempts to explain black-box prediction with interpretable instance-specific approximations e.g, LIME [223] and SHAP [174]. However, such approximations are known to be unstable [81, 147], expensive [235] and inaccurate [167]. Hence, it is desirable to consider modeling approaches that are inherently interpretable.

Amongst classical approaches, there are some models that have inherent interpretability e.g., Linear models, CART [29] and Generalized Additive Models (GAMs)[93]. GAMs [94], which have old roots in statistics, are considered a front-runner in the context of interpretable modeling. They consider an additive model of the main effects of the form: $g(\mathbb{E}[y]) =$

$\sum_{j \in [p]} f_j(x_j)$, where x_j denotes the jth feature in input $\boldsymbol{x} \in \mathbb{R}^p$, f_j is a univariate shape function and g denotes the link function that adapts the model to various settings such as regression or classification. GAMs are considered easy to interpret as the impact of each feature can be understood via visualizing the corresponding shape function e.g., plotting $f_j(x_j)$ vs x_j. However, such models often suffer in prediction performance when compared to black-box methods e.g., deep neural networks (DNNs). This can be attributed in part to the fact that GAMs do not consider interactions among covariates.

There has been some exciting work that aims to reduce this performance gap by considering GAMs with pairwise interactions [see, for example, 41, 63, 66, 172, 216, 276, and the references therein]. GAMs with pairwise interactions consider a model of the following form:

$$g(\mathbb{E}[y]) = \sum_{j \in [p]} f_j(x_j) + \sum_{(j,k) \in \mathcal{I}} f_{j,k}(x_j, x_k) \tag{2.1}$$

where $f_{j,k}$ is a bivariate shape function and $\mathcal{I} \subseteq \{(1,2), (1,3), ..., (p-1,p)\}$ denotes the set of all pairwise interactions. Under this model, $f_j(x_j)$ is the j-main effect and $f_{j,k}(x_j, x_k)$ is the (j,k)-th interaction effect. Pairwise interactions are considered interpretable as each of the bivariate shape function $f_{j,k}$ can be visualized as a heatmap on an x_j, x_k-plane. Despite their appeal, GAMs with pairwise interactions pose several challenges: (i) Learning all pairwise interaction effects of the order of p^2 lead to computational and statistical challenges. (ii) Performing component selection such that only a few of the components $\{f_j\}$ and $\{f_{j,k}\}$ are nonzero in an end-to-end fashion (while training) is a hard combinatorial optimization problem. We remind the reader that component selection is needed to aid interpretability. (iii) Imposing structural constraints on the interaction effects, e.g., hierarchy [26, 49, 66, 85, 195, 207, 276] makes the associated optimization task more complex.

In this chapter and the next, we introduce two different scalable optimization based approaches that allow component selection in additive models with interactions in an end-to-end fashion. In this chapter, we introduce a novel `GRAND-SLAMIN` framework that allows for a flexible way to do component selection in GAMs with interactions under additional structural constraints in an end-to-end fashion. We consider neural/tree shape functions. In the next chapter i.e., chapter 3, we introduce an alternative `ELAAN` framework that presents efficient optimization based formulations and algorithms for regression setting. These formulations are based on spline shape functions motivated from extensive prior statistical literature [see, e.g, 111, 185, 220, 283, 287, and references therein]. Chapter 3 also considers a large-scale practical application of Census Survey Response Rate Prediction, which requires such interpretable models to gather insights.

In this chapter, we introduce a novel formulation of GAMs with interactions with additional binary variables. Next, we *smooth* these binary variables so that we can effectively learn these components via first-order methods in smooth optimization (e.g, SGD). Our formulation appears to have an edge over existing methods in terms of (i) model flexibility for enhanced structural interpretability and (ii) computational efficiency. First, the binary variables allow us to impose in the model (a) component selection constraints and (b) additional structural constraints (e.g, hierarchy) via a unified optimization formulation. Both of constraints (a), (b) can aid interpretability, model compression, and result in faster inference and better statistical properties. Second, our smoothing procedure for the binary variables allows us to have customized algorithms that exploit sparsity in the forward and backward pass of the backpropagation algorithm.

For structural interpretability, we study two notions: weak and strong hierarchy [26, 49, 85, 195, 207].

$$\text{Weak Hierarchy}: \quad f_{j,k} \neq 0 \implies f_j \neq 0 \quad \text{or} \quad f_k \neq 0 \qquad \forall (j,k) \in \mathcal{I},\ j \in [p],\ k \in [p]. \tag{2.2}$$

$$\text{Strong Hierarchy}: \quad f_{j,k} \neq 0 \implies f_j \neq 0 \quad \text{and} \quad f_k \neq 0 \qquad \forall (j,k) \in \mathcal{I},\ j \in [p],\ k \in [p]. \tag{2.3}$$

Weak hierarchy allows an interaction effect $f_{j,k}$ to be selected if either main effect f_j or f_k is selected. Strong hierarchy allows for an interaction effect $f_{j,k}$ to be selected only if both main effects f_j and f_k are selected. Such hierarchy constraints are popular in high-dimensional statistics: (i) They lead to more interpretable models [26, 66, 183], (ii) They promote practical sparsity, i.e., reduce the number of features that need to be measured when making new predictions (see Sec. 2.5.2) — this can reduce future data collection costs [26, 279], (iii) Additional constraints can also help regularize a model, sometimes resulting in improved AUC (see Sec. 2.5.1). (iv) They can also reduce variance in estimation of main effects in the presence of interaction effects (see Sec. 2.5.4), allowing the user to have more "trust" on model interpretability explanations.

Contributions. To summarise, while it's well acknowledged that GAMs with sparse interactions are a useful flexible family of explainable models, learning them pose significant computational challenges due to the combinatorial nature of the associated optimization problem. Our technical contributions in this chapter be summarized as:

1. We propose a novel optimization formulation that makes use of indicator (binary) variables. The indicator variables allow us to impose both (a) component selection and (b) structural constraints in an end-to-end fashion. We consider a smooth and continuous parameterization of the binary variables so that the optimization objective is differentiable (for a smooth training loss) and hence amenable to first order methods such as SGD.
2. We show the flexibility of our framework by considering two different notions of hierarchy. While these constraints improve interpretability, they make the combinatorial problem more challenging [26, 98]. We propose end-to-end algorithms to train these models, making our approach quite different from existing neural-based toolkits [66, 276].
3. We exploit sparsity in the indicator variables during the course of the training for sparse forward and backward passes in a customized backpropagation algorithm in a GPU-compatible manner. This provides speedups on training times up to a factor of 10× over standard backpropagation.

4. We introduce a new open-source toolkit `GRAND-SLAMIN` and perform experiments on a collection of 16 real-world datasets to demonstrate the effectiveness of our toolkit in terms of prediction performance, variable selection and scalability. Our code is available at https://github.com/mazumder-lab/grandslamin.

2.2 Related Work

GAMs. GAMs have a long history in statistics [93] and have been extensively studied. They're often studied with smooth spline shape functions [see, e.g, 111, 185, 220, 283, 287, and references therein]. Some works have studied tree-based shape functions [171] and neural basis functions [4, 273].

GAMs with all pairwise interactions. In this thread, [63] study low-rank decomposition with neural network shape functions; [216] fit all pairwise interactions using shared neural bases.

Sparse GAMs with interactions. [165] introduced COSSO, which penalizes the sum of the Sobolev norms of the functional components, producing sparse models. In [114], we propose `ELAAN-I` (see next chapter), which is an ℓ_0-regularized formulation with smooth cubic splines. We demonstrate the usefulness of the approach in a regression setting in terms of compact component selection and efficiency on a large-scale Census survey response prediction. [41, 172, 199] explore tree-based shape functions for fitting additive models with

Table 2.1: *Relevant work on sparse GAMs with Interactions. Models in rows 1-2 have some variable selection but no hierarchy; models in row 3-5 have screening-based approaches for hierarchy.*

Paper	Selection Method		Structural Constraints		Reg.	Classif.		Shape Functions	Statistical Properties	Scalable
	Main	Interactions	Weak Hier.	Strong Hier.		Bin.	Multi			
EBM [199]	None	Greedy	✗	✗	✓	✓	✓	trees	✗	✓
NODE-GA^{2}M [41]	Entmax+Anneal		✗	✗	✓	✓	✓	trees	✗	✓
GAMI-Net [276]	Prune	Screening	Screening	✗	✓	✓	✗	neural	✗	✗
SIAN [66]	None	Screening	✗	Screening	✓	✓	✗	neural	✓	✗
`ELAAN`(Chapter 3)	Group L0		✗	Screening+Convex Relax.	✓	✗	✗	splines	✓	✓
`GRAND-SLAMIN`(Chapter 2)	Binary Variables		End-to-end	End-to-end	✓	✓	✓	trees/neural	✓	✓

Hier.=Hierarchy, Reg.=Regression, Classif.=Classification, Bin.=Binary, Relax.=Relaxation sparse interactions. [172] consider *all* main effects and a subset of pairwise interactions; the subset of interactions are selected via greedy stage-wise interaction detection heuristics. [199] provide an efficient implementation of the above approach as Explainable Boosting Machines (EBMs). [41] propose NODE-GA^{2}M: an end-to-end learning approach with differentiable neural oblivious decision (NODE) trees [212]. Component selection in NODE-GAM is achieved by constraining the number of trees, and each tree learns to use one or two features via entmax transformation [210].

Structural Constraints. Structural interpretability constraints such as hierarchical interactions, have been studied for both linear settings [26, 98, 164, 275] and nonparametric settings [66, 114, 215, 276]. We briefly review prior work on nonparametric hierarchical interactions as it relates to this paper. [276] proposed GAMI-Net, which is a multi-stage neural-based approach that fits *all* main effects and a subset of Top-k interaction effects, selected via a fast interaction screening method [172]. Amongst this screened set, interactions that satisfy the weak hierarchy are later used to fit interaction effects. They also prune some main and interaction effects after training based on a variation-based ranking measure. [66] proposed SIAN, which uses Archipelago [247] to measure the strength of each main and interaction effect from a *trained* DNN, and then screens (i.e., selects a subset of candidate main and interaction effects) using Archipelago scores to identify main effects and interaction effects that obey strong hierarchy. Then, it fits a GAM model with the screened main and interaction effects. [276] and [66] only support screening of interactions obeying hierarchy *before* the training for interaction effects is done. None of these approaches impose hierarchy *while* training with interactions. [114] with their `ELAAN-H` framework (see next chapter) also consider strong hierarchy in the presence of ℓ_0-regularized formulation with splines for the least squares loss (regression). `ELAAN-H` has a two-stage approach: It selects a candidate set of interactions and then applies commercial mixed integer programming solvers to learn sparse interactions under a hierarchy constraint. This approach would require customized algorithms to adapt to different loss objectives e.g., multiclass classification. To our knowledge, current techniques for sparse hierarchical (nonparametric) interactions are not based on end-to-end differentiable training: they can be limited in flexibility and scalability—a gap we intend to fill in this work.

Note [63] and [66] also consider higher-order interactions (beyond two-way ones), which can be hard to interpret. For convenience, Table 2.1 summarizes some relevant work on Sparse GAMs with interactions and possible structural constraints.

2.3 Problem Formulation

We first present in Sec. 2.3.1 an alternative formulation of GAMs with interactions using binary variables for imposing sparsity and structural constraints. Next, in Sec. 2.3.2, we present a smooth reformulation of the objective that can be solved with first-order gradient-based methods.

2.3.1 An optimization formulation with binary variables

We first present an alternative formulation of GAMs with interactions under sparsity with-/without additional structured hierarchy constraints. Let us consider the parameterization:

$$f = \sum_{j \in [p]} f_j(x_j) z_j + \sum_{(j,k) \in \mathcal{I}} f_{j,k}(x_j, x_k) q(z_j, z_k, z_{j,k}), \tag{2.4}$$

with main effects $f_j(\cdot)$, interaction effects $f_{j,k}(\cdot)$ and binary gates z_j and $q(z_j, z_k, z_{j,k})$. We consider three different parameterizations for $q(\cdot)$, satisfying the following different constraints:

$$\text{No structural constraint:} \quad q(z_j, z_k, z_{j,k}) \overset{\text{def}}{=} z_{j,k}, \tag{2.5}$$

$$\text{Weak hierarchy:} \quad q(z_j, z_k, z_{j,k}) \overset{\text{def}}{=} (z_j + z_j - z_j z_k) z_{j,k}, \tag{2.6}$$

$$\text{Strong hierarchy:} \quad q(z_j, z_k, z_{j,k}) \overset{\text{def}}{=} z_j z_k z_{j,k}. \tag{2.7}$$

The binary gates z_j and $q(z_j, z_k, z_{j,k})$ play the role of selection. In particular, when $z_j = 0$, the corresponding j-th main effect $f_j(\cdot)$ is excluded from our additive model (2.4). Similarly, when $q(z_j, z_k, z_{j,k}) = 0$, the corresponding (j, k)-th interaction effect $f_{j,k}(\cdot)$ is excluded. Then, we can formulate the regularized objective as:

$$\min_{\substack{\{f_j\},\{f_{j,k}\},\\ \{z_j\}\in\{0,1\}^p,\{z_{j,k}\}\in\{0,1\}^{|\mathcal{I}|}}} \hat{\mathbb{E}}[\ell(y, f)] + \lambda\Big(\sum_{j\in[p]} z_j + \alpha \sum_{(j,k)\in\mathcal{I}} z_{j,k}\Big), \tag{2.8}$$

where the first term denotes empirical loss over the training data, the penalty term controls model sparsity: $\lambda \geq 0$ is the selection penalty, $\alpha \in [1, \infty)$ controls the relative selection strength of main and interaction effects. We refer to the framework in (2.8) under the different constraints (2.5)-(2.7) as `GRAND-SLAMIN`[1]. We discuss extension of this framework to third-order interactions in Appendix Sec. 2.7.2. However, we do not consider third-order interactions in our experiments as third-order interactions are hard to interpret.

The formulation in (2.8) with binary variables z_j, $z_{j,k}$ and functions f_j, $f_{j,k}$ with any of the constraint sets (2.5)-(2.7) is a challenging discrete optimization problem (with binary variables) and is not amenable to differentiable training via SGD (for example). Sec. 2.3.2 explores approximate solutions to (2.8) using a smooth reformulation of the binary variables. Intuitively, we rely on continuous relaxations of the binary variables z's and parameterize f's with smooth tree-based shape functions. The reformulation allows us to use first-order methods.

2.3.2 A Smooth Reformulation

We discuss a smooth reformulation of the objective in (2.8). We describe an approach to parameterize the continuous relaxation of the binary variables z_j, $z_{j,k}$ with a Smooth-Step function [100] and use smooth tree-based shape functions to model $\{f_i\}$, $\{f_{j,k}\}$.

[1]`GRAND-SLAMIN` stands for GeNeRAlizeD Sparse Learning of Additive Models with INteractions.

2.3.2.1 Relaxing Binary Variables with Smooth Gates

We present an approach to smooth the binary gates z's in (2.8) using a smooth-step function [100], which we define next.

Smooth-Step Function. Smooth-step function is a continuously differentiable function, similar in shape to the logistic function. However, unlike the logistic function, the smooth-step function can output 0 and 1 exactly for sufficiently large magnitudes of the input. This function, originally proposed by [100], has been used for smoothing binary representations for conditional computation [100, 102, 117]. Let γ be a non-negative scalar parameter. The smooth-step function takes the form:

$$S(t) = \begin{cases} 0 & \text{if } t \leq -\gamma/2 \\ -\frac{2}{\gamma^3}t^3 + \frac{3}{2\gamma}t + \frac{1}{2} & \text{if } \gamma/2 \leq t \leq \gamma/2 \\ 1 & \text{if } t \geq \gamma/2 \end{cases} \tag{2.9}$$

The smooth-step function is continuously differentiable, similar to the logistic function. Additionally, it performs hard selection, i.e., outside $[-\gamma/2, \gamma/2]$, the function produces exact zeros and ones.

We parameterize each of the z's in (2.8) as $S(\mu)$, where $\mu \in \mathbb{R}$ is a learnable parameter and $S(\cdot)$ denotes the Smooth-step function. We parameterize the additive function as: $f = \sum_{j\in[p]} f_j(x_j)S(\mu_j) + \sum_{(j,k)\in\mathcal{I}} f_{j,k}(x_j, x_k)q(S(\mu_j), S(\mu_k), S(\mu_{j,k})))$ and optimize the following objective:

$$\min_{\substack{\{f_j\},\{f_{j,k}\},\\ \{\mu_j\}\in\mathbb{R}^p,\{\mu_{j,k}\}\in\mathbb{R}^{|\mathcal{I}|}}} \hat{\mathbb{E}}[\ell(y, f)] + \lambda(\sum_{j\in[p]} S(\mu_j) + \alpha \sum_{(j,k)\in\mathcal{I}} S(\mu_{j,k})). \tag{2.10}$$

Note that $S(\mu_j)$ and $S(\mu_{j,k})$ are continuously differentiable, so the formulation in (2.10) is amenable to first-order gradient-based methods (e.g, SGD).

Achieving binary gates. To encourage each of the $S(\mu_j)$'s and $S(\mu_{j,k})$'s to achieve binary state (and, not fractional) by the end of training, we add an entropy regularizer $\tau(\sum_{j\in[p]}\Omega(S(\mu_j))+\sum_{(j,k)\in\mathcal{I}}\Omega(S(\mu_{j,k})))$ where $\Omega(S(\mu)) = -(S(\mu)\log S(\mu)+(1-S(\mu))\log(1-S(\mu)))$ and $\tau \geq 0$ controls how quickly each of the gates $S(\mu)$ converges to a binary z.

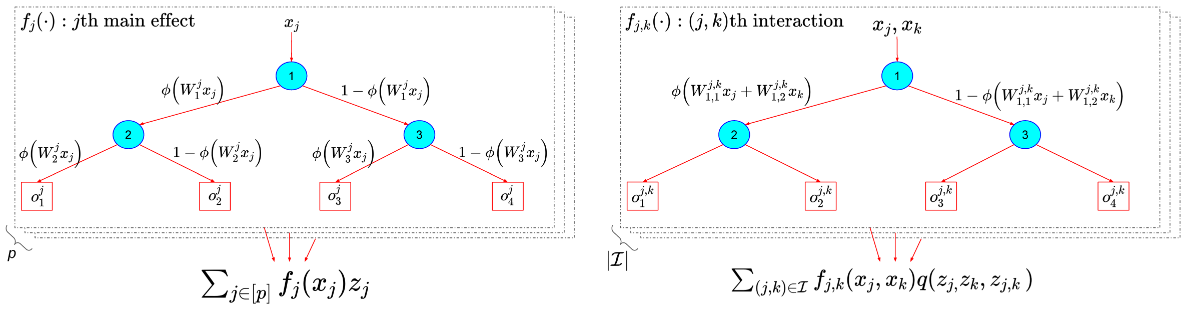

$f_j(\cdot)$: jth main effect
x_j
$\phi(W_1^j x_j)$
$1-\phi(W_1^j x_j)$
$\phi(W_2^j x_j)$
$1-\phi(W_2^j x_j)$
$\phi(W_3^j x_j)$
$1-\phi(W_3^j x_j)$
o_1^j
o_2^j
o_3^j
o_4^j
p
$\sum_{j\in[p]} f_j(x_j)z_j$
$f_{j,k}(\cdot)$: (j,k)th interaction
x_j, x_k
$\phi(W_{1,1}^{j,k}x_j + W_{1,2}^{j,k}x_k)$
$1-\phi(W_{1,1}^{j,k}x_j + W_{1,2}^{j,k}x_k)$
$o_1^{j,k}$
$o_2^{j,k}$
$o_3^{j,k}$
$o_4^{j,k}$
$|\mathcal{I}|$
$\sum_{(j,k)\in\mathcal{I}} f_{j,k}(x_j,x_k)q(z_j,z_k,z_{j,k})$

Figure 2.1: *Modeling main and interaction effects with soft trees. ϕ denotes the sigmoid activation function. We omit biases in split nodes for brevity. For interaction effect, $W_{i,1}^{j,k} \in \mathbb{R}$ and $W_{i,2}^{j,k} \in \mathbb{R}$ denote the weights in i-th node of the (j,k)-th tree.*

2.3.2.2 Soft trees

We use soft trees [79, 120, 131, 244]—based on hyperplane splits (univariate or bivariate) and constant leaf nodes — as shape functions to parameterize the main effects $f_j(\cdot)$ and the pairwise interaction effects $f_{j,k}(\cdot)$. See Figure 2.1 for an illustration. Soft trees were introduced as hierarchical mixture of experts by [131] and further developed by [79, 120, 244]. They allow for end-to-end learning [100, 115, 139]. They also have efficient implementations when learning tree ensembles [115]. A detailed definition of soft tress is given in Appendix 2.7.1. We also consider neural shape functions in Appendix 2.7.4.4.

2.3.2.3 Statistical Theory

We study novel statistical properties of our model, and present non-asymptotic prediction error bounds for our estimator without any hierarchy constraints in [116]. Different from earlier work, our results apply to learning with tree-based shape functions (for both main and interaction effects).

2.4 Efficient Implementation

We discuss a fast implementation of our approach. The key elements are: (i) Tensor parameterization of trees, (ii) Sparse backpropagation, and (iii) Complementary screening heuristics.

Tensor Parameterization of Additive Effects. Typically, in neural-based additive model implementations [4, 276], a separate network module is constructed for each shape function. The outputs from each model are sequentially computed and combined additively. This can create bottleneck in scaling these models. Some recent approaches e.g., [66] try to work around this approach by constructing a large block-wise network to compute representations of all shape functions simultaneously. However, this comes at a cost of large memory footprint. For tree-based shape functions, drawing inspiration from [115], we can implement a tensor-based formulation of all shape functions, leveraging the fact that each tree has the same depth. This parameterization can exploit GPU-friendly parallelization in computing all shape functions simultaneously without increasing memory footprints.

Sparse Backpropagation. We use first-order optimization methods (e.g., SGD and its variants) to optimize `GRAND-SLAMIN`. Typically, a main computational bottleneck in optimizing GAMs with all pairwise interactions via standard gradient-based backpropagation methods is the computation of forward pass and gradient computations with respect to all additive components (both main and interaction effects). This can hinder training large GAMs with interactions. We exploit the sparsity in `GRAND-SLAMIN` via the sparsity in the smooth-step function and its gradient during training.

Recall that $S(\mu_j)$'s and $S(\mu_{jk})$'s play a role of selection in a smoothed fashion. In the early stages of training, $S(\mu_j)$'s and $S(\mu_{jk})$'s are all in the range $(0, 1)$. As the optimization proceeds, due to the entropic regularization and selection regularization, $S(\mu_j)$'s and $S(\mu_{jk})$'s progressively achieve binary state $\{0, 1\}$ — the gradient with respect to μ_j and μ_{jk} also reaches 0 because of the nature of smooth-step function $S(\cdot)$. All the additive components corresponding to the selection variables that reached 0 can be removed from both the forward and the backward computational graph. This sparse backpropagation approach can provide large speedups on training times up to a factor of $10\times$ over standard backpropagation. Additionally, there is progressively a reduced memory footprint during the course of training in comparison to standard backpropagation.

The approach outlined above (use of Smooth-Step function for selection) when specialized to additive models allows us to implement GPU-friendly sparse backpropagation — this makes our work different from [100], which does not support GPU training.

Screening. We describe how screening approaches prior to training, can complement our sparse backpropagation approach when the number of all pairwise interactions is large e.g., of the order $100,000$. Fast screening methods based on shallow-tree like models are proposed in [172] for identifying prominent pairwise interactions. These are used by various toolkits e.g., EBM [199], GAMI-Net [276]. These screening approaches are complementary to our selection approach with indicator variables. We used CART [29] for each pairwise interaction and sorted the interaction effects based on AUC performance to select an initial screened set of interaction effects. In particular, we can consider $\mathcal{I}$ to be a screened subset (e.g., $10,000$) of all pairwise interaction effects of the order $100,000$. Then, we run our end-to-end learning framework under the component selection constraints on the main and screened interaction effects. We observe that such screening can be beneficial for multiple reasons: (i) The training time can be reduced further by 3× – 5×. (ii) The memory footprint of the model reduces by $10\times$. (iii) There is no loss in accuracy—the accuracy can sometimes improve with screening — see ablation study in Appendix Sec. 2.7.4.6. Note that even with screening, our approach is different from GAMI-Net as they directly screen to a much smaller set of interactions e.g., 500 and there is no component selection when training with these interactions.

2.5 Experiments

We study the performance of `GRAND-SLAMIN` on 16 real-world datasets and compare against relevant baselines for different cases. We make the following comparisons:

1. Performance comparison of `GRAND-SLAMIN` without/with structural constraints against existing toolkits for sparse GAMs with interactions.
   (a) Toolkits that support sparse interactions: EB^{2}M [199] and NODE-GA^{2}M [41]
   (b) Toolkits that support hierarchy constraints: GAMI-Net [276] and SIAN [66]
2. Variable selection comparison against the competing toolkits.
3. Computational scalability of `GRAND-SLAMIN` toolkit with sparse backpropagation.
4. Variance reduction with structural constraints

Additional results are included in Appendix Sec. 2.7.4 that study (i) comparison with full complexity models in 2.7.4.1, (ii) comparison with GAMs with all pairwise interactions in 2.7.4.2, (iii) comparison with Group Lasso selection approach in 2.7.4.3, (iv) choice of shape functions in 2.7.4.4, (v) effect of entropy on performance and component selection in 2.7.4.5, and (vi) effect of screening on training times, memory and performance in 2.7.4.6.

Datasets. We use a collection of 16 open-source classification datasets (8 binary, 6 multiclass and 2 regression) from various domains. We consider datasets with a wide range of number of all pairwise interactions $10 - 200000$. A summary of the datasets is in Table 2.7.1 in the Appendix.

Tuning Details. For all the experiments, we tune the hyperparameters using Optuna [5] with random search on a held-out validation set. We compute statistical averages across multiple runs for the optimal hyperparameters for all models. In particular, we report median test ROC AUC across 10 runs along with the mean absolute deviation (MAD). Additional details are in the Appendix 2.7.3.

2.5.1 Prediction Performance

Comparison with EB^{2}M and NODE-GA^{2}M. We first study the performance of our model in comparison to two tree-based state-of-the-art toolkits which support sparse GAMs with interactions without any structural constraints e.g., EB^{2}M and NODE-GA^{2}M. We report the ROC AUC performance in Table 2.2. Our model outperforms EB^{2}M in 10 out of 14 datasets. Our model is also competitive with NODE-GA^{2}M as it can outperform in 50% of the datasets. In summary, our results in Table 2.2 show that we are at par with state-of-the-art methods for unstructured component selection. Our key advantage is to do hierarchical interactions, which NODE-GA^{2}M and EB^{2}M can not support. Additionally, we can achieve faster training times (Sec. 2.5.3) and improve on variable selection (Sec. 2.5.2) than NODE-GA^{2}M and EB^{2}M.

Table 2.2: *Test ROC AUC of* `GRAND-SLAMIN`*, EB^{2}M and NODE-GA^{2}M. We report median along with mean absolute deviation across 10 runs.*

Dataset	EB^{2}M	NODE-GA^{2}M	`GRAND-SLAMIN`
Magic	93.12 ± 0.001	$\mathbf{94.27} \pm 0.13$	93.86 ± 0.30
Adult	91.41 ± 0.0004	$\mathbf{91.75} \pm 0.14$	91.54 ± 0.14
Churn	91.97 ± 0.005	89.62 ± 5.61	$\mathbf{92.40} \pm 0.41$ (SH)
Satimage	97.65 ± 0.0007	98.70 ± 0.07	$\mathbf{98.81} \pm 0.04$
Texture	99.81 ± 0.0004	$\mathbf{100.00} \pm 0.00$	$\mathbf{100.00} \pm 0.00$
MiniBooNE	97.86 ± 0.0001	$\mathbf{98.44} \pm 0.02$	97.77 ± 0.05 (WH)
Covertype	90.08 ± 0.0003	95.39 ± 0.12	$\mathbf{98.11} \pm 0.08$
Spambase	$\mathbf{98.84} \pm 0.01$	98.78 ± 0.06	98.55 ± 0.07 (SH)
News	73.03 ± 0.002	$\mathbf{73.53} \pm 0.06$	73.24 ± 0.04 (SH)
Optdigits	99.79 ± 0.0003	99.93 ± 0.02	$\mathbf{99.98} \pm 0.0$
Bankruptcy	$\mathbf{93.85} \pm 0.01$	92.02 ± 1.03	92.51 ± 0.54 (WH)
Madelon	88.04 ± 0.02	60.07 ± 0.82	$\mathbf{89.25} \pm 1.03$ (WH)
Activity	74.96 ± 8.77	$\mathbf{99.86} \pm 0.04$	99.24 ± 1.45
Multiple	$\mathbf{99.96} \pm 0.0002$	99.94 ± 0.02	99.95 ± 0.02

Structural constraints: Weak and Strong Hierarchy. Next, we study our method with structural constraints i.e., (2.6) for weak hierarchy or (2.7) for strong hierarchy. We compare against two competing neural-based state-of-the-art methods for sparse GAMs with hierarchical interactions: (i) GAMI-Net with support for weak hierarchy, and (ii) SIAN with

Table 2.3: *Test ROC AUC for* `GRAND-SLAMIN` *with structural constraints i.e., (2.6) or (2.7), GAMI-Net and SIAN. We report median across 10 runs along with mean absolute deviation.*

	GAMI-Net	SIAN	`GRAND-SLAMIN`	
Dataset\Model	WH	SH	WH	SH
Magic	91.72 ± 0.05	93.02 ± 0.06	93.16 ± 0.55	$\mathbf{93.37} \pm 0.16$
Adult	91.01 ± 0.04	90.67 ± 0.05	91.34 ± 0.32	$\mathbf{91.46} \pm 0.15$
Churn	90.05 ± 0.77	$\mathbf{92.98} \pm 0.20$	92.28 ± 0.75	92.40 ± 0.41
Spambase	$\mathbf{98.67} \pm 0.04$	98.28 ± 0.04	98.45 ± 0.15	98.55 ± 0.07
MiniBooNE	96.11 ± 0.41	95.90	$\mathbf{97.77} \pm 0.05$	97.62 ± 0.30
News	72.54 ± 0.05	72.28	73.15 ± 0.08	$\mathbf{73.24} \pm 0.04$
Bankruptcy	92.46 ± 0.12	90.71	$\mathbf{92.51} \pm 0.54$	90.45 ± 1.87
Madelon	88.14 ± 0.94	83.18	$\mathbf{89.25} \pm 1.03$	86.23 ± 1.89

WH=Weak Hierarchy, SH=Strong Hierarchy.
For SIAN, for some of the larger datasets (row 5-9),
we use the number for best trial as SIAN takes
$\sim$ 24 hours on V100 Tesla GPU.

support for strong hierarchy. We omit multiclass datasets as both GAMI-Net and SIAN do not support them. We report the ROC AUC performance in Table 2.3. Our models outperform GAMI-Net and SIAN in 7/8 datasets.

Additionally, our models are much more compact in terms of overall number of parameters — our tree-based shape functions have 100× and 10× smaller number of parameters than the neural-based shape functions used by GAMI-Net and SIAN respectively. Moreover, our toolkit is significantly faster than SIAN and GAMI-Net on larger datasets — see Sec. 2.5.3.

Additionally, we compare interpretable modeling toolkits with full complexity models e.g., deep neural network (DNN), in Appendix Sec. 2.7.4.1. We observed interpretable models to outperform full complexity models on these datasets. We also compare our toolkit that fits sparse components with toolkits that fit all pairwise interactions e.g., NA^{2}M, NB^{2}M and SPAM in Appendix Sec. 2.7.4.2. `GRAND-SLAMIN` generally outperform these methods with enhanced interpretability due to sparsity and structural constraints. We also study how our toolkit performs when we replace soft tree shape functions with MLP shape functions in Appendix Sec. 2.7.4.4. Interestingly, we observe that soft trees seem to have an edge when the parameters are matched with MLP.

Table 2.4: *Number of features used by* ***GRAND-SLAMIN*** *without/with additional structural constraints and competing approaches. Hyphen (-) indicates multiclass classification is not supported by GAMI-Net and SIAN.*

	EB^{2}M	NODE	GAMI	SIAN	GRAND-SLAMIN		
Dataset\Model		GA^{2}M	Net		None	WH	SH
Magic	10 ± 0	10 ± 0	10 ± 0	10 ± 0	10 ± 0	9 ± 1	**7** ± 0
Adult	14 ± 0	14 ± 0	14 ± 1	14 ± 0	13 ± 1	**11** ± 1	**11** ± 1
Churn	19 ± 0	19 ± 0	18 ± 2	19 ± 0	19 ± 0	**11** ± 1	12 ± 2
Satimage	36 ± 0	36 ± 0	–	–	36 ± 0	36 ± 0	**22** ± 2
Texture	40 ± 0	40 ± 0	–	–	40 ± 0	37 ± 2	**17** ± 2
MiniBooNE	50 ± 0	50 ± 0	**16** ± 12	34	50 ± 0	50 ± 0	28 ± 3
Covertype	54 ± 0	54 ± 0	–	–	**34** ± 1	54 ± 1	54 ± 0
Spambase	57 ± 0	57 ± 0	**52** ± 2	55 ± 1	57 ± 0	56 ± 3	54 ± 2
News	58 ± 0	58 ± 0	**47** ± 1	52	58 ± 0	58 ± 0	58 ± 0
Optdigits	64 ± 0	64 ± 0	–	–	64 ± 0	64 ± 0	**59** ± 1
Bankruptcy	95 ± 0	95 ± 0	60 ± 15	69	95 ± 0	60 ± 26	**7** ± 16
Madelon	500 ± 0	500 ± 0	61 ± 56	490	26 ± 19	**19** ± 15	24 ± 9
Activity	533 ± 0	346 ± 6	–	–	182 ± 15	440 ± 22	**159** ± 21
Multiple	649 ± 0	649 ± 0	–	–	648 ± 1	**629** ± 9	649 ± 0

WH=Weak Hierarchy, SH=Strong Hierarchy.

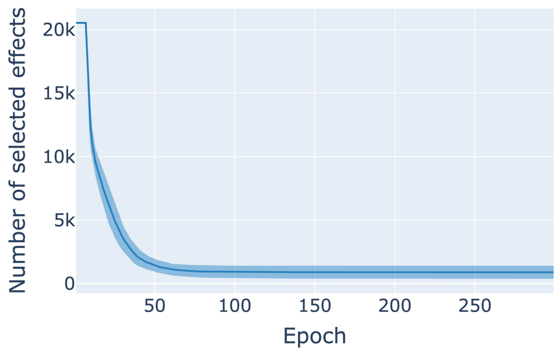
2.5.2 Variable Selection

We evaluate the performance of our models in terms of feature selection. We report the number of features selected in Table 2.4 by each toolkit for sparse GAMs with interactions. We see that GRAND-SLAMIN with structural constraints, in particular strong hierarchy, can significantly reduce the number of features selected by the GAMs with interactions model. For example, on Bankruptcy datasets, GRAND-SLAMIN achieves feature compression up to a factor of 8$\times$ over state-of-the-art GAM toolkits. Having fewer features reinforces the usefulness of additive models as being interpretable.

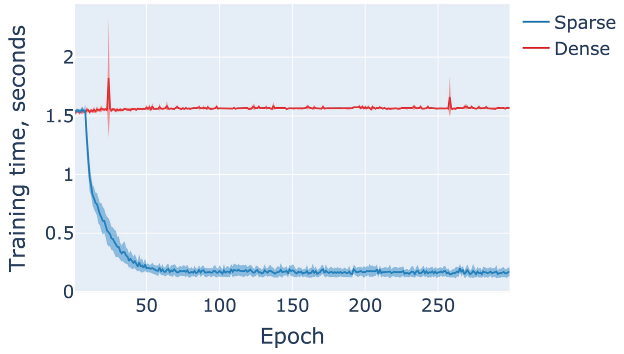
2.5.3 Computational Scalability

Next, we discuss the scalability of GRAND-SLAMIN.

Sparse backpropagation. We highlight the usefulness of our efficient approach with sparse backpropagation in Figure 2.2 on Activity dataset. We show in Figure 2.2[a] that during the



(a) *Number of selected effects at each epoch.*



(b) *Training time (seconds) for each epoch.*

Figure 2.2: *GRAND-SLAMIN with standard (dense) backpropagation vs sparse backpropagation on Activity dataset. (a) shows the number of nonzero effects: $\sum_j z_j + \sum_{(j,k)\in\mathcal{I}} q(z_j, z_k, z_{j,k})$ and (b) shows the time for each epoch during the course of training.*

course of training, the number of selected components (selected via binary variables z's and $q(\cdot)$'s) becomes progressively smaller. This leads to much faster computations at each epoch in Figure 2.2[b] due to sparse forward and backward passes. By exploiting sparsity during training, we can get $10\times$ faster training times than with standard backpropagation.

Comparison with other toolkits. Our toolkit is highly competitive in terms of training times with all existing tree-based and neural-based toolkits for sparse GAMs with interactions. For example, on Madelon dataset, we are $15\times$ faster than NODE-GA^{2}M, $20\times$ faster than EBM, $1300\times$ faster than SIAN and $25\times$ faster than GAMI-Net. See Appendix Sec. 2.7.4.7 for more detailed timing comparisons across multiple datasets. Note that, in addition, we can also handle the case of structured interactions — extending the flexibility of existing end-to-end training methods.

2.5.4 Variance Reduction with Structural Constraints

We provide a visualization study to further highlight an important contribution of our work. In particular, our framework can support models with structural constraints. Hence, we study the effect of these constraints on the stability of learning main effects (in the presence of interactions) when these structural constraints are imposed or not. For this exercise, we

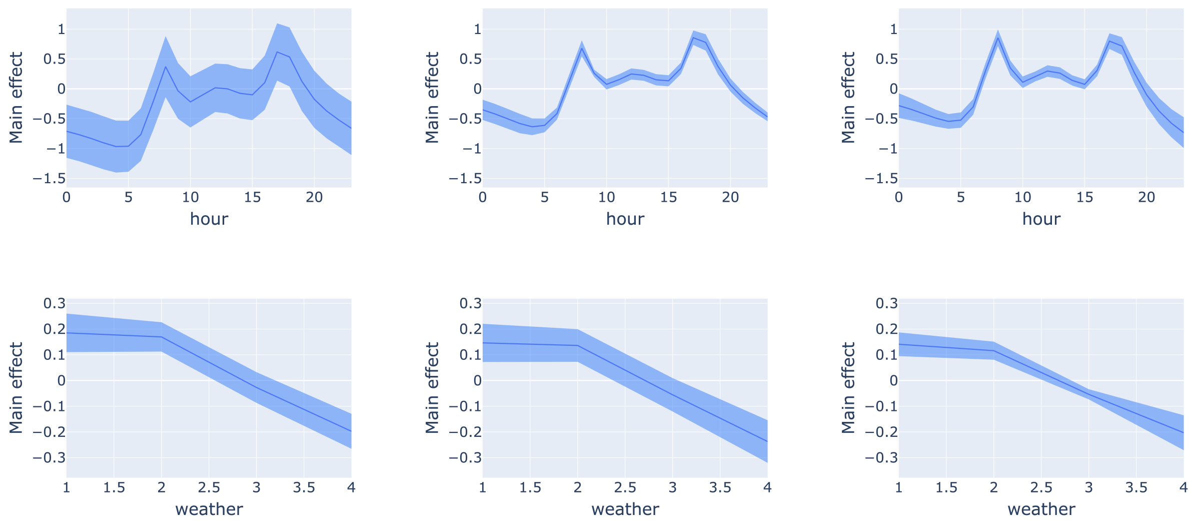


Figure 2.3: *Estimated main effects in the presence of interaction effects on bikesharing dataset [Left] without hierarchy, [Middle] weak hierarchy and [Right] strong hierarchy. Strong hierarchy has the smallest error bars.*

consider bikesharing dataset. We visualize some of the main effects in the presence/absence of hierarchy in Figure 2.3. Note that for visualization, we used the purification strategy [41, 155] post-training that pushes interaction effects into main effects if possible. We can observe in Figure 2.3 that when additional hierarchy constraints are imposed, the error bars are much more compact across different runs. This can potentially increase the trust you can have on the model for deriving interpretability insights. We show additional visualizations on another dataset (American Community Survey from US Census Planning Database 2022 [253]) to show the same behavior in Appendix Section 2.7.5.

2.6 Conclusion

We introduce `GRAND-SLAMIN`: a novel and flexible framework for learning sparse GAMs with interactions with additional structural constraints e.g., hierarchy. This is the first approach to do end-to-end training of nonparameteric additive models with hierarchically structured sparse interactions. Our formulation uses binary variables to encode combinatorial constraints.

For computational reasons, we employ smoothing of the indicator variables for end-to-end optimization with first-order methods (e.g., SGD). We propose sparse backpropagation, which exploits sparsity in the nature of the smoothing function in a GPU-compatible manner and results in 10$\times$ speedups over standard backpropagation. We present non-asymptotic prediction bounds for our estimators with tree-based shape functions. Numerical experiments on a collection of 16 real-world datasets demonstrate the effectiveness of our toolkit in terms of prediction performance, variable selection and scalability.

2.7 Appendix

2.7.1 Definition of Soft Trees

Formally, an interaction soft tree with the set of (internal) nodes $\mathcal{U}$ and the set of leaves $\mathcal{L}$ is a function such as $f(\cdot;\boldsymbol{W},\boldsymbol{o}):\mathbb{R}^2\mapsto\mathbb{R}$ where $\boldsymbol{W}\in\mathbb{R}^{|\mathcal{U}|\times 2}$ and $\boldsymbol{o}\in\mathbb{R}^{|\mathcal{L}|}$. The output is then given as $f((x_j,x_k);\boldsymbol{W},\boldsymbol{o})=\sum_{l\in\mathcal{L}}P(\{(x_j,x_k)\to l\})o_l$ where $P(\{(x_j,x_k)\to l\})$ is the proportion of (x_j,x_k) that is routed to leaf l. Particularly, $P(\{(x_j,x_k)\to l\})=\prod_{i\in\mathcal{A}(l)}r_{i,l}(x_j,x_k)$ where $\mathcal{A}(l)$ denotes the set of ancestors of l and $r_{i,l}(x_j,x_k)$ denotes the proportion of (x_j,x_k) is routed to leaf l from node i. These values are given as $r_{i,l}(x_j,x_k)=\phi(\boldsymbol{W}_i^T(x_j,x_k))$ if l belongs to the left subtree of i, and $r_{i,l}(x_j,x_k)=1-\phi(\boldsymbol{W}_i^T(x_j,x_k))$ if l belongs to the right subtree of i, where $\phi(\cdot)$ is the sigmoid activation function. A soft tree $f(\cdot;\boldsymbol{W},\boldsymbol{o}):\mathbb{R}\mapsto\mathbb{R}$ for a main effect is defined similarly.

2.7.2 Extension to third-order interactions

Our approach can be extended to model third-order interactions. In the case of third-order interactions, the structural constraints for strong and weak hierarchy can be described as follows:

$$\text{Strong Hierarchy}: \quad f_{j,k,l} \neq 0 \implies f_j \neq 0 \quad \text{and} \quad f_k \neq 0 \quad \text{and} \quad f_l \neq 0, \tag{2.7.1}$$

$$\text{Weak Hierarchy}: \quad f_{j,k,l} \neq 0 \implies f_j \neq 0 \quad \text{or} \quad f_k \neq 0 \quad \text{or} \quad f_l \neq 0. \tag{2.7.2}$$

These constraints can be modeled with binary variables as follows:

$$\text{Strong Hierarchy}: \; q(z_j, z_k, z_l, z_{j,k,l}) \overset{\text{def}}{=} z_j z_k z_l z_{j,k,l} \tag{2.7.3}$$

$$\text{Weak Hierarchy}: \; q(z_j, z_k, z_l, z_{j,k,l}) \overset{\text{def}}{=} (z_j + z_k + z_l - z_j z_k - z_j z_l - z_k z_l + z_j z_k z_l) z_{j,k,l} \tag{2.7.4}$$

Hence, our sparse selection and hierarchy constraints can add value in terms of model compactness and feature selection in the settings with third-order interactions as well. However, for third-order interactions, the number of third-order interactions are $O(p^3)$. Hence, for scalability, this would also require using a pre-training screening approach. Despite the fact that our approach is generalizable to third-order interactions, we do not consider such interactions because these are not considered to be easily interpretable.

2.7.3 Datasets, Computing Setup and Tuning

Datasets We use a collection of 16 open-source classification datasets (binary, multiclass and regression) from various domains, e.g., physics, computing, healthcare, life sciences, finance, and social networks. They are from Penn Machine Learning Benchmarks (PMLB) [200] and UCI databases [62]. For datasets with available training, validation and test splits, we used them in their original form. When no test set was available, we treated the original validation set as the test set and split the training set into 80% training and 20% validation. For remaining, we randomly split each of the dataset into 60% training, 20% validation and 20% testing sets. A summary of the 16 datasets considered is in Table 2.7.1.

Table 2.7.1: *Summary of Datasets*

Dataset	Domain	N	C	p	No. of interactions ($\lvert\mathcal{I}\rvert$)
Magic	Physics	19,020	2	10	55
Adult	Socio-economic	48,842	2	14	91
Churn	Business	5,000	2	19	190
Satimage	Physics	6,435	6	36	640
Texture	Image	5,500	11	40	780
MiniBooNE	Physics	130,065	2	50	1,225
Covertype	Life Science	581,012	7	54	1,431
Spambase	Computing	4,601	2	57	1,596
News	Social networks	39,797	2	61	1,830
Optdigits	Image	5,620	10	64	2,016
Bankruptcy	Finance	6,819	2	96	4,560
Madelon	NIPS-2003	2,600	2	500	124,750
Activity	Healthcare	4,480	4	533	141,778
Multiple	Image	2,000	10	649	210,276
Bike Sharing	Transportation	17,389	-	16	120
American Community Survey	Demographic	83,059	-	39	741

Computing Setup. We used a cluster running Ubuntu 7.5.0 and equipped with Intel Xeon Platinum 8260 CPUs and Nvidia Volta V100 GPUs. For all experiments of Sec. 2.5, each job involving `GRAND-SLAMIN`, EBM, Node-GAM, SIAN, GAMI-Net and DNN were run on 8 core, 32GB RAM. Jobs involving larger datasets ($p > 100$) were run on Tesla V100 GPUs.

Tuning. The tuning was done in parallel over the competing models and datasets. We tune the hyperparameter using Optuna [5] which optimizes the overall AUC on a validation set. We report the results on a held-out test set. A list of all the tuning parameters and their distributions is given for `GRAND-SLAMIN` below:

- Learning Rates: Discrete uniform in the set $\{0.05, 0.01, 0.005\}$ for Adam with multi-step decay rate of 0.9 every 25 epochs.
- Batch-size: Discrete uniform in the set $\{64, 256\}$.
- λ for selection: Discrete uniform in the set of 11 values $\{0, 1e-6, \cdots, 1e-3\}$.
- γ for Smooth-step: Discrete uniform in the set $\{0.01, 0.1, 1\}$.
- τ for Entropy Regularization: Discrete uniform in the set $\{0.001, 0.01, 0.1\}$.

- α for relative penalty on interactions: Discrete uniform in the set $\{1, 10\}$.
- Epochs: 1000 with early stopping (patience=50) based on validation loss.
- For Madelon, Activity and Multiple datasets, we used screening to reduce the initial set of interactions to 5000.

For other toolkits, we considered the tuning protocols outlined in the respective papers such that the parameter controlling the variable selection is tuned. For SIAN, we tuned over the threshold parameter θ such that the screened set of interactions is upper bounded by $\{250, 500, \cdots, 1000\}$. We set $\tau = 1$ such that the model satisfies strong hierarchy. For GAMI-Net, we tuned over the number of interactions. For small datasets ($|\mathcal{I}| < 1000$), we tuned over the set: $\{0.2|\mathcal{I}|, 0.4|\mathcal{I}|, \cdot, |\mathcal{I}|\}$. For large datasets ($|\mathcal{I}| < 1000$), we tuned over the set: $\{250, 500, \cdots, 1000\}$. For EBM, we tuned over the set of interactions $\{16, 32, 64, 128\}$ as done by authors in [41]. For NODE-GA^{2}M, we tuned the number of trees as this controls the maximum number of interactions selected by the model.

2.7.4 Additional Results

2.7.4.1 Comparison with full complexity models e.g., DNN

Here, we compare interpretable modeling toolkits based on pairwise interactions with full complexity models. In particular, we consider a deep neural network (DNN). We considered the same architecture for the DNN as used by the authors in [66] for similar comparisons. The architecture is a 4-layered ReLU-activated neural network. We show the test AUC performance of different models in Table 2.7.2. Across all datasets, we see that the full complexity DNN model underperforms interpretable models including `GRAND-SLAMIN`. However, it is noted that the choice of the architecture for DNN maybe contributing to the degradation in performance. There maybe other architecture choices for DNN e.g., neural networks with residual connections, that may perform better.

Table 2.7.2: *Test ROC AUC of **GRAND-SLAMIN**, EB^{2}M and NODE-GA^{2}M. We report median across 10 runs along with the mean absolute deviation (MAD).*

	Interpretable Models			Full Complexity Models
Dataset\Model	EB^{2}M	NODE-GA^{2}M	GRAND-SLAMIN	DNN
Magic	93.12 ± 0.001	$\mathbf{94.27} \pm 0.13$	93.86 ± 0.30	93.69 ± 0.04
Adult	91.41 ± 0.0004	$\mathbf{91.75} \pm 0.14$	91.54 ± 0.14	90.26 ± 0.04
Churn	91.97 ± 0.005	89.62 ± 5.61	$\mathbf{92.40} \pm 0.41$	90.28 ± 0.34
Satimage	97.65 ± 0.001	98.70 ± 0.07	$\mathbf{98.81} \pm 0.04$	98.67 ± 0.07
Texture	99.81 ± 0.0004	$\mathbf{100.00} \pm 0.00$	$\mathbf{100.00} \pm 0.00$	99.63 ± 0.09
MiniBooNE	97.86 ± 0.0001	$\mathbf{98.44} \pm 0.02$	97.77 ± 0.05	97.08 ± 0.38
Covertype	90.08 ± 0.0003	95.39 ± 0.12	$\mathbf{98.11} \pm 0.08$	93.83 ± 0.10
Spambase	$\mathbf{98.84} \pm 0.01$	98.78 ± 0.06	98.01 ± 4.70	98.09 ± 0.07
News	73.03 ± 0.002	$\mathbf{73.53} \pm 0.06$	73.24 ± 0.04	72.19 ± 0.08
Optdigits	99.79 ± 0.0003	99.93 ± 0.02	$\mathbf{99.98} \pm 0.00$	99.77 ± 0.04
Bankruptcy	$\mathbf{93.85} \pm 0.01$	92.02 ± 1.03	92.51 ± 0.54	88.38 ± 0.36
Madelon	88.04 ± 0.02	60.07 ± 0.82	$\mathbf{89.25} \pm 1.03$	63.60 ± 0.47
Activity	74.96 ± 8.77	$\mathbf{99.86} \pm 0.04$	99.24 ± 1.45	96.64 ± 1.93
Multiple	$\mathbf{99.96} \pm 0.0002$	99.94 ± 0.02	99.92 ± 0.02	99.94 ± 0.01

2.7.4.2 GRAND-SLAMIN versus GAMs with all pairwise interactions

Next, we compare GRAND-SLAMIN i.e., GAMs with sparse pairwise interactions against GAMs with all pairwise interactions toolkit:

1. NA^{2}M, i.e., Neural Additive Model [4] with pairwise interactions,

2. NB^{2}M, i.e., Neural Bases Model [216] with pairwise interactions,

3. SPAM, i.e., Scalable Polynomial Additive Model [63] with pairwise interactions.

For NA^{2}M, NB^{2}M, and SPAM, we tuned over learning rate in the set $\{0.1, 0.01, 0.001, 0.0001\}$ and number of epochs in the set $\{50, 100, 500\}$. We capped the time for each trial for NA^{2}M to 6 hrs.

We show results in Table 2.7.3. We outperform SPAM across many datasets. Notably, GRAND-SLAMIN improves by 21% over SPAM on Madelon. We are also competitive with NB^{2}M and NA^{2}M. For larger datasets e.g., Madelon, Activity and Multiple, NB^{2}M and NA^{2}M ran out of memory on a compute node with 2 V100 Tesla GPUs. Recall also that the

Table 2.7.3: *Test ROC AUC for* `GRAND-SLAMIN` *and GAMs with all pairwise interaction models, e.g., NA^{2}M, NB^{2}M and SPAM.*

Dataset	NA^{2}M (Dense)	NB^{2}M (Dense)	SPAM (Dense)	`GRAND-SLAMIN` (Sparse)
Magic	**94.46** ± 0.16	94.11 ± 0.08	91.75 ± 0.004	93.86 ± 0.30
Adult	90.81 ± 0.10	91.06 ± 0.03	89.65 ± 0.001	**91.54** ± 0.14
Churn	**93.03** ± 0.49	92.16 ± 0.42	88.41 ± 0.05	92.40 ± 0.41
Satimage	98.81 ± 0.05	**98.89** ± 0.03	97.94 ± 0.02	98.81 ± 0.04
Covertype	out of time	**98.36** ± 0.02	96.29 ± 0.02	98.11 ± 0.08
Spambase	98.38 ± 0.05	98.37 ± 0.06	97.78 ± 0.04	**98.55** ± 0.07
News	71.94 ± 0.30	72.54 ± 0.07	72.43 ± 0.06	**73.24** ± 0.04
Bankruptcy	87.83 ± 0.12	**93.01** ± 1.85	89.35 ± 0.90	92.51 ± 0.54
Madelon	out of memory	out of memory	68.59 ± 0.80	**89.25** ± 1.03
Activity	out of memory	out of memory	99.10 ± 0.04	**99.24** ± 1.45
Multiple	out of memory	out of memory	**99.98** ± 0.03	99.92 ± 0.02

goal of our work is to learn sparse components with/without structural constraints for easier interpretability.

2.7.4.3 Comparison with Group Lasso

Group Lasso [95] is also a possible choice for sparse selection, which is popularly used in high-dimensional statistics and machine learning. We compare against a version of Group Lasso:

$$\min_{\substack{\{f_i\}_{i\in[p]},\\ \{f_{i,j}\}_{i<j}}} \hat{\mathbb{E}}\left[l(y, \sum_{i\in[p]} f_i(x_i) + \sum_{i<j} f_{i,j})\right] + \lambda\left(\sum_{i\in[p]} \|f_i\|_2 + \sum_{i<j} \|f_{i,j}\|_2\right), \tag{2.7.5}$$

where $\hat{\mathbb{E}}$ denotes the empirical loss on the training dataset and $\|f_i\|_2$ denotes the regularization imposed via the group regularization on the leaf weights of the effect i. We compare performance between `GRAND-SLAMIN` and Group Lasso (with soft tree shape functions) in terms of test AUC and variable selection on a few datasets. The numbers are reported in Table 2.7.4. Overall, our approach significantly outperforms Group Lasso in terms of AUC performance and variable selection.

Table 2.7.4: *Comparison of Test ROC AUC of `GRAND-SLAMIN` with Group Lasso.*

Model / Dataset	Group Lasso (AUC)	`GRAND-SLAMIN` (AUC)	Group Lasso (#features)	`GRAND-SLAMIN` (#features)
Adult	91.19	**91.54** $\pm$ 0.14	14	**12**
Spambase	98.32	**98.81** $\pm$ 0.04	57	**44**
Madelon	65.22	**90.13** $\pm$ 1.03	500	**15**

2.7.4.4 `GRAND-SLAMIN` with different shape functions (Soft Trees vs MLPs)

Next, we compare `GRAND-SLAMIN` with different shape functions. In particular, we study the effect of using neural network shape functions instead of soft tree shape functions. We consider multilayer perceptrons (MLPs) as the functional form for each of the shape functions. We compare test AUC performance on 5 datasets in Table 2.7.5. Interestingly, it seems to us that soft trees seem to have an edge over MLPs across some datasets, when MLPs are matched in number of parameters to soft trees.

Table 2.7.5: *Soft tree shape functions vs MLP shape functions.*

Dataset\Model	`GRAND-SLAMIN` with *MLPs*	`GRAND-SLAMIN` with *Soft Trees*
Magic	93.13±0.12	**93.86**±0.30
Churn	92.33±0.56	**92.40**±0.41
MiniBooNE	97.41±0.21	**97.77**±0.05
Spambase	98.27±0.13	**98.55**±0.07
News	72.87±0.09	**73.24**±0.04

MLPs with varying complexity We further experimented with MLPs in our framework, where we vary the complexity of the individual components. We show training time on News dataset ($p = 61$, $\#Interaction = 1800$, $N = 40k$) in Table 2.7.6. The results show that the active set (learnt by learnable indicators) in the model is the primary contributing factor in terms of timing. The functions (1 or 2) $\rightarrow 64 \rightarrow 64 \rightarrow 1$ have at least $22\times$ more parameters than (1 or 2) $\rightarrow 64 \rightarrow 1$. However, the time only increases by $\sim 1.5\times$. We also show the AUC performance with MLPs with varying complexity in Table 2.7.7. Interestingly, we didn't observe any improvement in AUCs with more complex components. It seems to us that since

Table 2.7.6: *Training times for different choices of MLP shape functions.*

% Components Selected	MLPs with varying complexity	Training times
25% effects	(1 or 2) $\rightarrow$ 64 $\rightarrow$ 1	252.4 ± 0.4
	(1 or 2) $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 1	247.6 ± 0.8
	(1 or 2) $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 1	258.8 ± 10.7
	(1 or 2) $\rightarrow$ 128 $\rightarrow$ 128 $\rightarrow$ 1	268.5 ± 19.1
	(1 or 2) $\rightarrow$ 256 $\rightarrow$ 256 $\rightarrow$ 1	837.0 ± 43.6
75% effects	(1 or 2) $\rightarrow$ 64 $\rightarrow$ 1	343.3 ± 5.3
	(1 or 2) $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 1	422.7 ± 9.9
	(1 or 2) $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 1	506.0 ± 34.4
	(1 or 2) $\rightarrow$ 128 $\rightarrow$ 128 $\rightarrow$ 1	552.4 ± 151.4
	(1 or 2) $\rightarrow$ 256 $\rightarrow$ 256 $\rightarrow$ 1	1193.8 ± 0.3

Table 2.7.7: *AUC performance with MLP shape functions.*

MLPs with varying complexity	Test AUC
(1 or 2) $\rightarrow$ 64 $\rightarrow$ 1	**72.90** $\pm$ 0.09
(1 or 2) $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 1	72.48 ± 0.13
(1 or 2) $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 1	72.58 ± 0.17
(1 or 2) $\rightarrow$ 128 $\rightarrow$ 128 $\rightarrow$ 1	72.34 ± 0.11
(1 or 2) $\rightarrow$ 256 $\rightarrow$ 256 $\rightarrow$ 1	71.86 ± 0.37

our additive modeling framework is fitting low-dimensional components — 1-dimensional functions for main effects and 2-dimensional functions for pairwise interactions effects — the complexity of the function class doesn't need to be too large to get good model accuracy.

2.7.4.5 Effect of entropy regularization

Next, we study the impact of entropy regularization on performance and variable selection. It can be hypothesized that there might be a trade-off in the speed at which the binary state of the gate variables should be reached. On the one hand, suppression of the uninformative terms should happen fast as the early forward and backward passes of all p^2 interaction trees is computationally prohibitive. On the other hand, making this decision prematurely means that the model might not have had enough time to fit terms before suppressing them. We provide an ablation study for model performance as a function of entropy regularization in Table 2.7.8. We observe that when the entropy regularization is too high, the model suffers

Table 2.7.8: *Effect of entropy regularization τ on test AUC and component selection on Spambase. We report median and mean absolute deviation across 50 runs.*

τ	0.00001	0.0001	0.001	0.005	0.01	0.05	0.1	1.0
AUC	98.58±0.02	98.58±0.02	98.54±0.02	98.37±0.09	98.40±0.12	98.40±0.15	98.43±0.11	98.30±0.08
#Effects Selected	1653 ± 0	1235 ± 164	592 ± 173	558 ± 328	535 ± 352	507 ± 451	721 ± 528	1653 ± 26

in performance as some informative terms can also get suppressed too quickly. When the entropy regularization is too small, the model can produce very dense solutions, which can hurt interpretability as well as computational scalability. In a reasonable range of entropy regularization, there is a good region where the model produces a sparse solution for high AUC. Interestingly, we also observe that we didn't see a huge sensitivity of variable selection for a range of entropy values in the middle.

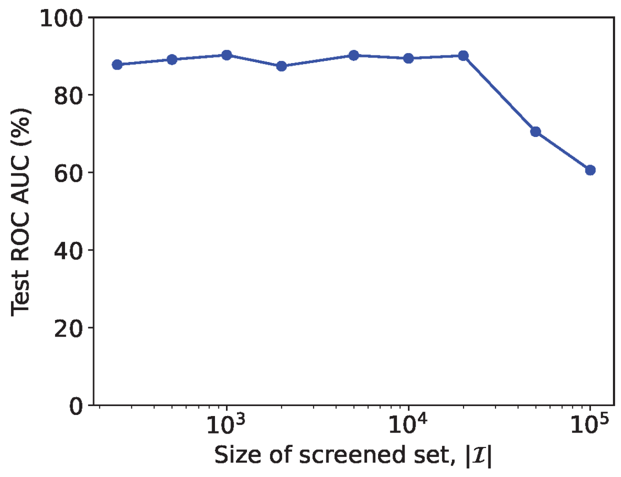


Figure 2.7.1: *Performance of* **`GRAND-SLAMIN`** *with various pre-training screening levels on Madelon.*

2.7.4.6 Effect of screening on performance

Here, we highlight that screening can help improve performance of our model. We consider a challenging Madelon dataset from the NIPS-2003 feature selection challenge, where only a very small subset of features are informative. The number of total features in the dataset are 500. The number of pairwise interactions in this dataset are $\sim 125k$. This is a large combinatorial space for variable selection in terms of interaction effects. A large portion of these interactions are non-informative. Pre-training screening rules similar to the ones used by [172] can effectively reduce the combinatorial space. Recall that the actual number of interactions selected by the model is much smaller than the screened set of interactions through variable selection by our model while training. We can observe from Fig. 2.7.1 that for a range of pre-training screening levels, our model can achieve almost 90% AUC performance — this performance is better than all existing approaches for sparse GAMs with interactions. In particular, the state-of-the art toolkit NODE-GAM that performs end-to-end variable seclection achieves 60% AUC.

Screening also plays a role in faster runtimes. For example, when the screened set is 10,000 interactions out of a total of 125k, the model can be $3\times - 5\times$ faster. Additionally with screening, the overall memory footprint of the model can be much smaller. The overall memory footprint of the model is dictated by the number of interaction effects. With screening, the initial memory footprint of the model can be reduced without a loss in performance.

2.7.4.7 Timing comparison

Our toolkit is highly competitive in terms of training times with all existing tree-based and neural-based toolkits for sparse GAMs with interactions. See Table 2.7.9 for timing comparisons.

2.7.5 Additional visualizations for variance reduction in estimation of main effects under structural constraints

Here we consider tract-level American Community Survey dataset from US Census Bureau Planning Database 2022 [253]. Following [114], we consider a reduced dataset with ~ 39 covariates (741 possible pairwise interactions) and consider the self-response as the regression target. We fit `GRAND-SLAMIN` with different structural constraints. We visualize the estimated main effects in the presence of interaction effects for these structural constraints for some features in Figure 2.7.2. We can observe that when additional hierarchy constraints are

Table 2.7.9: *Training time in seconds of* ***`GRAND-SLAMIN`****, EB^2M, NODE-GA^2M, GAMI-Net and SIAN. We report median across 10 runs. Hyphen (-) indicates either the toolkit does not support multiclass e.g., GAMI-Net, SIAN or does not fit interaction effects for multiclass e.g., EB^2M.*

Dataset\Model	EB^2M	NODE-GA^2M	GAMI-Net	SIAN	`GRAND-SLAMIN`
Magic	**140**	327	1567	608	430
Adult	**284**	612	3611	2396	1018
Churn	**33**	699	2340	298	35
Spambase	297	361	1979	2197	**133**
Miniboone	**1181**	1523	39334	64200	1662
Online	402	**325**	9030	3877	432
Bankruptcy	156	240	2630	3410	**57**
Madelon	403	899	5217	58500	**46**
Satimage	—	133	—	—	**52**
Texture	—	128	—	—	**98**
Optdigits	—	320	—	—	**55**
Covertype	—	**604**	—	—	3144
Activity	—	291	—	—	**110**
Multiple	—	1417	—	—	**184**

imposed, the error bars are much smaller across different runs.

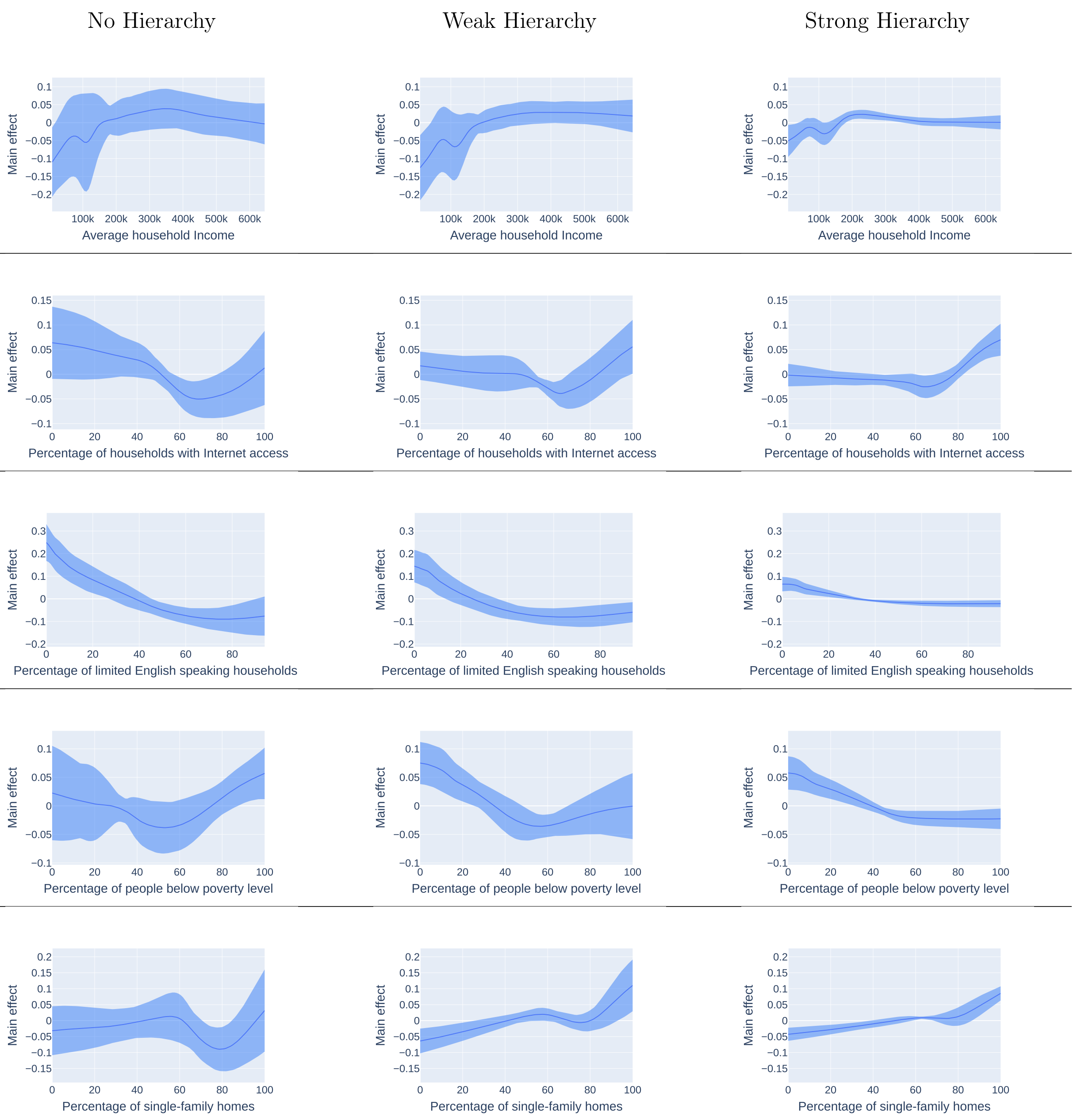


Figure 2.7.2: *Estimated main effects in the presence of interaction effects on American Community Survey dataset [Left] without hierarchy, [Middle] with weak hierarchy and [Right] with strong hierarchy. Strong hierarchy has the smallest error bars. We show visualization for 5 features: (a) Row 1: Average household income, (b) Row 2: Percentage of households with internet access, (c) Row 3: Percentage of limited English speaking households, (d) Row 4: Percentage of people below poverty level, (e) Row 5: Percentage of single-family homes.*

Chapter 3

Additive Models and Structured Interactions: Alternative Optimization Approaches

3.1 Introduction

We propose estimators based on Additive Models [93, 94], or AMs, with smooth nonlinear components that include nonlinear pairwise interactions between covariates. Given response $y \in \mathbb{R}$ and feature-vector $\mathbf{x} := (x_1, \ldots, x_p) \in \mathbb{R}^p$, we model the conditional mean function as

$$\mathbb{E}(y|\mathbf{x}) = \sum_{j \in [p]} f_j(x_j) + \sum_{j<k} f_{j,k}(x_j, x_k), \tag{3.1.1}$$

where f_j and $f_{j,k}$ are unknown smooth univariate and bivariate functions, respectively. Drawing inspiration from linear model settings [10], we propose new methodology to estimate the unknown functional components via an optimization problem with structural constraints arising from interpretability considerations. To obtain a sparse model with few predictors, that is, with many of the components $\{f_j\}$ and $\{f_{j,k}\}$ estimated as exactly zero, we present a novel ℓ_0-regularized approach[1], penalizing the number of nonzero components in the model. In addition, we explore a refined notion of interpretability in the context of interaction modeling – namely, hierarchical sparsity – inspired by its usage in linear models [for example, 26, 98]. To our knowledge, we present a novel exploration of computational and statistical perspectives of an ℓ_0 regularized approach for fitting sparse additive models with pairwise interactions.

Contributions. We propose a new family of estimators based on nonparametric additive models with interactions under combinatorial constraints that promote parsimony and interpretability. We present large-scale algorithms to compute these estimators – these algorithms significantly expand the current computational landscape for nonparametric additive models with pairwise interactions. Our approach addresses some of the key computational challenges posed by the large-scale setting of the Census dataset (next chapter), with $\approx 10^5$ nonparametric interaction components and $\approx 10^5$ observations. The code implementing our algorithms is available at https://github.com/mazumder-lab/elaan.

Related Work. There is an impressive body of methodological and theoretical work on using convex ℓ_1-based approaches to fit sparse nonparametric AMs without interactions [see, for example, 111, 185, 220, 283, 287, and the references therein]. Even with main-effects alone, these convex optimization-based methods face computational challenges for the problem-scales we seek to address[2]. This possibly limits practitioners from realizing the full potential of nonparametric AMs in large-scale settings arising in various practical applications. In terms of statistical properties, the ℓ_0-based estimators can offer improvements over their ℓ_1-counterparts, both on the prediction and the model selection fronts. To this end, we refer the reader to recent work by [104] demonstrating the advantages of using the ℓ_0-

[1]For examples of ℓ_0-based approaches and illustrations of their advantages over the ℓ_1-based counterparts in the context of linear regression, see [25, 99, 181, 182] and the references therein.

[2]Based on our experience, currently available software (for example, R package `SAM`) encounters numerical difficulties for instances where n is on the order of thousands, and p is on the order of hundreds.

regularized framework for grouped variable selection (which includes nonparametric AMs) *without* interactions.

In the presence of pairwise interactions, a setting we focus on, the problem of variable selection in nonparametric AMs becomes considerably more challenging. The approaches by Meier et al. [185] and Ravikumar et al. [220] consider additive models with main effects but no pairwise interactions. Lin and Zhang [165] introduced COSSO, which is a well-known method to fit model (3.1.1); however, the method appears to be suitable for low-dimensional settings – for example, the authors consider instances with $n \sim 500$, $p \sim 10$, and ~ 50 pairwise interactions[3].

COSSO penalizes the sum of the Sobolev norms of the functional components in representation (3.1.1) and, thus, is capable of producing sparse models. However, the convex COSSO penalty jointly controls sparsity and smoothness, potentially resulting in unwanted shrinkage interfering with component selection. In contrast, our proposed approach uses separate penalties for smoothness and sparsity, and encourages sparsity *directly* via an ℓ_0-based penalty on the number of components in the model.

More recently, some methods have been proposed that extend the scope of classical nonparametric AMs [93]. In a series of works Lou et al. [172], Nori et al. [199], Yang et al. [276] explore tree-based and neural-network-based approaches for fitting additive models with sparse interactions. Lou et al. [172] propose a two-stage approach using shallow tree-like models for the main and interaction effects. In the first stage, they use gradient boosting to fit the main effects – the boosting procedure cycles through all features in a round-robin fashion to fit *all* main effects. In the second stage, they use a scheme to select a small subset of pairwise interactions–these interaction effects are fitted with shallow tree-like models via gradient boosting. Nori et al. [199] provide an efficient implementation of the above approach as Explainable Boosting Machines (EBM) in the well-known *interpretml* toolkit. The EBM procedure is not based on optimizing a penalized estimation framework and, to our knowledge,

[3]There appears to be no open-source implementation for COSSO.

no statistical guarantees for it are known.

In another line of work, Yang et al. [276] proposed GAMI-Net, which uses neural networks to fit main and pairwise interaction effects. GAMI-Net uses a multi-stage approach: (i) fit all main effects, where each main effect is modeled using a multi-layer perceptron (MLP); (ii) select Top-k interactions based on an interaction detection method [172] on the residuals; (iii) fit the Top-k interaction effects simultaneously, where each interaction effect is again modeled as MLP; (iv) fine-tune all selected main effects and interaction effects toegther. GAMI-Net also prunes some components in steps (i) and (iii) based on some ranking measure. GAMI-Net uses screening methods prior to training to select a collection of main and interaction effects and, as such, does not jointly optimize the sparsity pattern *while* training.

Zschech et al. [296] independently compared various existing interpretable models and concluded that EBM [199] and GAMI-Net [276] appear to be the leading interpretable models. However, both models have some practical limitations. For example, EBM includes all main effects in the model (i.e., there is no feature selection). Neural-based approaches, such as GAMI-Net, are quite computationally expensive. On the Census dataset that we consider in the next chapter, GAMI-Net takes 3 days (using a 8-CPU machine) to compute one model if we consider 1000 tuning parameters, we are looking at a steep 3000 days of computation cost.

To the best of our knowledge, our algorithmic framework for sparse nonparametric AMs with interactions is novel.

Organization. Section 3.2 presents the statistical models pursued in this chapter. Section 3.2.3 discusses how to obtain solutions to the corresponding large-scale discrete optimization problems. Section 3.3 develops an extension of our method that incorporates strong hierarchy constraints. In Section 3.4, we present simulation studies comparing our methods with COSSO, EBM and GAMI-Net. Additional technical details are provided in the supplementary material.

3.2 Statistical Models and Methodology

We now discuss the statistical models we pursue in this work. Section 3.2.1 gives an overview of AMs with nonlinear main effects and pairwise interactions, along with optimization formulations associated with the estimation procedures. Section 3.2.2 presents our new models to incorporate sparsity in the main and interaction effects.

3.2.1 Smooth additive models with pairwise interactions

Given data $\{(y_i, \mathbf{x}_i)\}_1^n$, our key objective is to learn a multivariate conditional mean function $f(\mathbf{x}) := \mathbb{E}(y|\mathbf{x})$, where $f : \mathbb{R}^p \mapsto \mathbb{R}$ is an unknown smooth function [256]. It is well known that such functions become difficult to estimate even for moderately high p due to the curse of dimensionality – therefore we will focus on a smaller class of models corresponding to additive structures [93, 241]. A popular approach considered in the literature estimates a nonparametric additive model containing main-effects only, with $f(\mathbf{x}) = \sum_{j=1}^p f_j(x_j)$, where each f_j is an unknown univariate smooth function of the j-th coordinate in $\mathbf{x}$, namely x_j. In various applications, however, nonparametric AMs based on main effects alone may not lead to a sufficiently rich representation for predicting the outcome of interest: including interaction terms can lead to better predictive models while remaining interpretable to a practitioner [93]. AMs with pairwise interactions are a useful tool in applied statistical modeling with various applications in medical sciences and healthcare, e-commerce applications, recommender system problems, and sentiment analysis, among others [95].

An additive model with nonlinear main effects and pairwise interactions extends the traditional AM framework with main effects alone [94, 215], and is given by model (3.1.1), where the unknown components $\{f_j\}$ and $\{f_{j,k}\}$ need to be estimated from the data. This leads to two key challenges. The presence of $O(p^2)$-many unknown nonparametric functions results in statistical challenges even for a moderate value of p. Additional regularization (for example, in the form of sparsity in the components) may be necessary to obtain a reliable statistical model with good generalization properties (cf Section 3.2.2). Furthermore, as we mention in Section 3.1, estimating model (3.1.1) leads to severe computational challenges for large problems (for example, those with $n \approx 10^5$ and $p^2 \approx 10^5$, similar to the instances we consider in our applications) – we discuss how we address these challenges in Section 3.2.3.

We assume that the components $\{f_j\}$ and $\{f_{j,k}\}$ are smooth (for example, twice continuously differentiable). We let $\mathbf{f}_j = (f_j(x_{1j}), \ldots, f_j(x_{nj}))$ and $\mathbf{f}_{j,k} = (f_{j,k}(x_{1j}, x_{1k}), \ldots, f_{j,k}(x_{nj}, x_{nk}))$ denote the evaluations of the main effect component f_j and the interaction component $f_{j,k}$, respectively, at the n data points. Writing $\mathbf{y}$ for the response vector and using squared ℓ_2-loss as the data fidelity term, the task of learning (3.1.1) can be expressed as the following optimization problem:

$$\min_{\substack{f_j \in \mathcal{C}_1, \forall j \\ f_{j,k} \in \mathcal{C}_2, \forall j<k}} \frac{1}{n} \|\mathbf{y} - \sum_{j \in [p]} \mathbf{f}_j - \sum_{j<k} \mathbf{f}_{j,k}\|_2^2 + \lambda_1 \big[\sum_{j \in [p]} \Omega(f_j) + \sum_{j<k} \Omega(f_{j,k}) \big], \tag{3.2.1}$$

where $\mathcal{C}_1$ and $\mathcal{C}_2$ denote the sets of smooth candidate functions, Ω is a roughness penalty[4], and λ_1 is a non-negative regularization parameter controlling the smoothness of the fit. We show in Section 3.2.3 that problem (3.2.1) can be written as a finite-dimensional quadratic program by using cubic splines to model each of the main and interaction effects.

3.2.2 Parsimonious models via ℓ_0-penalization

Here we study ℓ_0-type estimators that limit the number of components in the additive models introduced earlier.

3.2.2.1 Sparse pairwise interactions

While a significant body of work has been devoted to studying sparsity in the context of linear models, sparsity in nonlinear models has received relatively less attention. Interestingly,

[4]For example, $\Omega(f_j) = \int f_j''(x_j)^2 dx_j$ and $\Omega(f_{ij}) = \int_{x_j x_k} (\partial^2 f_{j,k}/\partial x_j^2)^2 + (\partial^2 f_{j,k}/\partial x_j \partial x_k)^2 + (\partial^2 f_{j,k}/\partial x_k^2)^2 dx_j dx_k$.

we observe that the notion of parsimony is linked to the model being used and changes, for example, depending on whether we use a linear interaction model of the form $\mathbb{E}(y|\mathbf{x}) = \sum_j x_j\beta_j + \sum_{j<k} x_j x_k \beta_{jk}$ or model (3.1.1) that has nonlinear components.

To obtain an additive model with a small number of main and interaction effects, we consider an ℓ_0-penalized modification of optimization problem (3.2.1). We write

$$\Omega_{\text{gr}}(f) = \sum_{j\in[p]} \Omega(f_j) + \sum_{j<k} \Omega(f_{j,k}) \qquad \text{and} \qquad \mathbf{f} = \sum_{j\in[p]} \mathbf{f}_j + \sum_{j<k} \mathbf{f}_{j,k}, \tag{3.2.2}$$

to simplify the expressions, and propose an estimator that optimizes the following problem:

$$\min_{\substack{f_j\in\mathcal{C}_1,\forall j\\ f_{j,k}\in\mathcal{C}_2,\forall j<k}} \frac{1}{n}\left\|\mathbf{y}-\mathbf{f}\right\|_2^2 + \lambda_1\Omega_{\text{gr}}(f) + \lambda_2\Big[\sum_{j\in[p]} \mathbb{1}\big[\boldsymbol{f}_j \neq \mathbf{0}\big] + \alpha\sum_{j<k}\mathbb{1}\big[\boldsymbol{f}_{j,k}\neq\mathbf{0}\big]\Big], \tag{3.2.3}$$

where $\mathbb{1}[\cdot]$ is an indicator function, $\lambda_2 \in [0,\infty)$ controls the number of selected components, and $\alpha \in [1,\infty)$ controls the tradeoff between the number of main and interaction effects. We note that the finite-dimensional version of Problem (3.2.3) can be formulated as a mixed integer program (MIP) [264], and hence can be solved to optimality with modern commercial solvers (for example, Gurobi, Mosek) for small/moderate scale problems. Recently, tailored nonlinear branch-and-bound techniques [103] have been shown to be promising for solving large-scale instances of ℓ_0-sparse linear regression problems. Since we intend to compute solutions to (3.2.3) for a family of tuning parameters (λ_1, λ_2) at scale, we consider high-quality approximate solutions (cf Section 3.2.3). Once a good solution to (3.2.3) is available, we can employ MIP techniques to improve the solution and/or certify the quality of the solution, extending the local search techniques presented in [99] for ℓ_0-sparse linear regression problems. To our knowledge, our methodological investigation of estimator (3.2.3) is novel. We refer to this model as `ELAAN-I`[5].

While in this paper, we study a group ℓ_0 penalty in (3.2.3), one can also consider other non-convex penalty functions: for example, based on the (group) SCAD [70] and MCP

[5]`ELAAN-I` stands for End-to-end Learning Approach for Additive spliNes with Interactions.

[284]. Exploring the statistical and computational aspects of using such penalties would be interesting to pursue and is left as future work.

Related Work. In the special case of (3.2.3), when no interactions are present, several convex relaxations of the group ℓ_0-penalty have been studied [32, 185, 220, 287][6]. [104] consider ℓ_0-formulations for the setting without interaction effects and demonstrate the merits of using ℓ_0-based formulations over convex relaxation-based approaches. Despite the appeal of estimator (3.2.3), computational challenges appear to be a key limiting factor in exploring this model in practice. In Section 3.2.3, we present a new algorithm for problem (3.2.3).

Statistical Theory. We establish statistical guarantees for the resulting estimators in Ibrahim et al. [114]. In particular, we present general non-asymptotic oracle prediction error bounds, comparing the performance of our estimator to that of sparse approximations to the true regression function. Bounds of this type have been established for sparse nonparametric AMs with main effects; however, we are not aware of similar existing bounds for nonparametric models with pairwise interactions. The existing work has focused mainly on the Lasso-based estimators, which encourage sparsity in the main-effects by penalizing the magnitudes of the functional components [see 185, 243, and the references therein]. The use of ℓ_1-based relaxations, in lieu of ℓ_0-penalization, to induce sparsity in the main-effects can lead to unwanted shrinkage, which may interfere with variable selection. The approach in [104] controls the number of main-effects directly, demonstrating the benefits of ℓ_0-regularization both theoretically and empirically. Our analysis accounts for the interaction effects and focuses on the ℓ_0-penalized formulation, which poses additional theoretical challenges relative to the ℓ_0-constrained formulation. To our knowledge, the error bounds established in Ibrahim et al. [114] for sparse nonparametric AMs with pairwise interactions are novel. The established error bounds also have implications for model selection and our proposed approach is model

[6]In terms of existing implementations, R package `SAM` presents specialized algorithms for a convex relaxation of (3.2.3) without interactions. `SAM` however would not run on the dataset we consider here.

selection consistent.

3.2.3 Efficient Computations at Scale

We present specialized algorithms for obtaining solutions to Problem (3.2.3). Our approach scales to the problem-sizes with $n \approx 10^5$ and $p \approx 500$, which poses formidable computational challenges due to the presence of approximately 10^5 interaction effects[7]. To obtain good solutions at scale, we use techniques inspired by first order methods in continuous optimization [196] and a careful exploitation of the problem-structure. A high-level summary is presented below, with the details relegated to the Appendix.

Prior approaches. To appreciate the computational challenges of sparse nonlinear AMs with interactions, and nonparametric AMs in general, we provide a few examples of the problem instances that can be handled by prior state-of-the-art algorithms with publicly available implementations. Implementations based on R package `SAM` [287], the stepwise GAM function in R package `step.gam` (which performs greedy variable selection), and Python package `pyGAM`, take on the order of days to run and/or face numerical difficulties for instances with $n \approx 10^5$ to obtain a single solution without interactions. Furthermore, `pyGAM` is unable to do automated variable selection. [266] present an interesting approach for AMs that scales to large n settings (see R package `mgcv`) but doesn't appear to perform automated variable selection in the presence of a large number of features – Wood et al. [266] report instances containing fewer than 20 pre-specified main and interaction effects. EBM (199) is a state-of-the-art computational approach for AMs, but does not allow for feature selection as all the main effects are included in the estimated model. Additionally, the variant of EBM that selects interaction effects, results in many interaction effects, leading to sub-optimal support recovery – see Sections 3.4 for an illustration. The GAMI-Net method (276) is computationally expensive. Additionally, since it uses a multi-layered NN for every

[7]Note that, using 25 knots for every component, this leads to estimating around 2.5 million basis coefficients.

interaction effect it requires 100× more parameters when compared with the approaches we consider.

Algorithms for sparse nonlinear interactions: Problem (3.2.3). We represent the main and interaction effects as linear combinations of cubic spline basis functions (see the Appendix for the specific details). In particular, we let $\mathbf{f}_j = \mathbf{B}_j\boldsymbol{\beta}_j$, where $\mathbf{B}_j \in \mathbb{R}^{n\times K_j}$ is the model matrix and $\boldsymbol{\beta}_j \in \mathbb{R}^{K_j}$ is the vector of coefficients for each main effect component. Similarly, we let $\mathbf{f}_{j,k} = \mathbf{B}_{j,k}\boldsymbol{\theta}_{j,k}$, where $\mathbf{B}_{j,k} \in \mathbb{R}^{n\times K_{j,k}}$ is the model matrix and $\boldsymbol{\theta}_{j,k} \in \mathbb{R}^{K_{j,k}}$ is the vector of coefficients for each interaction effect component. Writing $\boldsymbol{\beta}$ for the vector obtained by stacking together the coefficients $\boldsymbol{\beta}_j$, $j \in [p]$ for the main-effects, and defining the vector $\boldsymbol{\theta}$ for the interaction effects analogously, we express the objective function in (3.2.1) as follows:

$$g_{\lambda_1}(\boldsymbol{\beta},\boldsymbol{\theta}) \overset{\text{def}}{=} \frac{1}{n}\Big\|\boldsymbol{y} - [\sum_{j\in[p]} \boldsymbol{B}_j\boldsymbol{\beta}_i + \sum_{j<k} \boldsymbol{B}_{j,k}\boldsymbol{\theta}_{j,k}]\Big\|_2^2 + \lambda_1[\sum_{j\in[p]} \boldsymbol{\beta}_j^T\boldsymbol{S}_j\boldsymbol{\beta}_j + \sum_{j<k} \boldsymbol{\theta}_{j,k}^T\boldsymbol{S}_{j,k}\boldsymbol{\theta}_{j,k}].$$

Here, $\boldsymbol{S}_j = \boldsymbol{D}_j^T\boldsymbol{D}_j$ and $\boldsymbol{S}_{j,k} = (\boldsymbol{D}_j^T\boldsymbol{D}_j)\otimes\boldsymbol{I}_k + \boldsymbol{I}_j\otimes(\boldsymbol{D}_k^T\boldsymbol{D}_k)$ are the smoothness penalty matrices for the main effects and the interaction components, respectively (further details are provided in the Appendix). For convenience, we use the same smoothness penalty λ_1 for both the main and the interaction effects, though in general they may be taken to be different. The above representation leads to the following form of the optimization problem (3.2.3):

$$\min_{\boldsymbol{\beta},\boldsymbol{\theta}} G(\boldsymbol{\beta},\boldsymbol{\theta}) \overset{\text{def}}{=} g_{\lambda_1}(\boldsymbol{\beta},\boldsymbol{\theta}) + \lambda_2\Big[\sum_{j\in[p]} \mathbb{1}\big[\boldsymbol{\beta}_j \neq \mathbf{0}\big] + \alpha\sum_{j<k}\mathbb{1}\big[\boldsymbol{\theta}_{j,k}\neq\mathbf{0}\big]\Big]. \tag{3.2.4}$$

We note that the indicator functions in the above equation are applied to vectors of basis coefficients corresponding to particular main or interaction effects. Hence, for example, when $\mathbb{1}\big[\boldsymbol{\beta}_j \neq \mathbf{0}\big]$ equals zero, the entire main effect $\mathbf{f}_j$ is also zero. As mentioned earlier, (3.2.4) can be expressed as a MIP and solved for small-to-moderate scale problems using modern MIP solvers. However, given the problem-sizes of interest and the fact that we seek to compute solutions to (3.2.4) for a family of tuning parameters, we discuss faster alternatives. We note that the objective in Problem (3.2.4) is a sum of a smooth convex loss function and a discontinuous regularizer separable across the components $\{\boldsymbol{\beta}_j\}$ and $\{\boldsymbol{\theta}_{i,k}\}$. Motivated by the strong empirical performance of cyclical coordinate descent (CD) methods [267] in ℓ_0-penalized linear regression [99], we explore block CD methods to obtain fast approximate solutions for the nonparametric setting with interactions (3.2.4). For convergence guarantees of this procedure, see [99] and references therein.

CD methods have old roots in optimization dating back to the foundations of the discipline: see for example, the review paper [267]. CD methods have close links with the well known Gauss-Seidel method (for solving linear/nonlinear systems of equations)—similar methods have been extensively used in additive models [91], where they are referred to as *backfitting*. CD schemes (including their block variants) arising from the optimization literature can be efficiently applied to optimization problems involving sparsity constraints: such problems arise in fitting additive models with sparsity-inducing penalties [95, 104, 220]. We note that there are other successful additive model fitting approaches, for example, smooth backfitting [179, 180], which have excellent theoretical properties. These works focus on the additive model setup without sparsity—extending these approaches to our setting of large-scale sparse additive models in a computationally efficient fashion is an interesting direction for future research.

3.2.4 Block Coordinate Descent for solving (3.2.4)

In our block CD method, the blocks correspond to the basis coefficients for either the main effects $\{\boldsymbol{\beta}_j\}$ or the interaction effects $\{\boldsymbol{\theta}_{j,k}\}$. Given an initialization $(\boldsymbol{\beta}_1^{(0)}, \cdots, \boldsymbol{\beta}_p^{(0)}, \boldsymbol{\theta}_{1,2}^{(0)}, \cdots, \boldsymbol{\theta}_{p-1,p}^{(0)})$, at every cycle, we sequentially sweep across the main effects and the interaction effects. If we denote the solution after t cycles by $(\boldsymbol{\beta}_1^{(t)}, \cdots, \boldsymbol{\beta}_p^{(t)}, \boldsymbol{\theta}_{1,2}^{(t)}, \cdots, \boldsymbol{\theta}_{p-1,p}^{(t)})$, then the block of coefficients for j-th main effect $\boldsymbol{\beta}_j^{(t+1)}$ at the cycle $t+1$ is obtained by optimizing (3.2.4) with respect to $\boldsymbol{\beta}_j$, with other variables held fixed:

$$\boldsymbol{\beta}_j^{(t+1)} \in \underset{\boldsymbol{\beta}_j \in \mathbb{R}^{K_j}}{\arg\min}\ G(\boldsymbol{\beta}_1^{(t+1)}, \cdots, \boldsymbol{\beta}_{j-1}^{(t+1)}, \boldsymbol{\beta}_j, \boldsymbol{\beta}_{j+1}^{(t)}, \cdots, \boldsymbol{\beta}_p^{(t)}, \quad \boldsymbol{\theta}_{1,2}^{(t)}, \cdots, \boldsymbol{\theta}_{p-1,p}^{(t)}). \tag{3.2.5}$$

Similarly, $\boldsymbol{\theta}_{j,k}^{(t+1)}$, the coefficients for the (j,k)-th interaction effect at cycle $t+1$ are updated as

$$\boldsymbol{\theta}_{j,k}^{(t+1)} \in \underset{\boldsymbol{\theta}_{j,k} \in \mathbb{R}^{K_{j,k}}}{\arg\min}\ G(\boldsymbol{\beta}_1^{(t+1)}, \cdots, \boldsymbol{\beta}_p^{(t+1)}, \quad \boldsymbol{\theta}_{1,2}^{(t+1)}, \cdots, \boldsymbol{\theta}_{(j,k)-1}^{(t+1)}, \boldsymbol{\theta}_{j,k}, \boldsymbol{\theta}_{(j,k)+1}^{(t)} \cdots, \boldsymbol{\theta}_{p-1,p}^{(t)}). \tag{3.2.6}$$

The block minimization problem (3.2.5) reduces to:

$$\boldsymbol{\beta}_j^{(t+1)} = \underset{\boldsymbol{\beta}_j \in \mathbb{R}^{K_j}}{\arg\min}\ \psi_j(\boldsymbol{r}^{(t)}; \boldsymbol{\beta}_j) := \frac{1}{n}\left\|\boldsymbol{r}^{(t)} - \boldsymbol{B}_j\boldsymbol{\beta}_j\right\|_2^2 + \lambda_1\ \boldsymbol{\beta}_j^T\boldsymbol{S}_j\boldsymbol{\beta}_j + \lambda_2\mathbb{1}[\boldsymbol{\beta}_j \neq \mathbf{0}], \tag{3.2.7}$$

where $\boldsymbol{r}^{(t)} = \boldsymbol{y} - (\sum_{j'=1}^{j-1}\boldsymbol{B}_{j'}\boldsymbol{\beta}_{j'}^{(t+1)} + \sum_{j'=j+1}^{p}\boldsymbol{B}_{j'}\boldsymbol{\beta}_{j'}^{(t)} + \sum_{j'<k'}\boldsymbol{B}_{j',k'}\boldsymbol{\theta}_{j',k'}^{(t)})$ denotes the residual. A solution to (3.2.7) can be computed in closed form via the following thresholding operator:

$$\boldsymbol{\beta}_j^{(t+1)} = \begin{cases} \mathbf{0} & \text{if } \ \psi_j(\boldsymbol{r}^{(t)}; \mathbf{0}) \le \min_{\boldsymbol{\beta}_j \neq \mathbf{0}} \psi_j(\boldsymbol{r}^{(t)}; \boldsymbol{\beta}_j) \\ \left(\boldsymbol{B}_j^T\boldsymbol{B}_j + n\lambda_1\boldsymbol{S}_j\right)^{-1}\boldsymbol{B}_j^T\boldsymbol{r}^{(t)} & \text{otherwise.} \end{cases} \tag{3.2.8}$$

Similarly, the sub-problem for the interaction effects is

$$\boldsymbol{\theta}_{j,k}^{(t+1)} = \underset{\boldsymbol{\theta}_{j,k} \in \mathbb{R}^{K_{j,k}}}{\arg\min}\ \psi_{j,k}(\boldsymbol{r}^{(t)}; \boldsymbol{\theta}_{j,k}) := \frac{1}{n}\left\|\boldsymbol{r}^{(t)} - \boldsymbol{B}_{j,k}\boldsymbol{\theta}_{j,k}\right\|_2^2 + \lambda_1\ \boldsymbol{\theta}_{j,k}^T\boldsymbol{S}_{j,k}\boldsymbol{\theta}_{j,k} + \alpha\lambda_2\mathbb{1}[\boldsymbol{\theta}_{j,k} \neq \mathbf{0}], \tag{3.2.9}$$

where $\boldsymbol{r}^{(t)} = \boldsymbol{y} - (\sum_{j'=1}^{p} \boldsymbol{B}_{j'}\boldsymbol{\beta}_{j'}^{(t+1)} + \sum_{(j',k')=1,2}^{(j,k)-1} \boldsymbol{B}_{j',k'}\boldsymbol{\theta}_{j',k'}^{(t+1)} + \sum_{(j',k')=(j,k)+1}^{p-1,p} \boldsymbol{B}_{j',k'}\boldsymbol{\theta}_{j',k'}^{(t)})$. A solution to this problem is given by

$$\boldsymbol{\theta}_{j,k}^{(t+1)} = \begin{cases} \boldsymbol{0} & \text{if} \quad \psi_{j,k}(\boldsymbol{r}^{(t)}; \boldsymbol{0}) \leq \min_{\boldsymbol{\theta}_{j,k} \neq \boldsymbol{0}} \psi_{j,k}(\boldsymbol{r}^{(t)}; \boldsymbol{\theta}_{j,k}) \\ \left(\boldsymbol{B}_{j,k}^T \boldsymbol{B}_{j,k} + n\lambda_1 \boldsymbol{S}_{j,k}\right)^{-1} \boldsymbol{B}_{j,k}^T \boldsymbol{r}^{(t)} & \text{otherwise.} \end{cases} \tag{3.2.10}$$

These block CD updates need to be paired with several computational devices in the form of active set updates, cached matrix factorizations, and warm-starts, among others. We draw inspiration from similar strategies used in CD-based procedures for sparse linear regression [77, 99], and adapt them to our problem. These devices are discussed in detail in the Appendix Section 3.2.5.

3.2.5 Scalability considerations for solving (3.2.4)

We draw inspiration from strategies used in CD-based procedures for sparse linear regression [77, 99], and adapt them to scale cyclic block coordinate descent in Section 3.2.3 to large problem instances of (3.2.4) . These include active set updates, cached matrix factorizations and warm-starts. They are explained in more detail below:

3.2.5.1 Active set updates

A main computational bottleneck in the block CD approach is the number of passes across the $O(p^2)$ blocks. However, as we anticipate a solution that is sparse (with few nonzero main and interaction effects), we use an active set strategy. We restrict our block CD procedure to a small subset $\mathcal{Q}$ of the $O(p^2)$ variables, with all blocks outside the active set being set to zero. Once the CD algorithm converges on the active set, we check if all blocks outside the active set satisfy the coordinate-wise optimality conditions[8]. If there are any violations, we

[8]That is, we check if optimal solutions to (3.2.5) and (3.2.6) are zero for all blocks outside $\mathcal{Q}$.

select the corresponding blocks, append them to the current active set, and then rerun our block CD procedure. As there are finitely many active sets, the algorithm is guaranteed to converge – in practice, with warm-start continuation (discussed below), the number of active set updates is quite small, and the algorithm is found to converge quite quickly.

3.2.5.2 Cached matrix factorizations

The updates (3.2.8) and (3.2.10) require computing a matrix inverse. In particular, if a block is nonzero, we need to compute a linear system solution of the form: $\boldsymbol{A}_l^{-1}\boldsymbol{b}_l$ where $\boldsymbol{A}_l = \boldsymbol{B}_l^T\boldsymbol{B}_l + n\lambda_1\boldsymbol{S}_l$. We note that matrix $\boldsymbol{A}_l$ is fixed throughout the CD updates and is independent of the choice of the sparsity regularization parameter λ_2 – hence, we pre-compute a matrix factorization for $\boldsymbol{A}_l$ (for example, an LU decomposition) and use it to compute the solution to the linear system. As the dimension of $\boldsymbol{A}_l$ equals the number of basis coefficients, which is small, this can be done quite efficiently.

3.2.5.3 Warm-starts

We use our CD procedure to compute a path of solutions to (3.2.4) for a 2D grid of tuning parameters $(\lambda_1, \lambda_2) \in \{\lambda_1^{(l)}\}_{l=0}^L \times \{\lambda_2^{(m)}\}_{m=0}^M$, where λ_1 corresponds to the smoothness parameter and λ_2 the sparsity parameter. Here, $\lambda_1^{(l)} > \lambda_1^{(l+1)}$ for all l, and $\lambda_2^{(m)} > \lambda_2^{(m+1)}$ for all m. When $\lambda_1 = \lambda_1^{(0)}$ (most regularized), we compute a sequence of solutions across the λ_2-values (from large to small values): a solution obtained at $(\lambda_1^{(0)}, \lambda_2^{(m)})$ is used to initialize our CD procedure for the value $(\lambda_1^{(0)}, \lambda_2^{(m+1)})$. As the number of nonzeros in a solution to (3.2.4) generally increases with λ_2-values, our CD procedure for $(\lambda_1^{(0)}, \lambda_2^{(m+1)})$ uses an active set that is slightly larger than the active set[9] corresponding to the solution at $(\lambda_1^{(0)}, \lambda_2^{(m)})$. Once we have traced a full path over λ_2, we use warm-starts in the lateral direction across the space of λ_1. For all $l \geq 0$, to obtain a solution to (3.2.4) at $(\lambda_1^{(l+1)}, \lambda_2^{(m)})$, we use the solution at $(\lambda_1^{(l)}, \lambda_2^{(m)})$ as a warm-start.

[9]This is usually taken to be 1-10% larger than the current active set, and is chosen in a greedy fashion from among the main and interaction effects lying outside the current active set.

3.3 Incorporating strong hierarchy constraints

In this section, we discuss the model with sparse interactions under strong hierarchy.

Problem (3.2.4) limits the total number of main and interaction effects and works well in our experiments in terms of obtaining a sparse model with good predictive performance. In terms of variable selection properties, however, (3.2.4) can lead to the inclusion of an interaction effect, say, $\{(j,k)\}$ where at least one of the corresponding main-effects $\{j\}$ or $\{k\}$ is excluded from the model. This may be somewhat problematic from an interpretation viewpoint—it may be desirable to enforce additional constraints in (3.2.4), such as the *hierarchy constraints* [26, 183].In this paper, we consider the *strong hierarchy* constraint, where an interaction effect $\{(j,k)\}$ is included in the model only if both the corresponding main effects are also included. In addition to improved interpretation, strong hierarchy can reduce the *effective* number of features in the model, subsequently reducing the operational costs associated with data collection [26, 98].

To enforce strong hierarchy into model (3.2.4), we consider the following estimator:

$$\min_{\boldsymbol{\beta},\boldsymbol{\theta}} \quad g_{\lambda_1}(\boldsymbol{\beta},\boldsymbol{\theta}) + \lambda_2\Big(\sum_{j\in[p]} \mathbb{1}[\boldsymbol{\beta}_j \neq \mathbf{0}] + \alpha \sum_{j<k} \mathbb{1}[\boldsymbol{\theta}_{j,k} \neq \mathbf{0}]\Big) \tag{3.3.1a}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_{j,k} \neq \mathbf{0} \implies \boldsymbol{\beta}_j \neq \mathbf{0} \ \ \& \ \ \boldsymbol{\beta}_k \neq \mathbf{0} \quad \forall j<k,\ j\in[p],\ k\in[p]. \tag{3.3.1b}$$

We note that (3.3.1) differs from (3.2.4) in the additional strong hierarchy constraint appearing in (3.3.1b). By using binary variables to model sparsity in the main/interaction effects and to encode the hierarchy constraint (3.3.1b), Problem (3.3.1) can be expressed as the following

MIP:

$$\min_{\boldsymbol{\beta},\boldsymbol{\theta},\boldsymbol{z}} \quad g_{\lambda_1}(\boldsymbol{\beta},\boldsymbol{\theta}) + \lambda_2\Big(\sum_{j\in[p]} z_j + \alpha \sum_{j<k} z_{j,k}\Big) \tag{3.3.2a}$$

$$\text{s.t.} \quad z_j, z_{j,k} \in \{0,1\}, \quad \|\boldsymbol{\beta}_j\|_2^2 \leq M z_j, \quad \|\boldsymbol{\theta}_{j,k}\|_2^2 \leq M z_{j,k} \quad \forall j<k,\ j,k \in [p], \tag{3.3.2b}$$

$$z_{j,k} \leq z_j, \quad z_{j,k} \leq z_k, \quad \forall j<k, \tag{3.3.2c}$$

where the BigM parameter M is a sufficiently large finite constant such that an optimal solution to (3.3.2) satisfies $\max_j \|\boldsymbol{\beta}_j\|_2^2 \leq M$ and $\max_{j,k} \|\boldsymbol{\theta}_{j,k}\|_2^2 \leq M$. Binary variable z_j (and $z_{j,k}$) indicates whether the corresponding main effect $\boldsymbol{\beta}_j$ (respectively, interaction effect $\boldsymbol{\theta}_{j,k}$) is zero or not; the constraint appearing in (3.3.2c) enforces the hierarchy constraint in (3.3.1b).

To our knowledge, ours is the first work to study estimator (3.3.2). The methodology presented here is of independent interest in the context of structured nonparametric learning with interactions. We refer to this model as `ELAAN-H`[10]. In Section 3.3.1 of the Appendix, we propose algorithms to obtain good solutions to Problem (3.3.2).

Related Work. Methodology for strong hierarchy in linear models has been studied in the statistics/machine learning literature [26, 98, 164, 275] – these works focus on the linear model setting, a special case of the nonlinear setting we consider here. [215] consider hierarchy constraints in the nonparametric setting via convex optimization schemes. To our knowledge, current techniques are unable to scale to the functional learning instances we consider in our paper.

As mentioned earlier, the EBM approach (199) does not support hierarchy constraints. GAMI-Net [276] follows a two-stage approach to fit nonparametric additive models with sparse (weak) hierarchical interactions. GAMI-Net performs a screening step after estimating the main effects with neural-network blocks and only consider interaction effects that satisfy the

[10]`ELAAN-H` stands for End-to-end Learning Approach for Additive spliNes with Hierarchy.

weak hierarchy principle amongst the screened main effect – an interaction effect can appear in the model if one of the main effects is in the model. As mentioned earlier, GAMI-Net is not based on a sparsity-inducing penalized optimization procedure; and can be computationally much more expensive than our estimators.

3.3.1 Algorithms for `ELAAN-H` with sparse hierarchical interactions: Problem (3.3.2)

Similar to the case of (3.2.4), and with the computational speed in mind, we present approximate methods to obtain good solutions to (3.3.2). As mentioned earlier, these approximate solutions can be used to initialize MIP-based approaches for (3.3.2) to improve the solution, and/or to certify the quality of these approximate solutions following Bertsimas et al. [25], Hazimeh and Mazumder [99].

Our first step is to reduce the number of main and interaction effects in (3.3.2) by making use of the family of solutions available from (3.2.4). We consider the union of supports available from the family of solutions obtained from (3.2.4), across the 2D grid of tuning parameters λ_1, λ_2. Let $\mathcal{M} \subset [p]$ and $\mathcal{I} \subset [p(p-1)/2]$ denote the sets of all nonzero main effects and interactions effects, respectively[11], encountered along the 2D path of solutions to (3.2.4). We form a reduced version of problem (3.3.2) with $z_i = 0, i \notin \mathcal{M}$ and $z_{j,k} = 0$ for all $(j,k) \notin \mathcal{I}$. Let us denote the reduced problem by $\mathcal{P}(\mathcal{M}, \mathcal{I})$. We consider a convex relaxation of $\mathcal{P}(\mathcal{M}, \mathcal{I})$, denoted by $\mathcal{P}^R(\mathcal{M}, \mathcal{I})$, where all binary variables $\{z_i\}$, $\{z_{j,k}\}$ in $\mathcal{P}(\mathcal{M}, \mathcal{I})$ are relaxed to the interval $[0,1]$. As the sizes of $\mathcal{M}$ and $\mathcal{I}$ are generally small, it is computationally feasible to solve the relaxation $\mathcal{P}^R(\mathcal{M}, \mathcal{I})$ – we let $\{z_i^R\}$ and $\{z_{j,k}^R\}$ denote a solution to this relaxation. Following [98], it can be shown that this solution satisfies the strong hierarchy constraint (almost surely). To obtain a feasible solution to Problem (3.3.2), we apply a relax-and-round procedure. For the rounding step, we consider a threshold $\tau \in (0,1)$ and obtain $\tilde{z}_i = \mathbb{1}[z_i^R > \tau]$

[11] If $\mathcal{I}$ includes an interaction (j,k) where main-effect j is not included in $\mathcal{M}$, we expand $\mathcal{M}$ to include j. This way, we make sure that all interaction effects have the corresponding main effects included in $\mathcal{M}$.

for all $i \in \mathcal{M}$ and $\tilde{z}_{j,k} = \mathbb{1}[z^R_{j,k} > \tau]$ for all $(j,k) \in \mathcal{I}$. We set $\tilde{z}_i = 0$, $i \notin \mathcal{M}$; and $\tilde{z}_{j,k} = 0$ for all $(j,k) \notin \mathcal{I}$. It can be verified that this rounding procedure maintains strong hierarchy. Finally, we perform a 'polishing' step where we solve (3.2.1) restricted to the support defined by $\{\tilde{z}_i\}_i$ and $\{\tilde{z}_{j,k}\}_{j,k}$.

Related Work. In contrast to Problem (3.2.4), the regularization penalty in problem (3.3.2) is not separable across the blocks due to the overlapping groups created by the strong hierarchy constraint. Hence, the CD-based procedures discussed for (3.2.4) do not apply to the hierarchical setting. To the best of our knowledge, there are no prior specialized algorithms for (3.3.2) that apply to the scale that we consider here. In fact, even in the linear regression setting, current algorithms for problems with a hierarchy constraint are somewhat limited in terms of the problem-scales they can address. The sole exceptions appear to be the convex optimization based approaches of [98, 164], which can address sparse linear regression problems with a large number of features and a small number of observations.

3.4 Simulations

In this section we study the empirical performance (estimation and prediction) of the `ELAAN-I` and `ELAAN-H` estimators on synthetic datasets.

3.4.1 Sparse additive model with interactions

Motivated by [161], we consider a problem with $p = 10$ features where the true underlying model is additive in a small number of main and interaction effects:

$$\begin{aligned} f^*(\mathbf{x}) = {} & g_1(x_1) + g_2(x_2) + g_3(x_3) + g_4(x_4) \\ & + g_1(x_3 x_4) + g_2\left(\frac{x_1 + x_3}{2}\right) + g_3(x_1 x_2), \end{aligned} \tag{3.4.1}$$

where functions $g_1(t) = t$, $g_2(t) = (2t-1)^2$, $g_3(t) = \frac{\sin(2\pi t)}{2-\sin(2\pi t)}$, and $g_4(t) = 0.1\sin(2\pi t) + 0.2\cos(2\pi t) + 0.3\sin^2(2\pi t) + 0.4\cos^3(2\pi t) + 0.5\sin^3(2\pi t)$ are defined on $[0,1]$. Note that the covariates $x_5, \cdots, x_{10}$ do not contribute to the response. Each of the covariates $x_1, \ldots, x_{10}$ are independently drawn from the uniform distribution $\mathcal{U}(0,1)$. We generate the responses as $y = f^*(\mathbf{x}) + \epsilon$, where the errors ϵ are drawn from a Gaussian distribution $\mathcal{N}(0, 0.2546^2)$.

We measure prediction accuracy using the integrated squared error, ISE $= \mathbb{E}_{\mathbf{x}}[(\hat{f}(\mathbf{x}) - f^*(\mathbf{x}))^2]$, estimated by Monte Carlo integration using $10,000$ test observations from the same distribution as the training ones, following the procedure in [161]. We vary the number of training observations from 100 to 400 and use 100 replications for each simulation setting. We compare our estimators to well-known benchmarks EBM [199], GAMI-Net [276], COSSO [165], and MARS[12] [78].

Table 3.4.1 presents the average test ISE and their standard errors (based on the replications). The tuning parameters for `ELAAN-I`, `ELAAN-H`, EBM and GAMI-Net are selected via 5-fold cross-validation. The results for MARS and COSSO are taken from Table 6 in [161]. Table 3.4.1 demonstrates that both of our estimators, `ELAAN-I` and `ELAAN-H`, outperform the competitors across the entire range of training set sizes under consideration.

Next, we study the support recovery performances of `ELAAN-I`, `ELAAN-H`, EBM and GAMI-Net. We evaluate models using discrepancy between the true and estimated support and separately report the average F1-score (defined in Section 3.5.2.1 of the Appendix) for the main and the interaction effects. The support recovery metrics are shown in Table 3.4.2. We observe that, overall, both `ELAAN-I` and `ELAAN-H` appear to outperform EBM and GAMI-Net in terms of the F1-score. Moreover, `ELAAN-H` is seen to be the best-performing method across all the simulation settings. In summary, our approaches work quite well in terms of variable selection.

Table 3.4.3 presents additional variable selection details of the two best-performing approaches: `ELAAN-I` and `ELAAN-H`. For each true main and interaction effect, we report the

[12]MARS is a stepwise forward–backward procedure for building functional ANOVA models.

Table 3.4.1: *Integrated Squared Errors for `ELAAN-I`, `ELAAN-H`, MARS, COSSO, EBM and GAMI-Net.*

$\mathbf{N_{train}}$ / **Model**	**100**	**200**	**400**
MARS	0.239 ± 0.008	0.109 ± 0.003	0.084 ± 0.001
COSSO	0.378 ± 0.005	0.094 ± 0.004	0.043 ± 0.001
EBM	0.274 ± 0.004	0.170 ± 0.002	0.100 ± 0.001
GAMI-Net	0.281 ± 0.007	0.114 ± 0.004	0.063 ± 0.003
`ELAAN-I`	$\mathbf{0.220} \pm 0.013$	$\mathbf{0.077} \pm 0.002$	$\mathbf{0.035} \pm 0.001$
`ELAAN-H`	$\mathbf{0.180} \pm 0.008$	$\mathbf{0.081} \pm 0.004$	$\mathbf{0.038} \pm 0.001$

Table 3.4.2: *Support recovery metric (F1-score) for the main effects and the interaction effects.*

$\mathbf{N_{train}}$	**Model**	**F1 (main)**	**F1 (Interactions)**
100	EBM	57.14 ± 0.00	51.47 ± 1.89
	GAMI-Net	61.77 ± 0.79	22.31 ± 1.27
	`ELAAN-I`	85.35 ± 1.24	40.30 ± 2.78
	`ELAAN-H`	$\mathbf{93.46} \pm 1.14$	$\mathbf{66.61} \pm 2.45$
200	EBM	57.14 ± 0.00	70.43 ± 2.07
	GAMI-Net	59.23 ± 0.41	33.01 ± 1.51
	`ELAAN-I`	90.55 ± 1.02	74.83 ± 1.58
	`ELAAN-H`	$\mathbf{98.52} \pm 0.43$	$\mathbf{82.43} \pm 1.38$
400	EBM	57.14 ± 0.00	83.47 ± 1.86
	GAMI-Net	58.11 ± 0.30	39.24 ± 2.16
	`ELAAN-I`	97.43 ± 0.62	87.34 ± 1.17
	`ELAAN-H`	$\mathbf{99.31} \pm 0.41$	$\mathbf{90.17} \pm 1.23$

Table 3.4.3: *Relative frequency of the true main and interaction effects appearing in the `ELAAN-I` and `ELAAN-H` models.*

$\mathbf{N_{train}}$	**100**		**200**		**400**		**1000**	
Components	`ELAAN-I`	`ELAAN-H`	`ELAAN-I`	`ELAAN-H`	`ELAAN-I`	`ELAAN-H`	`ELAAN-I`	`ELAAN-H`
$x^{(1)}$	87%	**97%**	79%	**100%**	89%	**100%**	90%	**100%**
$x^{(2)}$	67%	**93%**	88%	**100%**	99%	**100%**	100%	100%
$x^{(3)}$	**100%**	95%	**100%**	98%	100%	100%	100%	100%
$x^{(4)}$	99%	99%	**100%**	99%	**100%**	99%	100%	100%
$[x^{(1)}, x^{(2)}]$	69%	**100%**	100%	100%	100%	100%	100%	100%
$[x^{(1)}, x^{(3)}]$	18%	**100%**	74%	**94%**	97%	**99%**	100%	100%
$[x^{(3)}, x^{(4)}]$	0%	**96%**	14%	**33%**	42%	**59%**	**99%**	96%

relative frequency of their appearance in the estimated support across all replications. The results in Tables 3.4.2 and 3.4.3 suggest that `ELAAN-H` outperforms `ELAAN-I` in terms of the variable selection when the true model is hierarchical. In practice, when we do not know if the underlying truth obeys strong hierarchy, `ELAAN-H` tends to result in more compact models, as we demonstrate in the Census application in Section 4.3.

3.4.2 Large-scale setting with correlated features

Next, we evaluate our proposed methods on large-scale synthetic data with $p = 500$ correlated features. We draw $(x_1, ..., x_p)$ from a multivariate normal distribution $\mathcal{N}(0, \mathbf{\Sigma})$, where $\Sigma_{ij} = \sigma^{|i-j|}$ with $\sigma \in [0, 1]$. We generate the responses as $y = f^*(\mathbf{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.25)$ and

$$\begin{aligned}
f^*(\mathbf{x}) = {} & h_1(x_{26}) + h_2(x_{76}) + h_3(x_{126}) + h_4(x_{176}) + h_5(x_{226}) + h_6(x_{276}) \\
& + h_1(x_{326}) + h_2(x_{376}) + h_3(x_{426}) + h_4(x_{476}) \\
& + h_1(x_{26})\, h_2(x_{76}) + h_1(x_{26})\, h_3(x_{126}) + h_4(0.5(x_{126} + x_{176})) + h_4(x_{176})\, h_5(x_{226}) \\
& + h_4(x_{176})\, h_6(x_{276}) + h_5(x_{326}x_{376}) + h_6(x_{426}x_{476}) + h_4(x_{276}x_{476}),
\end{aligned}$$

with $h_1(t) = 0.5t$, $h_2(t) = 1.25\sin(t)$, $h_3(t) = 0.3\exp(t)$, $h_4(t) = 0.5t^2$, $h_5(t) = 0.9\cos(t)$, and $h_6(t) = 1/(1 + \exp(-t))$.

Table 3.4.4: *Integrated Squared Errors and support recovery metrics for EBM, GAMI-Net, `ELAAN-I` and `ELAAN-H` on large-scale synthetic data with different correlation strengths (σ).*

		Prediction Error	**Support Recovery**		
σ	**Model**	**ISE** ($\times 10^{-2}$)	**F1 (features)**	**F1 (main)**	**F1 (Interactions)**
0.1	EBM	379.1 ± 5.3	3.92 ± 0.00	3.92 ± 0.00	26.41 ± 3.26
	GAMI-Net	81.8 ± 7.3	56.57 ± 4.10	92.25 ± 2.44	11.65 ± 1.78
	`ELAAN-I`	8.9 ± 0.9	$\mathbf{98.95} \pm 0.69$	67.75 ± 4.77	$\mathbf{95.17} \pm 1.17$
	`ELAAN-H`	$\mathbf{7.4} \pm 0.8$	98.72 ± 0.59	$\mathbf{98.72} \pm 0.59$	93.78 ± 1.65
0.3	EBM	381.4 ± 4.24	3.92 ± 0.00	3.92 ± 0.00	28.26 ± 3.07
	GAMI-Net	83.2 ± 7.2	55.20 ± 4.52	94.20 ± 2.04	12.00 ± 2.29
	`ELAAN-I`	11.4 ± 1.6	$\mathbf{99.24} \pm 0.35$	65.90 ± 5.07	$\mathbf{95.10} \pm 1.08$
	`ELAAN-H`	$\mathbf{10.0} \pm 1.6$	98.16 ± 1.80	$\mathbf{98.16} \pm 1.80$	93.48 ± 2.41
0.5	EBM	386.4 ± 5.9	3.92 ± 0.00	3.92 ± 0.00	26.00 ± 3.40
	GAMI-Net	77.9 ± 6.2	58.06 ± 4.15	93.92 ± 2.11	13.24 ± 2.04
	`ELAAN-I`	12.4 ± 1.7	$\mathbf{100.00} \pm 0.00$	60.04 ± 5.75	$\mathbf{95.25} \pm 1.00$
	`ELAAN-H`	$\mathbf{10.6} \pm 1.8$	98.87 ± 0.54	$\mathbf{98.87} \pm 0.54$	94.32 ± 1.05
0.7	EBM	375.7 ± 3.3	3.92 ± 0.00	3.92 ± 0.00	17.20 ± 2.06
	GAMI-Net	78.4 ± 7.0	62.42 ± 4.18	94.80 ± 1.97	14.87 ± 2.67
	`ELAAN-I`	$\mathbf{9.3} \pm 0.8$	$\mathbf{99.43} \pm 0.31$	77.38 ± 4.71	$\mathbf{97.12} \pm 0.98$
	`ELAAN-H`	10.6 ± 1.8	97.57 ± 0.68	$\mathbf{97.57} \pm 0.68$	91.31 ± 1.52

We produce 10,000 training observations and evaluate the prediction performance as before, using ISE on a test set of size 10,000. We compare our estimators to EBM [199] and

GAMI-Net [276]. Table 3.4.4 presents the average prediction errors and support recovery metrics over 25 simulation replications, along with the corresponding standard errors. The tuning procedure for `ELAAN-I`, `ELAAN-H`, EBM and GAMI-Net is described in the Appendix. Table 3.4.4 demonstrates that our proposed models consistently and significantly outperform the competitors, exhibiting a 7-11 fold reduction in the prediction error.

3.4.3 Conclusion

We propose a framework to obtain models with good predictive accuracy while simultaneously delivering parsimonious and interpretable statistical models. In the next chapter, we consider a large-scale practical Census Survey Response Prediction with our `ELAAN-I`/`ELAAN-H` models.

3.5 Appendix

3.5.1 Computational details

3.5.1.1 Univariate and bivariate smooth function estimation with splines

We discuss how the univariate and bivariate smooth functions are used to model main effects and interaction effects in Section 3.2.1. The computational details for estimating these building blocks via quadratic optimization are outlined below.

One dimensional nonparametric function estimation. Suppose that we have observations $\{(y_i, u_i)\}_1^n$ corresponding to a univariate response y_i and a univariate predictor $u_i \in [0, 1]$. If the underlying relationship between y and u can be modeled via a twice continuously differentiable (i.e., smooth) function $m(u)$, we can estimate $m(u)$ as a function that minimizes the following objective: $\sum_{i=1}^n (y_i - m(u_i))^2 + \lambda \int (m''(u))^2 du$. This functional optimization problem can be reduced to a finite dimensional quadratic program by using a suitable basis representation for $m(u)$. For example, if all of the u_i's are distinct and $b_1(u), \ldots, b_n(u)$ are the basis functions [for example, cubic splines with knots at the observations u_i, 95, 256], then we can write $m(u) = \sum_{i=1}^{n} \gamma_i b_i(u)$. In this case, $\int (m''(u))^2 du = \boldsymbol{\gamma}^\top \mathbf{D} \boldsymbol{\gamma}$, where $\mathbf{D}$ is a positive semidefinite matrix with the (i, l)-th entry given by $\int b_i''(u) b_l''(u) du$.

Instead of using an n-dimensional basis representation for $m(u)$, which is computationally expensive and perhaps statistically redundant, one can choose a smaller number of basis functions, such as $K = O(n^{1/5})$; this leads to a low-rank representation of $\mathbf{D}$ [84, 162] and hence an improved computational performance. Reducing the number of basis elements from n to K leads to fewer parameters at the expense of a marginal increase in bias [256], which is often negligible in practice.

The evaluations of the function $m(u)$ at the n data points can be stacked together in the form of a vector $\mathbf{B}\boldsymbol{\beta}$, where $\mathbf{B}$ is an $n \times K$ matrix of basis functions evaluated at the observations, and $\boldsymbol{\beta} \in \mathbb{R}^K$ contains the corresponding basis coefficients that need to be estimated from the data. The univariate function fitting problem then reduces to a regularized least squares problem given by $\min_{\boldsymbol{\beta} \in \mathbb{R}^K} \{ \|\boldsymbol{y} - \boldsymbol{B}\boldsymbol{\beta}\|_2^2 + \lambda \boldsymbol{\beta}^T \boldsymbol{A} \boldsymbol{\beta} \}$.

While many choices of spline bases are available [94, 265], we use penalized B-splines for their appealing statistical and computational properties. Following [64], a second-order finite difference penalty on the coefficients of adjacent B-splines serves as a good discrete approximation to the integral of the squared second-order derivative penalty Ω discussed above. The regularized least squares problem takes the form

$$\min_{\boldsymbol{\beta}} \quad \|\boldsymbol{y} - \boldsymbol{B}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{l=1}^{K-2} (\Delta^2 \beta_l)^2, \tag{3.5.1}$$

where K is the number of the B-spline basis functions and $\Delta^2 \beta_l = \beta_l - 2\beta_{l+1} + \beta_{l+2}$ is the second-order finite difference of the coefficients of adjacent B-splines with equidistant knots. We note that the regularization term can be represented in matrix form as $\lambda \|\boldsymbol{D}\boldsymbol{\beta}\|_2^2$, where $\boldsymbol{D} \in \mathbb{R}^{(K-2) \times K}$ is a banded matrix with nonzero entries given by $d_{l,l} = 1$, $d_{l,l+1} = -2$ and $d_{l,l+2} = 1$ for $l \in [K-2]$.

Two dimensional nonparametric function estimation. To model the response as a function of two covariates one can again use reduced rank parameterizations, in the form of multivariate splines [256], thin-plate splines [132], or tensor products of B-splines [65, 265], for example. We use tensor products of B-splines to model the two-dimensional smooth functions of the form $m(u, v)$, with $(u, v) \in [0, 1]^2$ for concreteness. We start from a low-rank B-spline basis representation for the marginal smooth functions as $m_1(u) = \sum_{k=1}^{K} \beta_k b_k(u)$ and $m_2(v) = \sum_{l=1}^{L} \delta_l c_l(v)$, where $\{b_k\}$ and $\{c_l\}$ are B-spline basis functions, and $\{\beta_k\}, \{\delta_l\}$ are unknown basis coefficients. We convert the marginal smooth function m_1 into a smooth function of covariates u and v by allowing the coefficients β_k to vary in a smooth fashion with respect to v. Given that we have an available basis for representing smooth functions of v, we can write $\beta_k(v) = \sum_{l=1}^{L} \delta_{k,l} c_l(v)$ and then arrive at the tensor smooth given by $m(u, v) = \sum_{k=1}^{K} \sum_{l=1}^{L} \delta_{k,l} b_k(u) c_l(v)$. We note that the evaluations of the function $m(u, v)$ at the n data points can be stacked together in a vector written as $\boldsymbol{R\gamma}$. We denote the vectors evaluating the two marginal functional bases, $\{b_k(\cdot)\}$ and $\{c_l(\cdot)\}$, at the i-th data point as $\boldsymbol{B}_i \in \mathbb{R}^K$ and $\boldsymbol{C}_i \in \mathbb{R}^L$, respectively. In matrix notation, the model matrix $\boldsymbol{R} \in \mathbb{R}^{n \times KL}$ can be expressed as $\boldsymbol{R} = (\boldsymbol{B} \otimes \mathbf{1}_L) \odot (\mathbf{1}_K \otimes \boldsymbol{C})$, where operations $\otimes$ and $\odot$ denote Kronecker product and element-wise multiplication respectively. The basis coefficients $\delta_{k,l}$ are appropriately ordered into the vector $\boldsymbol{\gamma} \in \mathbb{R}^{KL}$ via the vectorization operation $\boldsymbol{\gamma} = \mathrm{vec}(\boldsymbol{\delta}) = [\delta_{1,1}, \cdots, \delta_{K,1}, \delta_{1,2}, \cdots, \delta_{K,2}, \cdots, \delta_{1,L}, \cdots, \delta_{K,L}]^\top$, where $\boldsymbol{\delta} = [\delta_{k,l}]_{k \in [K], l \in [L]}$ stores the basis coefficients in matrix form.

The smooth 2D estimate can be obtained by making use of tensor products and a discretized version of the smoothness penalty [65], given by:

$$\min_{\boldsymbol{\gamma}} \|\boldsymbol{y} - \boldsymbol{R\gamma}\|_2^2 + \lambda_b \sum_{k=1}^{K} \sum_{l=1}^{L-2} (\Delta_{(1)}^2 \delta_{k,l})^2 + \lambda_c \sum_{l=1}^{L} \sum_{k=1}^{K-2} (\Delta_{(2)}^2 \delta_{k,l})^2, \tag{3.5.2}$$

where $\Delta_{(1)}^2 \delta_{k,l} = \delta_{k,l} - 2\delta_{k,l+1} + \delta_{k,l+2}$ and $\Delta_{(2)}^2 \delta_{k,l} = \delta_{k,l} - 2\delta_{k+1,l} + \delta_{k+2,l}$. The regularization terms above can be compactly represented as quadratic forms in $\boldsymbol{\gamma}$ as follows:

$$\min_{\boldsymbol{\gamma}} \|\boldsymbol{y} - \boldsymbol{R}\boldsymbol{\gamma}\|_2^2 + \lambda_b \boldsymbol{\gamma}^T \boldsymbol{P}_b \boldsymbol{\gamma} + \lambda_c \boldsymbol{\gamma}^T \boldsymbol{P}_c \boldsymbol{\gamma}, \tag{3.5.3}$$

where $\boldsymbol{P}_b = (\boldsymbol{D}^T\boldsymbol{D}) \otimes \boldsymbol{I}_L$ and $\boldsymbol{P}_c = \boldsymbol{I}_K \otimes (\boldsymbol{D}^T\boldsymbol{D})$. The regularizer ensures that the coefficients in the same row (or column) of $\boldsymbol{\delta}$ vary regularly, leading to a smooth 2D surface.

3.5.1.2 A finite dimensional quadratic program

Problem (3.2.1) can be written as a finite-dimensional quadratic program. Using splines to model each of the main-effects and the interaction-effects, one can write the main-effects and interaction-effects as a linear combination of suitable bases elements. More specifically, $\mathbf{f}_j = \mathbf{B}_j \boldsymbol{\beta}_j$, where $\mathbf{B}_j \in \mathbb{R}^{n \times K_j}$ is the model matrix and $\boldsymbol{\beta}_j \in \mathbb{R}^{K_j}$ is the vector of coefficients for each main effect component. Similarly, $\mathbf{f}_{j,k} = \mathbf{B}_{j,k}\boldsymbol{\theta}_{j,k}$, where $\mathbf{B}_{j,k} \in \mathbb{R}^{n \times K_{j,k}}$ is the model matrix and $\boldsymbol{\theta}_{j,k} \in \mathbb{R}^{K_{j,k}}$ is the vector of coefficients for each interaction effect component. Here, K_j and $K_{j,k}$ denote the corresponding dimensions of the bases – in our implementation, all the values K_j are taken to be the same and all the $K_{j,k}$ are taken to be the same as well. We use a penalty function to control the smoothness (i.e., integral of the squared second derivative) of the 1D and 2D components. Writing $\boldsymbol{\beta}$ for the vector obtained by stacking together the coefficients $\boldsymbol{\beta}_j$, $j \in [p]$ for the main-effects, and defining the vector $\boldsymbol{\theta}$ for the interaction effects analogously, we express the optimization problem (3.2.1) as follows:

$$\min_{\boldsymbol{\beta},\boldsymbol{\theta}} \frac{1}{n}\Big\|\boldsymbol{y} - [\sum_{j\in[p]} \boldsymbol{B}_j\boldsymbol{\beta}_i + \sum_{j<k} \boldsymbol{B}_{j,k}\boldsymbol{\theta}_{j,k}]\Big\|_2^2 + \lambda_1 [\sum_{j\in[p]} \boldsymbol{\beta}_j^T \boldsymbol{S}_j \boldsymbol{\beta}_j + \sum_{j<k} \boldsymbol{\theta}_{j,k}^T \boldsymbol{S}_{j,k}\boldsymbol{\theta}_{j,k}].$$

Here, $\boldsymbol{S}_j = \boldsymbol{D}_j^T\boldsymbol{D}_j$ and $\boldsymbol{S}_{j,k} = (\boldsymbol{D}_j^T\boldsymbol{D}_j) \otimes \boldsymbol{I}_k + \boldsymbol{I}_j \otimes (\boldsymbol{D}_k^T\boldsymbol{D}_k)$ are the smoothness penalty matrices for the main effects and the interaction components, respectively. For convenience, we use the same smoothness penalty λ_1 for both the main and the interaction effects, though in general they may be taken to be different.

3.5.2 Additional Details for Simulations

3.5.2.1 Definition of F1-score

F1-score is a harmonic mean of precision and recall:

$$\text{F1-score} = 2 * \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{3.5.4}$$

where

$$\text{Precision} = \frac{\#\text{ of True Positives}}{\#\text{ of True Positives} + \#\text{ of False Positives}},$$
$$\text{Recall} = \frac{\#\text{ of True Positives}}{\#\text{ of True Positives} + \#\text{ of False Negatives}}$$

For example, if we have 10 main effects and the true support is given by $\{1,1,1,1,0,0,0,0,0,0\}$ and the recovered support is $\{1,1,1,0,0,1,1,0,0,0\}$, the F1-score is 66.67%.

Table 3.5.1: *Average integrated squared error for different configurations of basis elements. "No" in Smoothing column indicates $\lambda_1 = 0$. "Yes" in Smoothing column indicates λ_1 is tuned via cross-validation. Note that for all cases λ_2 is tuned via cross-validation.*

				$\mathbf{N_{train}}$	**100**	**200**	**400**	**1000**
Model	α	**Smoothing**	$\mathbf{K_i}$	$\mathbf{K_{ij}}$				
`ELAAN-I`	1.0	Yes	20	8x8	0.269 (0.014)	0.100 (0.004)	0.0444 (0.0016)	0.0161 (0.0005)
		Yes	10	5x5	0.193 (0.012)	0.075 (0.002)	0.0412 (0.0016)	0.0135 (0.0004)
		Yes	5	3x3	0.162 (0.005)	0.088 (0.002)	0.0579 (0.0007)	0.0445 (0.0003)
		No	5	3x3	0.177 (0.005)	0.100 (0.003)	0.0609 (0.0008)	0.0463 (0.0004)
	1.5	Yes	20	8x8	0.270 (0.013)	0.116 (0.006)	0.0405 (0.0013)	0.0157 (0.0004)
		Yes	10	5x5	0.220 (0.013)	0.077 (0.002)	0.0353 (0.0011)	0.0135 (0.0004)
		Yes	5	3x3	0.174 (0.006)	0.087 (0.002)	0.0577 (0.0005)	0.0448 (0.0003)
		No	5	3x3	0.188 (0.007)	0.095 (0.002)	0.0603 (0.0007)	0.0455 (0.0003)

3.5.2.2 Effect of number of basis functions

We study the effect of number of basis functions on the model performance. We consider 3 different configurations: $\{(K_i = 5, K_{ij} = 3\times3), (K_i = 10, K_{ij} = 5\times5), (K_i = 20, K_{ij} = 8\times8)\}$. The results are reported in Table 3.5.1. As expected, we observe that the performance generally degrades when the number of basis elements is too small (underfitting) or too large (overfitting). When the number of bases elements is large, it is important to use smoothing for regularization.

3.5.2.3 Tuning Details

Tuning Details for the Simulation in Section 3.4.1. We use 100 replications for each simulation setting, i.e., settings with different training set sizes, error distributions etc. For each replication, we select the best hyperparameters via 5-fold cross-validation on the training set. We use mean squared error as the tuning criterion. Next, we run the model on the full training set with the best hyperparameters to compute ISE performance on the test set.

For EBM and GAMI-Net, we tune the number of interactions in the range $[1, 45]$ for 50 trials. For GAMI-Net, we used 32 batch-size, 0.0001 learning rate, 500 epochs (for each stage) and 0.0 loss threshold.

For `ELAAN-I` and `ELAAN-H`, we used 10 knots for the main effects and 5 knots in each coordinate for the interaction effects (leading to $5 \times 5 = 25$ knots). For `ELAAN-I`, we tuned $\lambda_1 \in \{10^{-6}, \cdots, 10^{-2}\}$ and $\lambda_2 \in \{10^{-5}, \cdots, 10^{-1}\}$ with warm-starts. For `ELAAN-H`, we tuned $\lambda_1 \in \{10^{-6}, \cdots, 10^{-2}\}$ and $\lambda_2 \in \{10^{-4}, \cdots, 1\}$, $\tau \in \{0.01, \cdots, 1\}$ with warm-starts.

Tuning Details for the Simulation in Section 3.4.2. We use 25 replications for each simulation setting. For each replication, we select the best hyperparameters via validation tuning, using a validation set of size 2,000 with the mean squared error as the tuning criterion. Next, we run the model on the full training set with the best hyperparameters to compute ISE performance on the test set.

For EBM, we tune the number of interactions in the range $[1, 100]$. For GAMI-Net, we tuned the model with number of interactions in the set $\{5, 50\}$. For GAMI-Net, we used 200 batch-size, 0.0001 learning rate, 500 epochs (for each stage) and 0.0 loss threshold.

For `ELAAN-I` and `ELAAN-H`, we used 10 knots for the main effects and 6 knots in each coordinate for the interaction effects (leading to $6 \times 6 = 36$ knots). For ELAAN-I, we tuned $\lambda_1 \in \{10^{-7}, \cdots, 10^{-3}\}$ and $\lambda_2 \in \{10^{-4}, \cdots, 10^{0}\}$ with warm-starts. The ℓ_0 regularization path was terminated when the number of interactions reached 50. For ELAAN-H, we fix the smoothing parameter λ_1 to the optimal value available from ELAAN-I and tuned $\lambda_2 \in \{10^{-2}, \cdots, 10^{2}\}$ and $\tau \in \{0.01, \cdots, 1\}$ with warm-starts. For ELAAN-I and ELAAN-H, we also considered three choices for α in the set $\{1.0, 1.5, 2.0\}$.

For ELAAN-H, our algorithm, outlined in 3.3.1, approximately solves the problem under hierarchy with ℓ_0 by solving the convex relaxation for computational reasons. From variable section perspective, especially in the correlated setting considered in this simulation, the solution to the convex relaxation can overestimate the number of effects in the model e.g., the number of interactions. Hence, it can be beneficial to consider a solution along the regularization path with smaller number of selected components, but lies within a standard error of the best solution. The numbers reported for prediction error and support recovery for ELAAN-H in the Table 3.4.4 correspond to this choice of solution. This practice is common when using convex relaxations such as lasso in linear models.

Chapter 4

Additive Models and Structured Interactions: A Large-Scale Case Study for Census Survey Response Prediction

4.1 Introduction

Sample surveys and censuses are primary data sources in social science studies. However, low and often unpredictable response rates in surveys remain a continual source of concern [68, 236, 246] – see Figure 4.1.1 for an illustration on the American Community Survey. [246] discuss a multitude of factors that make parts of the population hard-to-survey – such factors are often used to improve sampling strategies, questionnaire designs, recruitment methods, and the language in which the interview is conducted, among others. [68] emphasize the usefulness of having an indicator for hard-to-survey areas to guide targeted surveying (including oversampling), staff recruitment strategies, and targeted follow-ups for non-responders. For major campaigns such as the decennial US Census, this approach can help guide resource allocation for advertisements and building community partnership activities. Eliciting responses from non-self-responding households through follow-up opera-

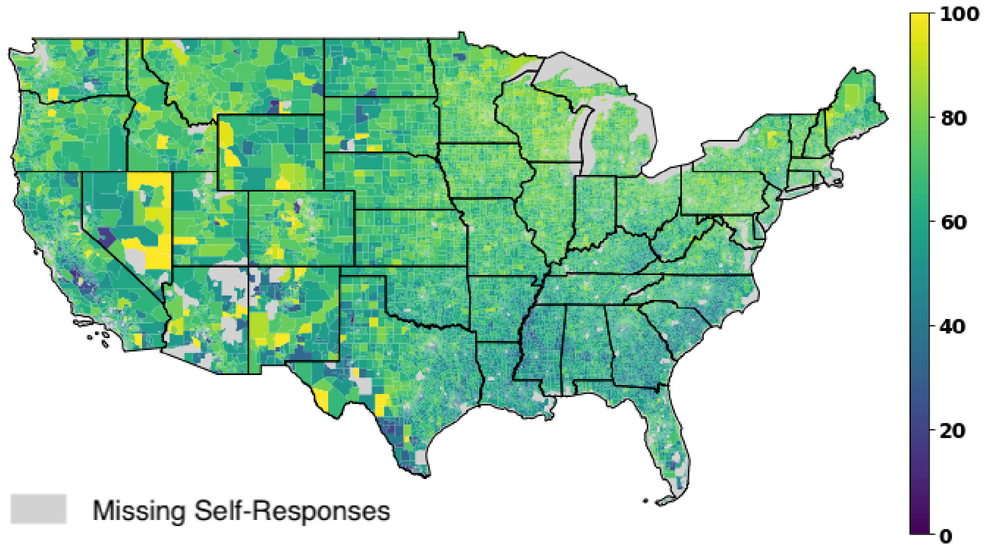


Figure 4.1.1: *The 2013-2017 American Community Survey self-response rates for all tracts in the continental United States. The North in general, and the Upper Midwest and the Northeast in particular, have higher self-response rates than the rest of the country. Tracts with lower self-response rates are visible in many states – in particular, in the South and in the Mountain region.*

.

tions can be very costly and time consuming. The Census Bureau estimates that a single percentage increase in the self-response rate amounts to roughly 85 million dollars saved in personal follow-up costs [12, 248]. Apart from the cost, the quality of proxy enumerations and imputations is typically significantly lower than that of self-responses [192].

Both the UK Office for National Statistics and the US Census Bureau have created measures that help quantify the difficulty in gathering data across different geographic areas. A report by [31] from the US Census Bureau introduced a hard-to-count (HTC) score for identifying difficult to enumerate segments of the population. The HTC score, which is based on 12 carefully chosen covariates (for example, housing variables and socio-demographic/economic indicators) has found its use in the planning for the 2010 Census and many other national surveys. The HTC score, however, has some limitations and was later improved to the low-response-score, or LRS [68]. The LRS plays a key role in the US Census Bureau's Response Outreach Area Mapper (ROAM) application[1] to identify hard-to-survey

[1]https://www.census.gov/library/visualizations/2017/geo/roam.html

areas. The LRS appears to have been partly motivated by the 2012 nationwide competition, organized by the Census Bureau in partnership with Kaggle, on predicting the mail-return rates. This competition aimed to solicit models highly predictive of the decennial census response rates, easily replicable, inherently interpretable, and consistent for use in the field at various levels of geography. Even though the winning models from the competition had good predictive performance, they lacked *reproducibility* and *interpretability*. Some of the winning models included covariates that are not publicly available; in particular, they were not chosen from within the US Census Bureau Planning Database. Additionally, the winning models included covariates that were good predictors but not actionable[2]. Interpretability suffered due to a multitude of factors. First, the winning models were based on complex ensemble methods (for example, random forests and gradient boosting). Second, these methods employed a large number of covariates (the best model had nearly 340), which hurt the model parsimony. After careful analysis, [68] proposed a linear model based on 25 covariates – LRS is the prediction from this model. While this model suffered in terms of predictive accuracy compared to the black-box machine learning methods, it was highly interpretable and led to useful actionable insights. For example, Erdman and Bates highlight three different block-groups (Columbia Heights, Trinidad, Anacostia) in the District of Columbia that have similar LRS predictions but their characteristics vary, indicating a need for different actions to increase self-response. Columbia Heights has a large Hispanic population, suggesting they could benefit from forms and advertising in Spanish. Anacostia has a stable population (low relocation to the region) and could benefit from community partnerships. On the other hand, Trinidad is characterized as a region in transition with high mobility of the young population. This area could potentially benefit from web-based advertisements for internet-based responses from the tech-savvy younger population.

We aim to predict self-response rates across all Census tracts in the US using operational, socio-economic, and demographic characteristics from the US Census Planning Database. Our

[2]These include covariates such as the nearest neighboring block group return rates and margins of error for various estimates. Models that incorporate such features are not actionable.

goal is to consider interpretable statistical models that deliver strong predictive performance and, thus, can be used to complement the currently used LRS metric. Our hope is that these models lend operational insights into factors influencing survey self-response rates to facilitate the downstream goal of having a cost-effective census with improved coverage.

Applied contributions. Our empirical analysis on the US Census Bureau Planning Database demonstrates the promise of our proposed estimators `ELAAN-I`/`ELAAN-H` (in Chapter 3) as potential tools to predict self-response rates in the US Census Bureau application.

Our models lead to improved prediction of the self-response rates when compared to the linear regression methods discussed in the US Census Bureau report. The improvements are due to *both* the departure from linearity and the presence of nonlinear interactions. Importantly, the prediction accuracy of our flexible models appears to be at par with (or slightly better than) black-box machine learning methods such as neural networks and gradient boosted decision trees, which topped the nationwide Kaggle competition organized by the Census Bureau.

Our framework results in simple models and allows for automated variable selection. Our models are substantially more compact than many competing benchmarks – we use 8-20 times fewer interaction effects than the corresponding linear models, and about half as many covariates as off-the-shelf machine learning methods (e.g, gradient boosting, feedforward neural networks, or explainable boosting machines) or the sparse *linear* models with interaction effects. Our models also use a fewer number of covariates (by about 30%) compared to sparse nonparametric additive models (without interactions). We use our models to gather useful operational insights into the factors that influence response rates in surveys across different segments of the population. In particular, our models automatically identify interactions between many of the key factors that have been used in prior publications from the Census Bureau [12] to derive meaningful clustering of the population. These interaction effects appear to result in improved predictions of survey response rates and help complement the

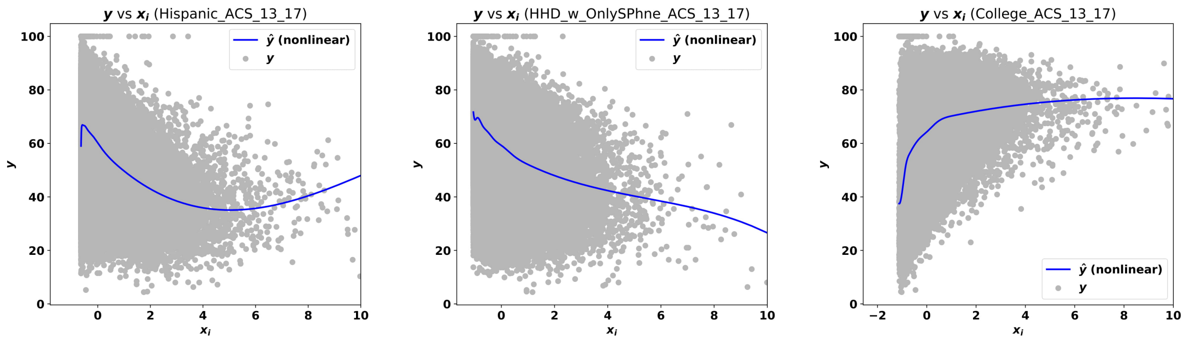


Figure 4.2.1: *Panels [Left]-[Right] illustrate marginal nonparametric fits for the self-response rate output variable versus three covariates. Each marginal fit, displayed on a scatter plot with a solid blue line, clearly suggests a nonlinear relationship of the output vs the individual covariate (we note that the covariates are standardized.) The x-axis corresponds to: [Left] "Persons of Hispanic origin in the ACS"; [Middle] "Number of households that have only a smartphone and no other computing device"; [Right] "Persons 25 years and over with college degree or higher in the ACS".*

existing studies by the Census Bureau [12, 142] on understanding different segments of the population. In summary, our work appears to address many of the challenges of the US Census application by delivering parsimonious models with good predictive performance.

4.2 Additional Motivation for Additive Models with Interactions

Before we dive into the large-scale study in the next section, we briefly summarize some insights from data that motivate the relevance of additive models with interactions with smooth non-parametric function under sparsity constraints.

Motivation for smooth functions. We first consider marginal behavior of various covariates from US Census Planning Database Tract on the self-response rate. For illustration, Figure 4.2.1 shows that the marginal fits for the self-response rate appear to be well-approximated by smooth nonlinear functions. This motivates potential for considering additive models with smooth coordinate functions.

The need for interactions. Exploratory analysis on US Census Planning Database Tract dataset suggests that interaction effects across features can lead to improved predictions of the survey self-response rate. For example, there is interaction between the percentages of "People who do not speak English well" and "Renters" in the area. In areas with a relatively high concentration of poor English speakers (e.g, $\geq 5.4\%$), the self-response rate decreases on an average by 0.33% for a unit increase in the percentage of renters. On the other hand, when the concentration of poor English speakers is relatively low (e.g, $\leq 1.6\%$), the self-response rate decreases at the rate of 0.21%. Similarly, there is a strong interaction effect between covariates "Single-unit households" and "Household moved in 2010 or later" in terms of predicting the low self-response rate. Indeed, Erdman and Bates [68] note the importance of incorporating interaction effects when predicting the self-response rate, although their paper does not pursue statistical modeling involving interactions.

4.2.0.1 Sparse pairwise interactions

For motivation, we consider Figure 4.2.2, which presents our findings on a 2019 US Census Bureau Planning Database dataset with 40 covariates[3]. These features include covariates used for the low-response-score [67], important covariates highlighted in Appendix C of the 2019 US Census Bureau Planning Database Documentation [251], plus some additional covariates capturing internet penetration and urbanization. Specifically, we observe that:

- `ELAAN-I` with *nonlinear* main-effects and interactions[4] results in a substantially more compact model than a *linear* model with interactions.
- `ELAAN-I` leads to better test predictions compared to its linear model counterpart.
- The best (based on validation tuning with prediction error) model with linear main and interaction effects contains 37 main and 555 interaction effects. On the other hand,

[3]These covariates were selected based on discussions with researchers at the US Census Bureau.

[4]This corresponds to an estimate available from (3.2.3). We present the model leading to the best prediction performance on the validation set – see Section 4.3 for details.

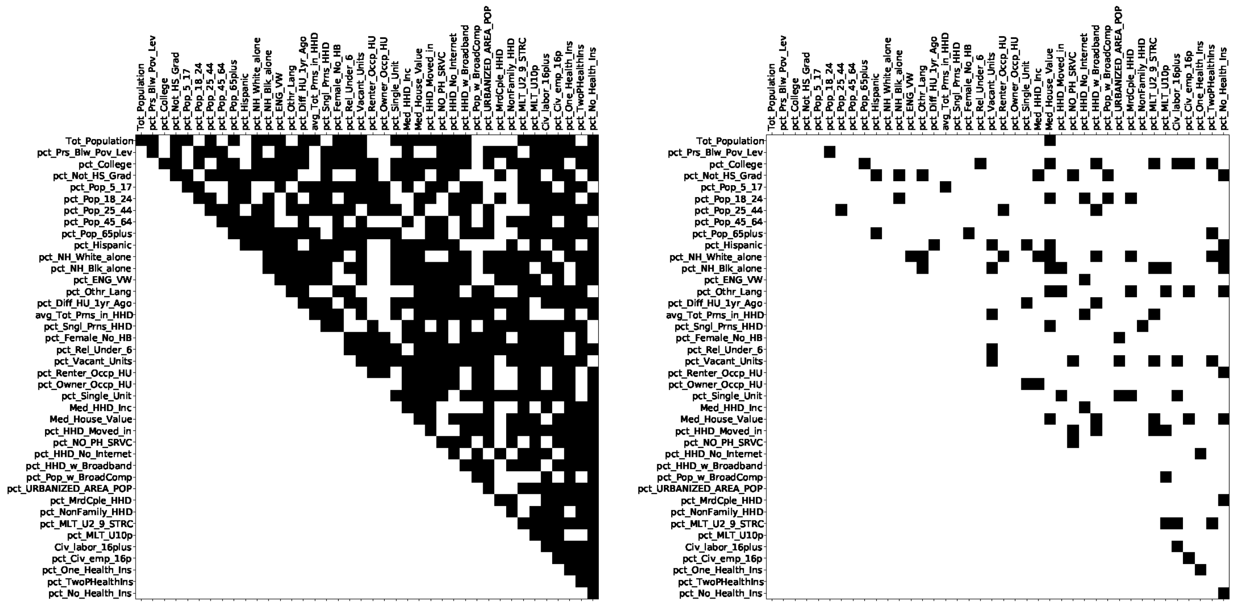


Figure 4.2.2: *Sparsity pattern of the main and interaction effects presented in a $p \times p$ matrix: a black square on the diagonal indicates the presence of a main effect, and an off-diagonal black square indicates the presence of an interaction effect in the joint model. [Left] Panel illustrates the sparsity pattern of a Lasso model with main and interaction effects. There are 37 main and 555 interaction effects in the optimal model. [Right] Panel illustrates the sparsity pattern of a nonlinear AM with main and interaction effects, i.e., model* (3.2.3)*. There are 8 main and 92 interaction effects in the optimal model. Model* (3.2.3) *has prediction performance similar to the Lasso model, with only 3 main and 33 interaction effects. Nonlinear models lead to much more compact and hence, easier to interpret models compared to linear models. Both models were trained on a 2019 US Census Bureau Planning Database dataset (predicting the tract-level self-response rate) with* $p = 40$ *covariates and* $74,000$ *observations.*

`ELAAN-I` selects a total of 36 nonlinear main and interaction effects to obtain a similar test prediction performance. `ELAAN-I` also uses fewer covariates compared to the linear model counterpart.

The above observations suggest that the nonlinear model with sparse interactions has advantages over its linear model counterpart for predicting survey self-response rates. As discussed in Section 4.3, the above findings generally carry over to the expanded dataset with a larger number of features.

Our approach scales to the problem-size of the Census application, with $n \approx 10^5$ and $p \approx 300$, which poses formidable computational challenges due to the presence of approximately

$50,000$ interaction effects[5].

4.3 Predicting the Census Survey Self-Response Rate

We now present our findings on the large-scale Census application. As mentioned in Section 4.1, obtaining interpretable models with good predictive capabilities is important in this application – such models can inform the planning of outreach campaigns and guide stakeholders in deciding the allocation of spending for different communications channels (for example, TV, radio, digital), advertising messages with appropriately tailored content, and the timing of spending during the campaigns [142]. Due to their opaque nature, predictive models that topped the nationwide competition were not actionable for this application. On the other hand, the linear models that drive the LRS have limited predictive power. We explore how our proposed approach balances simplicity and good prediction performance.

The data we use for this study is publicly available in the US Census Planning Database, which provides a range of demographic, socioeconomic, housing, and Census operational data [251]. The data in the Planning Database includes covariates from the 2010 Census and the 2013-2017 American Community Survey (ACS), aggregated at both the Census tract level and the block level. We use tract-level data, with approximately $74,000$ observations and 500 covariates. The response is the ACS self-response rate. We exclude the following covariates from our model: spatial covariates ("State", "County", "Tract", "Flag", "AIAN Land") and variables that serve as a proxy to the response (for example, "Low response score", "Number of housing units that returned first forms", "Replacement forms" or "Bilingual forms" in Census 2010). We also remove the margin of error variables corresponding to the ACS. After excluding these variables, we are left with $p = 295$ covariates[6].

[5]Note that, using 25 knots for every component, this leads to estimating around 1.25 million basis coefficients.

[6]To clarify, the results presented in this section are based on these 295 features, though our approach scales to $p = 500$ features.

4.3.1 Experimental setup

We randomly split the data into 58K for training, 7.2K for validation (used for selecting tuning parameters), and 7.2K observations for testing. We repeat this procedure 20 times with different random splits of the data and report the average numbers on the test sets. The features were standardized to have zero mean and unit variances. We use the squared ℓ_2 loss for training and evaluate the performance of the models in terms of root mean square (RMSE). We also study the variables selected by our algorithm: for example, the number of features retained and the associated interpretations they offer.

As noted in [68], the current approach for predicting the self-response rates–i.e., the LRS–is based on a linear regression model with around 25 *hand-selected* features. Erdman and Bates also use tailored variable transformations on some features to incorporate nonlinear effects. When the number of features, both main and pairwise interaction effects, increase, it is desirable to have an automated procedure such as the one we propose. As we use nonparametric models, which seek to learn nonlinearities in the main and interaction effects, manual feature engineering may not be necessary.

Benchmark Methods. We evaluate and compare our models against existing linear and nonparametric approaches. We consider the following linear models with main effects only (i.e., without interactions): **(i)** Ridge regression; **(ii)** Lasso regression [95]; **(iii)** ℓ_0-penalized regression with additional ridge regularization (denoted as $\ell_0 - \ell_2$), implemented via `L0Learn` [99]. In addition, we consider the following linear models with interaction effects: **(iv)** Lasso with main effects and all pairwise linear interactions, **(v)** `hierScale`: a convex optimization framework for learning sparse main-effects and interactions under a strong hierarchy constraint [98]. For (i), (ii), and (iv), we use Python's `scikit-learn` library [206].

In addition to the competing methods discussed above based on linear models, we consider nonparametric additive models with/without interactions. For AMs with main effects only, we compare with **(vi)** Additive Models with Lasso-based selection, which we fit using Python's scikit-learn library [206]. For additive models with interactions, we consider **(vii)** EBM: A boosting approach that uses trees for main and interaction effects [199]. **(viii)** GAMI-Net: a neural network-based additive model with structured interactions [276]. For EBM, we tune the learning rate in the range $[10^{-4}, 10^{-1}]$ and the number of interactions in the range $[10, 500]$ for 1000 trials. For GAMI-Net, we run the model once with a 200 batch-size, 0.0001 learning rate, 500 interaction effects, 500 epochs (for each stage) and 0.0001 loss threshold. We note that GAMI-Net took 3 days to complete using a machine wth 8-CPUs and 128GB RAM.

In addition to the linear and additive model benchmarks mentioned above, we also compare with state-of-the-art black-box machine learning methods such as **(ix)** Gradient boosted decision trees (GBDT) from XGBoost [45]; and **(x)** Neural networks: multilayer perceptron (MLP). The tuning parameters in both the models (ix) and (x) are selected using the Python hyperparameter optimization package `hyperopt` [20]. GBDT is tuned with respect to the maximum depth $[1 - 10]$, number of estimators $[10 - 200]$, learning rate $[10^{-4}, 1]$. Neural networks are tuned with respect to the number of dense layers $\{2, 3, 4, 5, 6, 7\}$, number of hidden units $\{64, 128, 256, 512\}$, dropout rate $\{0.1, 0.2, 0.3\}$, learning rates $\{0.1, 0.01, 0.001, 0.0001\}$ for Adam optimizer [137], batch sizes $\{64, 128\}$ and epochs $\{25, 50, 75, 100\}$. The number of tuning parameters for all the nonparametric models is capped at 1000.

Proposed Models. Among our proposed estimators, we consider: **(a)** `ELAAN`: ℓ_0-penalized AMs with nonlinear main effects. We also consider the following AMs containing both main and pairwise interaction effects: **(b)** `ELAAN-I`: ℓ_0-penalized AMs with both main and interaction effects, i.e., estimator (3.2.4); and **(c)** `ELAAN-H`: sparse hierarchical interactions, i.e., estimator (3.3.2).

All our algorithms for estimators (a)–(c) are implemented in Python. We use cubic B-splines with 10 knots for the main effects. For the interaction effects, we use tensor spline bases of degree 3 with 5 knots in each coordinate, leading to a total of $5 \times 5 = 25$ knots[7].

[7]We study the effect of number of knots on out-of-sample generalization on synthetic data (see Section

Table 4.3.1: *Comparisons of our methods with several benchmark models as discussed in the text. We display the average test RMSE for the different models, along with the corresponding number of covariates, and number of effects (main and interactions). Best metrics are highlighted in bold. Numbers after $\pm$ provide the standard errors. Asterisk(*) indicates statistical significance (p-value$<$0.05) over the best existing model, using a one-sided paired t-test. Models (ix), (x) are black-box models using higher order interactions, hence #Main and #Interactions are left as '-' (dash).*

Type	Model	RMSE	#Covariates	#Main	#Interactions
Linear Models (LM)	(i) Ridge	6.750 ± 0.013	295 ± 0	295 ± 0	-
	(ii) Lasso	6.741 ± 0.013	226 ± 7	226 ± 7	-
	(iii) L0Learn ($\ell_0 - \ell_2$)	6.752 ± 0.012	145 ± 3	145 ± 3	-
Linear Models with Interactions (LMI)	(iv) LMI with Lasso	6.514 ± 0.019	262 ± 2	75 ± 2	1592 ± 79
	(v) LMI with Strong Hierarchy (hierScale)	6.528 ± 0.018	276 ± 2	276 ± 3	4107 ± 179
Additive Models (AM)	(vi) AM with Lasso	6.548 ± 0.015	285 ± 1	285 ± 1	-
	(a) `ELAAN`	6.566 ± 0.014	184 ± 11	184 ± 11	-
Additive Models with Interactions (AMI)	(vii) EBM	6.475 ± 0.014	295 ± 0	295 ± 0	492 ± 1
	(viii) GAMI-Net	6.573 ± 0.015	246 ± 5	224 ± 4	150 ± 26
	(b) `ELAAN-I`	***6.442** ± 0.019	154 ± 8	**33** ± 5	201 ± 19
	(c) `ELAAN-H`	***6.425** ± 0.019	**133** ± 6	133 ± 6	255 ± 13
Nonparametric (Non-interpretable)	(ix) GBDT	6.481 ± 0.016	278 ± 0	-	-
	(x) Neural Networks (MLP)	6.505 ± 0.016	295 ± 0	-	-

Because the problem at hand has 295 covariates and 43,365 possible pairwise interactions, we need to be careful with implementation aspects while generating spline-transformed representations for all the interaction effects, which can be memory intensive.

For `ELAAN` and `ELAAN-I` we perform a tuning procedure with warm-starts over a 2D grid of parameters (λ_1, λ_2) – for details, see Section 3.2.5.3 of the Appendix. For both `ELAAN-I` and `ELAAN-H`, we set $\alpha = 1$ so that the main and the interaction effects have the same ℓ_0-penalty parameter. For `ELAAN-H`, we fix the smoothing parameter λ_1 to the optimal value available from `ELAAN-I`. Parameter λ_2 (as well as parameter τ defined in Section 3.3.1 of the Appendix) is chosen based on validation tuning. For all our methods (`ELAAN`, `ELAAN-I`, `ELAAN-H`), we cap the number of tuning-parameter values at 1000.

4.3.2 Comparing methods: prediction, sparsity and model structure

We discuss the performance of different estimators in terms of prediction error and model parsimony.

3.5.2.2 of the Appendix).

Additive models vs black-box ML methods. Table 4.3.1 reports the prediction errors (RMSE) on the test-set along with the number of nonzero features in the model. Importantly, we observe that the test performances of AMs with sparse interactions (both with and without the hierarchy constraints) are better than that of the best black box predictive ML models (ix), (x) that are based on GBDT, Neural Networks. Our model `ELAAN-H` delivers the best RMSE value and is closely followed by `ELAAN-I`.

`ELAAN-I`/`ELAAN-H` vs state-of-the-art for AMs with sparse interactions. Among the methods we compared, EBM and GAMI-Net were the only methods that could compute AMs with sparse interactions for the Census dataset. Interestingly, the test performance of both `ELAAN-I` and `ELAAN-H` is better than that of EBM and GAMI-Net. Interpretability (i.e., model sparsity and/or hierarchy) is another key factor differentiating the leading prediction methods. Our models select a smaller number of additive components than EBM and GAMI-Net. For example, `ELAAN-I` selects a total of 234 additive components. In contrast, EBM and GAMI-Net select 787 and 374 additive components, respectively. We also observe that our models are almost twice as compact in terms of the number of selected covariates.

While Table 4.3.1 presents a summary of the best models chosen based on the validation set prediction performance, a practitioner may also find it useful to study a family of models prioritizing models with fewer features (perhaps at the cost of a marginal deterioration in predictive performance) – see Section 4.3.3 for further discussion.

We briefly discuss computation times for different methods. EBM takes around one hour to fit a model with 500 pairwise interactions (for a single tuning parameter). Computing a GAMI-Net solution (for one tuning parameter corresponding to 500 interaction effects) takes around 3 days using 8 CPUs and 10 hours using a single V100 Tesla GPU. In contrast, our approaches are much faster: `ELAAN-I` takes on average 1.8 mins to compute a solution for a fixed tuning parameter[8]. When `ELAAN-H` is run on a reduced subset of 800 pairwise

[8]We use warm-start continuation across λ_2-values for a fixed λ_1: it takes approximately 3 hours to compute 100 solutions where the λ_2 regularization path is cut off when number of pairwise interactions reach 500.

interactions, it takes 1.5 hours to compute a path of 100 solutions. These numbers are reported on an 8-CPU 128GB RAM device.

Nonlinear models vs linear models. Table 4.3.1 suggests that nonlinear/nonparametric models have better predictive performance than their linear counterparts. Also, linear models with sparse main and interaction-effects appear to have an edge over sparse linear models without interactions. Another appealing aspect of sparse nonparametric AMs compared to linear models, is in model parsimony – we alluded to this aspect in Figure 4.2.2 while using a reduced number of features. The number of nonzero effects is significantly lower for the nonlinear AMs. For example, ℓ_0-sparse nonparametric AMs with no interactions (i.e., `ELAAN`) can achieve the same level of predictive performance as its linear model counterpart (i.e., Model (iii)) with significantly fewer covariates: 40 versus 145. Similarly, if we compare `hierScale` (i.e., linear main and interaction effects with strong hierarchical restrictions) to its nonparametric counterpart i.e., `ELAAN-H`, the number of main effects (and also, covariates) reduces from 276 to 133, and the number of interaction effects reduces from 4107 to 255. Interestingly, all the covariates selected by the nonparametric AMs with interaction models (with/without strong hierarchy) are contained in the set of covariates selected by the sparse linear models with both main and interaction effects.

Additive nonlinear models: interactions vs no interactions. Table 4.3.1 shows that ELAAN (AMs with no interactions) has more covariates than AMs with interactions, both `ELAAN-I` and `ELAAN-H`. By including interactions that obey the hierarchy principle, we select a smaller number of covariates: `ELAAN-I` has 154 covariates vs `ELAAN-H` has 133. The difference between `ELAAN` vs `ELAAN-I` is possibly because many nonlinear main effects attempt to explain the nonlinear interaction effects. This reduction points to the redundancy of some of the covariates when interaction effects are directly included in the model. The prediction performance of nonlinear AMs seems to improve significantly when pairwise interactions are included in the model.

Hierarchy vs no hierarchy. We observe that ELAAN-H, i.e., additive model with hierarchical interactions, achieves the best out-of-sample RMSE – this improves over ELAAN-I (interactions with no hierarchy). The improvement is statistically significant based on a paired t-test. We observe that ELAAN-I selects fewer effects (33 Main + 201 Interactions) when compared to ELAAN-H (133 Main + 255 Interactions). On the other hand, ELAAN-H obeys the hierarchy constraint and selects fewer covariates than ELAAN-I– this can aid in interpretability and be easier to operationalize from a practical standpoint.

Insights from visualizations. As illustrated by Figure 4.2.1, which displays some marginal nonlinear fits, an appealing aspect of nonlinear AMs is that they naturally allow the practitioner to gather insights into the associations between the response and a feature of interest, for example, by exploring the map $x_j \mapsto f_j(x_j)$. Similarly, the map $(x_j, x_k) \mapsto f_{j,k}(x_j, x_k)$ would shed light into how the interaction of (x_j, x_k) influences the output. Such interpretations are a by-product of our additive model framework; and may not be readily available via black-box ML methods such as Neural Networks and GBDT—see also [172] for related discussions advocating for the interpretability of AMs. These association plots, for example, can help stakeholders identify promising factors influencing self-response scores potentially informing policy decisions (e.g, targeted investments and optimizing operational costs).

To obtain a finer understanding of the performance of our models, we use visualization tools inspired by earlier works from the US Census Bureau [142, 250]. Figure 4.3.1 illustrates the tract self-response rates predicted by ELAAN-I i.e., model (3.2.4) on a map of the United States. Different from Figure 4.1.1, which shows actual data with missing values for some of the tracts[9], Figure 4.3.1 provides predicted self-response rates for all tracts based on our proposed model. Predictive models in general and our models in particular, allow us to forecast the response rates for tracts where gathered data is incomplete. It is worth noting that our regression-based models may be preferable over commonly used nearest neighbor

[9]Some responses are deliberately suppressed by the US Census Bureau for privacy considerations, to limit the disclosure of information about individual respondents and to reduce the number of estimates with unacceptable levels of statistical reliability [249]

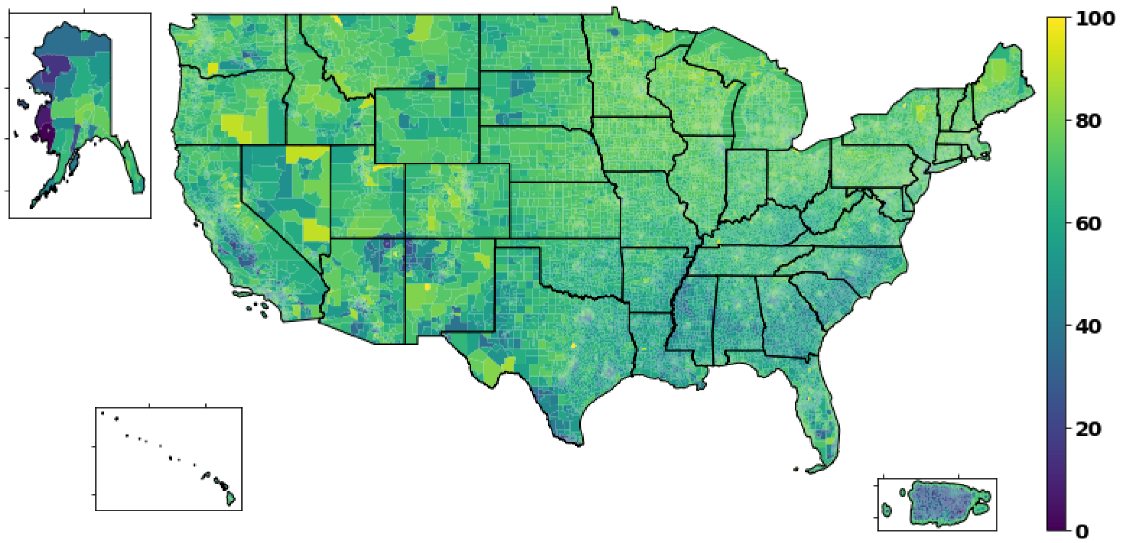


Figure 4.3.1: *Predicted ACS self-response rates for all tracts in the United States.*

.

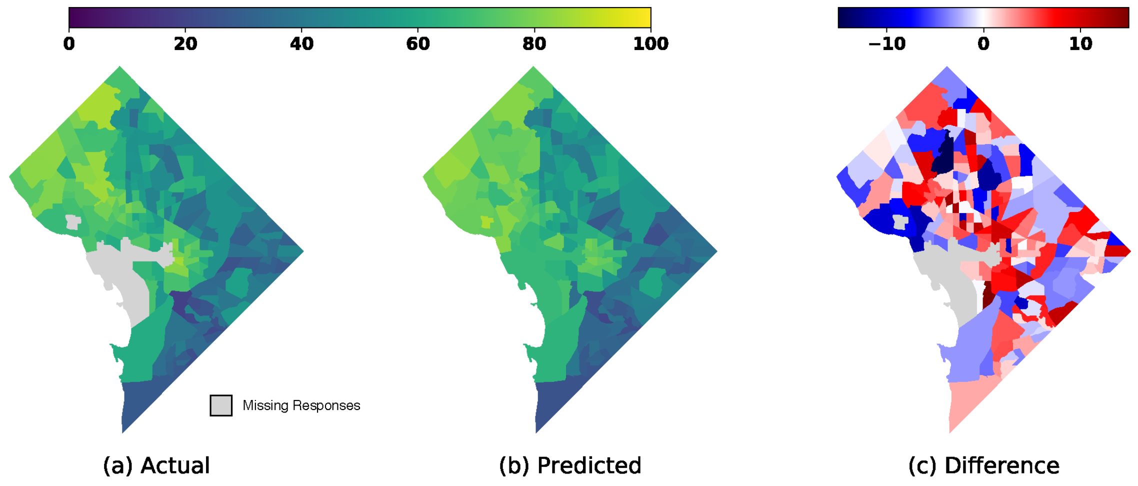


Figure 4.3.2: *ACS self-response rates for all tracts in the District of Columbia. a) Actual ACS self-response rates. b) Predicted ACS self-response rates for AMs with interactions* (3.2.4)*. c) Difference between the actual and predicted self-response rate: difference = actual - predicted.*

type methods: the latter models do not generally offer insights into factors associated with high/low response rates. Nearest neighbor based methods may also be ill-posed for isolated regions such as Alaska, Hawaii and Puerto Rico, each having at least 10% of their tracts censored.

Figure 4.3.2 displays, side by side, the actual and the predicted self-response rates for the tracts in Washington DC, as well as the corresponding differences. Both the actual and the

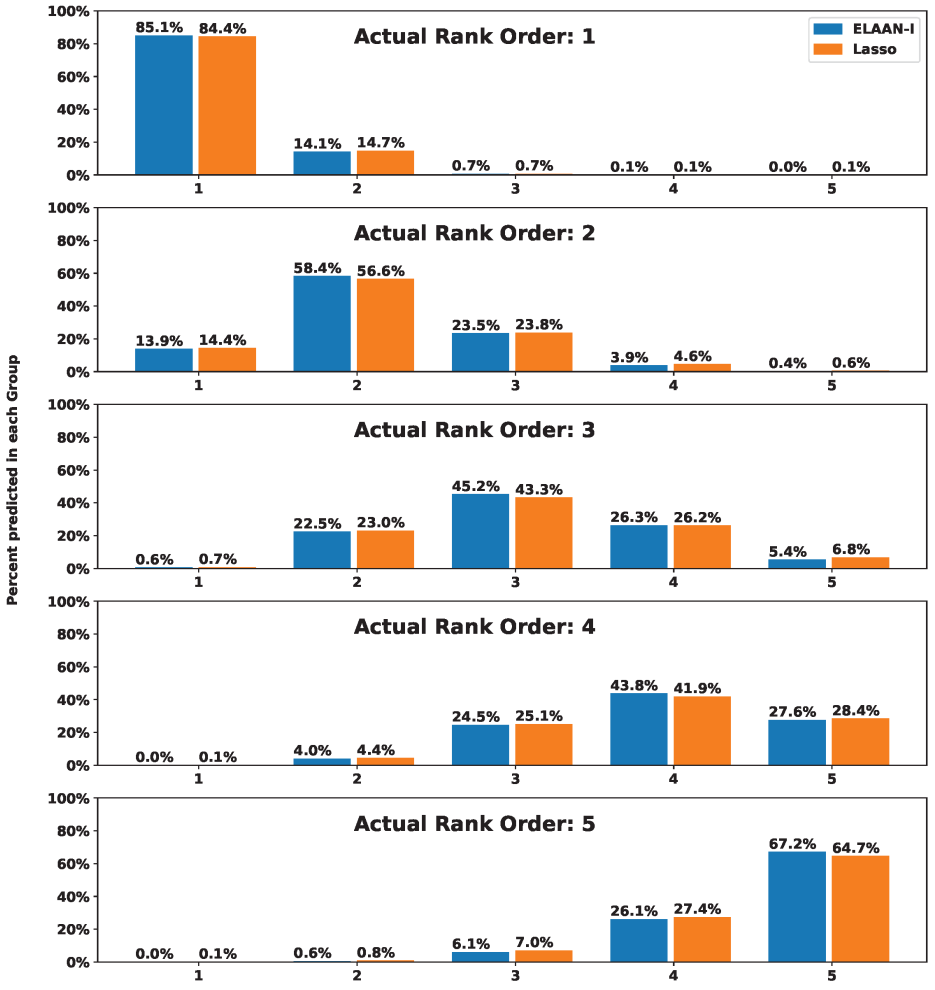


Figure 4.3.3: *Comparison of quintile groups based on Lasso (linear model) [142, 250] and our proposed `ELAAN-I`.*

.

predicted rates are higher than average in most parts of the Northwest and a portion of the Northeast DC, and lower in the Southeast and most of the Northeast DC.

Figure 4.3.3 shows how the sorting of the Census tracts into quintiles [142, 250] of the actual self-response rates compares with the corresponding sorting of the predictions made by: the Lasso (we use an ℓ_1-penalized linear model with main effects and no interactions), and `ELAAN-I`. For example, the top panel in Figure 4.3.3 shows that among the tracts in the first quintile of the actual self-response rates, 85.1% are correctly predicted by `ELAAN-I` to fall in the first quintile. The corresponding proportion for Lasso is 84.4%. The same panel shows that among the tracts in the first quintile of the actual self-response rates, 14.1% are incorrectly predicted to fall in the second quintile; this proportion is smaller than the one for the Lasso (14.7%). Similarly, in the second quintile of the actual self-response rates, a higher proportion (58.4%) is correctly identified by `ELAAN-I` to be in the second quintile; this is again an improvement over the 56.6% for the Lasso regression model. For all 5 quintiles of the actual self-response rates, `ELAAN-I` identifies a higher proportion to be in the correct quintile than Lasso, suggesting that our approach has an edge over the Lasso.

4.3.3 Interpreting important features

We now illustrate how our methodology can guide the practitioner in obtaining a set of features and deriving associated actionable insights into the factors contributing towards low response-rates in surveys. An important aspect of our regularized learning framework is that it provides an automated scheme to identify a collection of models, balancing out the complexity of the model and data-fidelity. To see this, Figure 4.3.4[left panel] plots the number of main and interaction effects against the associated prediction error for model (3.2.4). The plot illustrates that by trading off a little in the predictive performance (in terms of RMSE, which is shown by the red dashed line), we can limit the number of effects at any level, as desired by the practitioner who intends to obtain a more parsimonious representation of the factors impacting survey response rates. Specifically, if we would like to limit the number of main effects to under 20, Figure 4.3.4(b) shows the top 19 main effects in the order they enter the model along the regularization path. The definitions of the variables in Figure 4.3.4(b) are provided in the Appendix. Our investigation reveals that most (though not all) of these variables also appear in the models reported in prior studies by the Census Bureau – see for example [68] and [142]. Interestingly, some of the features discovered by our framework can be interpreted in the context of clusters defined in [12] or mindset solutions in [142], as discussed below.

Using cluster analysis on the 12 covariates from the HTC score, [12] categorize the tracts

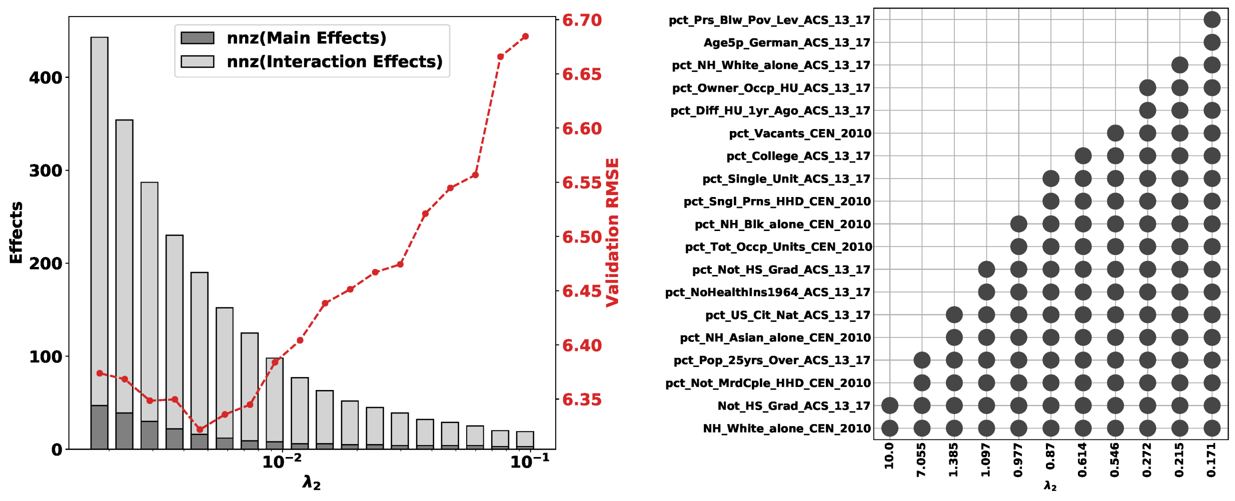


Figure 4.3.4: *Estimates available from our model* **ELAAN-I** *i.e., model* (3.2.4)*: ℓ_0-sparse AMs with main effects and interactions. [Left] number of nonzero effects vs the corresponding RMSE. [Right] Variables corresponding to main effects selected by the model along the regularization path. For visualization purposes, we focus on the top 19 main effects.*

into eight distinct clusters: All Around Average I and II, Economically Disadvantaged I and II, Ethnic Enclave I and II, Single Unattached Mobiles, and Advantaged Homeowners. From the cluster analysis in [12], we see that the reported clusters have a wide variation in response rate (the response rate is not used to identify the clusters). Each cluster has some key characteristics (pertaining to a subset of variables) that seem to drive the response rate. From the description of these clusters in [12], we see that the covariates automatically discovered by our framework, as listed in Figure 4.3.4[right panel], broadly cover the important characteristics upon which the clusters are based. For example, there is high variability across clusters with respect to the occupancy rate and home ownership status, which are captured by covariates “Owner occupied housing units” and “Total occupied units”. Racial diversity also appears to play a role in cluster formation – in particular, the covariates related to ethnic groups characterize the Ethnic Enclave clusters. Similarly, covariates related to poverty, for example, “Persons below poverty line”, lack of education (“No high school diploma”), and lack of health insurance (“No health insurance”), determine the Economically Disadvantaged clusters. Covariates related to high mobility, for example, “Different housing unit in prior year", high education ("College or higher degree holders"), marital status ("Single person households"), and nonspousal occupants ("Unmarried couples"), characterize the Single Unattached Mobiles cluster.

The above discussion suggests that there may be important interaction effects within the variable groups that characterize each cluster. We observe that our AMs with interactions model (3.2.4) automatically selects many of these effects. For instance, our model recovers several interaction effects between important covariates characterizing the Economically Disadvantaged I cluster. [12] notes that this cluster has high percentages of people in poverty, people receiving public assistance, and adults without a high school education. Our model identifies an interaction effect between "No high school graduation" and "No health insurance". The same cluster also has a large African American population and above average number of children under 18. Our model potentially captures this characteristic by selecting an interaction effect between "Non-Hispanic blacks" and "Population with ages 5-17". We also recover an interaction effect between "Non-Hispanic blacks" and "No high school graduation", which seems to be reflective of this cluster. Consider also the Advantaged Homeowners cluster characterized by stable homeowners who are predominantly white and live in single-unit spousal-households. Our model automatically selects a closely related interaction effect between "Non-Hispanic whites" and "Single units". Another example is the Single Unattached Mobiles cluster, which is characterized by young, highly educated population living in multi-units structures with high mobility [12]; the tracts are almost exclusively urban and have an above-average Asian population relative to the national average. Our model selects multiple interaction effects within this group of variables, for example: i) "Moved in 2010 or later" and "Multi-unit structures"; ii) "College or higher degree holders" and "Single person households"; iii) "People speaking Asian languages" and "Multi-unit structures"; iv) "People speaking Asian languages" and "Urban clusters".

Some of the covariates in Figure 4.3.4[right] can also be interpreted in terms of the mindset solutions in [142]. In a mindset solution, survey respondents are categorized using factor analysis into six mindsets based on their predicted self-response patterns and demographic characteristics. These mindsets are determined by the demographic characteristics such as age, race, income, home-ownership, presence of children in the household, marital status, internet use, English proficiency, and country of birth. The six mindsets are: Eager Engagers, Fence Sitters, Individuals with a Confidentiality, Head Nodders, Wary Skeptics and Disconnected Doubters. The covariates can be also interpreted in terms of eight segmented tracts: Responsive Suburbia, Main Street Middle, Country Roads, Downtown Dynamic, Student and Military Communities, Sparse Spaces, Multicultural Mosaic, and Rural Delta and Urban Enclaves – see [142] for additional details.

4.3.4 Conclusion

We study the usefulness of some of our estimators in a large-scale Census Survey Response Prediction. Our estimators e.g., `ELAAN-I` and `ELAAN-H` can give good predictive accuracy while simultaneously delivering parsimonious and interpretable statistical models. We hope that our models will help practitioners identify some key factors that influence survey response rates. Insights gathered from our models may be used for targeted follow-up surveys and allocation of resources for advertisements. We hope that our models will help inform the goal of achieving improved census coverage in a timely fashion, while optimizing the operational risks and costs. Our framework can complement and potentially offer an alternative to the LRS metric that is currently used in the US Census Bureau ROAM application.

4.4 Appendix

4.4.1 Definition of important Census/American Community Survey variables

4.4.1.1 Definition of the variables in Figure 4.2.2

- `Tot_Population_ACS_13_17`: U.S. resident population includes everyone who meets the ACS residence rules in the tract at the time of the ACS interview.
- `pct_Prs_Blw_Pov_Lev_ACS_13_17`: Percentage of the ACS eligible population that are classified as below the poverty level given their total family or household income within the last year.
- `pct_College_ACS_13_17`: The percentage of the ACS population aged 25 years and over that have a college degree or higher.
- `pct_Not_HS_Grad_ACS_13_17`: The percentage of the ACS population aged 25 years and over that are not high school graduates and have not received a diploma or the equivalent.
- `pct_Pop_5_17_ACS_13_17`: The percentage of the ACS population that is between 5 and 17 years old.
- `pct_Pop_18_24_ACS_13_17`: The percentage of the ACS population that is between 18 and 24 years old.
- `pct_Pop_25_44_ACS_13_17`: The percentage of the ACS population that is between 25 and 44 years old.
- `pct_Pop_45_64_ACS_13_17`: The percentage of the ACS population that is between 45 and 64 years old.

- `pct_Pop_65plus_ACS_13_17`: The percentage of the ACS population that is 65 years old or over.

- `pct_Hispanic_ACS_13_17`: The percentage of the ACS population that identify as “Mexican", “Puerto Rican", “Cuban", or “another Hispanic, Latino, or Spanish origin".

- `pct_NH_White_alone_ACS_13_17`: The percentage of the ACS population that indicate no Hispanic origin and their only race as “White" or report entries such as Irish, German, Italian, Lebanese, Arab, Moroccan, or Caucasian.

- `pct_NH_Blk_alone_ACS_13_17`: The percentage of the ACS population that indicate no Hispanic origin and their only race “Black, African American, or Negro" or report entries such as African American, Kenyan, Nigerian, or Haitian.

- `pct_ENG_VW_ACS_13_17`: The percentage of all ACS occupied housing units where no one ages 14 years and over speaks English only or speaks English “very well".

- `pct_Othr_Lang_ACS_13_17`: The percentage of the ACS population aged 5 years and over that speaks a language other than English at home.

- `pct_Diff_HU_1yr_Ago_ACS_13_17`: The percentage of the ACS population aged 1 year and over that moved from another residence in the U.S. or Puerto Rico within the last year.

- `avg_Tot_Prns_in_HHD_ACS_13_17`: The average number of persons per ACS occupied housing unit. This was calculated by dividing the total household population in the ACS by the total number of occupied housing units in the ACS.

- `pct_Sngl_Prns_HHD_ACS_13_17`: The percentage of all ACS occupied housing units where a householder lives alone.

- `pct_Female_No_HB_ACS_13_17`: The percentage of all ACS occupied housing units with a female householder and no spouse of householder present.

- `pct_Rel_Under_6_ACS_13_17`: The percentage of 2010 ACS family-occupied housing units with a related child under 6 years old; same-sex couple households with no relatives of the householder present are not included in the denominator.

- `pct_Vacant_Units_ACS_13_17`: The percentage of all ACS housing units where no one is living regularly at the time of interview; units occupied at the time of interview entirely by persons who are staying two months or less and who have a more permanent residence elsewhere are classified as vacant.

- `pct_Renter_Occp_HU_ACS_13_17`: The percentage of ACS occupied housing units that are not owner occupied, whether they are rented or occupied without payment of rent.

- `pct_Owner_Occp_HU_ACS_13_17`: The percentage of ACS occupied housing units with an owner or co-owner living in it.

- `pct_Single_Unit_ACS_13_17`: The percentage of all ACS housing units that are in a structure that contains only that single unit.

- `Med_HHD_Inc_ACS_13_17`: Median ACS household income for the tract.

- `Med_House_Value_ACS_13_17`: Median of ACS respondents' house value estimates for the tract.

- `pct_HHD_Moved_in_ACS_13_17`: The percentage of all ACS occupied housing units where the householder moved into the current unit in the year 2010 or later.

- `pct_NO_PH_SRVC_ACS_13_17`: The percentage of ACS occupied housing units that do not have a working telephone and available service.

- `pct_HHD_No_Internet_ACS_13_17`: Percentage of ACS households that have no Internet access.

- `pct_HHD_w_Broadband_ACS_13_17`: Percentage of ACS households that have broadband Internet access.

- `pct_Pop_w_BroadComp_ACS_13_17`: Percentage of people that live in households that have both broadband Internet access and a computing device of any kind in the ACS.
- `pct_URBANIZED_AREA_POP_CEN_2010`: The percentage of the 2010 Census total population that lives in a densely settled area containing 50,000 or more people.
- `pct_MrdCple_HHD_ACS_13_17`: The percentage of all ACS occupied housing units where the householder and his or her spouse are listed as members of the same household; does include same sex married couples.
- `pct_NonFamily_HHD_ACS_13_17`: The percentage of all ACS occupied housing units where a householder lives alone or with non relatives only; includes unmarried same-sex couples where no relatives of the householder are present.
- `pct_MLT_U2_9_STRC_ACS_13_17`: The percentage of all ACS housing units that are in a structure that contains two to nine housing units.
- `pct_MLT_U10p_ACS_13_17`: The percentage of all ACS housing units that are in a structure that contains 10 or more housing units.
- `Civ_labor_16plus_ACS_13_17`: Number of civilians ages 16 years and over at the time of the interview that are in the labor force in the ACS.
- `pct_Civ_emp_16plus_ACS_13_17`: The percentage of ACS civilians ages 16 years and over in the labor force that are employed.
- `pct_One_Health_Ins_ACS_13_17`: The percentage of the ACS population that have one type of health insurance coverage, including public or private.
- `pct_TwoPHealthIns_ACS_13_17`: The percentage of the ACS population that have two or more types of health insurance.
- `pct_No_Health_Ins_ACS_13_17`: The percentage of the ACS population that have no health insurance, public or private.

4.4.1.2 Definition of the variables in Figure 4.3.4(b)

- `pct_Prs_Blw_Pov_Lev_ACS_13 _17`: The percentage of the ACS eligible population that are classified as below the poverty level given their total family or household income within the last year, family size, and family composition.

- `Age5p_German_ACS_13_17`: Number of people ages 5 years and over who speak English less than "very well" and speak German at home in the ACS. Examples include Luxembourgian.

- `pct_NH_White_alone_ACS_13_ 17`: The percentage of the ACS population that indicate no Hispanic origin and their only race as “White" or report entries such as Irish, German, Italian, Lebanese, Arab, Moroccan, or Caucasian.

- pct_Owner_Occp_HU_ACS_13 _17: The percentage of ACS occupied housing units with an owner or co-owner living in it.

- `pct_Diff_HU_1yr_Ago_ACS_1 3_17`: The percentage of the ACS population aged 1 year and over that moved from another residence in the U.S. or Puerto Rico within the last year.

- `pct_Vacant_Units_CEN_2010`: The percentage of all 2010 Census housing units that have no regular occupants on Census Day; housing units with its usual occupants temporarily away (such as on vacation, a business trip, or in the hospital) are not considered vacant, but housing units temporarily occupied on Census Day by people who have a usual residence elsewhere are considered vacant.

- `pct_College_ACS_13_17`: The percentage of the ACS population aged 25 years and over that have a college degree or higher.

- `pct_Single_Unit_ACS_13_17`: The percentage of all ACS housing units that are in a structure that contains only that single unit.

- `pct_Sngl_Prns_HHD_Cen_2010`: The percentage of all 2010 Census occupied housing units where a householder lives alone.
- `pct_NH_Blk_alone_CEN_2010`: The percentage of the 2010 Census total population that indicate no Hispanic origin and their only race as “Black, African American, or Negro" or report entries such as African American, Kenyan, Nigerian, or Haitian.
- `pct_Tot_Occp_Units_ACS_13_17`: The percentage of all ACS housing units that are classified as the usual place of residence of the individual or group living in it.
- `pct_Not_HS_Grad_ACS_13_17`: The percentage of the ACS population aged 25 years and over that are not high school graduates and have not received a diploma or the equivalent.
- `pct_NoHealthIns1964_ACS_13_17`: Percentage of people age 19 to 64 with no health insurance in the ACS.
- `pct_US_Cit_Nat_ACS_13_17`: The percentage of the ACS population who are citizens of the United States through naturalization.
- `pct_NH_Asian_alone_Cen_2010`: The percentage of the 2010 Census total population that indicate no Hispanic origin and their only race as “Asian Indian", “Chinese", “Filipino", “Korean", “Japanese", “Vietnamese", or “Other Asian".
- `pct_Pop_25yrs_Over_ACS_13_17`: The percentage of the ACS population who are ages 25 years and over at time of interview.
- `pct_Not_MrdCple_HHD_Cen_2010`: The percentage of all 2010 Census occupied housing units where no spousal relationship is present.
- `Not_HS_Grad_ACS_13_17`: The percentage of the ACS population aged 25 years and over that are not high school graduates and have not received a diploma or the equivalent.

- `NH_White_alone_CEN_2010`: Number of people who indicate no Hispanic origin and their only race as “White” or report entries such as Irish, German, Italian, Lebanese, Arab, Moroccan, or Caucasian in the 2010 Census population.

Chapter 5

Tree Ensembles: Flexible Modeling and Efficient Training

5.1 Introduction

Decision tree ensembles are popular models that have proven successful in various machine learning applications and competitions [46, 68]. Besides their competitive performance, decision trees are appealing in practice because of their interpretability, robustness to outliers, and ease of tuning [92]. Training a decision tree naturally requires solving a combinatorial optimization problem, which can be challenging to scale to large instances. In practice, greedy heuristics are commonly used to get feasible solutions to the combinatorial problem; for example CART [29], C5.0 [214], and OC1 [193]. By building on these heuristics, highly scalable toolkits for learning tree ensembles have been developed, e.g., XGBoost [46] and LightGBM [134]. These toolkits are considered a defacto standard for training tree ensembles and have demonstrated success in various domains.

Despite their success, popular toolkits for learning tree ensembles lack modeling flexibility. For example, these toolkits support a limited set of loss functions, which may not be suitable for the application at hand. Moreover, these toolkits are limited to single task learning. In many modern applications, it is desirable to solve multiple, related machine learning tasks. In such applications, multi-task learning, i.e., learning tasks simultaneously, may be a more appropriate choice than single task learning [42, 51, 86, 143]. If the tasks are sufficiently related, multi-task learning can boost predictive performance by leveraging task relationships during training.

In this chapter, we propose a flexible modeling framework for training tree ensembles that addresses the aforementioned limitations. Specifically, our framework allows for training tree ensembles with any differentiable loss function, enabling the user to seamlessly experiment with different loss functions and select what is suitable for the application. Moreover, our framework equips tree ensembles with the ability to perform multi-task learning. To achieve this flexibility, we build up on differentiable trees [100, 139], which can be trained with first-order (stochastic) gradient methods.

Previously, soft tree ensembles have been predominantly explored for classification tasks with cross-entropy loss. In such tasks, they were found to be more expressive and compact than traditional tree ensembles [100]. However, state-of-the-art toolkits, e.g., TEL [100], are slow as they only support CPU training and are difficult to customize. Our proposed framework goes beyond the latter work on soft trees by supporting a diverse collection of loss functions for classification, regression, Poisson regression, zero-inflated models, overdispersed distributions, and multi-task learning. The framework also offers seamless support for arbitrary loss functions: the user can modify the loss function with a single line. We empirically observe that the ability to customize the loss can lead to a significant reduction in ensemble sizes (up to 20x). We separately investigate end-to-end feature selection in tree ensembles (next chapter). We also propose a custom tensor-based formulation of differentiable tree ensembles, leading to more efficient training on CPUs ($10\times$) and GPUs ($20\times$).

Contributions. Our contributions can be summarized as follows.

1. We propose a flexible framework for training differentiable tree ensembles with seamless support for new loss functions.

2. We introduce a novel, tensor-based formulation for differentiable tree ensembles that allows for efficient training on GPUs. Existing toolkits e.g., TEL [100], only support CPU training.

3. We extend differentiable tree ensembles to multi-task learning settings by introducing a new regularizer that allows for soft parameter sharing across tasks — current popular multi-task tree ensemble toolkits e.g., RF [30], GRF [9] do not allow for soft information sharing.

4. We introduce `FASTEL`[1] — a new toolkit (based on Tensorflow 2.0) for learning differentiable tree ensembles — and perform experiments on a collection of 28 open-source and real-world datasets, demonstrating that our toolkit can lead to 100x more compact ensembles and up to 23% improvement in out-of-sample performance, compared to tree ensembles learnt by popular toolkits such as XGBoost [46].

Organization. We summarize related work in Section 5.2. We then briefly review differentiable trees in Section 5.3.1. In Section 5.3.2, we present a careful tensor-based formulation of the tree ensemble, which allows for efficient training on both CPUs and GPUs. In Section 5.4, we discuss important examples of loss functions supported by our framework. In Section 5.5 , we propose a multitask learning formulation based on differentiable tree ensembles. In Section 5.6, we present an empirical study on a collection of real-world datasets.

5.2 Related Work

Learning binary trees has been traditionally done in three broad ways. The first approach relies on greedy construction and/or optimization via methods such as CART [29], C5.0 [214], OC1 [193], TAO [37]. These methods optimize a criterion at the split nodes based on the samples routed to each of the nodes. The second approach considers probabilistic

[1]https://github.com/ShibalIbrahim/FASTEL

relaxations/decisions at the split nodes and performs end-to-end learning with first order methods [79, 120, 150]. The third approach considers optimal trees with mixed integer formulations and jointly optimizes over all discrete/continuous parameters with MIP solvers [18, 19, 22, 292]. Each of the three approaches have their pros and cons. The first approach is highly scalable because of greedy heuristics. In many cases, the tree construction uses a splitting criterion different from the optimization objective [29] (e.g., gini criterion when performing classification) possibly resulting in sub-optimal performance. The second approach is also scalable but principled pruning in probabilistic trees remains an open research problem. The third approach scales to small datasets: some of the largest instances reported in prior work include number of samples $N \sim 10^4$, features $p \sim 10$ and tree depths $d \sim 4$.

Jointly optimizing over an ensemble of classical decision trees is a hard combinatorial optimization problem [112]. Historically, tree ensembles have been trained with two methods. One approach uses greedy heuristics for individual trees with ensembling done via bagging/boosting. For example, individual trees are trained with CART on bootstrapped samples of the data e.g., random forests (RF) [30] and its variants [9, 80]; or sequentially trained with gradient boosting: Gradient Boosting Decision Trees [92] and efficient variants [46, 134, 213, 231]. Despite the success of ensemble methods, interesting challenges remain: (i) RF tend to under-perform gradient boosting methods such as XGBoost [46]. (ii) The tree ensembles are typically very large, making them complex and difficult to interpret. Recent work by [37, 289] improve RF with local optimization methods such as alternating minimization. However, their implementation is not open-source. (iii) Open-source toolkits for gradient boosting are limited in terms of flexibility. They lack support for multi-task learning, missing responses or customized loss functions. Modifying these toolkits for custom applications often require significant effort, technical expertise and research investment.

The alternative approach for tree ensemble learning extends probabilistic/differentiable trees and performs end-to-end learning [100, 139]. These works build upon the idea of hierarchical mixture of experts introduced by [131] and further developed by [79, 120, 244] for greedy construction of trees. Some of these works [100, 139] propose using differentiable trees as an output layer in a cascaded neural network for combining feature representation learning along with tree ensemble learning for classification. In this chapter, we focus on learning tree (ensembles) with hyperplane splits and constant leaf nodes–this allows us to expand the scope of trees to flexible loss functions, and develop specialized implementations that can be more efficient. One might argue that probabilistic trees are harder to interpret and suffer from slower inference as a sample must follow each root-leaf path, lacking *conditional computation* present in classical decision trees. However, [100] proposed a principled way to get conditional inference in probabilistic trees by introducing a new activation function: this allows for routing samples through small parts of the tree similar to classical decision trees. We refer the reader to Section 5.6.1 for a study on a single tree and highlight that a soft tree with hyperplane splits and conditional inference has similar interpretability as that of a classical tree with hyperplane splits — see Figure 5.6.1. Additionally, a soft tree can lead to smaller optimal depths—see Appendix Section 5.8.1.

End-to-end learning with differentiable tree ensembles appears to have several advantages. (i) Training is easy to set up in public deep learning frameworks, e.g., Tensorflow [1] and PyTorch [204]. Differentiable tree ensembles allow for flexibility in loss functions without the need for specialized algorithms. For example, mixture likelihoods can be easily implemented in Tensorflow Probability [55], which allows for handling zero-inflated data. Similarly, multi-task loss objectives can also be handled. (ii) With a careful implementation, the tree ensemble can be trained efficiently on GPUs — this is not possible with earlier toolkits such as TEL [100]. (iii) Differentiable trees can lead to more expressive and compact ensembles [100]. This can have important implications for interpretability, latency and storage requirements during inference.

5.3 Optimizing Tree Ensembles

In this section, we first introduce background on soft trees. Later, in Section 5.3.2, we discuss the tensor-based formulation to perform efficient training on both CPUs and GPUs. Finally, in Section 5.3.3, we briefly discuss our `FASTEL` toolkit.

We assume a supervised multi-task learning setting, with input space $\mathcal{X} \subseteq \mathbb{R}^p$ and output space $\mathcal{Y} \subseteq \mathbb{R}^k$. We learn a mapping $f : \mathbb{R}^p \to \mathbb{R}^k$, from input space $\mathcal{X}$ to output space $\mathcal{Y}$, where we parameterize function f with a differentiable tree ensemble. We consider a general optimization framework where the learning objective is to minimize any differentiable loss function $g : \mathbb{R}^p \times \mathbb{R}^k \to \mathbb{R}$. The framework can accommodate different loss functions arising in different applications and perform end-to-end learning with tree ensembles.

Notation. For an integer $n \geq 1$, let $[n] := \{1, 2, ..., n\}$. We let 1_m denote the vector in $\mathbb{R}^m$ with all coordinates being 1. For a matrix $\boldsymbol{B} = ((B_{ij})) \in \mathbb{R}^{m \times n}$, let the j-th column be denoted by $\boldsymbol{B}_j := [B_{1j}, B_{2j}, ..., B_{mj}]^T \in \mathbb{R}^m$ for $j \in [n]$. A dot product between two vectors $\boldsymbol{u}, \boldsymbol{v} \in R^m$ is denoted as $\boldsymbol{u} \cdot \boldsymbol{v}$. A dot product between a matrix $\boldsymbol{U} \in R^{m,n}$ and a vector $\boldsymbol{v} \in \mathbb{R}^m$ is denoted as $\boldsymbol{U} \cdot \boldsymbol{v} = \boldsymbol{U}^T v \in \mathbb{R}^n$. A dot product between a tensor $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{p,m,n}$ and a vector $\boldsymbol{v} \in \mathbb{R}^m$ is denoted as $\boldsymbol{\mathcal{U}} \cdot \boldsymbol{v} = \boldsymbol{\mathcal{U}}^T \boldsymbol{v} \in \mathbb{R}^{p,n}$ where the transpose operation of a tensor $\boldsymbol{\mathcal{U}}^T \in \mathbb{R}^{p,n,m}$ permutes the last two dimensions of the tensor.

5.3.1 Preliminaries and Setup

We learn an ensemble of m differentiable trees. Let $\boldsymbol{f}^j$ be the jth tree in the ensemble. For easier exposition, we consider a single-task regression or classification setting—see Section 5.5 for an extension to the multi-task setting. In a regression setting $k = 1$, while in multi-class classification setting $k = C$, where C is the number of classes. For an input feature-vector $\boldsymbol{x} \in \mathbb{R}^p$, we learn an additive model with the output being an average of the outputs of all the trees:

$$\boldsymbol{f}(\boldsymbol{x}) = \frac{1}{m} \sum_{j=1}^{m} \boldsymbol{f}^j(\boldsymbol{x}). \tag{5.3.1}$$

The output, $\boldsymbol{f}(\boldsymbol{x})$, is a vector in $\mathbb{R}^k$ containing raw predictions. For multiclass classification, mapping from raw predictions to $\mathcal{Y}$ is done by applying a softmax function on the vector $\boldsymbol{f}(\boldsymbol{x})$ and returning the class with the highest probability. Next, we introduce the key building block of the approach: differentiable decision tree.

Differentiable decision trees for modelling $\boldsymbol{f}^j$. Classical decision trees perform hard sample routing, i.e., a sample is routed to exactly one child at every splitting node. Hard sample routing introduces discontinuities in the loss function, making trees unamenable to continuous optimization. Therefore, trees are usually built in a greedy fashion. In this section, we first introduce a single soft tree proposed by [131], which is utilized in [27, 79, 120] and extended to soft tree ensembles in [100, 109, 139]. A soft tree is a variant of a decision tree that performs soft routing, where every internal node can route the sample to the left and right simultaneously, with different proportions. This routing mechanism makes soft trees differentiable, so learning can be done using gradient-based methods. Notably, [100] introduced a new activation function for soft trees that allowed for conditional computation while preserving differentiability.

Let us fix some $j \in [m]$ and consider a single tree $\boldsymbol{f}^j$ in the additive model (5.3.1). Recall that $\boldsymbol{f}^j$ takes an input sample and returns an output vector (logit), i.e., $\boldsymbol{f}^j : X \in \mathbb{R}^p \rightarrow \mathbb{R}^k$. Moreover, we assume that $\boldsymbol{f}^j$ is a perfect binary tree with depth d. We use the sets $\mathcal{I}^j$ and $\mathcal{L}^j$ to denote the internal (split) nodes and the leaves of the tree, respectively. For any node $i \in \mathcal{I}^j \cup \mathcal{L}^j$, we define $A^j(i)$ as its set of ancestors and use the notation $\boldsymbol{x} \rightarrow i$ for the event that a sample $\boldsymbol{x} \in \mathbb{R}^p$ reaches i.

Routing. Internal (split) nodes in a differentiable tree perform soft routing, where a sample is routed left and right with different proportions. This soft routing can be viewed as a probabilistic model. Although the sample routing is formulated with a probabilistic model, the final prediction of the tree $\boldsymbol{f}$ is a deterministic function as it assumes an expectation over the leaf predictions. Classical decision trees are modeled with either axis-aligned splits [29, 214] or hyperplane (a.k.a. oblique) splits [193]. Soft trees are based on hyperplane splits, where the routing decisions rely on a linear combination of the features. Particularly, each internal node $i \in \mathcal{I}^j$ is associated with a trainable weight vector $\boldsymbol{w}_i^j \in \mathbb{R}^p$ that defines the node's hyperplane split. Given a sample $\boldsymbol{x} \in \mathbb{R}^p$, the probability that internal node i routes $\boldsymbol{x}$ to the left is defined by $S(\boldsymbol{w}_i^j \cdot \boldsymbol{x})$, where $S : R \rightarrow [0, 1]$ is an activation function. Now we discuss how to model the probability that $\boldsymbol{x}$ reaches a certain leaf l. Let $[l \swarrow i]$ (resp. $[i \searrow l]$) denote the event that leaf l belongs to the left (resp. right) subtree of node $i \in \mathcal{I}^j$. Assuming that the routing decision made at each internal node in the tree is independent of the other nodes, the probability that $\boldsymbol{x}$ reaches l is given by:

$$P^j(\{x \rightarrow l\}) = \prod_{i \in A^j(l)} r_{i,l}^j(\boldsymbol{x}), \tag{5.3.2}$$

where $r_{i,l}^j(\boldsymbol{x})$ is the probability of node i routing $\boldsymbol{x}$ towards the subtree containing leaf l, i.e., $r_{i,l}^j(x) := S(\boldsymbol{w}_i^j \cdot \boldsymbol{x})1[l \swarrow i] \odot (1 - S(\boldsymbol{w}_i^j \cdot \boldsymbol{x}))1[i \searrow l]$. Popular choices for S include logistic function [79, 109, 131, 139, 244] and smooth-step function (for conditional computation as in classical trees with oblique splits) [100]. Next, we define how the root-to-leaf probabilities in (5.3.2) can be used to make the final prediction of the tree.

Prediction. As with classical decision trees, we assume that each leaf stores a weight vector $\boldsymbol{o}_l^j \in R^k$ (learned during training). Note that, during the forward pass, $\boldsymbol{o}_l^j$ is a constant vector, meaning that it is not a function of the input sample(s). For a sample $\boldsymbol{x} \in \mathbb{R}^p$, we define the prediction of the tree as the expected value of the leaf outputs, i.e.,

$$\boldsymbol{f}^j(\boldsymbol{x}) = \sum_{l \in L} P^j(\{\boldsymbol{x} \rightarrow l\}) \boldsymbol{o}_l^j. \tag{5.3.3}$$

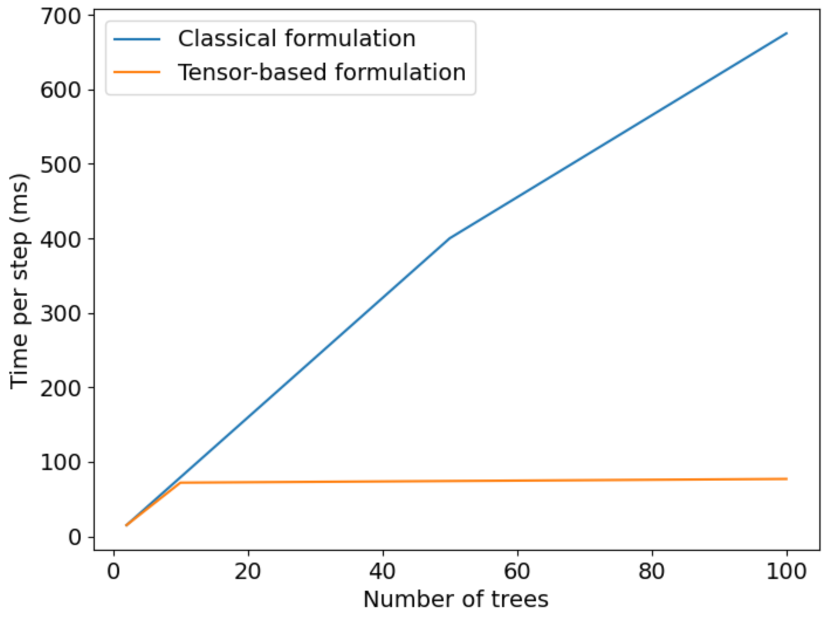


Figure 5.3.1: *Timing comparison of classical formulation against our tensor-based formulation of a tree ensemble. Classical formulation models trees in the ensemble individually as pointed out in Section 5.3.2. Tensor-based formulation with CPU training is up to* 10× *faster than classical formulation. Tensor-based formulation with GPU training leads to an additional 40% improvement, leading to an effective* 20× *gain over classical formulation.*

5.3.2 Efficient Tensor Formulation

Current differentiable tree ensemble proposals and toolkits, for example deep neural decision forests[2] [139] and TEL [100] model trees individually. This leads to slow CPU-training times and makes these implementations hard to vectorize for fast GPU training. In fact, TEL [100] does not support GPU training. We propose a tensor-based formulation of a tree ensemble that parallelizes routing decisions in nodes across the trees in the ensemble.

[2]https://keras.io/examples/structured_data/deep_neural_decision_forests/

This can lead to 10x faster CPU training times if the ensemble sizes are large e.g., 100. Additionally, the tensor-based formulation is GPU-friendly, which provides an additional 40% faster training times. See Figure 5.3.1 for a timing comparison on CPU training without/with tensor formulation. Next, we outline the tensor-based formulation.

We propose to model the internal nodes in the trees across the ensemble jointly as a "supernodes". In particular, an internal node $i \in \mathcal{I}^j$ at depth d in all trees can be condensed together into a supernode $i \in \mathcal{I}$. We define a learnable weight matrix $\boldsymbol{W}_i \in \mathbb{R}^{p,m}$, where each j-th column of the weight matrix contains the learnable weight vector $\boldsymbol{w}_i^j$ of the original j-th tree in the ensemble. Similarly, the leaf nodes are defined to store a learnable weight matrix $\boldsymbol{O}_l \in \mathbb{R}^{m,k}$, where each j-th row contains the learnable weight vector $\boldsymbol{o}_l^j$ in the original j-th tree in the ensemble. The prediction of the tree with supernodes can be written as

$$\boldsymbol{f}(\boldsymbol{x}) = \left(\sum_{l \in L} \boldsymbol{O}_l \odot \prod_{i \in A(l)} \boldsymbol{R}_{i,l} \right) \cdot \frac{1}{m} \mathbf{1}_m \tag{5.3.4}$$

where $\odot$ denotes the element-wise product, $\boldsymbol{R}_{i,l} = S(\boldsymbol{W}_i \cdot \boldsymbol{x}) 1[l \swarrow i] \odot (1 - S(\boldsymbol{W}_i \cdot \boldsymbol{x})) 1[i \searrow l] \in \mathbb{R}^{m,1}$ and the activation function S is applied element-wise. This formulation of tree ensembles via supernodes allows for sharing of information across tasks via tensor formulation in multi-task learning — see Section 5.5 for more details.

5.3.3 Toolkit

Our `FASTEL` toolkit is built in Tensorflow (TF) 2.0 and integrates with Tensorflow-Probability. The toolkit allows the user to write a custom loss function, and TF provides automatic differentiation. Popular packages, such as XGBoost, require users to provide first/second order derivatives. In addition to writing a custom loss, the user can select from a wide range of predefined loss and likelihood functions from Tensorflow-Probability. By relying on TF in the backend, our toolkit can easily exploit distributed computing. It can also run on multiple CPUs or GPUs, and on different platforms, including mobile platforms.

5.4 Flexible loss functions

Our framework can handle any differentiable loss function. Such flexibility is important as various applications require flexibility in loss functions beyond what is provided by current tree ensemble learning toolkits. Our framework is built on Tensorflow, which allows for scalable gradient-based optimization. This coupled with our efficient differentiable tree ensemble formulation gives a powerful toolkit to seamlessly experiment with different loss functions and select what is suitable for the intended application. A few examples of flexible distributions that our toolkit supports — due to compatibility with Tensorflow-Probability — are normal, Poisson, gamma, exponential, mixture distributions e.g., zero-inflation models [153], and compound distributions e.g., negative binomial [277]. Other loss functions such as those robust to outliers [11] can also be handled. To demonstrate the flexibility of our framework, we deeply investigate two specific examples: zero-inflated Poisson and negative binomial regression. These cannot be handled by the popular gradient boosting toolkits such as XGBoost [46] and LightGBM [134].

Zero-inflated Poisson Regression. Zero-inflation occurs in many applications, e.g., understanding alcohol and drug abuse in young adults [124], characterizing undercoverage and overcoverage to gauge the on-going quality of the census frames [278], studying popularity of news items on different social media platforms [191], financial services applications [153] etc. Despite the prevalence of these applications, there has been limited work on building decision tree-based approaches for zero-inflated data perhaps due to a lack of support in public toolkits. Therefore, practitioners either resort to Poisson regression with trees or simpler linear models to handle zero-inflated responses. A Poisson model can lead to sub-optimal performance due to the limiting equidispersion constraint (mean equals the variance). Others take a two-stage approach [36], where a classification model distinguishes the zero and non-zero and a second model is used to model the non-zero responses. This can be sup-optimal as errors in the first model can deteriorate the performance of the second model. We employ a more well-grounded approach by formulating the joint mixture model, where one part of the model tries to learn the mixture proportions (zero vs non-zero) and the other part models the actual non-zero responses. Such a mixture model permits a differentiable loss function when both components of the model are parameterized with differentiable tree ensembles and can be optimized with gradient descent method in an end-to-end fashion without the need for a custom solver. We provide an extensive study with our framework on small to large-scale real world zero-inflated datasets and demonstrate that such flexibility in distribution modeling can lead to significantly more compact and expressive tree ensembles. This has large implications for faster inference, storage requirements and interpretability.

We briefly review Poisson regression and then dive into zero-inflated Poisson models. Poisson regression stems from the generalized linear model (GLM) framework for modeling a response variable in the exponential family of distributions. In general, GLM uses a link function to provide the relationship between the linear predictors, $\boldsymbol{x}$ and the conditional mean of the density function: $g[\mathbb{E}(\mathrm{y}|\boldsymbol{x})] = \boldsymbol{\beta} \cdot \boldsymbol{x}$, where $\boldsymbol{\beta}$ are parameters and $g(\cdot)$ is the link function. When responses y_n (for $n \in [N]$), are independent and identically distributed (i.i.d.) and follow the Poisson distribution conditional on $\boldsymbol{x}_n$'s, we use $log(\cdot)$ as the link function and call the model a Poisson regression model: $log(\mu_n|\boldsymbol{x}_n) = \boldsymbol{\beta} \cdot \boldsymbol{x}_n$. We consider more general parameterizations with tree ensembles as given by

$$log(\mu_n|\boldsymbol{x}_n) = f(\boldsymbol{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}). \tag{5.4.1}$$

where f is parameterized with a tree ensemble as in (5.3.1) and $\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}$ are the learnable parameters in the supernodes and the leaves of the tree ensemble. When a count data has excess zeros, the equi-dispersion assumption of the Poisson is violated. The Poisson model is not an appropriate model for this situation anymore. [148] proposed zero-inflated-Poisson (ZIP) models that address the mixture of excess zeros and Poisson count process. The mixture is indicated by the latent binary variable d_n using a logit model and the density for the

Poisson count given by the log-linear model. Thus, $y_n = y_n^* \odot 1[d_n \neq 0]$, where the latent indicator $d_n \sim Bernoulli(\pi_n)$ with $\pi_n = P(d_n = 1)$ and $y_n^* \sim Poisson(\mu_n)$. The mixture yields the marginal probability mass function of the observed y_n given as:

$$ZIP(y_n|\mu_n, \pi_n) = \begin{cases} (1-\pi_n) + \pi_n e^{-\mu_n}, & \text{if } \; y_n = 0 \\ \pi_n e^{-\mu_n} \mu_n^{y_n}/y_n!, & \text{if } \; y_n = 1, 2, \cdots \end{cases} \tag{5.4.2}$$

where μ_n and π_n are modeled by

$$log\left(\frac{\pi_n}{1-\pi_n}|\boldsymbol{x}_n\right) = f(\boldsymbol{x}_n; \boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{U}}) \tag{5.4.3}$$

$$log(\mu_n|\boldsymbol{x}_n) = f(\boldsymbol{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}). \tag{5.4.4}$$

where $\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{U}}$ are the learnable parameters in the splitting internal supernodes and the leaves of the tree ensemble for the logit model for π_n and $\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}$ are the learnable parameters in the supernodes and the leaves of the tree ensemble for the log-tree model for μ_n respectively. The likelihood function for this ZIP model is given by

$$L(y_n, f(\boldsymbol{x}_n)) = \prod_{y_n=0} (1-\pi_n) + \pi_n e^{-\mu_n} \prod_{y_n>0} \pi_n e^{-\mu_n} \mu_n^{y_n}/y_n! \tag{5.4.5}$$

where $\mu_n = e^{f(\boldsymbol{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})}$ and $\pi_n = e^{f(\boldsymbol{x}_n, \boldsymbol{\mathcal{Z}}; \boldsymbol{\mathcal{U}})}/(1 + e^{f(\boldsymbol{x}_n; \boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{U}})})$. Such a model can be overparameterized and we observed that sharing the learnable parameters $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{W}}$ in the splitting internal supernodes across the log-mean and logit models can lead to better test performance — see Section 5.6 for a thorough evaluation on real-world datasets.

Negative Binomial Regression. An alternative distribution to zero-inflation modeling that can cater to over-dispersion in the responses is Negative Binomial (NB) distribution. A negative binomial distribution for a random variable y with a non-negative mean $\mu \in \mathbb{R}_+$ and dispersion parameters $\phi \in \mathbb{R}_+$ is given by:

$$NB(\mathrm{y}|\mu, \phi) = \binom{\mathrm{y} + \phi - 1}{\mathrm{y}} \left(\frac{\mu}{\mu + \phi}\right)^{\mathrm{y}} \left(\frac{\phi}{\mu + \phi}\right)^{\phi} \tag{5.4.6}$$

The mean and variance of a random variable $\mathrm{y} \sim NB(\mathrm{y}|\mu, \phi)$ are $\mathbb{E}[\mathrm{y}] = \mu$ and $\mathrm{Var}[\mathrm{y}] = \mu + \mu^2/\phi$. Recall that Poisson($\mu$) has variance μ, so $\mu^2/\phi > 0$ is the additional variance of the negative binomial above that of the Poisson with mean μ. So the inverse of parameter ϕ controls the overdispersion, scaled by the square of the mean, μ^2.

When the responses y_n (for $n \in [N]$) are i.i.d, and follow NB distribution conditioned on $\boldsymbol{x}_n$'s, we can use the $log(.)$ as a link function to parameterize the log-mean and log-dispersion as linear functions of the covariates $\boldsymbol{x}_n$. In our parameterization with Tree Ensembles, we model them as given by:

$$log(\mu_n|\boldsymbol{x}_n) = f(\boldsymbol{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}) \tag{5.4.7}$$

$$log\,(\phi_n|\boldsymbol{x}_n) = f(\boldsymbol{x}_n; \boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{U}}). \tag{5.4.8}$$

where $\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{U}}$ are the learnable parameters in the supernodes and the leaves of the tree ensemble for the log-mean and $\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}$ are the learnable parameters in the supernodes and the leaves of the tree ensemble for the log-dispersion model for ϕ_n respectively. Such a model can be overparameterized and we observed that sharing the learnable parameters $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{W}}$ in the splitting internal supernodes across the log-mean and log-dispersion models can lead to better out-of-sample performance. See Section 5.6.4 for empirical validation on a large-scale dataset.

5.5 Multi-task Learning with Tree Ensembles

Multi-task Learning (MTL) aims to learn multiple tasks simultaneously by using a shared model. Unlike single task learning, MTL can achieve better generalization performance through exploiting task relationships [38, 42]. One key problem in MTL is how to share model parameters between tasks [226]. For instance, sharing parameters between unrelated tasks can potentially degrade performance. MTL approaches for classical decision trees approaches e.g., RF [166], GRF [9] have shared weights at the splitting nodes across the tasks. Only the leaf weights are task specific. However this can be limiting in terms of performance, despite easier interpretability associated with the same split nodes across tasks.

To perform flexible multi-task learning, we extend our formulation in Section 5.3.2 by using task-specific nodes in the tree ensemble. We consider T tasks. For easier exposition, we consider tasks of the same kind: multilabel classification or multi-task regression. For multilabel classification, each task is assumed to have same number of classes (with $k = C$) for easier exposition — our framework can handle multilabel settings with different number of classes per task. Similarly, for regression settings, $k = 1$. For multi-task zero-inflated Poisson or negative binomial regression, when two model components need to be estimated, we set $k = 2$ to predict log-mean and logit components for zero-inflated Poisson and log-mean and log-dispersion components for negative binomial.

We define a trainable weight tensor $\boldsymbol{\mathcal{W}}_i \in \mathbb{R}^{T,p,m}$ for supernode $i \in \mathcal{I}$, where each t-th slice of the tensor $\boldsymbol{\mathcal{W}}_i[t, :, :]$ denotes the trainable weight matrix associated with task t. The prediction in this case is given by

$$\boldsymbol{f}(\boldsymbol{x}) = \left(\sum_{l \in L} \boldsymbol{\mathcal{O}}_l \odot \prod_{i \in A(l)} \boldsymbol{\mathcal{R}}_{i,l} \right) \cdot \frac{1}{m} \mathbf{1}_m \tag{5.5.1}$$

where $\boldsymbol{\mathcal{O}}_l \in \mathbb{R}^{T,m,k}$ denotes the trainable leaf tensor in leaf l, $\boldsymbol{\mathcal{R}}_{i,l} = S(\boldsymbol{\mathcal{W}}_i \cdot \boldsymbol{x}) 1[l \swarrow i] \odot (1 - S(\boldsymbol{\mathcal{W}}_i \cdot \boldsymbol{x})) 1[i \searrow l] \in \mathbb{R}^{T,m,1}$.

In order to share information across the tasks, our framework imposes a closeness penalty on the hyperplanes $\boldsymbol{\mathcal{W}}_i$ in the supernodes across the tasks. This results in the optimization formulation:

$$\min_{\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}} \sum_{t \in T} \sum_{\boldsymbol{x}, y_t} g_t(y_t, f_t(\boldsymbol{x})) + \lambda \sum_{s < t, t \in T} \left\| \boldsymbol{\mathcal{W}}_{:,s,:,:} - \boldsymbol{\mathcal{W}}_{:,t,:,:} \right\|^2, \tag{5.5.2}$$

where $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{\mathcal{I},T,m,p}$ denotes all the weights in all the supernodes, $\boldsymbol{\mathcal{O}} \in \mathbb{R}^{\mathcal{L},m,k}$ denotes all the weights in the leaves, and $\lambda \in [0, \infty)$ is a non-negative regularization penalty that controls how close the weights across the tasks are. For $\lambda = 0$, the model behaves similar to a single-task learning setting. When $\lambda \to \infty$, the model shares complete information in the splitting nodes and the weights across the tasks in each of the internal supernodes become the same — this is similar to hard parameter sharing. The latter case can be separately handled more efficiently by using the function definition in (5.3.4) for $\boldsymbol{f}(\boldsymbol{x})$ without any closeness regularization in (5.5.2). Our model can control the level of sharing across the tasks by controlling λ. In practice, we tune over $\lambda \in [1e-5, 10]$ and select the optimal value based on a validation set. This penalty assumes that the hyperplanes across the tasks should be equally close as we go down the depth of the trees. However this assumption maybe less accurate as we go down the tree. Empirically, we found that decaying λ exponentially as $\lambda/2^d$ with depth d of the supernodes in the ensemble can achieve better test performance.

5.6 Experiments

We study the performance of differentiable tree ensembles in various settings and compare against the relevant state-of-the-art baselines for each setting. The different settings can be summarized as follows: (i) Comparison of a single soft tree with state-of-the-art classical tree method in terms of test performance and depth. We include both axis-aligned and oblique classical tree in our comparisons. (ii) Flexible zero-inflation models with tree ensembles. We compare against Poisson regression with tree ensembles and gradient boosting decision trees (GBDT). We consider test Poisson deviance and tree ensemble compactness for model evaluation (iii) We evaluate our proposed multi-task tree ensembles and compare them against multioutput RF, multioutput GRF and single-task GBDT. We consider both fully observed and partially observed responses across tasks. (iv) We also validate our tree ensemble methods with flexible loss functions (zero-inflated Poisson and negative binomial regression) on a large-scale multi-task proprietary dataset.

Model Implementation. Differentiable tree ensembles in our toolkit are implemented in TensorFlow 2.0 using Keras interface.

Datasets. We use 27 open-source regression datasets from various domains (e.g., social media platforms, human behavior, finance). 9 are from Mulan [272], 2 are from UCI data repository [61], 12 are from Delve database [6] and the 5 remaining are SARCOS [255], Youth Risk Behavior Survey [124], Block-Group level Census data [252], and a financial services loss dataset from Kaggle[3]. We also validate our framework on a proprietary multi-task data with millions of samples from a multi-national financial services company.

5.6.1 Studying a Single Tree

In this section, we compare performance and model compactness of a single tree on 12 regression datasets from Delve database: abalone, pumadyn-family, comp-activ (cpu, cpuSmall) and concrete.

Competing Methods and Implementation. We focus on two baselines from classical tree literature: CART [29] and Tree Alternating Optimization (TAO) method proposed by [37]. The authors in [289] performed an extensive comparison of various single tree learners and demonstrated TAO to be the best performer. Hence, we include both axis-aligned and oblique decision tree versions of TAO in our comparisons. Given that the authors in [37, 289] do not provide an open-source implementation for TAO, we implemented our own version of TAO. For a fair comparison, we use binary decision trees for both axis-aligned and oblique versions of TAO. For more details about our implementation, see Appendix Section 5.8.1.

[3]https://bit.ly/3swGnTo

Table 5.6.1: *Test mean squared error of single axis-aligned and oblique decision trees on various regression datasets.*

	Axis-Aligned		Oblique	
Data	**CART**	**TAO**	**TAO**	**Soft Tree**
abalone	7.901E-03	8.014E-03	7.205E-03	**6.092E-03**
pumadyn-32nh	9.776E-03	9.510E-03	1.146E-02	**8.645E-03**
pumadyn-32nm	2.932E-03	2.741E-03	4.942E-03	**1.280E-03**
pumadyn-32fh	1.324E-02	1.307E-02	1.370E-02	**1.166E-02**
pumadyn-32fm	2.246E-03	2.177E-03	3.566E-03	**1.602E-03**
pumadyn-8nh	1.995E-02	1.983E-02	2.027E-02	**1.670E-02**
pumadyn-8nm	4.878E-03	4.584E-03	4.206E-03	**2.352E-03**
pumadyn-8fh	2.182E-02	2.200E-02	2.159E-02	**2.048E-02**
pumadyn-8fm	4.398E-03	4.347E-03	4.074E-03	**3.543E-03**
cpu	9.655E-04	1.475E-03	1.312E-03	**9.159E-04**
cpuSmall	1.450E-03	2.319E-03	1.171E-03	**9.159E-04**
concrete	7.834E-03	6.619E-03	1.166E-02	**4.139E-03**

Results. We present the out-of-sample mean-squared-error performance and optimal depths in Tables 5.6.1 and 5.8.1 (in Appendix Section 5.8.1) respectively. Notably, in all 12 cases, soft tree outperforms all 3 baseline methods in terms of test performance. The soft tree finds a smaller optimal depth in majority cases in comparison with its classical counterpart i.e., oblique TAO tree — See Table 5.8.1 in Appendix Section 5.8.1. This may be due to the end-to-end learning in a soft tree, unlike TAO that performs local search.

5.6.2 Zero-inflation

We consider a collection of real-world applications with zero-inflated data. The datasets include (i) yrbs: nationwide drug use behaviors of high school students as a function of demographics, e-cigarettes/mariyuana use etc.; (ii) news: popularity of news items on social media platforms [191] e.g., Facebook, Google+ as a function of topic and sentiments; (iii) census: number of people with zero, one, or two health insurances across all Census blocks in the ACS population as a function of housing and socio-economic demographics; (iv) fin-services-loss: financial services losses as a function of geodemographics, information on crime rate, weather.

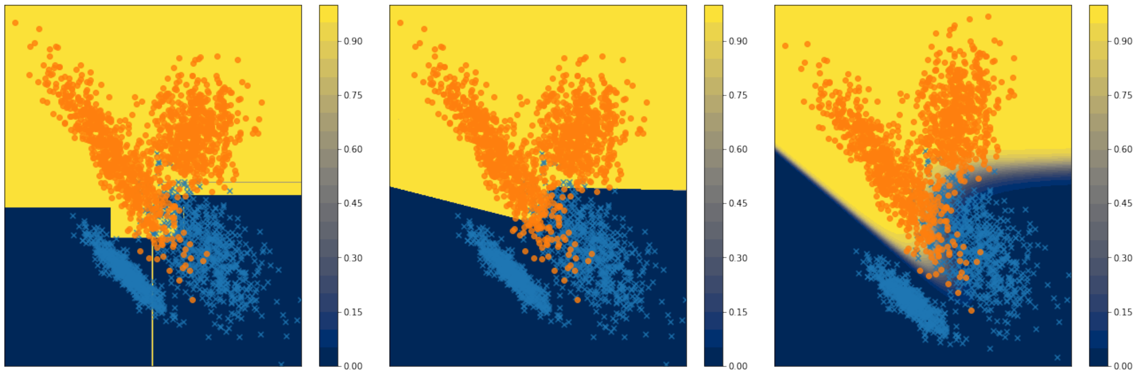


Figure 5.6.1: *Classifier boundaries for CART [Left], TAO (oblique) [Middle] and Soft tree [Right] on a synthetic dataset with* $N_{train} = N_{val} = N_{test} = 2500$ *from sklearn [33]. We tune for 50 trials over depths in the range* $[2-4]$ *for TAO (oblique) and soft trees and* $[2-10]$ *for CART. Optimal depths for CART, TAO, Soft tree are 5, 4 and 2 respectively. Test AUCs are 0.950, 0.957, and 0.994 respectively.*

Table 5.6.2: *Test poisson deviance across models on various datasets. Flexible modeling via zero-inflated Poisson for Soft Tree Ensembles leads to better poisson deviance.*

			GBDT	**Soft Trees**	
Data	**N**	**p**	**Poisson**		**ZIP**
yrbs-cocaine	12172	55	3.14E-02	3.00E-02	**2.82E-02**
yrbs-heroine	12711	55	1.81E-02	1.60E-02	**1.54E-02**
yrbs-meth	12690	55	2.38E-02	2.21E-02	**2.09E-02**
yrbs-lsd	9564	55	3.51E-02	3.59E-02	**3.43E-02**
news-facebook	81637	3	4.70E-03	**4.68E-03**	**4.68E-03**
news-google+	87495	3	5.97E-03	**5.93E-03**	**5.93E-03**
census-health0	220333	64	1.51E-04	**1.28E-04**	1.32E-04
census-health1	220333	64	**4.63E-04**	5.11E-04	4.66E-04
census-health2+	220333	64	2.73E-03	3.06E-03	**2.72E-03**
fin-services-losses	452061	300	**2.20E-03**	2.28E-03	**2.20E-03**
#wins	-	-	2	3	8

Competing methods. We consider Poisson regression with GBDT and differentiable tree ensembles. We also consider zero-inflation modeling with differentiable tree ensembles. We use GBDT from sklearn [33]. For additional details about the tuning experiments, please see Appendix Section 5.8.2.

Results. We present the out-of-sample Poisson deviance performance in Table 5.6.2. Notably, tree ensembles with zero-inflated loss function leads the chart. We also present the

Table 5.6.3: *Tree ensemble compactness (# trees and depth) for GBDT and Soft Tree Ensembles for different datasets. Flexible modeling via zero-inflated Poisson for Soft Tree Ensembles can lead to more compact tree ensembles, which can improve interpretability*

	#Trees			**Depth**		
	GBDT	**Soft Trees**		**GBDT**	**Soft Trees**	
Data	**Poisson**		**ZIP**	**Poisson**		**ZIP**
yrbs-cocaine	575	77	**13**	4	**2**	3
yrbs-heroine	1425	83	**4**	4	4	**2**
yrbs-meth	1475	16	**4**	4	**2**	3
yrbs-lsd	1225	**17**	45	4	3	**2**
news-facebook	200	**65**	85	4	4	4
news-google+	750	81	**74**	4	4	4
census-health0	1275	**10**	**10**	4	3	**2**
census-health1	1275	**17**	**17**	4	2	**2**
census-health2+	1375	73	**56**	8	4	**3**
fin-services-losses	1225	32	**4**	4	3	**2**
#wins	0	4	**7**	-	5	**8**

optimal selection of tree ensemble sizes and depths in Table 5.6.3. We can observe that zero-inflation modeling can lead to significant benefits in terms of model compression. Both tree ensemble sizes and depths can potentially be made smaller, which have implications for faster inferences, memory footprint and interpretability.

5.6.3 Multi-task Regression

We compare performance and model compactness of our proposed regularized multi-task tree ensembles on 11 multi-task regression datasets from Mulan (atp1d, atp7d, sf1, sf2, jura, enb, slump, scm1d, scm20d), and UCI data repository (bike) and SARCOS dataset.

Competing Methods. We focus on 4 tree ensemble baselines from literature: single-task soft tree ensembles, sklearn GBDT, sklearn multioutput RF [33] and r-grf package for GRF [9]. We consider two multi-task settings: (i) All Fully observed responses for all tasks, (ii) Partially observed responses across tasks. In the former case, we compare against RF and GRF. In the latter case, we compare against single-task soft tree ensembles and GBDT. Note the open-source implementations for RF and GRF do not support partially observed

Table 5.6.4: *Test MSE of RF, GRF and multi-task Differentiable Tree Ensembles on 11 multi-task regression datasets with fully observed responses across tasks.*

		Multi-task		
Data	**Task**	**RF**	**GRF**	**Soft Trees**
atp1d	1	2.242E-02	1.847E-02	**5.383E-03**
	2	2.498E-02	1.894E-02	**7.217E-03**
	3	1.127E-02	**9.625E-03**	1.128E-02
	4	1.574E-02	1.504E-02	**1.403E-02**
	5	2.040E-02	1.905E-02	**1.182E-02**
	6	1.571E-02	**1.333E-02**	1.527E-02
atp7d	1	3.244E-03	**2.691E-03**	8.965E-03
	2	**3.914E-03**	3.931E-03	4.456E-03
	3	1.231E-02	1.078E-02	**1.059E-02**
	4	**5.459E-03**	6.542E-03	6.001E-03
	5	**1.842E-03**	1.922E-03	3.358E-03
	6	**4.042E-03**	5.303E-03	5.033E-03
sf1	1	4.384E-02	**3.151E-02**	3.345E-02
	2	1.077E-02	4.638E-03	**3.828E-03**
	3	4.252E-02	**2.863E-02**	2.993E-02
sf2	1	7.883E-03	8.807E-03	**7.789E-03**
	2	3.247E-03	2.583E-03	**2.206E-03**
	3	**1.262E-03**	3.744E-03	3.595E-03
jura	1	3.027E-02	3.008E-02	**2.233E-02**
	2	1.483E-02	1.405E-02	**1.015E-02**
	3	7.896E-03	7.586E-03	**6.036E-03**
enb	1	2.473E-04	**1.865E-04**	2.063E-04
	2	2.830E-03	3.305E-03	**1.054E-03**
slump	1	1.732E-01	1.396E-01	**1.001E-01**
	2	1.224E-01	9.827E-02	**7.368E-02**
	3	3.878E-02	2.944E-02	**5.149E-03**
scm1d	1	3.040E-03	2.530E-03	**1.794E-03**
	2	3.397E-03	3.003E-03	**2.226E-03**
	3	4.178E-03	3.611E-03	**2.940E-03**
	4	3.991E-03	3.376E-03	**2.150E-03**
scm20d	1	4.457E-03	3.650E-03	**2.198E-03**
	2	4.766E-03	3.632E-03	**2.410E-03**
	3	4.892E-03	3.506E-03	**2.620E-03**
	4	5.573E-03	4.072E-03	**2.632E-03**
bike	1	3.466E-03	2.558E-03	**1.730E-03**
	2	4.636E-03	4.039E-03	**3.728E-03**
	3	5.123E-03	4.303E-03	**3.822E-03**
# wins	-	5	6	**26**

Table 5.6.5: *Tree ensemble sizes for soft trees, RF, and GRF.*

	atp1d	**atp7d**	**sf1**	**sf2**	**jura**	**enb**	**slump**	**scm1d**	**scm20d**	**bike**
RF	100	125	500	225	150	100	825	175	100	100
GRF	1050	950	350	100	150	100	350	50	100	250
Ours	10	15	12	44	17	13	54	49	25	93

responses for multi-task settings and GBDT does not have support for multi-task setting. We refer the reader to Appendix Section 5.8.3 for tuning experiments details.

Results. We present results for fully observed response settings in Table 5.6.4 and partially observed response settings in Table 5.6.6. In both cases, regularized multi-task soft trees lead the charts over the corresponding baselines in terms of out-of-sample mean squared error performance. For the fully observed response setting, we also show tree ensemble sizes in Table 5.6.5. We see a large reduction in the number of trees with out proposed multi-task tree ensembles.

5.6.4 Large-scale multi-task data from a multinational financial services company

We study the performance of our differentiable tree ensembles in a real-word, large-scale multi-task setting from a multinational financial services company. The system encompasses costs and fees for millions of users for different products and services. The dataset has the following characteristics: (i) It is a multi-task regression dataset with 3 tasks. (ii) Each task has high degree of over-dispersion. (iii) All tasks are not fully observed as each user signs up for a subset of products/services. The degree of missing responses on average across tasks is $\sim 50\%$. (iv) Number of features is also large (~ 600).

We validate the flexibility of our end-to-end tree-ensemble learning framework with soft trees on a dataset of 1.3 million samples. We study the following flexible aspects of our framework: (i) Flexible loss handling with zero-inflation Poisson regression and negative binomial regression for single-task learning. (ii) Multi-task learning with our proposed

Table 5.6.6: *Test MSE of GBDT, single-task and multi-task Soft Tree Ensembles on 11 multi-task regression datasets, with 50% missing responses per task.*

		Single-Task		**Multi-Task**
Data	**Task**	**GBDT**	**Soft Trees**	
atp1d	1	1.469E-02	3.091E-02	**1.295E-02**
	2	**7.698E-03**	2.598E-02	1.137E-02
	3	2.172E-02	2.915E-02	**1.807E-02**
	4	**5.905E-03**	**1.417E-02**	9.434E-03
	5	2.421E-02	5.631E-02	**2.105E-02**
	6	**5.646E-03**	5.880E-02	2.724E-02
atp7d	1	5.562E-02	6.261E-02	**3.601E-02**
	2	4.033E-02	2.216E-02	**1.067E-02**
	3	2.140E-02	4.989E-02	**1.680E-02**
	4	1.107E-02	2.089E-02	**9.926E-03**
	5	4.254E-02	6.476E-02	**3.053E-02**
	6	**4.195E-03**	2.416E-02	, 2.199E-02
sf1	1	**2.674E-02**	2.763E-02	2.732E-02
	2	5.825E-03	1.680E-02	**5.435E-03**
	3	3.030E-02	3.704E-02	**2.964E-02**
sf2	1	**1.003E-02**	1.179E-02	**1.009E-02**
	2	1.123E-02	6.843E-03	**2.809E-03**
	3	9.271E-03	1.039E-02	**8.064E-03**
jura	1	1.779E-02	1.934E-02	**1.503E-02**
	2	1.117E-02	1.665E-02	**9.304E-03**
	3	1.311E-02	1.514E-02	**1.262E-02**
enb	1	**1.509E-04**	2.738E-04	4.167E-04
	2	**1.071E-03**	1.227E-03	1.160E-03
slump	1	1.622E-01	**7.611E-02**	9.485E-02
	2	8.823E-02	1.050E-01	**4.734E-02**
	3	8.423E-03	**1.737E-03**	7.744E-03
scm1d	1	**1.598E-03**	1.903E-03	2.058E-03
	2	**1.952E-03**	2.703E-03	2.490E-03
	3	3.029E-03	3.194E-03	**2.919E-03**
	4	**2.666E-03**	3.656E-03	3.272E-03
scm20d	1	2.541E-03	2.672E-03	**2.533E-03**
	2	3.640E-03	3.174E-03	**3.146E-03**
	3	3.658E-03	4.015E-03	**3.201E-03**
	4	3.756E-03	4.115E-03	**3.670E-03**
bike	1	**2.122E-03**	2.558E-03	2.300E-03
	2	**3.680E-03**	4.038E-03	3.846E-03
	3	**3.731E-03**	4.303E-03	3.910E-03
sarcos	1	2.582E-04	**1.317E-04**	1.518E-04
	2	1.643E-04	8.310E-05	**8.307E-05**
	3	3.325E-04	**1.788E-04**	1.933E-04
# wins	-	14	5	**23**

regularized multi-task soft tree ensembles in the presence of missing responses across tasks. (iii) Flexible loss handling with zero-inflation Poisson/negative binomial regression in the

Table 5.6.7: *Out-of-sample performance of single-task and multi-task tree ensembles with flexible loss functions for zero-inflation/overdispersion. We evaluate performance with weighted Poisson deviance and AUC across tasks.*

		GBDT	**Soft Trees**					
		Single-task	**Single-task**			**Multi-task**		
Metric	**Task**	**Poisson**	**Poisson**	**ZIP**	**NB**	**Poisson**	**ZIP**	**NB**
Poisson Deviance	1	2.643E-04	2.624E-04	2.623E-04	2.623E-04	2.607E-04	**2.605E-04**	2.608E-04
	2	8.029E-04	8.050E-04	8.029E-04	8.044E-04	8.022E-04	**8.014E-04**	**8.014E-04**
	3	1.044E-03	1.045E-03	1.043E-03	1.042E-03	1.041E-03	**1.040E-03**	1.041E-03
AUC	1	0.710	0.721	0.722	0.721	0.730	**0.734**	0.727
	2	0.690	0.689	0.690	0.688	0.691	0.691	**0.692**
	3	0.684	0.683	0.686	0.685	0.687	**0.689**	**0.689**

context of multi-task learning.

We present our results in Table 5.6.7. We can see that we achieve the lowest Poisson deviance and highest AUC with multi-task regression via zero-inflated Poisson/negative binomial regression.

5.7 Conclusion

We propose a flexible and scalable framework for learning differentiable tree ensembles. Our framework supports a diverse set of loss functions and allows for easily adding new loss functions. It also has novel support for multi-task learning. For scalability, we propose a new tensor-based formulation of tree ensembles, which allows for 10x faster training on CPUs and also adds support for GPU training. We perform experiments on a collection of 28 open-source and real-world datasets, demonstrating that our new `FASTEL` toolkit can lead to 100x more compact ensembles and up to 23% improvement in out-of-sample performance, compared to tree ensembles learnt by popular toolkits such as XGBoost.

5.8 Appendix

Datasets. We use a collection of 27 open-source regression datasets from various domains (e.g., social media platforms, human behavior, financial risk data). 9 of these are from

Mulan: A Java library for multi-label learning (Mulan) [272], 2 of them are from University of California Irvine data repository (UCI) [61], 12 of them are from Delve database [6] and the 5 remaining are SARCOS [4] [255], Youth Risk Behavior Survey[5] [124], Block-Group level data from US Census Planning Database[6][252], and financial services loss data from Kaggle[7]. For scm1d and scm20d from Mulan[272], we consider the first 4 tasks (out of the 16 tasks in the original dataset). For SARCOS, we consider 3 torques for prediction (torque-3, torque-4 and torque-7; we ignore the other torques as those seem to have poor correlations with these.)

For all datasets, we split the datasets into 64%/16%/20% training/validation/test splits. We train the models on the training set, perform hyperparameter tuning on the validation set and report out-of-sample performance on the test set.

5.8.1 Tuning parameters and optimal depths comparison for a single tree in Section 5.6.1

TAO Implementation. We wrote our own implementation of the TAO algorithm proposed in [37]. We considered binary trees with TAO for both axis-aligned and oblique trees. In the case of axis-aligned splits, we initialize the tree with CART solution and run TAO iterations until there is no improvement in training objective. In the case of oblique trees, we initialize with a complete binary tree with random parameters for the hyperplanes in the split nodes and use logistic regression to solve the decision node optimization. We run the algorithm until either a maximum number of iterations are reached or the training objective fails to improve.

Tuning parameters. For CART and axis-aligned TAO, we tune the depth in the range $[2-20]$. We also optimize over the maximum number of iterations in the interval $[20-100]$.

[4]The original test set has significant data leakage as noted by https://www.datarobot.com/blog/running-code-and-failing-models/. Following their guidance we discard the original test set and use the original train set to generate train/validation/test splits.

[5]https://www.cdc.gov/healthyyouth/data/yrbs/data.htm

[6]https://www.census.gov/topics/research/guidance/planning-databases.html

[7]https://bit.ly/3swGnTo

For oblique TAO and soft tree, we tune the depth between $2-10$ and the number of iterations between $20-100$. Additionally, for soft tree, we also tune over the learning rates $[1e-5, 1e-2]$ with Adam optimizer and batch sizes $\{64, 128, 256, 512\}$. For a fair comparison, we run all 4 methods for 100 trials.

Optimal depths. We make a comparison of optimal depths between CART, TAO (both axis-aligned and oblique) and soft tree. The soft tree finds a smaller optimal depth in majority cases in comparison with its classical counterpart i.e., oblique TAO tree — See Table 5.8.1. This is hypothesized to be due to the end-to-end optimization done by soft tree as opposed to a local search performed by the TAO algorithm.

Table 5.8.1: *Optimal depth of a single axis aligned and oblique decision tree on various regression datasets.*

	Axis-Aligned		**Oblique**	
Data	**CART**	**TAO**	**TAO**	**Soft Tree**
abalone	5	5	4	**2**
pumadyn-32nh	6	6	4	**3**
pumadyn-32nm	8	8	5	**4**
pumadyn-32fh	**3**	**3**	**3**	5
pumadyn-32fm	6	6	5	**4**
pumadyn-8nh	6	6	5	**3**
pumadyn-8nm	9	8	7	**5**
pumadyn-8fh	5	5	4	**2**
pumadyn-8fm	7	7	6	**3**
cpu	10	8	**5**	**5**
cpuSmall	8	8	**5**	**5**
concrete	15	8	**5**	9

5.8.2 Tuning parameters for Sections 5.6.2

We use HistGBDT from sklearn [33] (GBDT in sklearn does not support Poisson regression). We tune over depths in the range $[2-20]$, number of trees between $50-1500$ and learning rates on the log-uniform scale in the interval $[1e-5, 1e-1]$. For differentiable tree ensembles, we tune number of trees in the range $[2, 100]$, depths in the set $[2-4]$, batch sizes $\{64, 128, 256, 512\}$, learning rates $[1e-5, 1e-1]$ with Adam optimizer and perform early stopping with a patience of 25 based on the validation set. For all models, we perform a random search with 1000 hyperparameter tuning trials.

5.8.3 Tuning parameters for Sections 5.6.3

We tune number of trees in the interval $[50 - 1500]$ for RF, GRF and GBDT. For RF and GBDT, we also tune over depths between $2 - 20$. For GBDT, we tune learning rates between $[1e - 5, 1e - 1]$. For GRF, we also tune over min_node_size in the set $[2 - 20]$ and $\alpha \in [1e - 3, 1e - 1]$. For single-task and multi-task trees, we tune over depths $[2 - 4]$, number of trees $[5 - 100]$, batch sizes $\{64, 128, 256, 512\}$, epochs $[20 - 500]$, Adam learning rates $[1e - 5, 1e - 2]$. We also optimize over the regularization penalty for multi-task soft decision trees $[1e - 5, 1e1]$. All single-task models (soft tree ensembles, GBDT) are tuned for 1000 trials per task. All multi-task models (RF, GRF, multi-task soft trees) are tuned for 1000 trials in total.

5.8.4 Tuning parameters for Sections 5.6.4

We use GBDT from XGBoost [46], where we tune number of trees in the interval $[50 - 1500]$, depths between $2 - 20$ and learning rates between $[1e - 4, 1e - 0]$. For single-task and multi-task trees, we tune over depths $[2 - 4]$, number of trees $[5 - 100]$, batch sizes $\{64, 128, 256, 512\}$, epochs $[20 - 200]$, Adam learning rates $[1e - 5, 1e - 2]$. We also optimize over the regularization penalty for multi-task soft decision trees $[1e - 5, 1e1]$. All single-task models (soft tree ensembles, GBDT) with Poisson, Zero-Inflated-Poisson, Negative Binomial are tuned for 1000 trials per task. All multi-task soft-trees are tuned for 1000 trials in total.

5.8.5 Exploration of different loss and training strategies for tree ensemble learning

In this section, we explore some additional ensemble learning techniques. There can be different loss objectives as well as training strategies to improve the learning of an ensemble of trees. These training strategies are motivated from literature on deep ensembles of neural networks [126, 286, 297]. We study these techniques in the context of soft decision tree ensembles and provide some empirical results.

Before we dive into these objectives, we first briefly summarize some notations. The input feature-vector is denoted by $\boldsymbol{x} \in \mathbb{R}^p$. The output is denoted by $\mathbf{y} \in \mathbb{R}^k$. We have an ensemble of m differentiable trees. Let $\boldsymbol{f}^j$ be the jth tree in the ensemble. For classification tasks, $\boldsymbol{f}^j$ can either be the logits or the probability over the classes. We denote the average of predictions in the tree ensemble as

$$\bar{\boldsymbol{f}}(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} \boldsymbol{f}^j(\boldsymbol{x}). \tag{5.8.1}$$

We explore five different techniques for learning a tree ensemble:

1. **Independent.** Train each of the base learner $\boldsymbol{f}^j$ independently. This can be thought of as training with the sum of individual objectives:

$$\frac{1}{m}\sum_{j\in[m]} \mathcal{L}(\mathbf{y}, \boldsymbol{f}^j(x)) \tag{5.8.2}$$

   At evaluation, the probability predictions are combined in an additive fashion.

2. **Joint.** Train all base learners jointly with the objective:

$$\mathcal{L}(\mathbf{y}, \bar{\boldsymbol{f}}(x)) \tag{5.8.3}$$

Here, we combine the predictions at the logit level as done in [100] as well as in chapter 6.

3. **Multi-Objective.** Following Jeffares et al. [126], the two above objectives are combined in a multi-objective strategy as follows:

$$(1-\alpha_1)\left(\frac{1}{m}\sum_{j\in[m]}\mathcal{L}(\mathbf{y},\boldsymbol{f}^j)\right)+\alpha_1\mathcal{L}(\mathbf{y},\bar{\boldsymbol{f}}) \tag{5.8.4}$$

   where $\alpha_1 \in [0,1]$ is a tuning parameter.

4. **Diversity Penalization.** Zhang et al. [286] proposes to explicitly penalize diversity as follows:

$$\frac{1}{m}\sum_{j\in[m]}\left(\mathcal{L}(\mathbf{y},\boldsymbol{f}^j)+\alpha_2\mathrm{KL}(\bar{\boldsymbol{f}}||\boldsymbol{f}^j)\right) \tag{5.8.5}$$

   This encourages the base learners' predictions to shrink towards an average, allowing to use any of the base learner at prediction time. This can have implications for faster inference times as well as interpretability.

5. **THOR.** Zuo et al. [297] suggest to randomly sample two base learners (per mini-batch of samples) during training:

$$\mathcal{L}(\mathbf{y},\boldsymbol{f}^j)+\mathcal{L}(\mathbf{y},\boldsymbol{f}^k)+\alpha_3\mathrm{KL}_{sym}(\boldsymbol{f}^j||\boldsymbol{f}^k) \tag{5.8.6}$$

   where j and k denote two randomly chosen base learners for a mini-batch of samples during backpropagation. This sparse backpropagation allows to reduce the training costs. At inference time, a single base learner can be selected.

Experiments. We compare the 5 training strategies described above. We consider a collection of 9 classification datasets from the Penn Machine Learning Benchmarks (PMLB)

Table 5.8.2: Test AUC on 9 classification tasks for different ensemble learning techniques.

Dataset\Method	Independent	Joint	Multi-Objective	Diversity-Penalization	THOR
pima	79.52	79.65	81.88	**82.88**	79.14
yeast	**85.45**	84.56	84.77	84.53	85.04
heart-c	87.52	87.75	89.69	**90.10**	89.82
diabetes	**82.79**	81.76	82.77	81.91	82.69
solar	87.63	87.09	**88.56**	88.24	88.53
vehicle	95.76	95.33	95.71	**95.88**	93.07
crx	93.56	91.45	**93.83**	92.93	93.54
flare	**76.65**	73.77	75.51	74.60	73.23
churn	92.76	91.45	92.66	**92.86**	92.58
Rank(avg)	3	5	**1**	2	4
Rank(rank)	2	5	**1**	3	4

repository [200].

We set the number of trees to 10 and depth of soft trees to 4. We consider 10 different seeds for splitting of data into training-validation-test splits. For each split, we perform 200 tuning trials to tune over the hyperparameters based on performance on the validation set. We tune over learning rates for Adam $\eta \in \{1e-5, 1e-4, 1e-3, 1e-2, 1e-1\}$, epochs $\in \{25, 50, 75, ..., 200\}$, $\alpha_1 \in \{0.001, 0.005, 0.01, ..., 50\}$, $\alpha_2 \in \{0.001, 0.005, 0.01, ..., 50\}$ and $\alpha_3 \in \{0.001, 0.005, 0.01, ..., 50\}$. We report the test accuracy across the 10 seeds in Table 5.8.2.

We rank the methods in two ways: (i) Compute average AUC across the datasets and then compute the rank, (ii) Compute the rank per dataset and then rank the methods based on the average rank. It appears that the multi-objective tends to lead the chart. Diversity Penalization and Independent learning approaches also appear promising. This suggests that we can further improve the end-to-end learning of soft tree ensembles by considering alternative training strategies.

Chapter 6

Tree Ensembles: End-to-end Feature Selection Approach for Learning Skinny Trees

6.1 Introduction

In the chapter, we consider feature selection in tree ensembles. While there have been various toolkits for learning tree ensembles [30, 46, 134, 213], which are excellent for building tree ensembles, they do not allow for feature selection during the training process.

Feature selection is a fundamental problem in machine learning and statistics and has widespread usage across various real-world tasks [35, 160, 269]. Popular tree ensemble toolkits only allow selecting informative features post-training based on feature importance scores, which are known to have drawbacks[1] in the context of feature selection [28, 242, 291]. Recently, there has been some work on optimization-based approaches for feature selection in trees. For example, [289] consider oblique decision trees (hyperplane splits at every node), and use ℓ_1-penalization to encourage coefficient sparsity at every node of the tree. This

[1]They are found to hurt performance in (a) settings where number of samples are smaller than features and (b) settings with correlated features (see Sec. 6.5).

achieves node-level feature selection and does not appear to be well-suited for tree-level or ensemble-level feature selection (See Sec. 6.6.1). Liu et al. [168] proposed ControlBurn that considers a lasso based regularizer for feature selection on a pre-trained forest. This can be viewed as a two-stage procedure (unlike an end-to-end training procedure we propose here), where one performs feature selection after training a tree ensemble with all original features, While these methods serve as promising candidates for feature selection, these works highlight that identifying relevant features *while* learning compact trees remains an open challenge—an avenue we address in this work.

In many real world problems there are costs associated with features reflecting time, money, and other costs related to the procurement of data [188, 285]. In this context, one would like to collect a compact set of features to reduce experimental costs. Additionally, selecting a compact set of relevant features can lead to enhanced interpretability [223], faster inference, decreased memory footprint, and even improved model generalization on unseen data [40].

In this chapter, we propose an end-to-end optimization framework for feature selection in tree ensembles where we jointly learn the trees and the relevant features in *one* shot. Our framework is based on differentiable (a.k.a. soft) tree ensembles [96, 115, 131, 139] where tree ensembles are learnt based on differentiable programming. These works, however, do not address feature selection in trees which is our focus. We use a sparsity-inducing penalty (based on the group $\ell_0 - \ell_2$-penalty) to encourage feature selection. While group $\ell_0 - \ell_2$ penalty has been found to be useful in recent work on high-dimensional linear models [104] and additive models [104, 113], their adaptation to tree ensemble presents unique challenges. To obtain high-quality models with good generalization-and-sparsity tradeoffs, we need to pay special attention to dense-to-sparse training, which differs from sparse-to-dense training employed in linear/additive models above. We demonstrate that our end-to-end learning approach leads to better accuracy and feature sparsity tradeoffs.

Contributions. We propose a novel end-to-end optimization-based framework for feature selection in tree ensembles. We summarize our contributions in the chapter as follows:

- We propose a joint optimization approach, where we simultaneously perform feature selection and tree ensemble learning. Our joint training approach is different from post-training feature selection in trees. Our approach learns (differentiable) tree ensembles with a budget on feature sparsity where the latter is achieved via a group ℓ_0-based regularizer.
- Our algorithmic workhorse is based on proximal mini-batch gradient descent (GD). We also discuss the convergence properties of our approach in the context of a nonconvex and nonsmooth objective. When our first-order optimization methods are used with *dense-to-sparse* scheduling of regularization parameter, we obtain tree ensembles with better accuracy and feature-sparsity tradeoffs.
- We introduce a new toolkit: `Skinny Trees`. We consider 15 synthetic and real-world datasets, showing that `Skinny Trees`can lead to superior feature selection and test AUC compared to popular toolkits. In particular, for 25% feature budget, `Skinny Trees` outperforms LightGBM by 10.2% (up to 37.7%), XGBoost by 3.1% (up to 17.4%), and Random Forests by 3% (up to 12.5%) in test AUC.

6.2 Related Work

We review some prior work on feature selection as they relate to our work. We group them into three major categories:

1. Filter methods attempt to remove irrelevant features before model training. These methods perform feature screening based on statistical measures that quantify feature-specific relevance scores [13, 43, 69, 208, 238, 239]. These scores consider the marginal effect of a feature over the joint effect of feature interactions.
2. Wrapper methods [7, 138, 190, 201, 221, 225, 240, 293] use the outcome of a model to determine the relevance of each feature. Some of these methods require recomputing

the model for a subset of features and can be computationally expensive. This category also includes feature selection using feature importance scores of a pre-trained model. Many tree ensemble toolkits [30, 46, 134, 213] produce feature-importance scores from a pre-trained ensemble. Lundberg and Lee [174] propose SHAP values as a unified measure of feature importance. Sharma et al. [232] uses SHAP values to select a subset of features that can be useful for secondary model performance characteristics e.g., fairness, robustness etc. Liu et al. [168] propose ControlBurn, which formulates an optimization problem with a Lasso-type regularizer to perform feature selection on a pre-trained forest.

3. Embedded methods *simultaneously* learn the model and the relevant subset of relevant features. Notable among these methods include ℓ_0-based procedures [97, 104, 113, 116] and their variants based on lasso [220, 245, 287] in the linear and additive model settings. Some distributed and stochastic greedy methods have also been explored for subset selection [see, for example, 135, and references therein]. Embedded nonlinear feature selection methods have been explored for neural networks. For example, Chen et al. [47] use an active-set style algorithm for cardinality-constrained feature selection. Other approaches include the use of (group) lasso type methods [56, 73, 154, 230], or reparameterizations of ℓ_0-penalty with stochastic gates [173, 274].

We propose an embedded approach that simultaneously performs feature selection and tree ensemble learning. This joint training approach can be useful for compression, efficient inference, and/or generalization.

Organization. The rest of the chapter is organized as follows. Section 6.3 presents a formulation for feature selection in soft tree ensembles. Section 6.4 discusses our optimization algorithm and its convergence properties. Later, we discuss a scheduling approach that can result in better accuracy and feature sparsity tradeoffs. Sections 6.5 and 6.6 perform experiments on a combination of 15 synthetic and real-world datasets to highlight the usefulness of our proposals.

6.3 Problem Formulation

Feature selection plays a ubiquitous role in modern statistical regression, especially when the number of predictors is large relative to the number of observations. We describe the problem of global feature selection. We assume a data-generating model $p(\mathbf{x}; \mathbf{y})$ over a p-dimensional space, where $\mathbf{x} \in \mathbb{R}^p$ is the covariate and $\mathbf{y}$ is the response. The goal is to find the best function $\boldsymbol{f}(\mathbf{x})$ for predicting $\mathbf{y}$ by minimizing:

$$\min_{\boldsymbol{f} \in \mathcal{F}, \mathcal{Q}} \ \mathbb{E}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}_{\mathcal{Q}}))] \tag{6.3.1}$$

where $\mathcal{Q} \subseteq \{1, 2, \cdots, p\}$ is an unknown (learnable) subset of features of size at most K, $\boldsymbol{f}$ is a learnable non-parametric function from the function class $\mathcal{F}$, and $L : \mathbb{R}^p \times \mathbb{R}^c \to \mathbb{R}$ is a loss function. Recall, in a regression setting $c = 1$, while in multi-class classification setting $c = C$, where C is the number of classes. The principal difficulty in solving (6.3.1) lies in the joint optimization of $(f, \mathcal{Q})$—the number of subsets $\mathcal{Q}$ grows exponentially with p. In addition, the family of functions $\mathcal{F}$ needs to be sufficiently flexible (here, $\mathcal{F}$ is the class of soft tree ensembles with fixed ensemble size m and depth d).

In the context of tree ensembles, our goal for global feature selection is to select a small subset of features across *all* the trees in the ensemble. More specifically, we consider the framework with response $\boldsymbol{y}$ and prediction $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})$, where the function $\boldsymbol{f}$ is parameterized by learnable hyperplane parameters $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{p,m,|\mathcal{I}|}$ across all the split nodes and learnable leaf parameters $\boldsymbol{\mathcal{O}}$ across all trees. Note that m is the number of trees in the ensemble and $|\boldsymbol{\mathcal{I}}|$ represents the number of (split) nodes in each tree. This parameterization of soft trees points to a group structure in $\boldsymbol{\mathcal{W}}$, where the whole slice $\boldsymbol{\mathcal{W}}_{k,:,:}$ in the tensor formulation has to be zero to maintain feature sparsity both across all split nodes in each tree and across the trees in the ensemble — see Fig. 6.3.1. This is a natural feature-wise non-overlapping group structure and allows adaptation of the grouped selection problem in linear models [24, 104] to soft tree ensembles.

Mixed Integer Problem (MIP) Formulation. Let us consider the tensor $\boldsymbol{\mathcal{W}}$. Using binary variables to model feature selection we obtain a regularized loss function:

$$\min_{\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}},\boldsymbol{z}} \hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})] + \lambda_0 \sum_{k\in[p]} z_k, \tag{6.3.2}$$

$$\text{s.t. } ||\boldsymbol{\mathcal{W}}_{k,:,:}||(1-z_k) = 0, z_k \in \{0,1\} \quad k \in [p],$$

where, the binary variable z_k controls whether the k-th feature is on or off via the constraint $||\boldsymbol{\mathcal{W}}_{k,:,:}||(1-z_k) = 0$. $\hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}))] := (1/N)\sum_{n\in[N]} L(\boldsymbol{y}_n, \boldsymbol{f}(\boldsymbol{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}))$ is the empirical loss; and λ_0 is regularization strength. Note MIP formulations [21] can also be

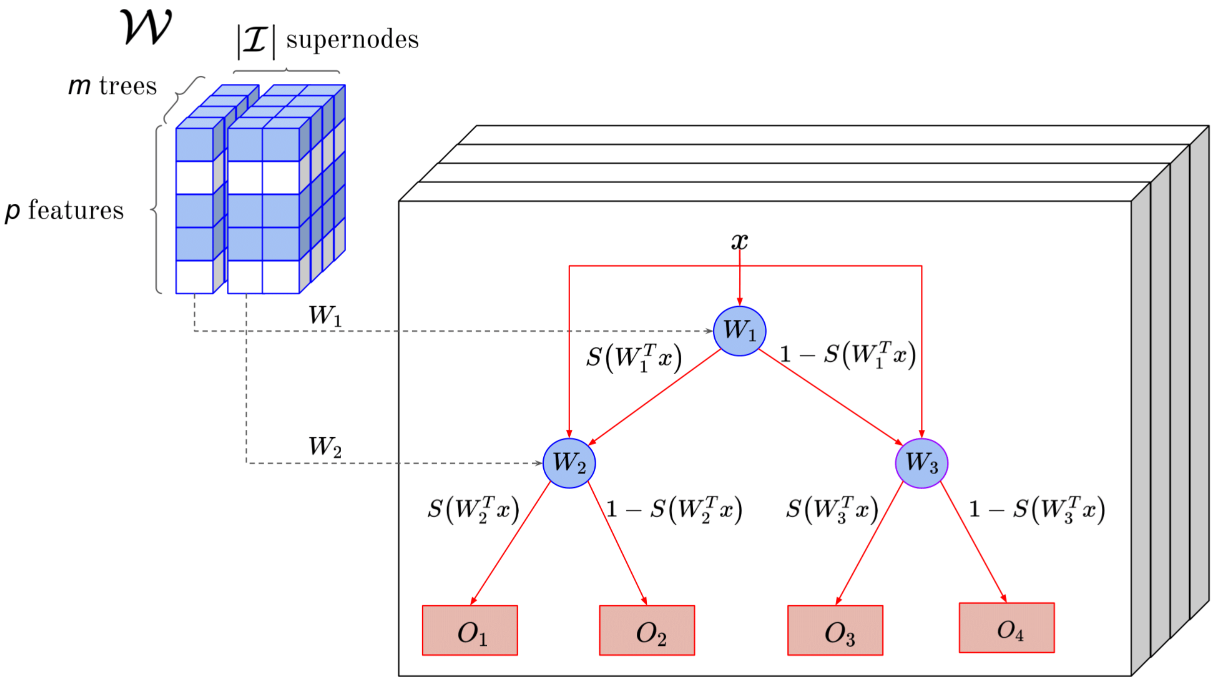


Figure 6.3.1: *Illustration of* `Skinny Trees`. *Each horizontal slice $\boldsymbol{\mathcal{W}}_{k,:,:}$ depicts a single feature. White slices indicate features filtered out by the ensemble while training. Each vertical slice (along the depth of the page), $\boldsymbol{\mathcal{W}}_{:,:,j} = \boldsymbol{W}_j$ corresponds to parameters in j-th (splitting) supernode (blue circles) in the ensemble, eventually producing the routing decisions. The red squares depict leaf nodes. $S(\cdot)$ denotes an activation function, which can be Sigmoid [131] or Smooth-Step [96].*

setup with classical trees under feature selection, but they would be difficult to scale beyond small problems.

Unconstrained formulation of Problem (6.3.2). For computation we consider a penalized version of (6.3.2) involving variables $(\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})$ with the grouped ℓ_0 (pseudo) norm encouraging feature sparsity. We perform end-to-end training via first-order methods (see Sec. 6.4 for details). It has been observed in the linear model setting that a vanilla (group) ℓ_0 penalty may result in overfitting [104]. A possible way to ameliorate this problem is to include an additional ridge regularization for shrinkage [104, 113, 182]. We consider the following group $\ell_0 - \ell_2$ regularized problem:

$$\begin{aligned} \min\nolimits_{\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}} \quad & \hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}))] \\ & + \lambda_0 \sum\nolimits_{k \in [p]} 1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}] + (\lambda_2 / m|\mathcal{I}|) ||\boldsymbol{\mathcal{W}}||_2^2 \end{aligned} \tag{6.3.3}$$

where $1[\cdot]$ is the indicator function, $\lambda_0 \geq 0$ controls the number of features selected, and $\lambda_2 \geq 0$ controls the amount of shrinkage of each group. We normalize λ_2 by the product $m|\boldsymbol{\mathcal{I}}|$ for convenience in hyperparameter tuning.

6.4 End-to-end Optimization Approach

We propose a fast approximate algorithm to obtain high-quality solutions to Problem (6.3.3). We use a proximal (mini-batch) gradient-based algorithm [149] that involves two operations. A vanilla mini-batch GD step is applied to all model parameters followed by a proximal operator applied to the hyperplane parameters $\boldsymbol{\mathcal{W}}$. This sequence of operations on top of backpropagation makes the procedure simple to implement in popular ML frameworks e.g. Tensorflow [1], and contributes to overall efficiency.

6.4.1 Proximal mini-batch gradient descent

We first present the proximal mini-batch GD algorithm for solving Problem (6.3.3) in Algorithm 1. We also discuss computation of the *Prox* operator in line 7 of Algorithm 1. *Prox* finds the global minimum of the optimization problem:

$$\begin{aligned} \boldsymbol{\mathcal{W}}^{(t)} = \text{argmin}_{\boldsymbol{\mathcal{W}}} \quad & (1/2\eta)||\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}^{(t)}||_2^2 \\ & + \lambda_0 \sum\nolimits_{k\in[p]} 1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq 0] \end{aligned} \tag{6.4.1}$$

where $\boldsymbol{\mathcal{Z}}^{(t)} = \boldsymbol{\mathcal{W}}^{(t-1)} - \eta\nabla_{\boldsymbol{\mathcal{W}}} h$. Problem 6.4.1 decomposes across features and a solution for the k-th feature can be found by a hard-thresholding operator given by:

$$H_{\eta\lambda_0}(\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}) = \boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:} \odot 1\left[||\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}|| \geq \sqrt{2\eta\lambda_0}\right]. \tag{6.4.2}$$

We use feature-wise separability for faster computation. The cost of *Prox* is of the order $\mathcal{O}(v)$, where v is the total number of hyperplane parameters being updated (i.e. $v = pm|\mathcal{I}|$). This cost is negligible compared to the computation of the gradients with respect to the same parameters. We implement the optimizer in standard deep learning APIs.

To our knowledge, our proposed approach (and algorithm) for group ℓ_0-based nonlinear

Algorithm 1 *Proximal Mini-batch Gradient Descent for Optimizing (6.3.3).*

```
Input: Data: $X, Y$;
Input: Hyperparameters: $\lambda_0, \lambda_2$, epochs, batch-size, learning rate ($\eta$);
 1: Initialize ensemble with $m$ trees of depth $d$ ($|\mathcal{I}| = 2^d - 1$): $\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}$
 2: for epoch $= 1, 2, \ldots,$ epochs do
 3:     for $batch = 1, \ldots, N$/batch-size do
 4:         Randomly sample a batch: $\boldsymbol{X}_{batch}, \boldsymbol{Y}_{batch}$
 5:         Compute gradient of loss $h$ w.r.t. $\boldsymbol{\mathcal{O}}$, $\mathbf{W}$.
 6:         Update leaves: $\boldsymbol{\mathcal{O}} \leftarrow \boldsymbol{\mathcal{O}} - \eta\nabla_{\boldsymbol{\mathcal{O}}} h$
 7:         Update hyperplanes: $\boldsymbol{\mathcal{W}} \leftarrow Prox(\boldsymbol{\mathcal{W}} - \eta\nabla_{\boldsymbol{\mathcal{W}}} h, \eta, \lambda_0))$
 8:     end for
 9: end for
where $h = \hat{\mathbb{E}}[L(\boldsymbol{Y}_{batch}, \boldsymbol{F}_{batch}] + (\lambda_2/m|\mathcal{I}|)||\boldsymbol{\mathcal{W}}||_2^2$
```

feature selection in soft tree ensembles is novel. Note that Chen et al. [47] considers group ℓ_0 based cardinality constrained formulation for feature selection in neural networks. However, their active-set style approach is very different from our iterative-hard-thresholding based approach. Their toolkit also appears to be up to 900$\times$ slower than our toolkit. In the context of neural network pruning, modifications of iterative-hard-thresholding based approaches [129, 209] have appeared for individual weight pruning in neural networks which is different from feature-selection.

6.4.2 Convergence Analysis of Algorithm 1

We analyze the convergence properties of Algorithm 1 in [118]. For simplicity, we assume that the outcomes are scalar, i.e. $c = 1$ and consider the least squares loss. We also analyze the full-batch algorithm, i.e., we assume batch-size $= N$. Theorem 1 in Ibrahim et al. [118] shows that Algorithm 1 is a convergent (descent) method for a suitably selected learning rate. Here, we note that Problem (6.3.3) is non-convex and non-smooth. Moreover, the activation function and therefore $\hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}))]$ are not twice differentiable everywhere. These lead to technical challenges in proof, given in Appendix in Ibrahim et al. [118].

6.4.3 Dense-to-Sparse Learning (DSL)

Prior work in feature selection [154] recommend an interesting multi-stage approach: Train a dense model completely and then learn a progessively sparser model. At each sparsity level, the model is trained till convergence. This approach appears to effectively leverage favorable generalization properties of the dense solution and preserves them after drifting into sparse local minima [154]—in particular, this seems to perform better than sparsely training from scratch. However, this approach can be expensive as it requires learning a dense model completely before starting the sparse-training process—the training runtime is likely to increase for higher sparsity settings (i.e., fewer number of features).

To reduce the computational cost of the above approach, we propose single-stage approach

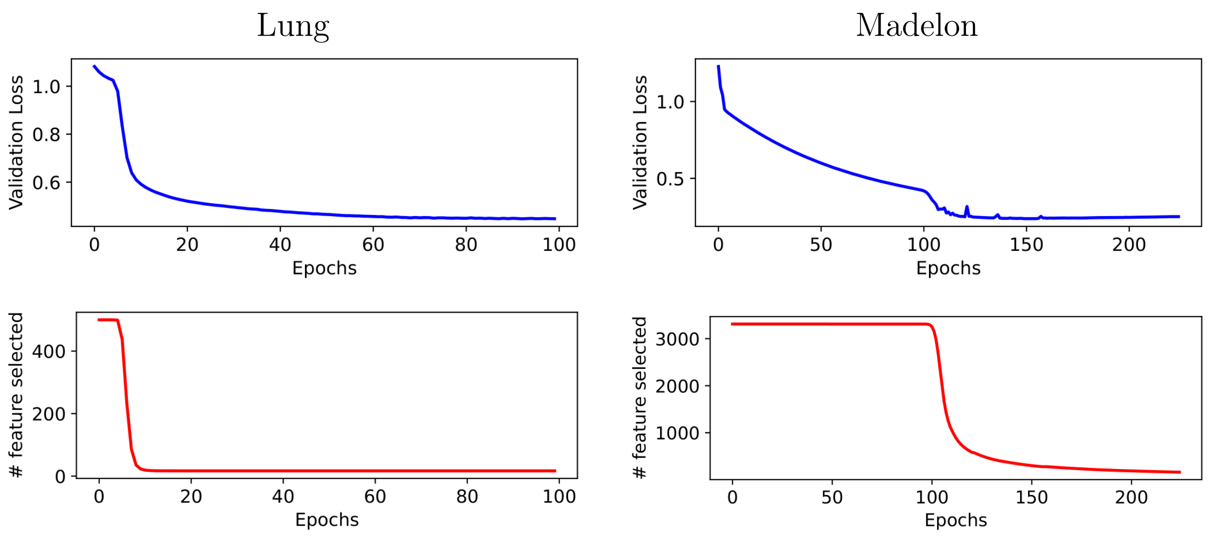


Figure 6.4.1: *Trajectory of validation loss and feature sparsity during training with dense-to-sparse learning.*

based on dense-to-sparse learning (DSL). To this end, we anneal the sparsity-inducing penalty λ_0 from small to large values $(0 \to \gamma)$ during the course of training. We use an exponential annealing scheduler of the form: $\lambda_0 = \gamma(1 - \exp(-s * t))$, where γ is the largest value of regularization penalty (corresponds to a fully sparse soft tree), s controls the rate of increase of the regularization penalty and t denotes the iteration step.

We show the trajectory of the validation loss and the number of features selected during training with dense-to-sparse learning in Figure 6.4.1. We empirically observed this scheduler to result in better out-of-sample accuracy and feature-sparsity tradeoffs (see Figure 6.6.1 in Sec. 6.6.5).

6.5 Synthetic Experiments

We first evaluate our proposed method using data with correlated features. In real-world, high-dimensional datasets, features are often correlated. Such correlations pose challenges for feature selection. Existing tree ensemble toolkits, e.g., XGBoost and Random Forests, based on feature importance scores, may produce misleading results [291]—any of the

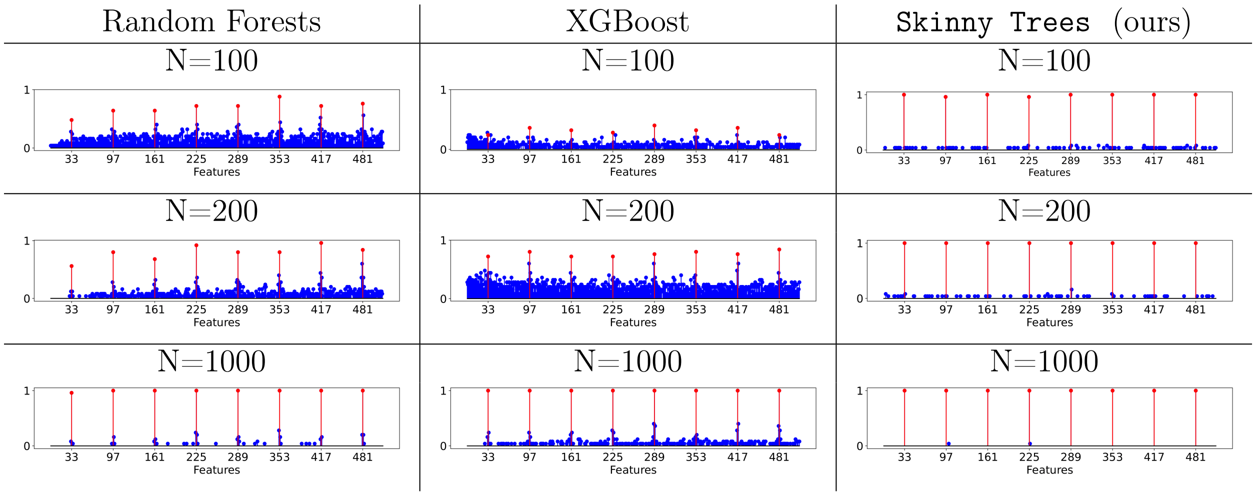


Figure 6.5.1: *Features selected by Random Forests, XGBoost and* `Skinny Trees` *for different sample sizes*

Table 6.5.1: *Test MSE, feature sparsity and support recovery metrics (F1-score) for a linear setting with correlated design matrix.* `Skinny Trees` *outperforms feature-importance-based methods across all metrics.*

σ	p	N	Model	Test MSE	#features	F1-score
0.7	512	100	RF	6.49 ± 0.19	79 ± 17	0.21 ± 0.02
			XGBoost	8.65 ± 0.27	32 ± 9	0.18 ± 0.03
			`Skinny Trees`	**0.65** ± 0.12	**12** ± 1	**0.86** ± 0.04
		200	RF	4.90 ± 0.15	40 ± 8	0.35 ± 0.03
			XGBoost	5.97 ± 0.12	110 ± 21	0.18 ± 0.02
			`Skinny Trees`	**0.34** ± 0.00	**11** ± 1	**0.89** ± 0.03
		1000	RF	2.97 ± 0.03	11 ± 1	0.84 ± 0.02
			XGBoost	1.81 ± 0.02	24 ± 1	0.50 ± 0.02
			`Skinny Trees`	**0.26** ± 0.00	**8** ± 0	**1.00** ± 0.00
0.5	256	100	RF	6.24 ± 0.13	42 ± 11	0.35 ± 0.03
			XGBoost	7.93 ± 0.27	35 ± 10	0.25 ± 0.02
			`Skinny Trees`	**0.45** ± 0.06	**10** ± 1	**0.89** ± 0.03
		200	RF	4.40 ± 0.13	18 ± 3	0.61 ± 0.04
			XGBoost	5.61 ± 0.13	67 ± 12	0.26 ± 0.02
			`Skinny Trees`	**0.31** ± 0.00	**12** ± 1	**0.87** ± 0.04
		1000	RF	2.90 ± 0.02	9 ± 0	0.94 ± 0.01
			XGBoost	1.45 ± 0.01	10 ± 0	0.91 ± 0.01
			`Skinny Trees`	**0.26** ± 0.00	**8** ± 0	**1.00** ± 0.00

correlated features can work as a splitting variable, and the feature importance scores can get distributed (and hence deflated) among the correlated features. Below, we consider a setting with correlated features and demonstrate the strong performance of `Skinny Trees` in terms of true support recovery on synthetic data.

We evaluate our approach in a setting where the underlying data comes from a sparse linear model. We generate the data matrix, $\boldsymbol{X} \in \mathbb{R}^{N\times p}$ with samples drawn from a multivariate normal distribution $\mathcal{N}(0, \boldsymbol{\Sigma})$ where entries of the covariance matrix $\boldsymbol{\Sigma}$ are given by $\Sigma_{ij} = \sigma^{|i-j|}$. We construct the response variable $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^* + \epsilon$ where $\epsilon_i, i \in [N]$ are drawn independently from $\mathcal{N}(0, 0.5)$. The locations of nonzero entries of $\boldsymbol{\beta}^*$ are equi-spaced in $[p]$, with each nonzero entry one, and $\|\boldsymbol{\beta}^*\|_0 = 8$. We experiment with a range of training set sizes $N \in \{100, 200, 1000\}$, correlation strengths $\sigma \in \{0.5, 0.7\}$, and number of total features $p \in \{256, 512\}$. We evaluate the final performance averaged across 25 runs in terms of (i) test MSE, (ii) number of features selected and (iii) support recovery (computed via the F1-score between the true and recovered support). More details are in Appendix Sec. 6.8.1.

`Skinny Trees` significantly outperforms both Random Forests and XGBoost in all three measures across various settings. With `Skinny Trees`, we observe a 5-15 fold improvement in MSE performance and $9\% - 65\%$ improvement in the support recovery metric (F1-score). Table 6.5.1 shows that even if the features are correlated, `Skinny Trees` successfully recovers the true support with high probability. We also visualize this in Figure 6.5.1. Indices corresponding to those in the true support are depicted in red. This confirms the usefulness of our end-to-end feature selection approach.

6.6 Real Data Experiments

We study the performance of `Skinny Trees` on real-world datasets and compare against popular competing methods. We make the following comparisons: (i) Single `Skinny Tree` vs other single tree baseline approaches with a limit on number of features, (ii) `Skinny Trees` vs dense soft trees, (iii) `Skinny Trees` vs wrapper-based feature selection tree toolkits, (iv) `Skinny Trees` vs neural network based embedded feature selection toolkits, (v) Ablation study for dense-to-sparse learning for feature selection.

Implementation. Skinny Trees are implemented in TensorFlow Keras. Our code for Skinny Trees is available at https://github.com/mazumder-lab/SkinnyTrees.

Datasets. We use 14 open-source classification datasets (binary and multiclass) from various domains with a range of number of features: $20 - 100000$. Dataset details are in Table 6.8.2 in Appendix.

Tuning, Toolkits, and Details. For all the experiments, we tune the hyperparameters using Optuna [5] with random search. The number of selected features affects the AUC. Therefore, to treat all the methods in a fair manner, we tune the hyperparameter that controls the sparsity level using Optuna which optimizes the AUC across different K's (budget on number of selected features) e.g., $0.25p$ or $0.50p$ on a held-out validation set. Details are in the Appendix.

6.6.1 Studying a single tree

We first study feature selection for a single tree on 4 classification tasks. We study the performance of Skinny Tree (a single soft tree with group $\ell_0 - \ell_2$ regularization).

Competing Methods. We compare against:

1. Decision tree with hyperplane splits (TAO [37]) using ℓ_1 regularization for node-level feature selection.

2. Soft Tree with a Group Lasso [230, 282] regularization given by $\frac{\lambda_1}{\sqrt{m|\mathcal{I}|}}\sum_{k\in[p]} \|\boldsymbol{\mathcal{W}}_{k,:,:}\|_2$.

Results. The numbers for classical-tree based TAO with ℓ_1 regularization and soft tree with Group Lasso regularization are shown in Table 6.6.1 for 50% sparsity budget. Results for Skinny Tree are also shown. We see a huge gain in test AUC performance across all 4 datasets with Skinny Tree in comparison with TAO and group lasso variant of a soft tree.

Table 6.6.1: *Test AUC for TAO with ℓ_1 regularization, single soft tree with Group Lasso and* `Skinny Tree` *(a single soft tree with Group $\ell_0 - \ell_2$).*

	Classical Tree TAO	Soft Tree w/ Group Lasso	`Skinny Tree`
Churn	58.36	76.23	**89.35**±0.15
Satimage	58.53	83.89	**88.66**±0.05
Texture	58.90	93.83	**98.42**±0.01
Mice-protein	57.13	87.88	**99.19**±0.00

This confirms that in the context of feature selection at the ensemble level, a node-level ℓ_1 penalty is sub-optimal. Similarly, it also suggests that joint selection and shrinkage using Group Lasso can be less useful than Group $\ell_0 - \ell_2$.

6.6.2 `Skinny Trees` vs Dense Soft Trees

In this section, we compare our sparse trees with dense soft trees. For dense soft trees, we use FASTEL [115] (an efficient state-of-the-art toolkit for training soft tree ensembles). We present test AUC performances in Table 6.6.2. `Skinny Trees` matches or outperforms dense soft trees in 10 datasets. Notably, we observe a 30% gain in test AUC on Madelon dataset with `Skinny Trees`. We also observe 11% improvements in test AUC on Cll and Gli datasets. Additionally, sparse trees achieve $1.3\times - 620\times$ feature compression on 10 datasets. Note that

Table 6.6.2: *Test AUC for* `Skinny Trees` *vs* dense *Soft Trees. We also report feature compression.*

	Dense Trees	`Skinny Trees`	Compression
Churn	91.15±0.09	**93.20**±0.08	1.8×
Gisette	**99.81**±0.003	**99.81**±0.002	1.5×
Arcene	89.57±0.11	**90.80**±0.30	2×
Dorothea	90.67±0.03	**92.15**±0.25	2.7×
Madelon	65.32±0.15	**95.44**±0.05	26×
Smk	**84.10**±0.16	79.29±0.22	253×
Cll	81.70±0.82	**92.86**±0.31	189×
Gli	88.65±0.90	**99.80**±0.07	619×
Lung	99.40±0.09	**99.80**±0.03	253×
Tox	99.19±0.04	**99.74**±0.02	189×

Table 6.6.3: *Test AUC (%) performance of* `Skinny Trees` *and feature-importance-based toolkits for* **trees** *for 25% feature budget ($K = 0.25p$). Bold and italics indicates best and runner-up models respectively.*

Case	Dataset	Random Forests	XGBoost	LightGBM	CatBoost	`Skinny Trees`
$N < p$	Lung	93.80±0.28	86.38±0.48	80.83±1.87	*94.72*±0.56	**99.80**±0.03
	Tox	94.52±0.14	*97.10*±0.09	95.94±0.54	95.95±0.14	**99.74**±0.02
	Arcene	74.80±0.36	76.36±0.16	*76.92*±0.36	76.64±0.22	**80.80**±0.30
	Cll	94.08±0.27	*94.21*±0.18	55.17±1.14	**94.41**±0.26	92.86±0.31
	Smk	77.78±0.20	76.88±0.40	67.29±0.91	*78.44*±0.41	**79.29**±0.22
	Gli	87.35±1.08	82.37±1.47	71.28±2.05	*91.31*±0.73	**99.80**±0.07
	Dorothea	*89.71*±0.12	89.09±0.09	88.14±0.18	88.50±0.27	**90.87**±0.02
$N > p$	Churn	83.79±0.24	*88.68*±0.06	86.33±0.08	83.73±0.06	**91.38**±0.08
	Satimage	97.62±0.005	**98.23**±0.01	94.00±0.05	95.11±0.05	*98.05*±0.01
	Texture	99.60±0.003	*99.94*±0.001	96.14±0.03	94.90±0.07	**99.97**±0.002
	Mice-protein	99.30±0.01	**99.77**±0.01	89.59±0.22	95.03±0.07	*99.59*±0.02
	Isolet	99.17±0.002	99.86±0.002	97.62±0.003	*99.89*±0.001	**99.94**±0.01
	Madelon	94.11±0.02	*94.65*±0.01	86.46±0.08	**96.41**±0.01	94.14±0.09
	Gisette	98.99±0.004	*99.64*±0.004	98.09±0.50	99.57±0.01	**99.81**±0.002
	Average	91.75	91.65	84.56	91.76	**94.72**

in soft trees, feature compression has a direct impact on model compression—this has reduced storage requirements and results in faster inference. We observed up to 10× faster inference times for `Skinny Trees` compared to dense soft trees for compression rates of 1.5×−620×.

6.6.3 `Skinny Trees` vs Classical Trees

We compare `Skinny Trees` against wrapper methods for feature selection as available from ensembles of classical trees (e.g., Random Forests, XGBoost, LightGBM, and CatBoost) on real-world datasets. For `Skinny Trees`, we use the combined dense-to-sparse scheduler. The tuning protocol and hyperparameters for all methods are reported in the Appendix Sec. 6.8.2.3. The results are in Table 6.6.3. `Skinny Trees` leads on 10 datasets. In contrast, other methods lead on 2 datasets. In terms of test AUC, `Skinny Trees` outperforms LightGBM by 10.2% (up to 37.7%), XGBoost by 3.1% (upto 17.4%), Random Forests by 3% (up to 12.5%) and CatBoost by 3% (up to 8.5%). Overall, `Skinny Trees` provides a strong alternative to existing wrapper-based methods.

Additional comparison with ControlBurn [168] is included in Appendix Sec. 6.8.2.6.

Table 6.6.4: *Test AUC (%) performance of* `Skinny Trees` *and embedded feature selection methods from* **neural networks** *(LassoNet, AlgNet, DFS) for 25% feature budget.*

Case	Dataset	LassoNet	AlgNet	DFS	`Skinny Trees`
$N < p$	Lung	99.56±0.02	56.72±1.34	*98.05*±0.36	**99.80**±0.03
	Tox	99.63±0.03	51.01±0.70	*99.13*±0.24	**99.74**±0.02
	Arcene	66.26±0.29	51.00±1.77	69.37±0.59	**80.80**±0.30
	Cll	**95.04**±0.24	64.96±2.54	92.85±0.32	*92.86*±0.31
	Smk	**85.56**±0.31	53.92±2.16	*79.62*±0.29	79.29±0.22
	Gli	*97.78*±0.56	61.37±4.27	92.25±0.64	**99.80**±0.07
	Dorothea	out of mem.	81.74±0.91	*85.18**	**90.87**±0.02
$N > p$	Churn	67.34±1.58	70.10±0.91	*85.70*±0.52	**91.38**±0.08
	Satimage	94.73±0.19	95.30±0.20	*97.39*±0.04	**98.05**±0.01
	Texture	98.02±0.40	76.24±1.94	*99.63*±0.04	**99.97**±0.002
	Mice-protein	94.90±0.26	89.07±0.59	*99.04*±0.03	**99.59**±0.02
	Isolet	99.64±0.01	70.21±2.92	*99.92*±0.00	**99.94**±0.01
	Madelon	81.15±2.53	68.55±1.42	*92.73*±0.45	**94.14**±0.09
	Gisette	99.81±0.002	73.49±1.54	*99.72**	**99.81**±0.002
	Average	90.43**	68.83	92.18	**94.72**

*DFS is very time-consuming to run, we report the test AUC for best trial (based on validation AUC) during tuning on Gisette and Dorothea.
**Adjusted Average: $\frac{90.72}{(94.72*14-90.87)/13} * 94.72 = 90.43$.

`Skinny Trees` also outperforms ControlBurn, achieving 2% (up to 6%) improvement in AUC.

6.6.4 `Skinny Trees` vs Neural Networks

In this chapter, we pursue embedded feature selection methods for *tree ensembles*. However, for completeness, we compare `Skinny Trees` against some state-of-the-art embedded feature selection methods from neural networks, namely LassoNet [154], AlgNet [56] and DFS [47]. Details are in Appendix Sec. 6.8.2.4.

Results. We report AUC performance for 25% feature budget in Table 6.6.4. `Skinny Trees` leads across many datasets. In terms of test AUC, `Skinny Trees` outperforms LassoNet by 4.3% (up to 24%), AlgNet by 25.9% (up to 49%), and DFS by 2.6% (up to 11.4%).

6.6.5 Dense-to-Sparse Learning

We perform an ablation study in which we compare the predictive performance achieved with dense-to-sparse learning (DSL) over a range of feature selection budgets. Tuning details are in the Appendix Sec. 6.8.2.5. The results are reported in Figure 6.6.1. Interestingly, we improve in test AUC across a range of feature selection budgets with dense-to-sparse learning over fixed regularization tuning.

6.6.6 Discussion on training times.

`Skinny Trees` is very competitive in terms of training times in comparison to existing toolkits. We compared timings on a single Tesla V100 GPU. For example, on dorothea dataset, `Skinny Trees` trained in under 3 minutes for optimal hyperparameter setting. XGBoost took 10 minutes. In contrast, DFS took 45 hours.

6.7 Conclusion

We introduce an end-to-end optimization approach for joint feature selection and tree ensemble learning. Our approach is based on differentiable trees with group $\ell_0 - \ell_2$ regularization. We use a simple but effective proximal mini-batch gradient descent algorithm and present

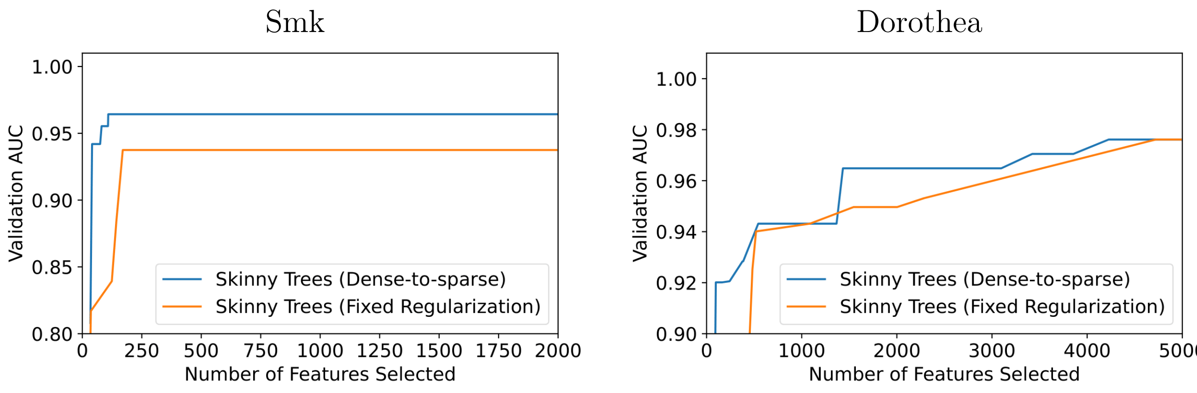


Figure 6.6.1: *Performance without/with Dense-to-sparse learning for different feature selection budgets.*

convergence guarantees. We propose a dense-to-sparse regularization scheduling approach that can lead to better feature-sparsity-vs-accuracy tradeoffs. We demonstrate on various datasets that our toolkit `Skinny Trees` can improve feature selection over several state-of-the-art wrapper-based feature selection methods in trees and embedded feature selection methods in neural networks.

6.8 Appendix

Notation. Table 6.8.1 lists the notation used throughout the chapter.

Table 6.8.1: *List of notation used.*

Notation	**Space or Type**	**Explanation**
$[n]$	Set	The set of integers $\{1, 2, ..., n\}$.
$\mathbf{1}_m$	$\mathbb{R}^m$	Vector with all coordinates equal to 1.
$\boldsymbol{U}$	$\mathbb{R}^{m,n}$	Matrix with elements $((U_{ij}))$
$\boldsymbol{u} \cdot \boldsymbol{v}$	$\mathbb{R}$	A dot product between two vectors $\boldsymbol{u}, \boldsymbol{v}$.
$\boldsymbol{U} \cdot \boldsymbol{v}$	$\mathbb{R}^n$	A dot product between a matrix $\boldsymbol{U} \in R^{m,n}$ and a vector $\boldsymbol{v} \in \mathbb{R}^m$ is denoted as $\boldsymbol{U} \cdot \boldsymbol{v} = \boldsymbol{U}^T v \in \mathbb{R}^n$.
$\mathcal{X}$	$\mathbb{R}^p$	Input feature space.
$\mathcal{Y}$	$\mathbb{R}^c$	Output (label) space.
m	$\mathbb{Z}_{>0}$	Number of trees in `Skinny Trees`.
$\boldsymbol{f}(\boldsymbol{x})$	Function	The output of `Skinny Trees`, a function that takes an input sample and returns a logit which corresponds to the average of the output of all the trees in the ensemble. Formally, $\boldsymbol{f} : \mathcal{X} \rightarrow \mathbb{R}^c$.
$\boldsymbol{f}^j(\boldsymbol{x})$	Function	A single perfect binary tree which takes an input sample and returns a logit, i.e., $\boldsymbol{f}^j : \mathcal{X} \rightarrow \mathbb{R}^c$.
d	$\mathbb{Z}_{>0}$	The depth of tree $\boldsymbol{f}^j$.
$\mathcal{I}^j$	Set	The set of internal (split) nodes in $\boldsymbol{f}^j$.
$\mathcal{I}$	Set	The set of internal (split) supernodes in $\boldsymbol{f}$.
$\mathcal{L}^j$	Set	The set of leaf nodes in $\boldsymbol{f}^j$.
$\mathcal{A}(i)$	Set	The set of ancestors of node i.
$\{x \rightarrow i\}$	Event	The event that sample $x \in \mathbb{R}^p$ reaches node i.
$\boldsymbol{w}_i$	$\mathbb{R}^p$	Weight vector of internal node i (trainable). Defines the hyperplane split used in sample routing.
$\boldsymbol{W}_i$	$\mathbb{R}^{p,m}$	Matrix of all weights in the internal supernode i of the ensemble in the tensor-formulation.
$\boldsymbol{\mathcal{W}}$	$\mathbb{R}^{p,m,\|\mathcal{I}\|}$	Tensor of all weights across all internal supernodes in the ensemble.
$\boldsymbol{\mathcal{W}}_{k,:,:}$	$\mathbb{R}^{m,\|\mathcal{I}\|}$	Matrix of all weights for k-th feature/covariate across all internal supernodes in the ensemble.
S	Function	Activation function $\mathbb{R} \rightarrow [0, 1]$
$S(\boldsymbol{w}_i \cdot \boldsymbol{x})$	$[0, 1]$	Probability (proportion) that internal node i routes x to the left.
$S'(v)$	Function	The derivative of $S(v)$
$[l \swarrow i]$	Event	The event that leaf l belongs to the left subtree of node $i \in \mathcal{I}$.
$[l \searrow i]$	Event	The event that leaf l belongs to the right subtree of node $i \in \mathcal{I}$.
$\boldsymbol{o}_l$	$\mathbb{R}^c$	Leaf l's weight vector (trainable).
$\boldsymbol{O}_l$	$\mathbb{R}^{m,c}$	Matrix of weights in superleaf l.
$\boldsymbol{\mathcal{O}}$	$\mathbb{R}^{m,c,\|\mathcal{L}\|}$	Tensor of weights across the superleaves in the ensemble.
L	Function	Loss function for training (e.g., cross-entropy).
$\mathcal{Q}$	Set	Unknown (learnable) subset of features of size at most K.
z_k	$\{0, 1\}$	Binary variable controls whether k-th feature is on or off in Problem (2).
$\boldsymbol{z}$	$\{0, 1\}^p$	Binary vector controlling which features are on or off in Problem (2).
λ_0	$\mathbb{R}_{\geq 0}$	Non-negative ℓ_0 regularization parameter controlling the number of features selected in Problem (3)
λ_2	$\mathbb{R}_{\geq 0}$	Non-negative ℓ_2 regularization parameter controlling the shrinkage in Problem (3)
λ_1	$\mathbb{R}_{\geq 0}$	Non-negative ℓ_0 regularization parameter controlling the number of features selected and shrinkage in Problem (6.8.1)
γ	$\mathbb{R}_{\geq 0}$	Non-negative scaling parameter for the exponential ramp-up of ℓ_0-penalty in dense-to-sparse learning.
s	$\mathbb{R}_{\geq 0}$	Non-negative temperature parameter for controlling the ramping rate of the ℓ_0-penalty in dense-to-sparse learning.
η	$\mathbb{R}_{\geq 0}$	Learning rate parameter for proximal mini-batch gradient descent.

6.8.1 Experimental Details for section 6.5

Data Design. We generate the data matrix, $\boldsymbol{X} \in \mathbb{R}^{N \times p}$ with samples randomly drawn from a multivariate normal distribution $\mathcal{N}(0, \boldsymbol{\Sigma})$ with a correlated design matrix $\boldsymbol{\Sigma}$ whose values are defined by $\Sigma_{ij} = \sigma^{|i-j|}$. We construct the response variable $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^* + \epsilon$ where the values of the noise $\epsilon_i, i = [N]$ are drawn independently from $\mathcal{N}(0, 0.5)$. We use a known sparsity $\boldsymbol{\beta}^*$ with equi-spaced nonzeros, where $\|\boldsymbol{\beta}^*\|_0 = 8$.

Simulation Procedure. We experiment with a range of training set sizes $N \in \{100, 200, 1000\}$, correlation strengths $\sigma \in \{0.5, 0.7\}$, and number of total features $p \in \{256, 512\}$. For each setting, we run 25 simulations with randomly generated samples. We evaluate on $10,000$ test samples drawn from the data generating model described above. Out of N samples, we allocate 80% for training and 20% for validation for model selection. For each simulation, we perform 500 tuning trials (hyperparameters are given below) and select the model with the smallest validation mean squared error (MSE). We evaluate the final performance in terms of (i) test MSE, (ii) number of features selected and (iii) support recovery via f1-score between the true support and the recovered feature set. We compute averages and standard errors across the 25 simulations to report final results.

`Skinny Trees` with DSL.

- Number of trees: Discrete uniform with range $[1, 50]$,
- Depths: Discrete uniform with range $[1, 5]$,
- Batch sizes: $16b$ with b uniform over the range $\{1, 2, \ldots, 8\}$,
- Number of Epochs: Discrete uniform with range $[5, 500]$,
- λ_0: exponentially ramped up $0 \rightarrow 100$ with temperature distributed as Log uniform in the range $[10^{-4}, 0.1]$ for group $\ell_0 - \ell_2$ with DSL,
- λ_2: Log uniform over the range $[10^{-2}, 10^{2}]$ for group $\ell_0 - \ell_2$,
- Learning rates, lr: Log uniform over the range $[10^{-3}, 10^{-1}]$.

XGBoost, Random Forests.

- Number of trees: Discrete uniform with range $[1, 100]$ for XGBoost and Random Forests,
- Depths: Discrete uniform with range $[1, 10]$ for XGBoost and Random Forests,
- Learning rates: Discrete uniform with range $[10^{-4}, 1.0]$ for XGBoost,
- Feature importance threshold: Log uniform over the range $[10^{-7}, 10^{-1}]$ for XGBoost and Random Forests.
- Subsample: Uniform over the range $[0.5, 1.0]$.

6.8.2 Experimental Details for section 6.6

6.8.2.1 Datasets, Computing Setup and Tuning Setup

Table 6.8.2: *Classification datasets*

Dataset	N	p	C
Churn	5,000	20	2
Satimage	6,435	36	6
Texture	5,500	40	11
Mice-protein	1,080	77	8
Isolet	7,797	617	26
Madelon	2,600	500	2
Lung	203	3,312	5
Gisette	7,000	5,000	2
Tox	171	5,748	4
Arcene	200	10,000	2
CLL	111	11,340	3
SMK	187	19,993	2
GLI	85	22,283	2
Dorothea	1,150	100,000	2

Datasets. We consider 5 classification datasets from the Penn Machine Learning Benchmarks (PMLB) repository [200]. These are Churn, Satimage, Mice-protein, Isolet and Texture; Mice-protein and Isolet were used in prior feature selection literature [154]. We used fetch_openml in Sklearn to download the full datasets. We used 4 datasets from

NIPS2003 feature selection challenge [83]. These are Arcene, Madelon, Gisette, Dorothea. We used 5 datasets (Smk, Cll, Gli, Lung, Tox) from the feature selection datasets given in this repo[2]. The datasets contain continuous, categorical and binary features. The metadata was used to identify the type of features and categorical features were one-hot encoded. For first non-NIPS2003 datasets, we randomly split each of the dataset into 64% training, 16% validation and 20% testing sets. For the datasets from NIPS2003, we split the training set into 80% training and 20% validation and treated the original validation set as the test set. The labels for original test set were unavailable, hence we discarded the original test set. The continuous features in all datasets were z-normalized based on training set statistics. A summary of the 14 datasets considered is in Table 6.8.2.

Computing Setup. We used a cluster running Ubuntu 7.5.0 and equipped with Intel Xeon Platinum 8260 CPUs and Nvidia Volta V100 GPUs. For all experiments of Sec. 6.5 and 6.6, each job involving `Skinny Trees`, Random Forests, XGBoost, LightGBM, CatBoost and neural networks were restricted to 4 cores and 64GB of RAM. Jobs involving `Skinny Trees` and neural networks on larger datasets (Gisette, Dorothea) were run on Tesla V100 GPUs.

Tuning. The tuning was done in parallel over the competing models and datasets. The number of selected features affects the AUC. Therefore, to treat all the methods in a fair manner, we tune the hyperparameter that controls the sparsity level using Optuna [5] which optimizes the overall AUC across different Ks. A list of all the tuning parameters and their distributions is given for every experiment below.

6.8.2.2 Experimental Details for Sec. 6.6.1

Classical tree: TAO Implementation. Given that the authors of TAO do not open-source their implementation, we have written our own implementation of the TAO algorithm

[2]https://jundongl.github.io/scikit-feature/datasets.html

proposed by [37], following the procedure outlined in Sec. 3.1 of [288]. For TAO, we use oblique (i.e. hyperplane splits) decision trees with constant leaves. We take as an initial tree a complete binary tree with random parameters at each node. We perform TAO updates until maximum number of iterations are reached (i.e. there is no other stopping criterion). TAO uses an ℓ_1-regularized logistic regression to solve the decision node optimization (using LIBLINEAR [71]) where $\lambda \geq 0$ parameter (controlling node-level sparsity of the tree) is used as a regularization parameter ($C = 1/\lambda$). We tune depth in the range $[1, 10]$, $\lambda \in [10^{-5}, 10^{5}]$ and number of maximum iterations in the range $[20, 100]$. We perform 500 tuning trials. We find the optimal trial that satisfies 50% sparsity budget.

Soft tree with group lasso. We compare group ℓ_0-based method with a competitive benchmark: the convex group lasso regulurizer, popularly used in high-dimensional statistics literature [282]. We consider group-lasso regularization in the context of soft trees: $(\lambda_1/\sqrt{m|\mathcal{I}|})\sum_{k\in[p]} \|\boldsymbol{\mathcal{W}}_{k,:,:}\|_2$. Although group lasso has not been used in soft trees, it has been used for feature selection in neural networks [230]. However, their proposal to use GD on a group ℓ_1 regularized objective does not lead to feature selection as highlighted by [154]. For a fairer comparison, we implement our own proximal mini-batch GD method for group lasso in the context of soft trees, which actually leads to feature selection.

We consider the following optimization problem with group-lasso regularization in the context of soft trees:

$$\min_{\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}} \ \hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})] + (\lambda_1/\sqrt{m|\mathcal{I}|})\sum_{k\in[p]} \|\boldsymbol{\mathcal{W}}_{k,:,:}\|_2 . \tag{6.8.1}$$

where λ_1 is a non-negative regularization parameter that controls both shrinkage and sparsity. We solve it with the algorithm presented in Alg. 2.

Algorithm 2 *Proximal Mini-batch Gradient Descent for Optimizing (6.8.1).*

```
Input: Data: $X, Y$;    Hyperparameters: $\lambda_1$, epochs, batch-size ($b$), learning rate ($\eta$);
1: Initialize ensemble with $m$ trees of depth $d$ ($|\mathcal{I}| = 2^d - 1$): $\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}$
2: for epoch $= 1, 2, \ldots,$ epochs do
3:     for batch $= 1, \ldots, N/b$ do
4:         Randomly sample a batch: $\boldsymbol{X}_{batch}, \boldsymbol{Y}_{batch}$
5:         Compute gradient of loss $g$ w.r.t. $\boldsymbol{\mathcal{O}}$, $\mathbf{W}$, where $g = \hat{\mathbb{E}}[L(\boldsymbol{Y}_{batch}, \boldsymbol{F}_{batch}]$.
6:         Update leaves: $\boldsymbol{\mathcal{O}} \leftarrow \boldsymbol{\mathcal{O}} - \eta\nabla_{\boldsymbol{\mathcal{O}}} g$
7:         Update hyperplanes: $\boldsymbol{\mathcal{W}} \leftarrow$ S-Prox$(\boldsymbol{\mathcal{W}} - \eta\nabla_{\boldsymbol{\mathcal{W}}} g, \eta, \lambda_1))$
8:     end for
9: end for
```

S-Prox in Algorithm 2 finds the global minimum of an optimization problem of the form:

$$\boldsymbol{\mathcal{W}}^{(t)} = \text{argmin}_{\boldsymbol{\mathcal{W}}} \frac{1}{2\eta} \left\| \boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}^{(t)} \right\|_2^2 + \frac{\lambda_1}{\sqrt{m|\mathcal{I}|}} \sum_{k=1}^{p} \|\boldsymbol{\mathcal{W}}_{k,:,:}\|_2 \tag{6.8.2}$$

where $\boldsymbol{\mathcal{Z}}^{(t)} = \boldsymbol{\mathcal{W}}^{(t-1)} - \eta\nabla_{\boldsymbol{\mathcal{W}}} g$ and $g = \hat{\mathbb{E}}[L(\boldsymbol{Y}_{batch}, \boldsymbol{F}_{batch}]$. This leads to a feature-wise soft-thresholding operator given by:

$$S_{\frac{\eta\lambda_1}{\sqrt{m|\mathcal{I}|}}}(\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}) = \begin{cases} \boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:} - \frac{\eta\lambda_1}{\sqrt{m|\mathcal{I}|}} \frac{\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}}{\left\|\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}\right\|} & \text{if } \left\|\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}\right\| \geq \frac{\eta\lambda_1}{\sqrt{m|\mathcal{I}|}} \\ 0 & \text{otherwise} \end{cases} \tag{6.8.3}$$

In these experiments, we used a single tree and mini-batch PGD *without* any dense-to-sparse scheduler for both models i.e, (i) Soft tree with group lasso (ii) `Skinny Tree`. We tune the key hyperparameters, which are given below.

- Batch sizes: $64 * b$ with b uniform over the range $\{1, 2, \ldots, 64\}$,
- Learning rates for mini-batch PGD: Log uniform over the range $[10^{-2}, 10]$,
- Number of Epochs: Discrete uniform with range $[5, 1000]$,
- λ_0: Log uniform over the range $[10^{-3}, 10]$ for group $\ell_0 - \ell_2$ for `Skinny Tree`,
- λ_2: Log uniform over the range $[10^{-2}, 10^2]$ for group $\ell_0 - \ell_2$ for `Skinny Tree`,
- λ_1: Log uniform over the range $[10^{-3}, 10]$ for group lasso.

6.8.2.3 Tuning Details for Sec. 6.6.2 and 6.6.3

We describe the tuning grid for these experiments below.

`Skinny Trees`.

- Number of trees: Discrete uniform with range $[1, 100]$,
- Depths: Discrete uniform with range $[1, 5]$,
- Batch sizes: Uniform over the set $\{16, 64, 256, 1024\}$,
- Number of Epochs: Discrete uniform over the range $[5, 1000]$ (Note tuning over epochs is important to achieve some trials with desired sparsity for dense-to-sparse learning, we do not use any validation loss early stopping as that is less robust during averages across runs/seeds in terms of feature support.),
- λ_0: exponentially ramped up $0 \rightarrow 1$ with temperature distributed as Log uniform in the range $[10^{-4}, 1]$ for group $\ell_0 - \ell_2$ with DSL,
- λ_2: Log uniform over the range $[10^{-3}, 1]$ for group $\ell_0 - \ell_2$,
- Learning rate (lr) for minibatch PGD: Log uniform over the range $[10^{-2}, 10]$.

XGBoost, LightGBM, CatBoost, Random Forests.

- Number of trees: Discrete uniform with range $[1, 300]$ for XGBoost, LightGBM, CatBoost and Random Forests,
- Depths: Discrete uniform with range $[1, 10]$ for XGBoost, LightGBM, CatBoost and Random Forests,
- Learning rates: Discrete uniform with range $[10^{-4}, 1.0]$ for XGBoost, LightGBM and CatBoost,
- Feature importance threshold: Log uniform over the range $[10^{-7}, 10^{-1}]$ for XGBoost, LightGBM, CatBoost and Random Forests,
- Subsample: Uniform over the range $[0.5, 0.9]$ for XGBoost, LightGBM, CatBoost and Random Forests.

- Bagging Frequency: Uniform over the set $\{1, \cdots, 7\}$ for LightGBM.

FASTEL (Dense soft trees).

- Number of trees: Discrete uniform with range $[1, 100]$,
- Depths: Discrete uniform with range $[1, 5]$,
- Batch sizes: Uniform over the set $\{16, 64, 256, 1024\}$,
- Number of Epochs: Discrete uniform over the range $[5, 1000]$,
- Learning rates (lr) for minibatch SGD: Log uniform over the range $[10^{-2}, 10]$.

6.8.2.4 Experimental Details for Sec. 6.6.4

In this chapter, we pursue embedded feature selection methods for tree ensembles. However, for completeness, we also compare `Skinny Trees` against some state-of-the-art embedded feature selection methods from neural networks.

Competing Methods. We compare against the following baselines.

1. LassoNet [154] is based on a ResNet-like [105] architecture with residual connections. Feature selection is done using hierarchical group lasso.

2. AlgNet [56] is based on adaptive lasso and uses proximal full-batch gradient descent for feature selection in a tanh-activated feedforward network.

3. DFS [47] solves cardinality constrained feature selection problem with an active-set style method.

We describe the tuning details for these neural network models below. We perform 2000 tuning trials for each method. For DFS, we capped number of trials completed in total 200 GPU hours. DFS is really slow for even medium sized datasets (in terms of feature dimensions).

LassoNet [154].

- ResNet architecture with 2-layered relu-activated feedforward network with (linear) skip connections.
- Number of hidden units: Discrete uniform in the set $\{\frac{p}{3}, \frac{2}{3}p, p, \frac{4}{3}p\}$,
- Batch sizes: Discrete uniform in the set $\{64, 128, 256, 512\}$,
- λ: Log uniform over the range $[1, 10000]$.
- Tuning protocol: 1000 for dense training stage and 1000 for each successive sparse training stages with early stopping with a patience of 25. We consider 100 sequential stages (100 values for λ).

AlgNet [56].

- Tanh-activated 4-layered feedforward neural network with number of hidden units chosen uniformly from a discrete set $\{\frac{p}{3}, \frac{2}{3}p, p, \frac{4}{3}p\}$,
- Learning rates (lr) for proximal GD: Log uniform over the range $[0.01, 1]$.
- λ: Log uniform over the range $[0.1, 1000]$.
- Number of Epochs: Discrete uniform with range $[5, 2000]$,

DFS [47].

- ReLU-activated 4-layered feedforward neural network with number of hidden units chosen uniformly from a discrete set $\{\frac{p}{3}, \frac{2}{3}p, p, \frac{4}{3}p\}$,
- Learning rates (lr) for Adam: Log uniform over the range $[0.001, 0.1]$.
- Weight decay: 0.0025 for feature selection layer and 0.005 for remaining layers.
- k: Number of features selected in the range $[1, 0.5p]$.
- Number of Epochs: 500 with early stopping.

6.8.2.5 Tuning Details for Sec. 6.6.5

We perform 2000 tuning trials for each method.

`Skinny Trees`.

- Number of trees: Discrete uniform with range $[1, 100]$,
- Depths: Discrete uniform with range $[1, 5]$,
- Batch sizes: Uniform over the set $\{16, 64, 256, 1024\}$,
- Number of Epochs: Discrete uniform with range $[5, 1000]$,
- λ_0 (without Dense-to-sparse learning): Log uniform in the range $[1, 10^4]$ for group $\ell_0 - \ell_2$,
- λ_0 (with Dense-to-sparse learning): exponentially ramped up $0 \to 1$ with temperature s distributed as Log uniform in the range $[10^{-4}, 1]$ for group $\ell_0 - \ell_2$ with DSL,
- λ_2: Log uniform over the range $[10^{-3}, 1]$ for group $\ell_0 - \ell_2$,
- Learning rates (lr): Log uniform over the range $[10^{-2}, 10]$.

Table 6.8.3: *Comparison of test AUC (%) performance of* `Skinny Trees` *against ControlBurn for* 25% *feature selection budget on binary classification tasks.*

Dataset	ControlBurn	`Skinny Trees`
Churn	85.66±0.25	**91.38**±0.08
Madelon	94.23±0.08	94.14±0.09
Gisette	99.36±0.01	**99.81**±0.002
Arcene	77.89±0.32	**80.80**±0.30
Smk	**79.94**±0.32	79.29±0.22
Gli	98.62±0.16	**99.80**±0.07
Dorothea	84.85±0.28	**90.87**±0.02
Average	88.65	**90.87**

6.8.2.6 Comparison with ControlBurn

We also compare against ControlBurn [168] on binary classification tasks. ControlBurn does not support multiclass classification. ControlBurn is another post-training feature selection method, which formulates an optimization problem with group-lasso regularizer to perform feature selection on a pre-trained forest. It relies on a commercial solver to optimize their formulation. We report the results for 25% feature selection budget in Table 6.8.3. We can observe that `Skinny Trees` outperforms ControlBurn across many datasets, achieving 2% (up to 6%) improvement in AUC.

We note the hyperparameter tuning for ControlBurn below:

- Method: bagboost,
- Depths: Discrete uniform with range $[1, 10]$,
- α: Log uniform over the range $[10^{-7}, 1.0]$.

6.8.3 Measuring Statistical Significance

We follow the following procedure to test the significance of all models. For all models, we tune over hyperparameters for 2000 trials. We select the optimal trial (within the desired feature sparsity budget) based on validation set. Next, we train each model for 100 repetitions with the optimal hyperparameters and report the mean results on test set along with the standard errors.

Chapter 7

Sparse Mixture of Experts: A Cardinality Constrained Routing Approach with Trees and Local Search

7.1 Introduction

The Sparse Mixture of Experts (Sparse-MoE) framework has led to state-of-the-art performance in various applications such as natural language processing (NLP) [8, 60, 72, 233, 295], vision [227, 268], time-series analysis [121], multi-task learning [101, 140, 176], and multimodal learning [194]. Sparse-MoE consists of a set of trainable experts (neural networks) and a trainable sparse gate. The sparse gate in Sparse-MoE selects an appropriate subset of experts on a per-sample basis, which allows for faster computation [233] and enhances interpretability [72, 121].

The literature on Sparse-MoE has traditionally focused on Top-k gating, which selects k out of m experts using a Top-k operation [72, 233, 295]. Top-k gating is simple and efficient because it allows sparse training. However, as highlighted by prior literature [72, 101, 295], the non-continuous nature of Top-k makes it susceptible to stability and convergence issues.

Alternative gating strategies exist in the literature, based on reinforcement learning [16] or post-processing via linear assignment [50, 158]. However, these strategies also face challenges in terms of efficiency and interpretability; see related work in Section 7.2 for more details. Random routing strategies [224, 297] alternatively bypass learning of the gating function altogether. Although computationally efficient, these strategies lead to performance degradation [50]. Recent work [101] demonstrates that differentiable gating in Sparse-MoE can improve stability and performance compared to popular non-differentiable gates. However, it suffers from expert collapse in some cases as we observed in our experiments.

In this chapter and the next, we propose three new approaches for improving routing in Sparse-MoE. These methods are designed to improve conditional computation at inference time and/or training time. First, we introduce a novel differentiable sparse gate *COMET*[1] that improves over existing state-of-the-art sparse gates [72, 101, 224, 233, 295]. Second, we argue that the combinatorial nature of expert selection in Sparse-MoE presents a serious challenge for first-order methods. In particular, the performance of these methods is highly dependent on initialization, and they can get stuck in low-quality routing solutions. Thus, we propose a new permutation-based local search method for Sparse-MoEs, which can help first-order methods escape low-quality initializations or solutions. Our local search approach is general and can be applied to any sparse gate, including Top-k [233], Hash routing [224], DSelect-k [101], and our proposed gate *COMET*. The above two methods allow sparse inference. However, these methods can only partially support conditional training with customized implementations. Hence, we consider an alternative sampling-based gate, discussed in detail in the next chapter, that can lead to purely sparse training as well as sparse inference.

COMET. Our proposed *COMET* gate is the first decision-tree-based selection mechanism for sparse expert selection — decision trees naturally perform per-sample routing (i.e., each sample follows a root-to-leaf path). Our gate has several advantages: (i) it is differentiable and can be optimized using first-order optimization methods e.g., stochastic gradient descent;

[1] *COMET*: This stands for Cardinality cOnstrained Mixture of Experts with Trees.

(ii) it allows (partially) conditional training, i.e., dense-to-sparse training; (iii) it enforces a cardinality constraint, i.e., selects (at most) k out of the n experts at inference time; (iv) it has superior predictive performance over state-of-the-art gates such as Hash routing, Top-k, and DSelect-k.

Local Search. The learning problem underlying Sparse-MoEs is of combinatorial nature, which poses additional challenges compared to non-MoE machine learning models. Popularly used optimization methods, such as SGD, may lead to low-quality solutions in Sparse-MoE, as we demonstrate in our numerical experiments in Section 7.5. To this end, we propose a permutation-based local search method, which can help first-order methods escape bad initializations and lead to better sample routing for any sparse gate e.g., Top-k, Hash routing, DSelect-k and even *COMET*. To the best of our knowledge, we are the first to explore local search methods in the context of Sparse-MoE. We provide empirical evidence through ablation studies and large-scale experiments to demonstrate that permutation-based local search (i) pushes learning towards better gate/expert initializations in early optimization stages (see Section 7.4.4); (ii) effectively reduces the budget needed for hyperparameter tuning by up to 100× for some popular gates e.g., Hash Routing and Top-k (see Section 7.5.1.3); (iii) leads to SOTA performance in terms of prediction and expert selection when combined with *COMET*, across various applications (see Section 7.5).

Contributions. As discussed earlier, it is well-known in the literature that popular sparse gates are challenging to train and may suffer from stability and performance issues. In this context, our contributions can be summarized as follows:

- We propose *COMET*, a novel tree-based sparse gate that simultaneously has the following desirable properties: (a) differentiable, (b) allows (partially) conditional training i.e., dense-to-sparse training, and sparse inference, (c) satisfies per-sample cardinality constraint (selects at most k out of the m experts per-sample, where k is a user-specified parameter).
- Popular first-order methods used to optimize Sparse-MoEs are heavily influenced by expert

and gate initializations, and may get stuck in low-quality solutions. Hence, we introduce a novel permutation-based local search method that can complement first-order methods by helping them escape bad initializations or solutions. Our local search method is general and can be applied to any gate, e.g., Hash routing, Top-k, and *COMET*.

- We perform extensive experiments on recommender systems, vision and natural language processing tasks to highlight that *COMET* and permutation-based local search can give boosts in predictive performance and reduce hyperparameter tuning.

7.2 Related work

Sparse-Mixture-of-Expert. The MoE framework was introduced by [123], and since then has been extensively studied — see e.g., [122, 128, 130]. More recently, [233] proposed a *Sparse*-MoE framework, based on the Top-k gate, and showed good performance on natural language processing tasks. It was further improved upon by [72, 290, 295]. However, Top-k gate does not optimize the core expert selection problem as pointed out by [50]. Additionally, as highlighted by prior literature [72, 101, 295], the non-continuous nature of Top-k makes it vulnerable to training stability and convergence issues.

With *BASE* Layers, [50, 158] formulate Sparse-MoE as an assignment problem where they post-process the gate output for balanced expert selection. [16] formulates the expert selection as a reinforcement learning problem. Others [224, 297] proposed random routing strategies that do not learn the gating function during training. These methods are also promising as they have been shown to outperform models that learn routing through Top-k, e.g., in Switch Transformers [72]. Lastly, [101] introduced DSelect-k, a differentiable gate based on binary encodings, which improves over Top-k in terms of stability and statistical performance.

Conditional Computation. In addition to the Sparse-MoE framework, there are other related works that also study conditional computation, i.e., the setup where only some parts of neural network are activated based on the input — see e.g., [16, 17, 119, 258]. These works rely on heuristics where the training and inference models are different. More recently, [100] introduced conditional computation in differentiable (a.k.a. soft) trees [79, 100, 109, 115, 120, 123]. Their proposal allows routing samples through small parts of the tree; thus allowing for conditional computation with customized algorithms. Our work builds upon this approach to solve the cardinality-constrained expert selection problem in Sparse-MoE. Note that [100] does not address sparse expert selection in Sparse-MoE.

Local Search and Permutation Learning. There is an extensive optimization literature on local search, e.g., [14, 99]. However, such methods have not been used in Sparse-MoE. Here, we survey permutation learning methods that are most relevant to our proposal. This work uses differentiable relaxations of permutation via Sinkhorn operators [2, 186]. These earlier works use these relaxations in other contexts e.g., ranking in [2] and sorting in [186]. We use permutation learning as a local search to complement first-order optimization methods to improve sample routing in Sparse-MoE.

7.3 Learning Sparse Mixture of Experts with decision trees

Problem Setup of Sparse-MoE. We first review the Sparse-MoE objective. We assume that the task has an input space $\mathcal{X} \subseteq \mathbb{R}^p$ and an output space $\mathcal{Y} \subseteq \mathbb{R}^u$. Denote the m-dimensional simplex by $\Delta_m = \{w \in \mathbb{R}^m : \sum_{i\in[m]} w_i = 1, w \geq 0\}$. In the MoE framework, the prediction function has two components: (i) a set of m experts (parametrized by neural networks) $f_i : \mathcal{X} \to \mathbb{R}^u$ for any $i \in [m] := \{1, 2, \ldots, m\}$, and (ii) a gate $g : \mathcal{X} \to \Delta_m$ that outputs weights in the probability simplex. Given a sample $x \in \mathcal{X}$, the corresponding output of the MoE is a convex combination of the experts with weights $g(x)$: $\sum_{i\in[m]} f_i(x)g(x)_i$.

The goal of Sparse-MoE paradigm is to develop a gate that selects a convex combination of at most k out of the m experts. The output of the gate can be thought of as a probability vector g with at most k nonzero entries, where $g(\cdot)_i$ is the weight assigned to the expert f_i. The underlying optimization problem (also in [101]) is:

$$\min_{f_1,\cdots,f_m,g} \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} \ell\left(y, \sum_{i\in[m]} f_i(x)g(x)_i\right), \tag{7.3.1a}$$

$$\text{s.t.} \quad \|g(x)\|_0 \leq k, \quad g(x) \in \Delta_m, \quad \forall x \in \mathcal{X}. \tag{7.3.1b}$$

$\|g(\cdot)\|_0$ denotes the number of nonzero entries in the vector $g(\cdot)$, $\ell(\cdot,\cdot)$ is the associated loss function such that $\ell : \mathcal{Y} \times \mathcal{X} \to \mathbb{R}$, and N denotes the size of training samples $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^N$.

The cardinality constraint in (7.3.1b) ensures that the gate selects at most k experts. Some popular gates e.g., Top-k impose exact cardinality constraint instead of an inequality constraint in (7.3.1b). However, the inequality constraint can allow for sparser expert selection as observed in prior work [101] and in our experiments (Section 7.5). Problem (7.3.1) is a combinatorial optimization problem that is not amenable to stochastic gradient descent due to the cardinality constraint in (7.3.1b). In the next sections, we discuss our formulation that ensures the cardinality constraint and the simplex constraints are satisfied despite optimization with gradient-based methods.

The rest of this section is organized as follows. In Section 7.3.1, we discuss a high-level overview of our novel tree-based framework, which equivalently sets up the cardinality-constrained objective in problem (7.3.1) as a weighted sum of decision trees. Next in Section 7.3.2, we provide background on a single decision tree that selects a single expert per-sample while (i) allowing for smooth optimization, and (ii) conditional computation support — routing samples to a single leaf. Later in Section 7.3.3, we dive deeper into our novel tree-based framework that combines such trees to satisfy the cardinality constraint for $k \geq 1$ without violating the simplex constraint. We additionally highlight important aspects regarding leaf parameterization and regularization. Next, in Section 7.3.4, we discuss how our method handles settings where experts are non-powers of 2. We then discuss in Section 7.3.5 an implementation of *COMET* for numerically stable training.

7.3.1 Sparse-MoE with k decision trees

The cardinality constrained MoE objective (7.3.1) can be formulated equivalently using a set of decision trees. Classical decision trees are naturally suited to route each sample to a single leaf with a chain of hierarchical decisions. In the case of $k = 1$, we propose a single decision tree to route samples, where each leaf node is associated with an expert. In cases where $k > 1$, we instantiate k different decision trees and combine their output in a way that enforces the cardinality and simplex constraints in (7.3.1b).

Given that classical decision trees are not amenable to differentiable training with first-order methods, we use a variant [100] of differentiable (a.k.a. soft) decision trees [100, 109, 115, 123, 139]. We build upon this work to solve the cardinality-constrained problem (7.3.1). We first provide a summary of a single soft tree (with conditional computation support) in Section 7.3.2. This serves as a building block for selecting a single expert per-sample.

7.3.2 Preliminaries: Differential Decision Tree with Conditional Computation

In this section, we provide a brief summary of a variant [100] of a differentiable (a.k.a. soft) tree [109, 120, 123, 130, 139], which we use to enable single-expert selection in Sparse-MoEs. We extend it in the next section to solve the cardinality constrained problem for a general case $k \geq 1$.

Differentiable decision trees are similar to classical decision trees with hyperplane splits [193]. However, they route each sample to left and right with different proportions, i.e., each sample reaches all leaves. Traditionally, differentiable decision trees have been unamenable to conditional computation as they cannot route a sample exclusively to the left or to the

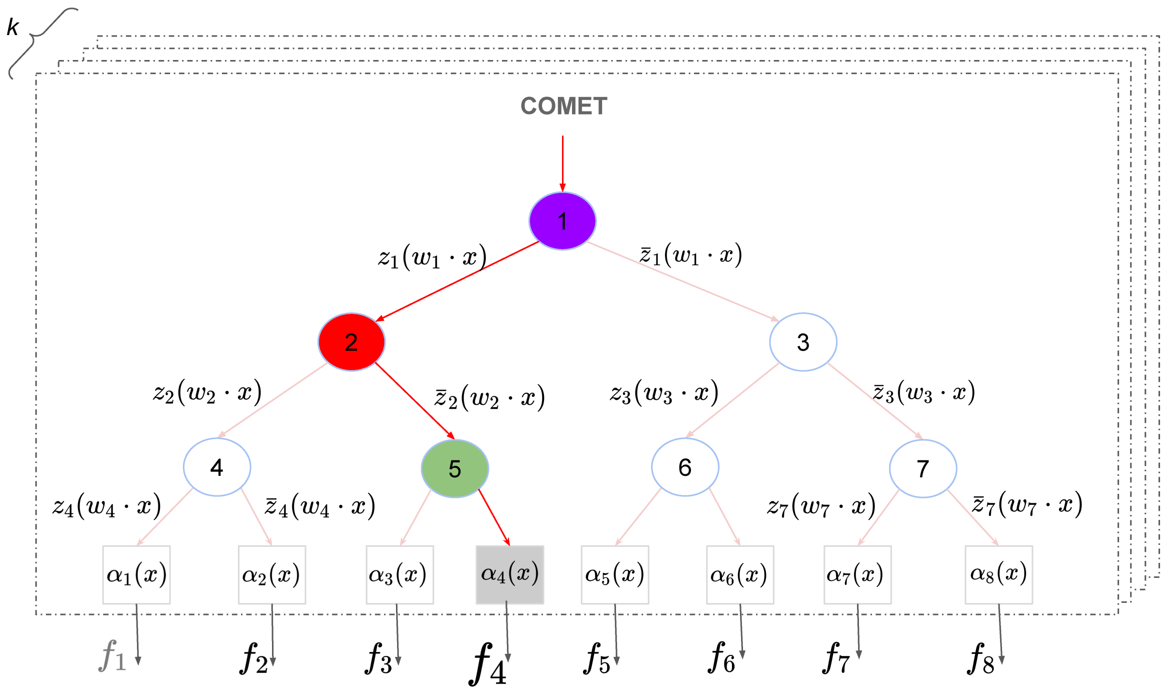


Figure 7.3.1: *COMET for 8 experts. Note $z_q(w_q \cdot x)$ denotes the binary state $\{0, 1\}$ for $h(w_q \cdot x)$ achieved due to Smooth-step activation function and entropic regularization.*

right. Recent work [100] introduced a variant of the differentiable tree model that supports conditional computation. Here, we discuss a brief summary of this variant.

We denote a single tree by $v : \mathcal{X} \rightarrow \Delta_m$, which maps an input sample $x \in \mathcal{X}$ to a probability vector v over Δ_m. Here, m corresponds to the number of root-to-leaf paths (also equal to number of experts in the MoE paradigm). Let v be a binary tree with depth d — note our framework can naturally support cases where number of experts is non-powers of 2, see Section 7.3.4 for more details. Let $\mathcal{I}$ and $\mathcal{L}$ denote sets of the internal (split) nodes and the leaves of the tree, respectively. For any node $q \in \mathcal{I} \cup \mathcal{L}$, we define $T(q)$ as its set of ancestors. Let $\{x \rightarrow q\}$ denote that a sample $x \in \mathbb{R}^p$ reaches q.

Sample Routing. Following prior work [100, 109, 139], we will discuss sample routing using a probabilistic model. While sample routing is discussed using probability, differentiable trees are deterministic. Differentiable trees are based on hyperplane splits [193], where a *linear* combination of the features is used in making routing decisions. In particular, we assign a trainable weight vector $w_q \in \mathbb{R}^p$ with each internal node, which parameterizes the node's hyperplane split. Let $h : \mathbb{R} \to [0, 1]$ be an activation function. Given a sample $x \in \mathbb{R}^p$, the probability that internal node q routes x to the left is defined by $h(w_q \cdot x)$.

Now we summarize how to model the probability that x reaches a certain leaf l [100]. Let $[l \swarrow q]$ (resp. $[q \searrow l]$) denote the event that leaf l belongs to the left (resp. right) subtree of node $q \in \mathcal{I}$. The probability that x reaches l is given by: $\Pr(\{x \to l\}) = \prod_{q \in T(l)} r_{q,l}(x)$, where $r_{q,l}(x)$ is the probability of node q routing x towards the subtree containing leaf l, i.e., $r_{q,l}(x) := h(w_q \cdot x)^{\mathbf{1}[l \swarrow q]}(1 - h(w_q \cdot x))^{\mathbf{1}[q \searrow l]}$. Note that the vector $v(x)$ given by

$$v(x) = [\Pr(\{x \to l_1\}), \cdots, \Pr(\{x \to l_m\})] \in \Delta_m, \tag{7.3.2}$$

defines a per-sample distribution over the m leaves (or experts).

Next, we discuss how the split probabilities $\{h(w_q \cdot x), 1 - h(w_q \cdot x)\}$ can achieve binary state with a particular choice of activation function h — this is crucial for achieving sparse expert selection (and conditional computation) in the Sparse-MoE paradigm.

Smooth-Step Activation Function. The common choice for activation function h in soft tree literature is a logistic function[79, 109, 130, 139]. However, it can not perform hard routing i.e., output exact zeros. This implies that any sample x will reach every node in the tree with a positive probability, leading to a dense v. Thus, computing the output of mixture of experts will require computation over every expert. [100] proposed a Smooth-step activation function for a variant of soft trees. Despite being continuously differentiable, Smooth-step activation function (also given in 2.9) can produce a sparse v (after an initial warm-up period of soft routing) for hard routing. The Smooth-step activation function can output exact zeros and ones per-sample, thus allowing for conditional computation. This is crucial for a sparse expert selection per-sample in Sparse-MoE paradigm. Additionally, this choice of activation function may allow for (partially) conditional training with customized sparse backpropagation algorithms in soft trees (as shown in [100]), which is an important consideration for training large-scale Sparse-MoE models.

For cardinality-constrained Sparse-MoE learning with trees (not studied in [100]), the goal for each tree is to perform hard routing for all samples. Therefore, we add additional regularization on $\{h(w_q \cdot x), 1 - h(w_q \cdot x)\}$ to encourage convergence of v to a one-hot state (discussed in more detail in Section 7.3.3).

7.3.3 Cardinality constraint with k trees

Next, we discuss how to achieve the cardinality constraint ($k \geq 1$) in Sparse-MoE with decision trees in the presence of simplex constraint. This key ideas are given as follows:

- We consider k decision trees, where each tree j selects a single expert via $v^{(j)}(\cdot)$ as defined in (7.3.2).
- With the experts selected as above, we need to decide the relative weights assigned to each expert. We use auxiliary functions $\alpha^{(j)}(\cdot)$, where $\alpha_i^{(j)}$ is a linear function $\beta_i^{(j)} \cdot x$ of the input. $\alpha_i^{(j)}$ reflects a linear weighting function (in the log space) for i-th expert (or leaf) in j-th tree.

See Figure 7.3.1 as an example. Next, we define the prediction function for Sparse-MoE with k decision trees to form *COMET*.

***COMET* Prediction with k Out of m Experts.** The prediction function for Sparse-MoE with $k \geq 1$ is a weighted sum of the predictions of i-th expert (or leaf) across k trees. To this end, we define the weight for i-th expert as follows

$$g(x; \alpha, v)_i = \frac{\sum\limits_{j \in [k]} \exp(\alpha_i^{(j)}(x)) v_i^{(j)}(x)}{\sum\limits_{j \in [k]} \sum\limits_{i \in [m]} \exp(\alpha_i^{(j)}(x)) v_i^{(j)}(x)}, \tag{7.3.3}$$

where $v_i^{(j)}(x)$ is the probability that a sample x will reach expert f_i in the j-th tree. Using (7.3.3), the prediction function for Sparse-MoE with $k \geq 1$ is given by $\hat{y} = \sum_{i \in [m]} f_i(x) g(x; \alpha, v)_i$.

We present the following proposition (proof in Appendix 7.7.1):

Proposition 7.3.1. *For any α, if $v^{(j)}$ outputs a binary vector for every j, the function $g(x; \alpha, v)$ satisfies the cardinality and simplex constraints in (7.3.1b).*

Accelerating Convergence of $v^{(j)}$ to One-Hot Encoding with Entropic Regularization. In the Sparse-MoE setup, the goal is to achieve a one-hot vector state for $v^{(j)}$ quickly — this ensures the cardinality constraint (i.e., to select at most k experts) is respected by the k trees. To encourage faster convergence towards a one-hot vector, we add a per-tree per-sample entropy regularizer, $\lambda\Omega(v^{(j)}(x))$ to the loss objective, where $\Omega(v^{(j)}(x)) = -\sum_{i\in[m]} v_i^{(j)}(x)\log(v_i^{(j)}(x))$. Entropy regularizers are used in [101, 186] to get binary representations.

Dense-to-Sparse Learning. *COMET* can support conditional training only partially with customized implementations. At the start of training, it uses all the available experts as $v^{(j)}$ is completely dense, so conditional training is not possible. As training proceeds, $v^{(j)}$ becomes sparser due to Smooth-step function and entropic regularization, eventually achieving binary state. From this stage onwards, the gate satisfies the cardinality constraint per-sample, i.e, each sample gets routed to at most k experts. Hence, sparse training can proceed to refine the solution quality. Empirically, we observe that a small number of epochs are sufficient for the optimizer to reach the sparse training phase.

7.3.4 Non-powers of 2

Typically, in Sparse-MoE, each expert is assigned to a separate machine for efficiency [72, 295]. This may mean that the number of experts could be defined by the number of machines — machines may not necessarily be available in powers of 2. Our gate naturally handles cases where the number of experts are not chosen to be powers of 2. We propose merging child nodes at the leaf level. In such instances, we have imperfect binary decision trees (Fig. 7.7.1 in Appendix) with m nodes, with $2^d - m$ nodes in the $(d-1)$-th level, and $2m - 2^d$ nodes in the d-th level. Additional details are in Appendix 7.7.2. In contrast to other differentiable gates (e.g., DSelect-k [101]), our proposed gate *COMET* does not require any additional regularization to encourage the simplex constraint in (7.3.1b).

7.3.5 Stable numerical implementation

Next, we discuss a stable numerical implementation of *COMET* gate. *COMET* introduces additional exponential functions in the expert weights (or leaf nodes of the decision trees) — see (7.3.3). More exponential functions are known to cause instabilities in Sparse-MoE models. For example, [295] introduced router z-loss in Switch Transformers to encourage smaller logits. However, this may have a performance tradeoff. In our implementation of *COMET*, we can mitigate instability issues arising from additional exponential functions using the following approach: (i) convert root-to-leaf probabilities to the log-space, $\log v_i^{(j)}(x)$, (ii) compute $\alpha_i^{(j)} + \log v_i^{(j)}(x)$, (iii) subtract the maximum, i.e., $\max_{i,j}(\alpha_i^{(j)} + \log v_i^{(j)}(x))$ from each element, (iv) apply a two-way softmax operation to get $g(x)$.

7.4 Local search

Expert selection is a challenging combinatorial problem that is known to be NP-hard. Although first-order heuristics can usually provide fast solutions, they rely heavily on initialization and are sometimes prone to arriving at low-quality solutions. To this end, we propose a permutation-based local search method that complements first-order methods in optimizing Sparse-MoEs. In both large-scale experiments and ablation studies, we see that the incorporation of local search can improve the performance of *any* gating method and can significantly reduce the number of tuning trials.

Our approach derives inspiration from the local search methods commonly used along with the first-order methods to help escape local minima in sparse linear models[14, 99]. We note that this is the first attempt in the literature to incorporate local search methods in the context of Sparse-MoE. Moreover, unlike common local search methods in literature, our proposed search method is differentiable. We want to highlight that our local search method is useful for any existing sparse gate, e.g., Hash routing, Top-k, and our proposed *COMET*. We hypothesize that our permutation-based approach can help navigate the optimization loss surface for various gates.

The rest of the section is organized as follows. In section 7.4.1, we formulate a refined cardinality-constrained Sparse-MoE objective with additional binary variables to add support for permutation-based local search. Then, in section 7.4.2, we provide background on permutation and its differentiable relaxation. Next in section 7.4.3, we outline our differentiable optimization approach for the refined Sparse-MoE objective and some additional practical considerations for computational efficiency. Later, in Section 7.4.4, we provide an ablation study to support our hypothesis that the local search can help escape bad initializations.

7.4.1 Permutation-based Local Search

In this section, we formulate a refined objective for the cardinality-constrained Sparse-MoE objective that adds support for permutation-based local search.

Let us denote by $\mathcal{S}_m$ the set of all permutations of the set $[m]$. Given any permutation $\sigma \in \mathcal{S}_m$, we permute the m experts accordingly and assign i-th weight $g(x)_i$ to $\sigma(i)$-th expert instead of i-th expert. With this permutation, the prediction for Sparse-MoE could be written as: $\hat{y} = \sum_{i \in [m]} f_{\sigma(i)}(x) g(x)_i$. We note that due to symmetry between experts and weights, permuting the experts is essentially same as permuting the weights. To see this, we can write $\sum_{i \in [m]} f_{\sigma(i)}(x) g(x)_i = \sum_{j \in [m]} f_j(x) g(x)_{\sigma^{-1}(j)}$, where σ^{-1} is the inverse map of σ, which is also a permutation.

For a permutation σ, we can define a corresponding permutation matrix $\boldsymbol{P}^{\sigma}$, by setting $P^{\sigma}[i,j] = \mathbf{1}\{\sigma(j) = i\}$, where $\mathbf{1}\{\cdot\}$ is an indicator function. Then it is easy to see that

$\sum_{j\in[m]} f_j(x)g(x)_{\sigma^{-1}(j)} = \sum_{j\in[m]} f_j(x)(\boldsymbol{P}^\sigma g(x))_j$. The refined Sparse-MoE problem is

$$\min_{f_1,\cdots,f_m,g,\boldsymbol{P}} \quad \frac{1}{N}\sum_{(x,y)\in\mathcal{D}} \ell\left(y, \sum_{i\in[m]} f_i(x)(\boldsymbol{P}g(x))_i\right), \tag{7.4.1a}$$

$$\text{s.t.} \qquad \|g(x)\|_0 \le k, \quad g(x)\in\Delta_m, \quad \forall x\in\mathcal{X}, \tag{7.4.1b}$$

$$\boldsymbol{P}\in\mathcal{P}_m^{\text{local}}, \tag{7.4.1c}$$

where $\mathcal{P}_m^{\text{local}}$ is a localized set of permutations in the full set of permutations, which we denote by $\mathcal{P}_m$. For example, one may only allow for $\mathcal{P}_m^{\text{local}} = \mathcal{P}_2$, which only allows interchanging (swapping) two columns similar to "swap" operations shown to be useful in the sparse regression literature [99]. Besides optimizing the gates and experts, formulation (7.4.1) performs local search by optimizing over the permutation matrix. Specifically, the goal of local search here is to find a permutation P that leads to a better solution, i.e., one with a lower objective. Intuitively, if SGD is stuck at a low-quality solution, the permutation may be able to escape the solution by a better reordering of the experts. Standard local search, e.g., bruteforce search may be computationally expensive. Therefore, we resort to a differentiable method that can be optimized efficiently.

7.4.2 Preliminaries: Permutation and a differentiable relaxation

In this section, we briefly summarize how the permutation learning problem is parameterized and later optimized. To parametrize the permutation matrix in the problem, a natural consideration is through the linear assignment problem [141]. To illustrate this, consider m people are to complete m tasks and a matrix $\boldsymbol{U}\in\mathbb{R}_{\ge 0}^{m\times m}$, the goal is to assign each task to one person so as to maximize the utility given that the utility of assigning task j to person i is U_{ij}. This leads to the following optimization problem

$$M(\boldsymbol{U}) = \operatorname*{arg\,max}_{\boldsymbol{P}\in\mathcal{P}_m} \langle \boldsymbol{P},\boldsymbol{U}\rangle_F := \sum_{i\in[m]}\sum_{j\in[m]} P_{ij}U_{ij}. \tag{7.4.2}$$

The operator M here is called the Matching operator, which maps a nonnegative matrix $\boldsymbol{U}$ to a permutation matrix $\boldsymbol{P}$.

Problem (7.4.2) is a combinatorial optimization problem, which admits the following linear relaxation [23]:

$$\max_{\boldsymbol{B}\in\mathcal{B}_m} \langle \boldsymbol{P}, \boldsymbol{U}\rangle_F := \sum_{i\in[m]}\sum_{j\in[m]} P_{ij}U_{ij}, \tag{7.4.3}$$

where $\mathcal{B}_m$ denotes the set of double stochastic matrices $\mathcal{B}_m = \{\boldsymbol{B} \in \mathbb{R}^{m\times m} : \sum_{i\in[m]} B_{ij} = 1, \sum_{j\in[m]} B_{ij} = 1, B_{ij} \in [0,1]\}$, which is a convex hull of the set of permutation matrices $\mathcal{P}_m$.

However, this is still not a differentiable parametrization as problem (7.4.3) might end up with multiple solutions. To this end, Mena et al. [186] proposes a smooth version[2] of the permutation learning objective in (7.4.3):

$$S(\boldsymbol{U}/\tau) = \arg\max_{\boldsymbol{B}\in\mathcal{B}_m} \langle \boldsymbol{B}, \boldsymbol{U}\rangle_F - \tau \sum_{i,j\in[m]} B_{ij}\log B_{ij}, \tag{7.4.4}$$

and solves it using Sinkhorn operator $S(\cdot)$ [2], defined by the following recursion:

$$S^0(\boldsymbol{U}) = \exp(\boldsymbol{U}), \tag{7.4.5a}$$

$$S^r(\boldsymbol{U}) = \mathcal{T}_{col}(\mathcal{T}_{row}(S^{r-1}(\boldsymbol{U}))), \tag{7.4.5b}$$

$$S(\boldsymbol{U}) = \lim_{r\to\infty} S^r(\boldsymbol{U}), \tag{7.4.5c}$$

where $\mathcal{T}_{row}(\boldsymbol{U}) = \boldsymbol{U} \oslash (\boldsymbol{U}\mathbf{1}_m\mathbf{1}_m^T)$, and $\mathcal{T}_{col}(\boldsymbol{U}) = \boldsymbol{U} \oslash (\mathbf{1}_m\mathbf{1}_m^T\boldsymbol{U})$ are the row and column-wise normalization operators of a matrix, with $\oslash$ denoting the element-wise division and $\mathbf{1}_m$ a column vector of ones. The sinkhorn procedure in (7.4.5) allows differentiable training with first-order methods, making it appealing as a local search method for Sparse-MoE.

As shown in [186], $M(\boldsymbol{U})$ can be obtained as $\lim_{\tau\to 0^+} S(\boldsymbol{U}/\tau)$, and thus $\lim_{\tau\to 0^+, r\to\infty} S^r(\boldsymbol{U}/\tau)$. In practice, we set a max number of iterations R for normalization in (7.4.5b) as well as a

[2]Note that (7.4.4) is an entropy-regularized version of (7.4.3). Since the entropy term is strictly concave, problem (7.4.4) has a unique minimizer and thus the parametrization is differentiable.

small positive number $\tau > 0$, and use $S^R(\boldsymbol{U}/\tau)$ to approximate the limit (7.4.5c). In this way, we are able to parametrize the permutation matrix $\boldsymbol{P}$ in (7.4.1) as a differentiable function $S^R(\boldsymbol{U}/\tau)$ of learnable matrix $\boldsymbol{U}$. However, additional considerations are needed to ensure that a hard permutation matrix can be achieved quickly in a few epochs — this is important in Sparse-MoE paradigm for computational reasons and a well-defined measure of sparsity. We discuss these in the next section.

7.4.3 Practical considerations for optimization

Next, we discuss some empirical considerations for the end-to-end learning approach that are important for Sparse-MoE.

Need for a hard permutation matrix. We would like to have a hard permutation matrix at inference time and ideally during the course of training, for exact sparsity and computational efficiency considerations. First, the gate does not perform sparse inference if the learnt permutation matrix is not a hard matrix. For example, even if $g(\cdot)$ is sparse, the refined weights $\boldsymbol{P} \cdot g(\cdot)$ are not a sparse vector if $\boldsymbol{P}$ is not a binary matrix. This would result in a dense mixture of experts. Second, some sparse gates perform dense-to sparse-training (partially conditional training), e.g., DSelect-k, *COMET*, or variants of Top-k [198]. If the learnt permutation matrix is not hard, then sparse training cannot proceed in the later stages of optimization. To this end, we employ a two-stage optimization approach: (i) in the first stage, we simultaneously train the network (experts and gates) and the permutation for a small number of epochs. (ii) In the second phase, the permutation matrix is fixed and only the remaining network (experts and gate) is trained. Therefore, local search is only used in the early stages of training. Empirically, we observe that a small number of epochs $(1 - 10)$ is sufficient to learn a good permutation in the first stage and improve solution quality. Since local search is restricted to the first stage, the computational efficiency of gates that perform dense-to-sparse training is not affected by much — please refer to Appendex

7.7.3.3 for additional discussion.

In the two-stage approach outlined above, there is a transition from a soft to a hard matrix between the two stages. As we mentioned earlier, we use $S^R(\boldsymbol{U}/\tau)$ to approximate $M(\boldsymbol{U})$ as a limit of $R \to \infty, \tau \to 0^+$. In practice, the transition could be not continuous, as this approximation does not always reach a hard permutation matrix given that R is finite and τ is nonzero. Therefore, at the transition point, we propose to convert the "soft" permutation matrix $S^R(\boldsymbol{U}/\tau)$ to a hard one via the linear assignment problem given in (7.4.2), by invoking $\boldsymbol{U}$ as $S^R(\boldsymbol{U}/\tau)$. In addition, empirically, small R can lead to numerical instabilities for small τ [186]. Therefore, to decrease deviance of $S^R(\boldsymbol{U}/\tau)$ from the closest hard permutation matrix, we introduce two schedulers on R and τ that increase R for decreased τ: (i) Ramp up (linearly) R from 20 to 150, (ii) Ramp down (linearly in log-scale) τ from 10^{-3} to 10^{-7}.

Although the above schedulers decrease the deviance between soft and its closest hard permutation matrix at the transition point, the method still appears to suffer from pseudo-convergence. In particular, we observed, some row-columns can converge to fractional entries i.e., a 2x2 sub-block having all entries with 0.5. Therefore, we introduce small separate row-wise and column-wise entropic regularizations to mitigate such degenerate cases:
$\zeta \sum_{i\in[m]}(\Omega(\mathcal{S}^R(\boldsymbol{U}/\tau)_i) + \Omega(\mathcal{T}_{row}(\mathcal{S}^R(\boldsymbol{U}/\tau))_i))$, where $\zeta \geq 0$.

Implicit localization. In the spirit of common local search approaches, a potential optimization approach could alternate between optimization of network (experts and gates) and permutation matrix. However, this is unnecessary because the differentiable relaxation of permutation is also amenable to first-order methods. Therefore, our approach jointly optimizes both the network and the permutation matrix. We noted earlier that the search space for permutation is "localized" out of the full set of permutation matrices $\mathcal{P}_m$. This localization is implicitly imposed through the smooth optimization of the permutation matrix via Sinkhorn. The permutation matrix learning relies on the initialization for $\boldsymbol{U}$ and at each gradient step the $\boldsymbol{U}^{(t)}$ is naturally expected to not deviate drastically from $\boldsymbol{U}^{(t-1)}$. Since the permutation matrix is updated for a limited number of steps in first stage, intuitively it cannot deviate significantly from the initial permutation matrix. This also defines an implicit neighborhood.

7.4.4 Ablation study for local search

In this section, we provide an ablation study to provide evidence that the permutation-based local search can complement first-order optimization methods for routing in Sparse-MoE. The study highlights that local search can improve solution quality through escape out of bad initializations in the first stages of optimization for different types of routing strategies: (a) fixed gates, (b) trainable gates. We perform this study on a subsampled (200k) MovieLens dataset and use the same MoE architecture with 16 experts as the one described in Appendix. We trained models for only 10 epochs without/with local search, where in the latter case we fixed the number of epochs for permutation learning to 5 epochs and $\zeta = 10^{-5}$. We used a batch size of 512 and learning rate of 2.5×10^{-5}. We repeat the training with 100 different random initializations and compute averages along with their standard errors.

Fixed Gates. In fixed gating strategies e.g., random hash routing (Hash-r), the samples are pre-assigned to experts. For example, in natural language processing tasks, tokens or words in vocabulary are clustered randomly [224] *before* training begins into groups and each group of words are assigned to a random expert in the set of experts. In our experiments on recommender systems, we randomly pre-assigned samples to experts based on user index for Hash-r (and *Hash-r+*). It is possible that the same group of users could be better aligned with another expert based on expert and user embedding initializations. Permutation-based local search can potentially find better assignment of each group to a more suited expert. We provide empirical evidence to demonstrate that local search indeed can find better loss. We report the average out-of-sample loss achieved by both Hash-r and *Hash-r+* in Table 7.4.1. Learning permutation appears to help map each pre-assigned cluster of users to a more

Table 7.4.1: *Test loss ($\times 10^{-2}$) achieved for different gates without and with (marked with +) local search in early stages of optimization. Asterisk(*) indicates statistical significance (p-value$<$0.05) over the corresponding gate without permutation with a one-sided unpaired t-test.*

Strategy	Smoothness	Gate	Test Loss ↓
Pre-assigned	-	Hash-r	57.434 ± 0.025
		Hash-r+	$^*\mathbf{57.000} \pm 0.037$
Trainable	Non-differentiable	Top-k	53.345 ± 0.033
		Top-k+	$^*\mathbf{53.140} \pm 0.031$
	Differentiable	*COMET*	52.034 ± 0.007
		COMET+	$^*\mathbf{52.017} \pm 0.005$

suitable expert based on expert initialization for second stage of optimization.

Trainable Gates. For trainable gates, we also study the effect of local search on non-differentiable (Top-k) and differentiable gates (*COMET*). We fixed $k = 2$ for both types of gates and followed the same training protocol for 10 epochs. For *COMET* (and *COMET+*), we fixed $\gamma = 0.01$ (for Smooth-step) and $\lambda = 1$ (for entropic regularization). For *Top-k+*and *COMET+*, we fixed the number of epochs for permutation learning as 5. We repeated this exercise for 100 different random initializations of the experts and gates. We report the average out-of-sample objective achieved by both types of gates in Table 7.4.1. We can observe that local search appears to complement first-order optimization methods by learning better initializations in the first stage of Sparse-MoE optimization for later learning.

The practical significance of local search achieving a better test objective across many initializations for various gates can be seen in terms of reducing hyperparameter tuning overhead as discussed in Section 7.5.1.3.

7.5 Experiments

We study the performance of *COMET* and *COMET+* in recommender systems and image datasets in Section 7.5.1 and *COMET-BERT* in natural language processing tasks in 7.5.2. We also study the effect of local search for various gates. We denote our methods in *italics*.

Appendix contains more details.

7.5.1 Experiments on Recommender Systems and Image Datasets

We study the performance of *COMET* and *COMET+* in recommender systems and image datasets. We compare with state-of-the-art gates and baselines including Softmax, Top-k, DSelect-k and Hash routing (Hash-r) on recommender systems (MovieLens [88], Jester[82], Books [294]) and image datasets (Digits [53, 197], MultiMNIST [228], MultiFashionMNIST [101], CelebA[170]). We also include an ablation study in Section 7.5.1.2 that shows that *COMET* achieves good performance with much less trials than existing popular gates e.g., Hash routing and Top-k. Additionally, in Section 7.5.1.3, we show that *Hash-r+*,*Top-k+*, and *COMET+* with local search can potentially achieve good performance with much less trials than Hash-r, Top-k and *COMET* respectively.

Implementation. We provide an open-source implementation of *COMET* and *COMET+*: https://github.com/mazumder-lab/COMET.

Experimental Setup. Although our exposition in Section 7.3 was for a single-task setting, the same gate can also be used in multi-task learning — multi-task requires multi-gate MoE architecture [176], where each task has a separate trainable gate, but tasks have to select from a common set of experts. We briefly summarize the key aspects for each dataset. For MovieLens/Books/Jester we have two tasks: classification task predicts whether user watches/read/rates a particular movie/book/joke, regression problem predicts user's rating. Loss is the convex combination of the two binary cross-entropy (for classification) and mean squared error (for regression) with task weights: $\{\alpha, 1-\alpha\}$. We separately present results for two different α's: $\alpha \in \{0.1, 0.9\}$. For MultiMNIST/MultiFashionMNIST, there are two multi-class classification tasks, which are equally weighted. For CelebA, there are 10 binary classification problems, which are equally weighted. Lastly, for Digits dataset, we have a

Table 7.5.1: *Tess Loss ($\times 10^{-2}$) and actual no. of experts per sample (k_a) at inference for COMET, COMET+ and benchmark gates across various recommender system datasets. Asterisk(*) indicates statistical significance (p-value<0.05) over the best existing gate, using a one-sided unpaired t-test.*

Dataset	m	Model	Test Loss ↓	k_a ↓
Books ($\alpha = 0.1$)	9	Softmax	244.47 ± 0.14	9.00 ± 0.00
		Hash-r	247.43 ± 0.14	1.00 ± 0.00
		Hash-r+	247.33 ± 0.23	1.00 ± 0.00
		Top-k	247.87 ± 0.17	4.00 ± 0.00
		Top-k+	247.88 ± 0.14	4.00 ± 0.00
		DSelect-k	246.43 ± 0.36	1.09 ± 0.00
		COMET	$^*\mathbf{240.79} \pm 0.14$	2.81 ± 0.11
		COMET+	240.82 ± 0.19	3.03 ± 0.08
Books ($\alpha = 0.9$)	9	Softmax	73.88 ± 0.02	9.00 ± 0.00
		Hash-r	75.02 ± 0.03	1.00 ± 0.00
		Hash-r+	75.06 ± 0.03	1.00 ± 0.00
		Top-k	74.78 ± 0.03	4.00 ± 0.00
		Top-k+	74.86 ± 0.03	4.00 ± 0.00
		DSelect-k	75.98 ± 0.13	1.07 ± 0.00
		COMET	73.62 ± 0.03	2.94 ± 0.10
		COMET+	$^*\mathbf{73.55} \pm 0.02$	3.15 ± 0.08
MovieLens ($\alpha = 0.9$)	16	Softmax	42.26 ± 0.01	16.00 ± 0.00
		Hash-r	46.91 ± 0.02	1.00 ± 0.00
		Hash-r+	46.84 ± 0.03	1.00 ± 0.00
		Top-k	41.83 ± 0.02	2.00 ± 0.00
		Top-k+	41.74 ± 0.02	2.00 ± 0.00
		DSelect-k	40.82 ± 0.02	1.94 ± 0.06
		COMET	40.76 ± 0.02	1.76 ± 0.06
		COMET+	$^*\mathbf{40.69} \pm 0.02$	1.66 ± 0.06
MovieLens ($\alpha = 0.1$)	16	Softmax	75.52 ± 0.02	16.00 ± 0.00
		Hash-r	79.41 ± 0.02	1.00 ± 0.00
		Hash-r+	78.92 ± 0.05	1.00 ± 0.00
		Top-k	76.52 ± 0.04	2.00 ± 0.00
		Top-k+	75.12 ± 0.04	2.00 ± 0.00
		DSelect-k	73.91 ± 0.05	1.94 ± 0.03
		COMET	73.91 ± 0.04	1.94 ± 0.03
		COMET+	$^*\mathbf{73.67} \pm 0.04$	1.98 ± 0.03
Jester ($\alpha = 0.1$)	16	Softmax	68.17 ± 0.03	16.00 ± 0.00
		Hash-r	67.47 ± 0.01	1.00 ± 0.00
		Top-k	68.38 ± 0.05	2.00 ± 0.00
		Top-k+	68.00 ± 0.07	2.00 ± 0.00
		DSelect-k	67.06 ± 0.03	1.96 ± 0.02
		COMET	67.12 ± 0.04	1.98 ± 0.02
		COMET+	$^*\mathbf{66.91} \pm 0.03$	2.00 ± 0.00
Jester ($\alpha = 0.9$)	16	Softmax	21.936 ± 0.002	16.00 ± 0.00
		Hash-r	22.083 ± 0.004	1.00 ± 0.00
		Top-k	21.958 ± 0.007	2.00 ± 0.00
		Top-k+	21.961 ± 0.006	2.00 ± 0.00
		DSelect-k	21.930 ± 0.005	2.00 ± 0.00
		COMET	21.946 ± 0.005	2.00 ± 0.00
		COMET+	$^*\mathbf{21.906} \pm 0.005$	2.00 ± 0.00

multi-class single-task classification cross-entropy objective. Full details about datasets and MoE architectures are in Appendix.

We used Adam for optimization, and we tuned the key hyperparameters using random grid search. Note that for *Hash-r+*, *COMET+* and *Top-k+*, we only allocate a very small portion of the epochs (1-10) for permutation learning. Full details about the hyperparameter tuning are given in Appendix.

7.5.1.1 Performance of *COMET* and *COMET+*

In Tables 7.5.1 and 7.5.2, we report the (average) test loss and the average number of selected experts per sample across multiple recommender and vision datasets. The results indicate that *COMET* and *COMET+* lead on many datasets, outperforming popular state-of-the-art gating methods e.g., Hash-r, Top-k and DSelect-k in test loss. Our proposed gate *COMET* can outperform standard routing techniques (without local search). Even without local search,

Table 7.5.2: *Tess Loss ($\times 10^{-2}$) and actual no. of experts per sample (k_a) at inference for COMET, COMET+ and benchmark gates across various image datasets. Asterisk(*) indicates statistical significance (p-value<0.05) over the best existing gate, using a one-sided unpaired t-test.*

Dataset	m	Model	Test Loss ↓	k_a ↓
MultiFashionMNIST	5	Softmax	34.21 ± 0.09	5.00 ± 0.00
		Top-k	33.82 ± 0.09	2.00 ± 0.00
		Top-k+	$^{*}\mathbf{33.62} \pm 0.08$	2.00 ± 0.00
		DSelect-k	35.49 ± 0.10	1.00 ± 0.00
		COMET	33.70 ± 0.09	1.49 ± 0.07
		COMET+	$^{*}\mathbf{33.67} \pm 0.09$	1.54 ± 0.07
CelebA	6	Softmax	35.10 ± 0.32	6.00 ± 0.00
		Top-k	34.48 ± 0.24	2.00 ± 0.00
		Top-k+	34.54 ± 0.23	2.00 ± 0.00
		DSelect-k	35.39 ± 0.12	1.00 ± 0.00
		COMET	33.96 ± 0.15	1.00 ± 0.08
		COMET+	$^{*}\mathbf{33.93} \pm 0.16$	1.00 ± 0.08
Digits	8	Softmax	17.48 ± 0.07	8.00 ± 0.00
		Top-k	17.46 ± 0.09	2.00 ± 0.00
		Top-k+	17.29 ± 0.08	2.00 ± 0.00
		DSelect-k	17.18 ± 0.06	1.15 ± 0.06
		COMET	17.19 ± 0.06	1.07 ± 0.04
		COMET+	$\mathbf{17.08} \pm 0.09$	1.06 ± 0.04
MultiMNIST	16	Softmax	6.88 ± 0.06	16.00 ± 0.00
		Top-k	6.84 ± 0.05	4.00 ± 0.00
		Top-k+	6.70 ± 0.08	4.00 ± 0.00
		DSelect-k	6.64 ± 0.07	3.40 ± 0.09
		COMET	$\mathbf{6.48} \pm 0.07$	3.49 ± 0.09
		COMET+	6.49 ± 0.06	3.53 ± 0.08

COMET is getting relatively good solutions. We hypothesize that the good performance of *COMET* is due to a combination of factors including differentiability, and k-decision trees formulation. With local search, *COMET+* can sometimes further enhance solution quality. We also provide task-specific metrics (AUC/Accuracy/MSE) in Tables 7.7.3 in Appendix 7.7.4. We observe *COMET+* can improve AUC by up to 13% over Hash routing and Top-k, and 9% over DSelect-k. We observe that Top-k gate does not uniformly outperform the Softmax across multiple datasets. However, *Top-k+* significantly improves the performance of Top-k across multiple datasets. In fact with the permutation module, *Top-k+* outperforms Softmax in all cases, so sparsity in gating seems to be beneficial on all these datasets.

Inference Sparsity. We see that *COMET* and *COMET+* can sometimes lead to a smaller number of experts selected than that for Top-k. This leads to smaller number of FLOPs at inference time (see Appendix 7.7.3.2). For some settings, DSelect-k appears to arrive at a sparser selection than *COMET+*; however, in these cases, DSelect-k loses significantly in terms of performance. We observed expert collapsing in DSelect-k in such cases.

Timing Discussion. For cost complexity of *COMET*, please see Appendix 7.7.3.1. Additionally, we discuss the computational aspects of the local search in Appendix 7.7.3.3.

7.5.1.2 Reducing Hyperparameter Search with *COMET*

Here, we study how our differentiable *COMET* gate (that performs dense-to-sparse training) can be beneficial in terms of hyperparameter tuning over popular gates such as Hash routing and Top-k. We perform a large set of tuning trials and perform a bootstrapping procedure (discussed in Appendix 7.7.5) to see whether *COMET* helps in reducing the hyperparameter tuning overload. *COMET* can achieve the same level of performance as popular gates with much lesser number of hyperparameter trials. This indicates that *COMET* is not too heavily dependent on a very restricted set of hyperparameter values. We visualize this for various datasets in Fig. 7.5.1. We see tuning reduction by a factor of $5\times - 100\times$ for *COMET* over popular gates.

7.5.1.3 Effect of Local Search on Hyperparameter Tuning

Here, we study how local search can be beneficial in terms of hyperparameter tuning. We study this effect for Hash-r, Top-k and *COMET*. We visualize this in Fig. 7.5.2 for MovieLens for both *Hash-r+*, *Top-k+*and *COMET+*. We observe that we can achieve comparable performance with much smaller number of trials. We see tuning reduction by a factor of $3\times-100\times$ for *Hash-r+*, $20\times-100\times$ for *Top-k+* and $2\times-5\times$ for *COMET+*. This suggests that permutation-based local search helps escape out of bad initializations. Such favorable

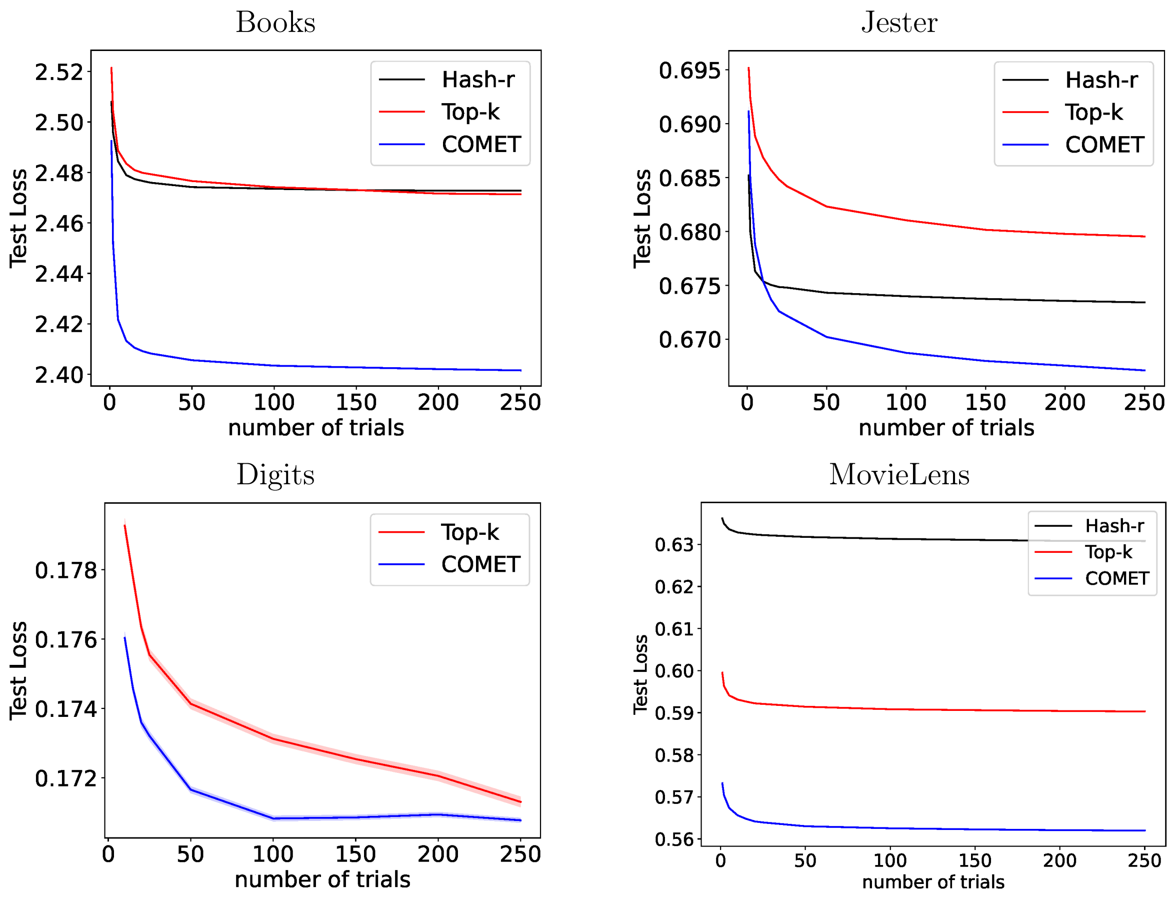


Figure 7.5.1: *Sensitivity of COMET to hyperparameter tuning. COMET can achieve the same level of performance as popular gates (e.g., Hash-r and Top-k) with significantly lesser number of hyperparameter trials. We see tuning reduction by* $5\times-100\times$ *for COMET over Top-k and Hash routing.*

properties of local search in terms of reducing the hyperparameter tuning load for existing gates can be beneficial for Large Language Models.

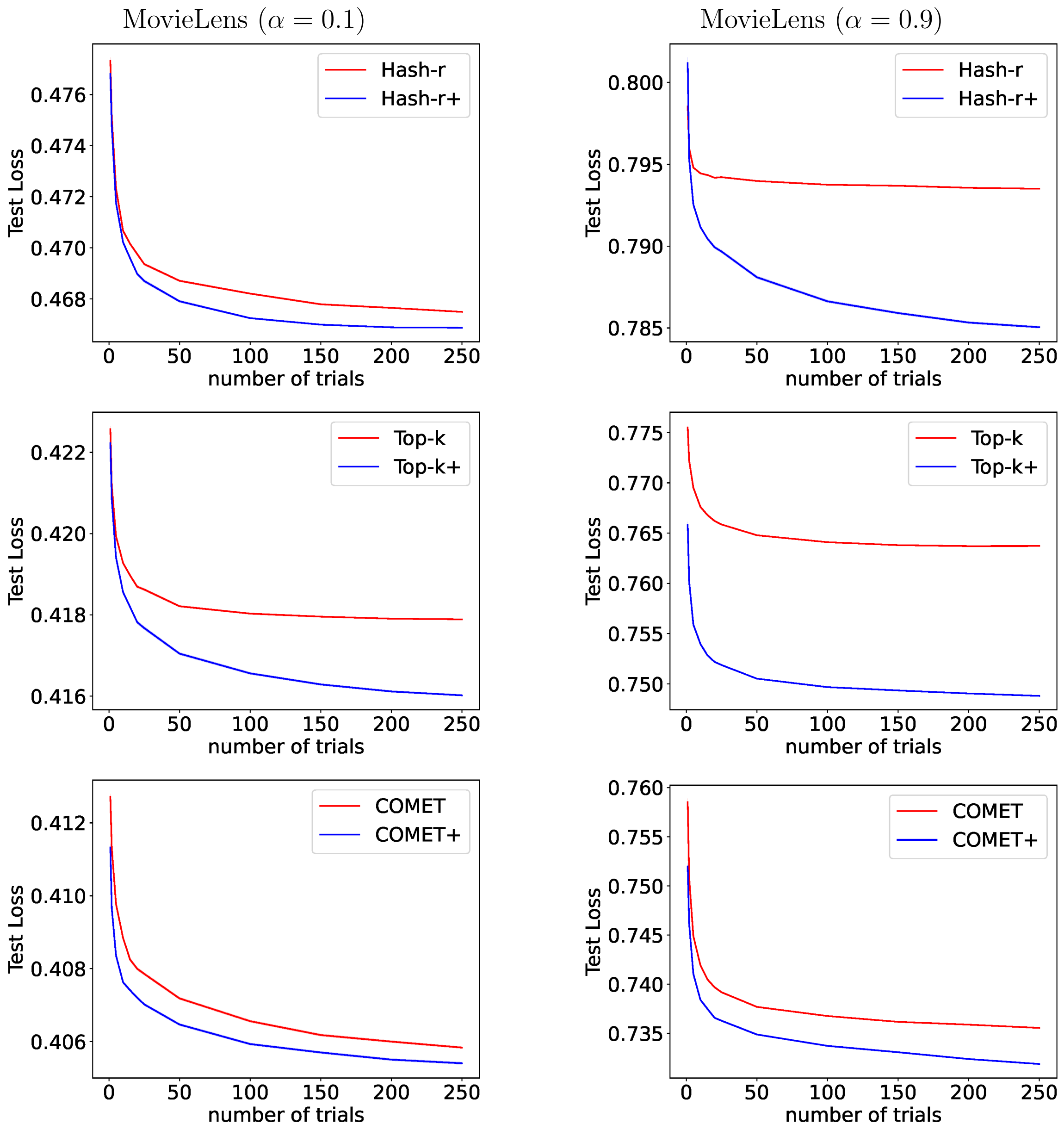


Figure 7.5.2: *Effect of local search on hyperparameter tuning.* Comparison of *Hash-r+* vs Hash-r, *Top-k+* vs Top-k and *COMET+* vs *COMET* on MovieLens with two different task weight settings. Local search appears to achieve the same level of performance with much lesser number of hyperparameter trials. We see tuning reduction by a factor of $3\times-100\times$ for *Hash-r+*, $20\times-100\times$ for *Top-k+* and $2\times-5\times$ for *COMET+*.

7.5.2 Experiments on NLP Tasks

In this section, we consider a setting where a pretrained large model (non-MoE based) needs to be distilled for a more efficient inference while preserving or improving the best performance. Following [298], we study a distillation setting, where BERT [54] is distilled into its Sparse-MoE based variant. Specifically, the FFN layers are replaced with MoE layers — this can result in a $\sim 2\times$ smaller number of (effective) parameters with per-sample sparse routing (for $k = 1$), thus allowing for more efficient inference.

Following [298], we use an importance-weight guided distillation strategy: (i) Finetune BERT on a downstream task. (ii) Compute importance weights in FFN layers to construct an MoE-based variant of BERT. (iii) Distill BERT into MoE-based variant on the downstream task with a layer-wise discrepancy loss. [298] used Hash routing in their MoEBERT model. We propose *COMET-BERT* (MoE based BERT model with *COMET*/ *COMET+* gating) and evaluate the performance on the GLUE benchmarks [257] and SQuAD benchmark [219]. More details about the benchmarks are given in Appendix.

Implementation. We implemented *COMET-BERT* in HuggingFace [263] and adapted the codebase of [298]. Unlike Hash routing, our gates can also cater to $k \geq 1$. However, for consistent comparison in terms of inference, we set $k = 1$. Tuning details are outlined in Appendix. Code for *COMET-BERT* is available at https://github.com/mazumder-lab/COMET-BERT.

Results. We report the performance metrics in Table 7.5.3 for 7 GLUE datasets and SQuAD dataset. *COMET-BERT* outperforms MoEBERT in 5/7 benchmarks on GLUE datasets. *COMET-BERT* also outperform MoEBERT significantly on SQuADv2.0. Notably, in 5 of these datasets (CoLA, MRPC, QNLI and MNLI, SQuAD v2.0), *COMET-BERT* achieves SOTA performance when distilling BERT, (when compared with all distillation methods in literature with same number of effective parameters for inference).

Table 7.5.3: *Performance metrics on the GLUE and SQuAD development sets. Models are trained without data augmentation. Both models have 66M (effective) parameters for inference.*

	GLUE							**SQuAD**
	RTE	**CoLA**	**MRPC**	**SST-2**	**QNLI**	**QQP**	**MNLI**	**v2.0**
	Acc	Mcc	F1	Acc	Acc	F1/Acc	m/mm	F1/EM
MOEBERT[3]	70.8	55.4	91.0	**93.2**	90.9	**88.5/91.4**	84.7	76.8/73.6
COMET-BERT	**71.1**	**57.0**	**91.3**	93.0	**91.2**	88.4/91.3	**85.5**	**78.4/75.3**

7.6 Conclusion

In summary, we propose two new approaches for improving routing in Sparse-MoE. First, we introduce a new differentiable gate *COMET*, which relies on a novel tree-based sparse expert selection mechanism. *COMET* allows optimization with first-order methods, offers explicit control over the number of experts to select, allows (partially) conditional training and sparse inference. Second, in this work, we argue that combinatorial nature of expert selection in Sparse-MoE makes sparse routing optimization challenging with first-order methods. Thus, we propose a new local search method that can help any gate including ours (*COMET*) escape "bad" initializations. Our large-scale experiments on recommender systems, vision and natural language processing tasks show *COMET* and *COMET+*: (i) achieve statistically significant improvements in prediction (up to 13% improvement in AUC) and expert selection over popular sparse gates. (ii) reduce tuning up to a factor of 100× to achieve the same level of performance as popular gates e.g., Top-k and Hash routing. (iii) help *COMET-BERT* achieve state-of-the-art results for distilling BERT on GLUE and SQuAD benchmarks.

[3]Numbers are based on re-run of the official codebase (https://github.com/SimiaoZuo/MoEBERT) with hash routing with the optimal hyperparameters reported in [290].

7.7 Appendix

7.7.1 Proof for Proposition 7.3.1

Proof. First off, it is straightforward to see that $g(x; \alpha, v)$ satisfies the simplex constraint in (7.3.1b):

$$\sum_{i=1}^{m} g(x; \alpha, v)_i = \sum_{i=1}^{m} \frac{\sum_{j=1}^{k} \exp(\alpha_i^{(j)}(x)) \cdot v_i^{(j)}(x)}{\sum_{j=1}^{k} \sum_{i=1}^{m} \exp(\alpha_i^{(j)}(x)) v_i^{(j)}(x)} = 1. \tag{7.7.1}$$

It remains to show that $\|g(x; \alpha, v)\|_0 \le k$ under the given conditions. Recall the hierarchical binary encoding $v^{(j)}$ outputs a one-hot vector for each sample x as the routing decision. Let us denote by $\hat{i}_j$ the expert number selected by j-th tree. For now, let us assume that $\hat{i}_j$ are different for any j. Then, we have

$$g(x; \alpha, v)_{\hat{i}_j} = \exp(\alpha_{\hat{i}_j}^{(j)}(x)) \Big/ \sum_{j \in [k]} \exp(\alpha_{\hat{i}_j}^{(j)}(x)), \tag{7.7.2}$$

i.e., the weights are restricted on the k experts selected by the k trees, and the weights form a softmax activation of logits $\alpha_{\hat{i}_j}^{(j)}$'s. Given that the j-th tree selects $\hat{i}_j$-th expert, we know that if $i \notin \{\hat{i}_1, \hat{i}_2, \dots, \hat{i}_k\}$, $v_i^{(j)} = 0$ for any j, and thus $g(x; \alpha, v)_i = 0$, and this means that the support of $g(x; \alpha, v)$ is contained in the set $\{\hat{i}_1, \hat{i}_2, \dots, \hat{i}_k\}$, which has at most k elements. Therefore, the cardinality constraint in (7.3.1b) holds. $\square$

7.7.2 *COMET* for non-powers of 2

COMET has a natural way to cater to settings where number of experts are non-powers of 2. Recall we have m experts. Let $d = \lceil \log_2 m \rceil$, then $2^{d-1} < m \le 2^d$. Let $\mathcal{T}$ be a full binary tree with depth d and 2^d leaf nodes. We collapse $2(2^d - m)$ leaf nodes to their parents in the $(d-1)$-th level. Each time we collapse two leaf nodes, we get a new node in the $(d-1)$-th level, and the total number of nodes reduce by one. Therefore, we get a tree with

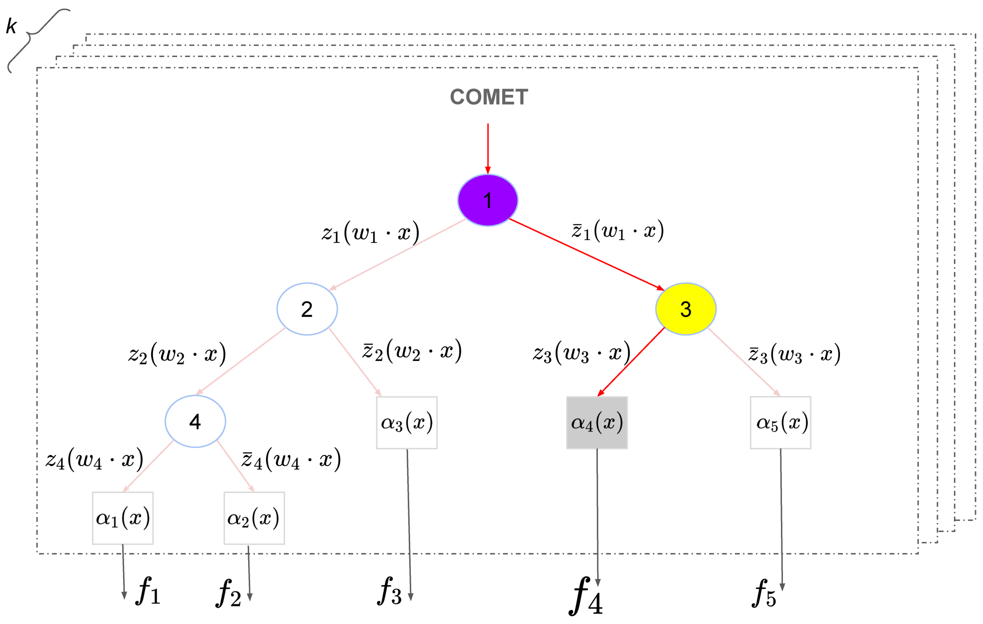


Figure 7.7.1: *COMET for 5 (non-powers of 2) experts.*

m nodes, with $2^d - m$ nodes in the $(d-1)$-th level, and $2m - 2^d$ nodes in the d-th level. This is visualized in Figure 7.7.1 for number of experts equal to 5.

7.7.3 Timing Discussion

7.7.3.1 Cost Complexity

We compare the cost of forward pass *COMET* vs Top-k. We consider a single-task, m experts (ith expert is f_i), p features, desired sparsity $k \ll m$, and some shared layers.

Table 7.7.1: *Time complexity of Top-k and COMET.*

Cost breakdown	Top-k	*COMET* (training)	*COMET* (inference)
Experts	$kO(f_i)$	$k_aO(f_i)$	$k_aO(f_i)$
Gate	$O(pm)+O(m+k\log m)$	$O(kpm)$	$O(kp\log m)$
Shared layers	$O(\text{Shared Layers})$	$O(\text{Shared Layers})$	$O(\text{Shared Layers})$

Cost for Top-k. Here the desired sparsity (k) is always equal to the actual sparsity (k_a) during training and inference, i.e. $k_a \equiv k$. The cost of computing $g(\cdot)$ is given by

$O(pm) + \mathcal{O}(k \log m + m)$. If the cost of computing expert f_i is $O(f_i)$, then the cost of the full model is $k\mathcal{O}(f_i) + O(pm) + \mathcal{O}(k \log m + m) + \mathcal{O}(\text{shared layers})$. Here the main cost is from how many f_i's are evaluated per sample.

Cost for *COMET*. For *COMET*, the cost of the gate is $O(kpm)$. The cost of the full model is $k_a\mathcal{O}(f_i) + O(kpm) + \mathcal{O}(\text{shared layers})$. After a very few epochs (e.g., 4-5), when *COMET* has reached the desired sparsity, we have $k_a \leq k$.

Note that during inference, *COMET* has smaller cost: All root-to-leaf paths in *COMET* do not require evaluation, hence the gate cost reduces from $O(kpm)$ to $O(kp\log(m))$. Hence, due to fewer expert evaluations and smaller gate cost, *COMET* is more efficient at inference time than Top-k. Also see Table 7.7.2 for FLOP counts at inference time for different gating methods.

7.7.3.2 FLOPs Comparison

We compare the FLOP counts of different gating methods (across different models/datasets) to compare inference speed—see Table 7.7.2 below. Inference FLOPs per sample - number of floating point operations that a model performs per sample - is a standard measure to evaluate the inference speed for Sparse-MoE e.g., in [72] etc.

We show that gates that learn sparse routing decisions per sample, e.g., Top-k, DSelect-k, *COMET*, significantly reduce the number of FLOPs ($3\times - 6\times$) at inference time in comparison to dense gates e.g., Softmax. Additionally, we see that in all 4 cases, *COMET* has smaller number of FLOPs ($1.1\times - 1.6\times$) than the highly popular Topk gate. We also outperform DSelect-k in some cases in number of FLOPs. While in some cases, we have larger number of FLOPs than DSelect-k, our AUC is higher (up to 9%) in these cases.

Table 7.7.2: *#FLOPs per-sample at inference time for COMET against benchmarks (Softmax, Top-k, DSelect-k) across various datasets.*

Dataset	Model	FLOPs
Books	Softmax	1195K
	Top-k	402K
	DSelect-k	**214K**
	COMET	326*K*
MovieLens	Softmax	2255K
	Top-k	413K
	DSelect-k	399K
	COMET	**362K**
MultiFashionMNIST	Softmax	7.49M
	Top-k	3.03M
	DSelect-k	**1.58M**
	COMET	2.35M
CelebA	Softmax	22.02M
	Top-k	8.82M
	DSelect-k	5.64M
	COMET	**5.47M**

7.7.3.3 Effect of local search on computation

Inference. Note that the permutation matrix is global and not sample specific. At inference time, multiplying permutation matrix $\boldsymbol{P}$ with $g(x)$ amounts to a reordering of the expert indices – hence, additional cost for this permutation is negligible compared to evaluation of $f(x)$ and $g(x)$.

Training. In the first stage of *COMET* training (a few epochs ~ 5), the training is dense (requiring all experts per sample). For *COMET+*, we also learn the permutation matrix during this stage. There is a small additional computational cost: (a) permutation matrix of size $m \times m$, where m is the number of experts, e.g., 16; (b) cost of Sinkhorn operator which constitutes row/column sum normalizations. This cost is marginal compared to the cost of evaluating the experts $f_i's$, each of which is an MLP/CNN. In the second stage of training, where the samples are being routed to a small $k(=2)$ subset of experts per-sample, there is no additional cost for *COMET* vs *COMET+*. To show an example, for MovieLens 200k, where we learn permutation matrix in first 5 epochs, the total time for 50 epochs (on 4

Table 7.7.3: *Test AUC/Accuracy/MSE for COMET+ and benchmark gates on recommender systems and image datasets.*

Recommender Systems			
	Gate	Task 1 (Test AUC)	Task 2 (Test MSE)
Books (alpha=0.1)	Softmax	56.70±0.16	2.6470±0.0016
	Hash-r	54.55±0.07	2.6791±0.0017
	Top-k	55.28±0.07	2.6783±0.0026
	DSelect-k	59.19±0.36	2.6667±0.0038
	COMET+	**68.18**±0.24	**2.6063**±0.0018
Books (alpha=0.9)	Softmax	77.85±0.01	2.6195±0.0019
	Hash-r	77.32±0.05	2.7152±0.0050
	Top-k	77.46±0.03	2.6942±0.0016
	DSelect-k	77.07±0.09	2.7581±0.0092
	COMET+	**77.95**±0.02	**2.6158**±0.0021
MovieLens (alpha=0.9)	Softmax	90.92±0.01	0.7585±0.0005
	Hash-r	88.95±0.02	0.8065±0.0004
	Top-k	91.25±0.01	0.7635±0.0008
	DSelect-k	91.65±0.02	0.7455±0.0006
	COMET+	**91.70**±0.01	**0.7437**±0.0006
MovieLens (alpha=0.1)	Softmax	85.50±0.03	0.7867±0.0029
	Hash-r	84.27±0.07	0.8279±0.0003
	Top-k	87.12±0.04	0.8005±0.0004
	DSelect-k	**88.16±0.07**	0.7734±0.0005
	COMET+	88.02±0.01	**0.7707**±0.0005
Jester (alpha=0.9)	Softmax	97.350±0.004	0.7460±0.0003
	Hash-r	97.323±0.003	0.7530±0.0003
	Top-k	97.346±0.004	0.7456±0.0006
	DSelect-k	97.361±0.004	0.7464±0.0005
	COMET+	**97.362**±0.004	**0.7439**±0.0006
Jester (alpha=0.1)	Softmax	97.22±0.01	0.7380±0.0003
	Hash-r	97.01±0.01	0.7301±0.0002
	Top-k	97.38±0.01	0.7412±0.0005
	DSelect-k	**97.45**±0.00	0.7273±0.0003
	COMET+	**97.45**±0.00	**0.7257**±0.0004

Image Datasets		
	Gate	Test Accuracy (Averaged across tasks)
MultiFashionMNIST	Softmax	87.99±0.04
	Top-k	88.03±0.03
	DSelect-k	87.42±0.04
	COMET+	**88.12**±0.04
CelebA	Softmax	83.84±0.15
	Top-k	83.95±0.17
	DSelect-k	83.53±0.06
	COMET+	**84.27**±0.08
Digits	Softmax	93.53±0.12
	Top-k	95.34±0.04
	DSelect-k	95.41±0.03
	COMET+	**95.45**±0.04
MultiMNIST	Softmax	98.01±0.03
	Top-k	98.01±0.02
	DSelect-k	98.03±0.02
	COMET+	**98.07**±0.02

GPUs) is given by: 494s for *COMET* and 496s for *COMET+*. Note 50 epochs were sufficient to achieve convergence for both gates.

7.7.4 Task-specific metrics corresponding to Tables 7.5.1 and 7.5.2

We provide task-specific metrics for all recommender systems and image datasets in Table 7.7.3. We observe *COMET+* can give superior AUC performance by up to 13% over Hash routing and Top-k, and 9% over DSelect-k.

7.7.5 Bootstrapping Procedure for studying hyperparameter tuning

We performed 500 tuning trials and performed a bootstrapping procedure as outlined below:

- Randomly sample s ($s \in \{1, 2, 5, 10, 15, \cdots, 250\}$) trials from the bag of a larger set of 500 trials.
- Find the trial with the best validation loss.
- Compute the test loss for that trial.
- Repeat this exercise for 1000 times.
- Compute the average test loss across the best selected trials.

7.7.6 Additional Details for Section 5.1

7.7.6.1 Datasets

MovieLens. MovieLens [88] is a movie recommendation dataset containing records for $\sim 4,000$ movies and $\sim 6,000$ users. Following [259], for every user-movie pair, we construct two tasks. Task 1 is a binary classification problem for predicting whether the user will watch a particular movie. Task 2 is a regression problem to predict the user's rating (in $\{1, 2, \cdots, 5\}$) for a given movie. We use 1.6 million samples for training and 200, 000 for each of the validation and testing sets.

Jester. Jester [82] is a joke recommendation dataset containing records for $\sim 74k$ users and ~ 100 jokes. This gives a dataset of 7.4 million records. Similar to MovieLens above, for every user-joke pair, we construct two tasks. Task 1 is a binary classification problem for predicting whether the user will rate a particular joke. Task 2 is a regression problem to predict the user's rating (in $[-10, 10]$) for a given joke. We use 5.1 million samples for training and 1.1 million samples for each of the validation and testing sets.

Books. Books [294] is a book recommendation dataset containing records for $\sim 105k$ users and $\sim 340k$ books. We filter users and books with each atleast 5 records. This gives a subset of $18,960$ users and $31,070$ books. This gives a subset of 556,724 records. Similar to MovieLens above, for every user-book pair, we construct two tasks. Task 1 is a binary classification problem for predicting whether the user will read a particular book. Task 2 is a regression problem to predict the user's rating (in $\{1, 2, \cdots, 10\}$) for a given book. We use 389,706 samples for training and 83,509 for each of the validation and testing sets.

Digits. We use a mixture of MNIST [53] and SVHN [197] datasets. MNIST is a database of $70,000$ handwritten digits. SVHN is a much harder dataset of $\sim 600,000$ images obtained from house numbers in Google Street View images. We divided the dataset into training, validation and testing as follows: MNIST (#train: 50,000, #validation: 10,000, #test: 10,000) and SVHN (#train: 480,420, #validation: 75,000, #test: 75,000). We combined the corresponding splits to get the train, validation and test sets for the mixture.

MultiMNIST/MultiFashionMNIST. We consider multi-task variants of MNIST/MultiFashionMNIST [53]. The datasets are constructed in a similar fashion as given in [101, 228]: (i) uniformly sample two images from MNIST and overlay them on top of each other, and (ii) shift one digit towards the top-left corner and the other digit towards the bottom-right corner (by 4 pixels in each direction). This procedure leads to 36×36 images with some overlap between the digits. We consider two classification tasks: Task 1 is to classify the top-left item and Task 2 is to classify the bottom-right item. We use 100,000 samples for training, and $20,000$ samples for each of the validation and testing sets.

CelebA. CelebA [170] is a large-scale face attributes dataset with more than $200,000$ celebrity images, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter. We consider 10 of the face attributes in a multi-task learning setting. We use $\sim 160,000$ images for training, and $\sim 20,000$ for each of validation and testing.

7.7.6.2 Architectures

MovieLens. We consider a multi-gate MoE architecture, where each task is associated with a separate gate. The MoE architecture consists of a shared bottom subnetwork comprising two embedding layers (for users and movies). The 128-dimensional embeddings from both layers are concatenated and fed into an MoE Layer of 16 experts, where each expert is a ReLU-activated dense layer with 256 units, followed by a dropout layer (with a dropout rate of 0.5). For each of the two tasks, the corresponding convex combination of the experts is fed into a task-specific subnetwork. The subnetwork is composed of a dense layer (ReLU-activated with 256 units) followed by a single unit that generates the final output of the task.

Books/Jester. We consider a multi-gate MoE architecture, where each task is associated with a separate gate. The MoE architecture consists of a shared bottom subnetwork comprising two embedding layers (for users and books/jokes). The 64-dimensional embeddings from both layers are concatenated and fed into an MoE Layer of $9/16$ experts, where each expert is a ReLU-activated dense layer with 128 units, followed by a dropout layer (with a dropout rate of 0.5). For each of the two tasks, the corresponding convex combination of the experts is fed into a task-specific subnetwork. The subnetwork is composed of a dense layer (ReLU-activated with 256 units) followed by a single unit that generates the final output of the task.

Digits. We use a single-gate MoE with 8 experts. Each of the experts is a CNN that is composed (in order) of: (i) convolutional layer 1 (kernel size = 5, number of filters = 10, ReLU-activated) followed by max pooling, (ii) convolutional layer 2 (kernel size = 5, number of filters = 20, ReLU-activated) followed by max pooling, and (iii) a ReLU-activated dense layer with 50 units. The subnetwork specific to the prediction task is composed of a stack of 3 dense layers: the first two have 50 ReLU-activated units and the third has 10 units followed by a softmax.

MultiMNIST/MultiFashionMNIST. We use a multi-gate MoE with 16/5 experts. Each of the experts is a CNN that is composed (in order) of: (i) convolutional layer 1 (kernel size = 5, #filters = 10, ReLU-activated) followed by max pooling, (ii) convolutional layer 2 (kernel size=5, #filters = 20, ReLU-activated) followed by max pooling, and (iii) a sequence of 2 ReLU-activated dense layers with 50 units each. The subnetwork specific to each of the 2 tasks is composed of a stack of 3 dense layers: the first two have 50 ReLU-activated units and the third has 10 units followed by a softmax.

CelebA. We use a multi-gate MoE with 6 experts. Each of the experts is a CNN that is composed (in order) of: (i) convolutional layer 1 (kernel size = 3, #filters = 4, ReLU-activated) followed by max pooling, (ii) convolutional layer 2 (kernel size=3, #filters = 4, ReLU-activated) followed by max pooling, (iii) convolutional layer 3 (kernel size=3, #filters = 4, ReLU-activated) followed by max pooling, and (iv) convolutional layer 4 (kernel size=3, #filters = 1, ReLU-activated) followed by max pooling, and (v) flatten layer. The subnetwork specific to each of the 2 tasks is composed of a dense layer followed by a sigmoid.

7.7.6.3 Hyperparameters and Tuning

We performed 500 tuning trials for each gate with a random search over the hyperparameter space described below (for each dataset). For each gate, we selected Top 5% of the trials based on validation loss. We report the (average) test loss for the Top 5% trials along with the standard errors in Tables 7.5.1 and 7.5.2.

MovieLens.

- Learning Rates: Uniform in the log range $[5 \times 10^{-5}, 5 \times 10^{-4}]$ for Adam.
- Batch-size: 512.
- Epochs: 100 with early stopping (patience=25) based on validation set.
- γ: Discrete uniform in the set $\{0.01, 0.1, 1, 5, 10\}$ for DSelect-k and *COMET*. γ is fixed to 10 for *COMET+*.
- Entropy: Discrete uniform in the set $\{0.05, 0.1, 0.5, 1, 5, 10\}$ for DSelect-k and *COMET* and *COMET+*.
- Number of epochs for permutation learning: Discrete uniform in the set $\{1, \cdots, 10\}$ for *COMET+* and *Top-k+*.
- ζ (for permutation): 10^{-4}
- m (number of experts): 16.
- k: 2 for all sparse (trainable) gates.
- For Hash-r (and *Hash-r+*), users are randomly pre-allocated to experts (similar to how words in vocabulary are pre-allocated randomly in LLMs)
- Number of tuning trials per gate: 500

Books.

- Learning Rates: Uniform in the log range $[5 \times 10^{-5}, 5 \times 10^{-4}]$ for Adam.
- Batch-size: 2048.
- Epochs: 100 with early stopping (patience=25) based on validation set.
- γ: Discrete uniform in the set $\{0.1, 0.5, 1, 5, 10\}$ for DSelect-k and *COMET*. γ is fixed to 0.5 for *COMET+*.
- Entropy: Discrete uniform in the set $\{1, 5, 10, 50, 100\}$ for DSelect-k and *COMET* and *COMET+*.
- Number of epochs for permutation learning: Discrete uniform in the set $\{1, \cdots, 10\}$ for

COMET+ and *Top-k+*.

- ζ (for permutation): 10^{-4}
- m (number of experts): 9.
- k: 4 for all sparse (trainable) gates.
- For Hash-r (and *Hash-r+*), users are randomly pre-allocated to experts (similar to how words in vocabulary are pre-allocated randomly in LLMs)
- Number of tuning trials per gate: 500

Jester.

- Learning Rates: Uniform in the log range $[5 \times 10^{-5}, 5 \times 10^{-4}]$ for Adam.
- Batch-size: 2048.
- Epochs: 100 with early stopping (patience=25) based on validation set.
- γ: Discrete uniform in the set $\{0.001, 0.02, 0.1, 1, 5, 10\}$ for DSelect-k and *COMET*. γ is fixed to 0.01 for *COMET+*.
- Entropy: Discrete uniform in the set $\{0.05, 0.1, 0.5, 1, 5, 10\}$ for DSelect-k and *COMET* and *COMET+*.
- Number of epochs for permutation learning: Discrete uniform in the set $\{1, \cdots, 10\}$ for *COMET+* and *Top-k+*.
- ζ (for permutation): 10^{-4}
- m (number of experts): 16.
- k: 2 for all sparse (trainable) gates.
- For Hash-r (and *Hash-r+*), users are randomly pre-allocated to experts (similar to how words in vocabulary are pre-allocated randomly in LLMs)
- Number of tuning trials per gate: 500

Digits.

- Learning Rates: Uniform in the log range $[1 \times 10^{-5}, 5 \times 10^{-4}]$ for Adam.
- Batch-size: 512.

- Epochs: 200 with early stopping (patience=25) based on validation set.
- γ: Discrete uniform in the set $\{0.001, 0.01, 0.1, 1\}$ for DSelect-k and *COMET*. γ is fixed to 0.001 for *COMET+*.
- Entropy: Discrete uniform in the set $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$ for DSelect-k and *COMET* and *COMET+*.
- Number of epochs for permutation learning: Discrete uniform in the set $\{1, \cdots, 10\}$ for *COMET+* and *Top-k+*.
- ζ (for permutation): 10^{-4}
- m (number of experts): 8.
- k: 2 for all sparse (trainable) gates.
- Number of tuning trials per gate: 500

MultiMNIST.

- Learning Rates: Uniform in the log range $[1 \times 10^{-4}, 1 \times 10^{-3}]$ for Adam.
- Batch-size: 512.
- Epochs: 200 with early stopping (patience=25) based on validation set.
- γ: Discrete uniform in the set $\{0.001, 0.01, 0.1, 1, 5, 10\}$ for DSelect-k and *COMET*. γ is fixed to 0.01 for *COMET+*.
- Entropy: Discrete uniform in the set $\{0.0001, 0.001, 0.01, 0.1, 1, 5, 10\}$ for DSelect-k and *COMET* and *COMET+*.
- Number of epochs for permutation learning: Discrete uniform in the set $\{1, \cdots, 10\}$ for *COMET+* and *Top-k+*.
- ζ (for permutation): 10^{-4}
- m (number of experts): 16.
- k: 4 for all sparse (trainable) gates.
- Number of tuning trials per gate: 500

MultiFashionMNIST.

- Learning Rates: Uniform in the log range $[1 \times 10^{-4}, 1 \times 10^{-3}]$ for Adam.
- Batch-size: 512.
- Epochs: 200 with early stopping (patience=25) based on validation set.
- γ: Discrete uniform in the set $\{0.001, 0.01, 0.1, 1, 5, \}$ for DSelect-k and *COMET*. γ is fixed to 0.01 for *COMET+*.
- Entropy: Discrete uniform in the set $\{0.001, 0.01, 0.1, 1, 5\}$ for DSelect-k and *COMET* and *COMET+*.
- Number of epochs for permutation learning: Discrete uniform in the set $\{1, \cdots, 10\}$ for *COMET+* and *Top-k+*.
- ζ (for permutation): 10^{-4}
- m (number of experts): 6.
- k: 2 for all sparse (trainable) gates.
- Number of tuning trials per gate: 500

CelebA.

- Learning Rates: Uniform in the log range $[1 \times 10^{-4}, 1 \times 10^{-3}]$ for Adam.
- Batch-size: 512.
- Epochs: 100 with early stopping (patience=25) based on validation set.
- γ: Discrete uniform in the set $\{0.001, 0.01, 0.1, 1, 5\}$ for DSelect-k and *COMET*. γ is fixed to 5 for *COMET+*.
- Entropy: Discrete uniform in the set $\{0.001, 0.01, 0.1, 1, 5\}$ for DSelect-k and *COMET* and *COMET+*.
- Number of epochs for permutation learning: Discrete uniform in the set $\{1, \cdots, 10\}$ for *COMET+* and *Top-k+*.
- Entropy for permutation: 10^{-4}
- k: 2 for all sparse gates.
- Number of tuning trials per gate: 100

7.7.7 Additional Details for Section 5.2

7.7.7.1 Datasets

GLUE. General Language Understanding Evaluation (GLUE) benchmark [257], is a collection of natural language understanding tasks. Following previous works on model distillation, we consider SST-2 [237], CoLA [260], MRPC [57], STSB [39], QQP, and MNLI [262] and exclude STS-B [39] and WNLI [157] in the experiments. The datasets are briefly summarized below:

- SST-2 [237] is a binary single-sentence classification task that classifies movie reviews to positive or negative;
- CoLA [260] is a linguistic acceptability task;
- MRPC [57] is a paraphrase detection task;
- QQP is a duplication detection task;
- MNLI [262], QNLI [218], and RTE [52] are natural language inference tasks.

Dataset details are summarized in Table 7.7.4.

Table 7.7.4: *Summary of GLUE benchmark.*

Corpus	Task	#Train	#Dev	#Test	#Label	Metrics
Single-Sentence Classification (GLUE)						
CoLA	Acceptability	8.5k	1k	1k	2	Matthews correlation
SST-2	Sentiment	67k	872	1.8k	2	Accuracy
Pairwise Text Classification (GLUE)						
MNLI	NLI	393k	20k	20k	3	Accuracy
RTE	NLI	2.5k	276	3k	2	Accuracy
QQP	Paraphrase	364k	40k	391k	2	Accuracy/F1
MRPC	Paraphrase	3.7k	408	1.7k	2	Accuracy/F1
QNLI	QA/NLI	108k	5.7k	5.7k	2	Accuracy

SQuAD. We evaluate our sparse routing approaches on question answering dataset: SQuAD v2.0 [219]. This task is treated as a sequence labeling problem, where we predict the probability of each token being the start and end of the answer span. Statistics of the question answering dataset (SQuAD v2.0) are summarized in Table 7.7.5.

Table 7.7.5: *Summary of SQuAD benchmark.*

Corpus	Task	#Train	#Dev	Metrics
SQuAD v1.1	Question Answering	87.6k	10.6k	F1/EM
SQuAD v2.0	Question Answering	130k	11.9k	F1/EM

7.7.7.2 Tuning Procedure for MoEBERT and *COMET-BERT*

Following [298], we followed the 3-step process as outlined in the MoEBERT codebase[4]:

- We finetuned BERT on each downstream task for a set of 50 random hyperparameter trials over the following set:
  - Learning Rate: Discrete uniform over the set $\{1\times10^{-5}, 2\times10^{-5}, 3\times10^{-5}, 5\times10^{-5}\}$
  - Batch size: Discrete uniform over the set $\{8, 16, 32, 64\}$
  - Weight Decay: Discrete uniform over the set $\{0, 0.01, 0.1\}$
  - Epochs: 10

  Note that this step matched the performance numbers reported for BERT-base in Table 1 of [298]. We used the best model (for each dataset) for the remaining steps below.

- Compute importance weights in FFN layers to construct an MoEBERT/COMET-BERT model, where FFN layers are replaced with MoE layers with the weight assignment strategy in [298].

- Distill BERT into MoEBERT or *COMET-BERT* on the downstream task with a layer-wise discrepancy loss. For MoEBERT, we used the optimal hyperparameters reported (based on $\sim$ 1000 trials per dataset) in Table 7 of Appendix in [298]. For *COMET-BERT*, we performed 100 tuning trials via random search with each *COMET* and *COMET+* and picked the best results based on development datasets. The hyperparameters were randomly selected from the following sets:

[4]https://github.com/SimiaoZuo/MoEBERT

- Learning Rate: Discrete uniform over the set $\{1\times10^{-5}, 2\times10^{-5}, 3\times10^{-5}, 5\times10^{-5}\}$
- Batch size: Discrete uniform over the set $\{8, 16, 32, 64\}$
- Weight Decay: Discrete uniform over the set $\{0, 0.01, 0.1\}$
- Distillation Regularization ($\lambda_{distill}$ in [298]): Discrete uniform over the set $\{1, 2, 3, 5\}$.
- γ (for smooth-step for *COMET*): Discrete uniform over the set $\{0.01, 0.1, 1.0\}$.
- λ (for entropy regularization for *COMET*): Discrete uniform over the set $\{0.05, 0.1, 0.5, 1, 5, 10\}$.
- Epochs: 50 for small datasets (CoLA, RTE, MRPC) and 25 for large datasets (SST-2, MNLI, QQP, QNLI, SQuADv2.0). Best model was recovered on development set on best checkpoint.

Chapter 8

Sparse Mixture of Experts: An Effective Sampling-based Routing Approach

8.1 Introduction

The Sparse Mixture of Experts (Sparse MoE) [233] is a promising framework for scaling up model training. Sparse-MoE consists of a set of trainable experts (neural networks) and a trainable gate. The gate adaptively selects a subset of experts on a per-input basis during model training. This adaptive selection makes it possible to train Sparse MoE models that are orders of magnitude larger than densely activated models (i.e., models that use all parameters to process each input) without significant increase in training costs [72, 156, 233]. In addition, these sparsely activated models can learn faster than their compute-matched dense models. For example, Switch Transformers [72], based on the Top-k gate, can learn $7\times$ faster than dense T5 model [217]—the number of FLOPs used per-input during training is the same for both models. Such Sparse MoE models have shown state-of-the-art performance in many learning tasks [8, 60, 72, 227, 233].

The choice of gate plays a central role in Sparse-MoEs. For efficient training, the literature in Sparse-MoEs has been primarily based on Top-k gating [233], which selects k out of

m experts using a top-k operation. Top-k gating is simple and efficient because it allows conditional training: in backpropagation, for each input example, only the gradients of the loss with respect to k experts need to be computed. With a customized implementation, conditional training can lead to large computational savings. There have been various follow-up works that have proposed variants of Top-k gating [48, 156, 227, 290, 295]. These approaches fundamentally rely on a top-k operation to exploit conditional training. Prior literature [72, 101, 227] has highlighted performance and stability issues with Top-k gating. Some differentiable gating approaches [101, 117, 229] were proposed to mitigate these issues. However, these approaches are more expensive per training step as they require gradients with respect to more than k experts.

In this chapter, we focus on improving k-sparse gating in Sparse MoEs. Specifically, we propose *MOESART*: a novel sampling-based approach for conditional training in Sparse MoEs. We model the gating function as a learnable parameterized softmax distribution. To achieve sparsity, we sample k times from the parameterized distribution, identifying the selected experts per-input. We then assign weights to the selected experts; these weights are adjusted to ensure that the resulting prediction is a good approximation of the standard softmax gate. In expectation, the adjusted gate weights serve as a good sparse approximation of the gate probabilities of the dense softmax.

Although Top-k is considered to be a k-sparse approximation of the classical (dense) MoE models [50], we empirically show that our k-sparse approximation approach appears to be superior than Top-k based approximations for learning gating. The greedy nature of Top-k leads to a biased estimation strategy. Top-k gating can potentially suffer from selection bias [203], where a suitable expert is ranked too low by the gate for an input example to be sufficiently exposed to during training. This approach may ignore potentially informative experts whose contribution is overshadowed by the top-k operation. Consequently, it might not fully leverage the diversity and richness of the expert pool, while optimizing the combinatorially challenging gating problem. In contrast, our approach can mitigate this selection bias as any expert can be selected because of sampling during the course of training.

Contributions. Our technical contributions are as follows:

- We propose *MOESART*: a novel sparse gating approach for Sparse Mixture of Experts. Unlike existing gates, *MOESART* aims at learning a good k-sparse approximation of the classical, softmax gate [123]. We achieve this through sampling and carefully designed expert re-weighting strategies. We empirically show that our training approach learns a better sparse approximation of the classical (dense) MoE models than Top-k style gates.
- *MOESART* maintains k-sparsity during both training and inference, hence allowing for conditional training and inference. Conditional training allows sparse backpropagation, where for each input example, only the gradients of the loss w.r.t. k experts need to be computed. This is similar to Top-k gating [233].
- On standard recommender systems and vision datasets, *MOESART* achieves 15%-16% improvement in performance of MoE models over SOTA k-sparse gates: Top-k [233], V-MoE [227], Expert Choice [290], and X-MoE [48].
- We also present results for a 1 billion MoE-Transformer on two machine translation tasks: German $\rightarrow$ English (IWSLT'14) and English $\rightarrow$ French (WMT'14). *MOESART* improves by 2.1 BLEU points over Top-k gate on German $\rightarrow$ English translation task and performs comparably on English $\rightarrow$ French translation task.
- In distillation of pre-trained BERT (non-MoE model) into MoEBERT variant for efficient inference, *MOESART* consistently improves over Top-k on 7 GLUE and 2 SQuAD benchmarks by 0.5% (up to 1.5%).

8.2 Related work

The MoE framework was introduced by Jacobs et al. [123]. Shazeer et al. [233] proposed a *Sparse*-MoE framework which routes each input to a subset of experts and showed good performance on NLP tasks. This sparked off a series of works on gating strategies in the

Sparse-MoE paradigm. These can be categorized and summarized as follows:

- *Sparse gates.* Sparse gates exactly activate a user-specified number of experts in each training step on a minibatch of inputs. Shazeer et al. [233] proposed Top-k gating, which selects k experts per input. Fedus et al. [72], Lepikhin et al. [156], Ruiz et al. [227] proposed variants of Top-k gate and showed better performance on various tasks. Chi et al. [48] proposed to compute gating scores on a low-dimensional hypersphere. Zhou et al. [290] proposed to let experts select top inputs.
- *Gating as assignment.* Lewis et al. [158] and Clark et al. [50], Liu et al. [169] formulate selection as an assignment problem using linear programming and optimal transport for balanced assignment. All these approaches are also sparse but more expensive than Top-k style gates.
- *Differentiable gates.* Hazimeh et al. [101] and Ibrahim et al. [117] propose differentiable gates, which improve over Top-k in terms of stability and statistical performance. Although these gates allow conditional inference, they can only partially allow for conditional training (during a later stage of training with customized implementations) and are thus more expensive during training. Sander et al. [229] proposed a differentiable relaxation of the Top-k operator. Their smoothing approach does not always guarantee exact k-sparsity. It also requires solving an optimization subproblem per-input during inference, which can be more costly.

The goal of our work is to propose a new k-sparse gating approach such that the per-minibatch compute cost is the same as Top-k style gates. A related line of work focuses on stochastic k-subset selection in non-MoE settings. Chen et al. [44], Paulus et al. [205], Xie and Ermon [270] propose differentiable methods for sampling k-subsets from a categorical distribution, based on generalizations of the Gumbel-softmax trick [125, 177]. If these methods were to be applied to Sparse-MoE, they would perform dense training (i.e., the gradients of all experts, even if not selected, will be computed during backpropagation), hence limiting their utility. In contrast, *MOESART* exploits sparsity for conditional training.

8.3 Routing in Mixture of Experts

We first review the classical MoE learning paradigm. We assume that the task has an input space $\mathcal{X} \subseteq \mathbb{R}^p$ and an output space $\mathcal{Y} \subseteq \mathbb{R}^u$. In the MoE framework, the prediction function has two components: (i) a set of m experts (neural networks) $f_i : \mathcal{X} \to \mathbb{R}^u$ for any $i \in [m] := \{1, 2, \dots, m\}$, and (ii) a gate $g : \mathcal{X} \to \Delta_m$ that outputs weights in the probability simplex $\Delta_m = \{g \in \mathbb{R}^m : \sum_i g_i = 1, g \geq 0\}$. Given a sample $x \in \mathcal{X}$, MoE combines the expert outputs as follows: $\sum_{i=1}^m f_i(x)g(x)_i$. Recall that classical (dense) MoE minimizes the following objective:

$$\min_{\{f_i\},g} \hat{\mathbb{E}} \left[\ell \left(y, \sum_{i \in [m]} f_i(x)g(x)_i \right) \right], \tag{8.3.1}$$

where $\hat{\mathbb{E}}$ denotes expectation over the training dataset $\mathcal{D} = \{(x_1, y_1), \cdots, (x_N, y_N)\}$, $\ell(\cdot)$ denotes the loss function used during training, and $g(x)$ is a softmax gate, which is typically expressed as $g(x) = \text{softmax}(Ax + b)$, where $A \in \mathbb{R}^{m \times p}$ and $b \in \mathbb{R}^m$ denote learnable gate parameters.

Different from the classical MoE above, Sparse-MoEs use a gate that selects a convex combination of k out of the m experts per-input, where typically $k \ll m$. Next, we discuss some state-of-the-art sparse gates for parameterizing $g(\cdot)$.

- **Top-k**: The Top-k gate [233] is defined as $g(x) := \text{softmax}(\text{Top}k(Ax + b, k))$, where for any vector v, $\text{Top}k(v, k)_i := v_i$ if v_i is in the top k elements of v, and $-\infty$ otherwise. This is also used by state-of-the-art open-source model Mixtral-8x7B [127].
- **V-MoE**: Lepikhin et al. [156], Ruiz et al. [227] proposed to reorder the softmax$(\cdot)$ and Top$k$ operation in the Top-k gate. Ruiz et al. [227] parameterizes the gate as follows: $g(x) := \text{Top}k(\text{softmax}(Ax + b + \epsilon), k)$, where $\epsilon \sim \mathcal{N}(0, \frac{1}{m^2})$ and for any vector v, $\text{Top}k(v, k)_i := v_i$ if v_i is in the top k elements of v, and $\text{Top}k(v, k)_i := 0$ otherwise. We denote this gate as V-MoE.

- **S-MoE:** Fedus et al. [72] parameterize the gate with multiplicative noise as follows: $g(x) := \text{Top}k(\text{softmax}(Ax\epsilon + b), k)$, where $\epsilon \sim \mathcal{U}(0.98, 1.02)$.
- **Expert Choice**: Zhou et al. [290] define their gate on a minibatch as: $G(X_{\mathcal{B}}) := \text{Top}k(\text{softmax}(AX_{\mathcal{B}} + b)^T, k')^T$, where for any vector v, $\text{Top}k(v, k')_i := v_i$ if v_i is in the top k' elements of v, and zero otherwise. Note that $k' = |\mathcal{B}| * k/m$, where k is the (average) sparsity level per-input such that each expert $i \in [m]$ selects k' samples in a batch of size $|\mathcal{B}|$.
- **X-MoE**: Chi et al. [48] estimate the gating scores on a low-dimensional hypersphere as follows: $g(x) := \text{Top}k(\text{softmax}(((APx)/(\|A\|_{L_2} \|Px\|_{L_2}))/\tau), k)$, where $P \in \mathbb{R}^{\frac{m}{2} \times p}$, $A \in \mathbb{R}^{m \times \frac{m}{2}}$, $\tau \in \mathbb{R}_+$ are learnable.

Both Top-k (or its variants above) and Softmax gates have their pros and cons. Top-k style gates allow for conditional training, i.e., in the forward pass, for each minibatch of size B, only kB (instead of mB) expert evaluations (i.e., $f_j(x)$) are required, and hence in backpropagation, only the gradients of the loss with respect to kB elements need to be computed, allowing computational savings. However, the discontinuous nature and selection bias in Top-k can lead to challenges during optimization. In contrast, the softmax gate is smooth, hence can be easier to optimize. However, the softmax gate can be computationally expensive during both training and inference: the gate score for each expert is non-zero; hence, all experts $f_i(x)$ are used per-input x.

Motivation. Our approach tries to combine the benefits of the two approaches. We propose to learn a sparse approximation of the softmax gate via sampling and reweighting such that our gate has the following desirable properties: (i) Per-step training costs are similar to Top-k, which is a central consideration in Sparse MoE models. (ii) As we demonstrate in our experiments, sampling can reduce the selection bias prevalent in greedy Top-k based selection approaches, especially in the early stages of optimization, reducing the likelihood of bad solutions.

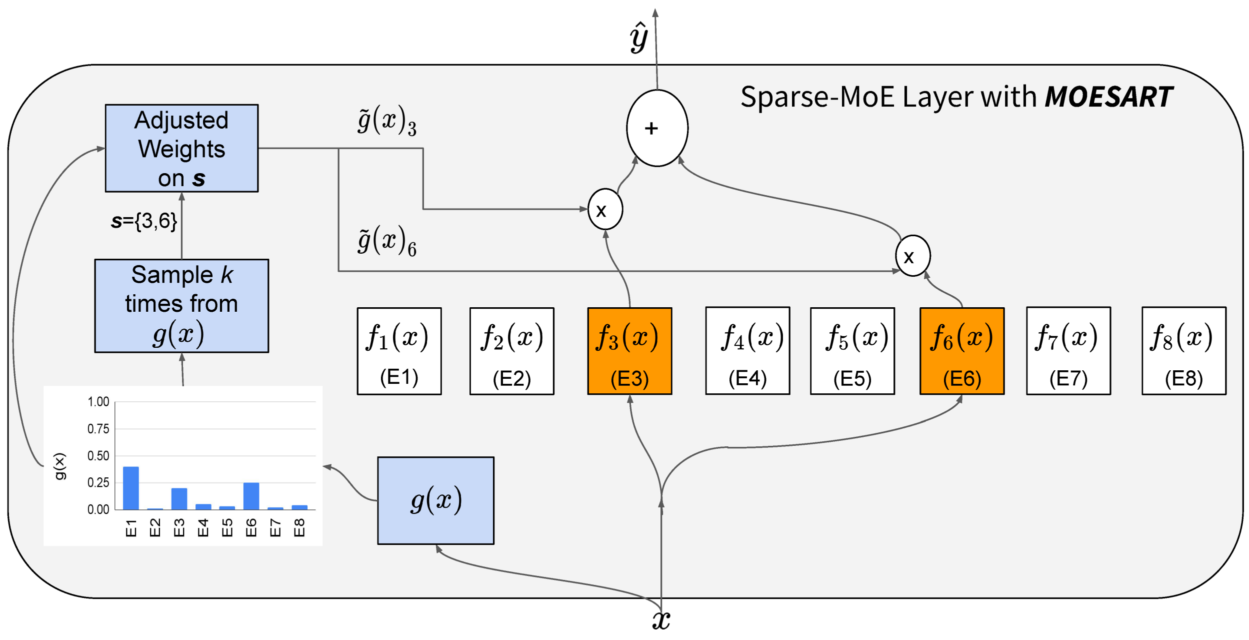


Figure 8.3.1: *Sparse-MoE with MOESART gate.*

8.3.1 *MOESART* for Conditional Computing

We formulate a sparse approximation of the classical MoE objective in Problem (8.3.1). The goal is to train a sampled version of the softmax gate to approximate the softmax gate during the course of training. The high-level idea (visualized in Fig. 8.3.1) can be summarized as follows. For each input x, we sample a subset of experts from a parameterized distribution $g(x)$, and use a modified version of $g(x)$ on the *sampled set* as a sparse approximation for conditional computation.

Let us denote the dense softmax gate by the function $g(\cdot)$. For each input x, we sample k times from the discrete set $\{1, \cdots, m\}$ with distribution $g(x)$. We define a random vector $r(x) = (r_1, \cdots, r_m) \in \mathbb{Z}^m_{\geq 0}$, which denotes the number of times each expert is sampled. We denote the unique subset of sampled experts by $s(x)$, which can be derived from $r(x)$ as follows: $s(x) := \{i : r(x)_i > 0\}$. The cardinality of set $s(x)$ is given by $|s(x)| \leq k$. For instance, as shown in Fig. 8.3.1, $s(x) = \{3, 6\}$ and $r(x) = (0, 0, 1, 0, 0, 1, 0, 0)$ represents a sample of size $k = 2$, where 3rd and 6th expert were sampled once. Similarly, $s(x) = \{1, 5\}$ and $r(x) = (1, 0, 0, 0, 2, 0, 0, 0)$, depict the case where the first expert was sampled once, and fifth expert was sampled twice. Note that $\sum_{i=1}^{m} r(x)_i = k$.

We seek to minimize the following objective:

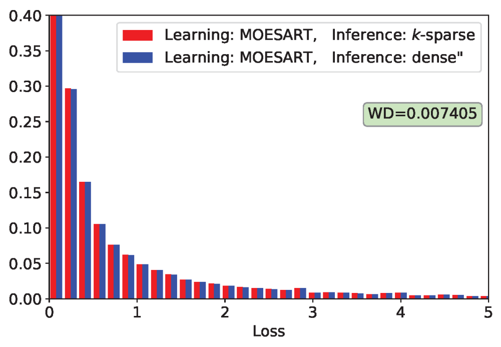
$$\min_{\{f_i\},g} \hat{\mathbb{E}} \left[\hat{\mathbb{E}}_{r(x)} \left[\ell \left(y, \sum_{i \in s(x)} f_i(x) \tilde{g}(x)_i \right) \right] \right], \tag{8.3.2}$$

where the outer expectation in Problem (8.3.2) is over the training set $\mathcal{D}$ and the inner expectation is computed with respect to a subset of experts in the set $s(x)$ for each input x. $g(x) = \text{softmax}(Ax + b)$ and $\tilde{g}(x)$ is a modified version of $g(x)$ on the set $s(x)$ — more details are given in Section 8.3.1.1. Additional regularization can be added to the objective, which is discussed in Section 8.3.2.

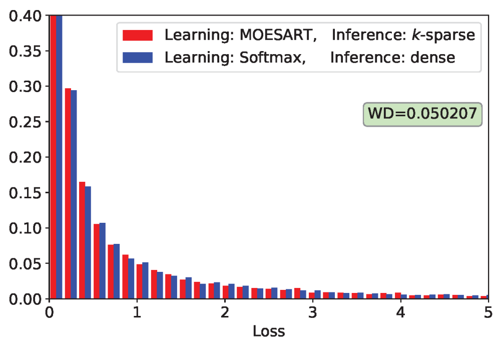
Next, we empirically show how our *MOESART* objective, *without* any additional regularization, learns a good k-sparse approximation to the dense softmax gate in terms of the empirical loss, superior than that learnt by Top-k gate.

***MOESART* is a good k-sparse approximation.** *MOESART* leads to a more effective sparse approximation than that learnt by Top-k. We empirically show this on SVHN dataset for an MoE architecture with 8 convolutional experts. The details of the architecture are outlined in Appendix Section 8.6.2.9. We trained MoE models with 3 different gating strategies: Classical MoE (softmax), Top-k (sparse) and *MOESART* (sparse). We optimized with Adam [136] with cross-entropy loss for 250 epochs. We set $k = 2$ for Top-k and *MOESART*.

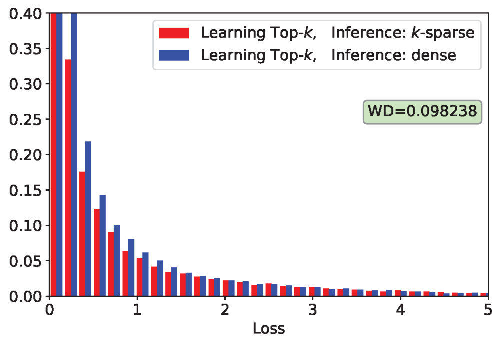
We visualize the different loss distributions on held-out data in Figure 8.3.2. In particular, we visualize the distributions, when each of the 3 gates are used to perform dense or sparse inference post-training. We also show the discrepancy between different distributions using Wasserstein Distance [133, 211]. We can see in Figure 8.3.2a that *MOESART* has a small distribution discrepancy when the model performs sparse or dense inference. In comparison, we can see a much bigger discrepancy when the model performs dense inference when the model is trained with Top-k gating — see Figure 8.3.2c. We also see in Figure 8.3.2e that



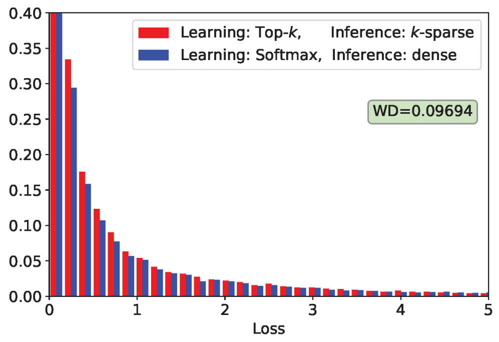
(a) *MOESART: sparse vs dense inference*



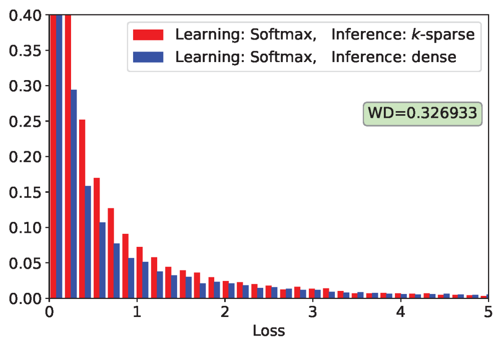
(b) *MOESART vs Softmax*



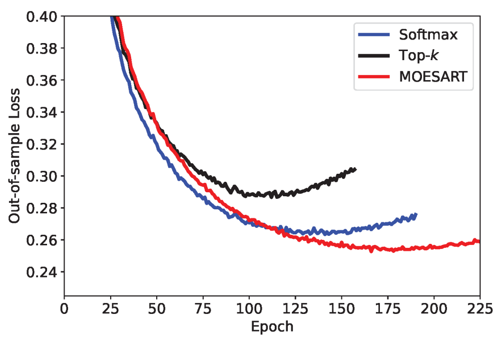
(c) *Top-k: sparse vs dense inference*



(d) *Top-k vs Softmax*



(e) *Softmax: sparse vs dense inference*



(f) *Out-of-sample loss during training*

Figure 8.3.2: *Comparison of empirical loss distributions on held-out samples from SVHN dataset when MoE models are trained with three different objectives: Softmax (classical MoE), Top-k and MOESART (no additional regularization). The y-axis in 8.3.2a-8.3.2d is zoomed in to highlight differences between distributions. We evaluate with k-sparse and dense approximations at inference time for all three models. Wasserstein Distance (WD) is shown to quantify the distance between loss distributions. We can see that MOESART provides a good k-sparse approximation to its dense counterpart (see 8.3.2a) as well as to the classical MoE (see 8.3.2b). Note in 8.3.2f that classical MoE, MOESART, and Top-k achieve held-out objective of* 0.259, 0.251 *(lowest) and* 0.282 *respectively.*

when the model is trained densely with classical MoE objective, a large discrepancy between loss distributions occurs when the model performs sparse versus dense inference — note that we used Top-k as a sparse approximation at inference time in this setting. We also compare in Figure 8.3.2b and 8.3.2d distribution discrepancy of the loss distribution of *MOESART* and Top-k with that of classical MoE. Interestingly, we see *MOESART* is closer to classical MoE loss distribution than Top-k. Finally, in Figure 8.3.2, we show the out-of-sample loss during the course of training. *MOESART* achieves a much smaller objective than Top-k; interestingly, it even surpasses the densely-trained classical MoE.

8.3.1.1 Computing $\tilde{g}$ during training and inference

Here, we outline approaches to compute $\tilde{g}$ during training and inference. $\tilde{g}$ in (8.3.2) is a modification of g on the sampled indices from g, which serves three purposes: (a) $\tilde{g}$ is sparse as it's only non-zero on the sampled indices, hence allowing conditional computation. (b) $\tilde{g}$ allows gradient to backpropagate to g — recall that just sampling from parameterized distribution g doesn't allow gradient computation with respect to gate parameters in g. (c) $\tilde{g}$ aims to approximate g in expectation.

Training. We first introduce some notation. Let $\{a_i\} \in \mathbb{R}^p \ \forall i \in [m]$ and $b_i \in \mathbb{R}^m$ be learnable gate parameters and $\tau > 0$ be a temperature hyperparameter. Let $o(x) = (Ax + b)/\tau$ be gate logits, where $A := (a_1 \ \cdots \ a_m)^T$. Equivalently, $o_i(x) := (a_i^T x + b_i)/\tau \ \forall i \in [m]$, and $g(x)_i := \exp(o_i(x))/\sum_{j=1}^{m} \exp(o_j(x))$. Next, we refer to $\tilde{o}(x)$ as adjusted logits after the sampling process. $\tilde{g}(x)$ is a softmax-transformed version of adjusted logits $\tilde{o}(x)$. Note that $\tilde{g}(x)$ will be sparse with cardinality $\|\tilde{g}(x)\|_0 \leq k$. For notational convenience, we drop the dependence on x, and write $o(x) = o, \tilde{o}(x) = \tilde{o}, s(x) = s, g(x) = g, \tilde{g}(x) = \tilde{g}$. We first outline some natural choices for $\tilde{o}$ below and highlight their limitations.

(i) $\tilde{o}_i := \{o_i \text{ if } i \in s \ \}$ or $\{-\infty \text{ if } i \notin s\}$. This corresponds to $\tilde{g}_i = g_i / \sum_{j \in s} g_j \ \forall i \in s$.

(ii) $\tilde{o}_i := \{o_i + \log(r_i) \text{ if } i \in s \ \}$ or $\{-\infty \text{ if } i \notin s\}$. This gives $\tilde{g}_i = g_i r_i / \sum_{j \in s} g_j r_j \ \forall i \in s$.

(iii) $\tilde{o}_i := \{o_i + \log(r_i) - \log(k g_i) \text{ if } i \in s\}$ or $\{-\infty \text{ if } i \notin s\}$. This gives $\tilde{g}_i = 1/k \ \forall i \in s$.

The above approaches have limitations, which make them either less appealing or ill-posed. If we first consider (i)-(ii), we get $\tilde{g}_i$ to be biased estimators of g_i — see Appendix Section 8.6.2.1 for examples. If we consider (iii), the adjusted logit corrects the true logit o_i by the *expected* number of occurrences of an expert i. This correction makes $\tilde{g}_i$ an unbiased estimator of g_i — see proof in Appendix Section 8.6.1; however, the correction makes $\tilde{g}_i$ independent of g_i, causing gradients with respect to gate parameters A and b to be zero — hence gate parameters can not be updated.

Therefore, we propose an alternative strategy, which tends to have a smaller bias than the strategy in (i)-(ii), and has non-zero gradients with respect to gate parameters unlike (iii). The idea is to follow a randomized adjustment strategy for the sampled logits o_i $\forall i \in s$. We randomly sample an index z uniformly in the subset s. We propose the following adjustment strategy:

$$\tilde{o}_i \coloneqq \begin{cases} o_i + \log(r_i), & i = z \\ o_i + \log(r_i) - \log((k-1)g_i), & i \in s \backslash \{z\} \\ -\infty & i \notin s \end{cases} \tag{8.3.3}$$

This strategy tends to have a similar bias in comparison with the strategies in (i)-(ii) for uniform distribution g and have a much smaller bias than (i)-(ii) for non-uniform distributions for g — see ablation study in Appendix Section 8.6.2.1. Note that gate probability g is expected to have a range of distributions across inputs, which makes our proposed strategy less biased during training. Additionally, this approach has non-zero gradients with respect to the gate parameters A and b, allowing the gate to learn unlike the unbiased strategy in (iii) above. We empirically observed that sampling without replacement performed significantly better than sampling with replacement in the context of Sparse MoE training — see an ablation study in Section 8.4.3. We also show that our proposed strategy in (8.3.3) significantly outperforms (i-ii) in the context of Sparse MoE training on image datasets in Table 8.6.1 in

Appendix Section 8.6.2.2.

Note that all the approaches in (i)-(iii) and the one we propose require $k > 1$; for $k = 1$, these approaches have zero gradients with respect to the gate parameters, this issue arises in Top-k gate as well as highlighted by Ruiz et al. [227]; Our motivation to consider $k > 1$ is also based on earlier work [60, 127, 156, 227, 233, 290, 295]. They consider $k > 1$ for good trade-off between predictive performance and the training/serving efficiency.

Inference. There are two important considerations in Sparse MoE models at inference time: (a) Similar to conditional training, k-sparse inference is crucial for efficient serving of large-scale MoE models. As highlighted by the Figure 8.3.2a, there is a small discrepancy between the loss distributions when the models perform sparse or dense inference when trained with *MOESART* objective. Hence using k-sparse solution works well at inference. (b) The gating should be deterministic for better interpretability and reproducibility. Therefore, we follow a k-sparse deterministic strategy at inference time. Instead of sampling from $g(x)$, we select the indices corresponding to the top k elements of $g(x)$. We use the adjustment strategy in (iii) above — note that this leads to an equally weighted average of the top k expert predictions per input. This deterministic inference strategy led to a smaller out-of-sample loss in comparison to using sampling at inference.

8.3.2 Additional Trimmed Lasso regularization

We can also add (per-input) Trimmed Lasso regularization [22] in the objective (8.3.2). Trimmed Lasso regularization is defined as $\lambda \sum_{j>k} T(g(x))_j$, where $T(v)$ sorts the elements of v in descending order, and $\lambda \geq 0$ is a non-negative penalty. This regularization can encourage $g(x)$ to accumulate gate probability mass per-input in the top k elements, making the sampling process more deterministic per-input by the end of training. Note that this regularization doesn't affect the k-sparse training characteristics of the *MOESART* objective (8.3.2).

8.3.3 *MOESART* versus S-MoE(S).

Fedus et al. [72] also considered a sampling version of S-MoE i.e., S-MoE(S) in their Appendix. In one experiment, they showed that S-MoE(S) can degrade performance compared to Top-k based S-MoE. Although both approaches i.e., *MOESART* and S-MoE(S) perform sampling, there are key differences in parameterization and regularization:

(i) S-MoE(S) considers the following parameterization: $g(x) = g(x) \odot \mathbf{1}_S$ where $\mathbf{1}_S$ denotes a vector such that only k sampled indices are non-zero that are in the set S. This does not obey the simplex constraint. In contrast, *MOESART* is based on weight adjustment strategy in (8.3.3) and these adjusted weights always lie on the sparse simplex. The bias of S-MoE(S) is higher than that for *MOESART*, which can degrade performance of S-MoE(S).

(ii) S-MoE(S) is stochastic during training and inference. *MOESART* uses top-k indices at inference.

(iii) There is no trimmed lasso regularization in S-MoE(S). Trimmed lasso regularization can also boost performance on some tasks (Appendix Section 8.6.2.4).

MOESART consistently outperforms S-MoE(S) as shown in Appendix Section 8.6.2.8.

8.4 Experiments

We study the performance of *MOESART* on recommender systems and image datasets in Section 8.4.1 and NLP tasks in 8.4.2. We include some ablation studies in Section 8.4.3. Our experiments are run on academic-level compute resources.

8.4.1 Experiments on Recommender Systems and Images

We study the performance of *MOESART* in recommender systems and image datasets. We compare with state-of-the-art k-sparse gates, including Top-k [233], V-MoE [227], S-MoE

[72, 295], Expert Choice [290] and X-MoE [48]. Note that Expert Choice obeys the k-sparsity on average across samples in a minibatch. We also include softmax gate as a baseline.

Datasets. We consider multitask versions of two recommendation system datasets: MovieLens [88] and Books [294]. For MovieLens and Books, we have two tasks: classification task predicts whether user watches/reads a particular movie/book, regression problem predicts user's rating. For image datasets, we consider two multitask datasets: Multi-MNIST [228] and Multi-FashionMNIST. There are two multi-class classification tasks [101]. Full details about each dataset are in Appendix Section 7.7.6.

Experimental setup. Athough our exposition in Section 8.3 was for a single-task setting, the same gate can be used in multi-task learning — multi-task requires multi-gate MoE architecture [176], where each task has a separate trainable gate, but tasks have to select from a common set of experts. Total loss is the convex combination of loss for each task. For recommender systems, we train the network with a convex combination of the task-specific losses: binary cross-entropy (for classification) and mean squared error (for regression) with task weights (TW): $(\alpha, 1-\alpha)$. We separately present results for two different task weight settings. For Multi-MNIST and Multi-FashionMNIST, we train with an equally weighted combination of cross-entropy losses. We used Adam optimizer, and we tuned the key hyperparameters using random grid search. After tuning, we train each model for 50 repetitions (using random initialization) and report the averaged test loss and task-specific metrics along with their standard errors. Full details about the respective MoE architectures and hyperparameter tuning for all gates are given in Appendix Section 7.7.6.

Results. In Table 8.4.1, we report the test loss, task-specific metrics and the number of experts for each input used during training across multiple recommender and image datasets. The results indicate that *MOESART* lead on all datasets with statistical significance, outperforming all state-of-the-art sparse gates e.g., Top-k, V-MoE, Expert Choice and X-MoE. Notably, *MOESART* can achieve 16% reduction in test loss on Multi-MNIST over all sparse gates. Similarly, we can observe that *MOESART* can achieve 15% (relative)

Table 8.4.1: *Test loss, task-specific metrics and number of experts activated per sample (k/s)* while *training for MOESART and existing purely sparse gating methods: (1) Top-k [233], (2) V-MoE [227], (3) S-MoE [72], (4) Expert Choice [290] and (5) X-MoE [48] across various datasets. Bold indicates statistical significance (p-value<0.05) over the best existing sparse gate, using a one-sided unpaired t-test.*

Recommender Systems					
Dataset	Gate	Test Loss ($\times 10^{-2}$) $\downarrow$	Task-1 AUC $\uparrow$	Task-2 MSE $\downarrow$	$k/s\downarrow$
Books (TW=(0.1,0.9))	Softmax	248.96 ± 0.11	55.31 ± 0.08	2.697 ± 0.001	9
	Top-k	246.97 ± 0.11	56.64 ± 0.08	2.675 ± 0.001	4
	V-MoE	253.21 ± 0.15	55.11 ± 0.06	2.744 ± 0.002	4
	S-MoE	249.88 ± 0.13	56.68 ± 0.09	2.707 ± 0.001	4
	Expert Choice	314.54 ± 0.49	56.22 ± 0.10	3.426 ± 0.005	4
	X-MoE	264.16 ± 0.25	56.18 ± 0.10	2.866 ± 0.003	4
	MOESART	$\mathbf{242.47} \pm 0.09$	$\mathbf{64.99} \pm 0.13$	$\mathbf{2.626} \pm 0.001$	4
Books (TW=(0.9,0.1))	Softmax	74.69 ± 0.04	77.48 ± 0.04	2.697 ± 0.002	9
	Top-k	74.63 ± 0.03	77.42 ± 0.02	2.683 ± 0.002	4
	V-MoE	75.96 ± 0.04	76.89 ± 0.05	2.768 ± 0.003	4
	S-MoE	75.30 ± 0.05	77.13 ± 0.05	2.718 ± 0.002	4
	Expert Choice	82.83 ± 0.06	76.75 ± 0.03	3.403 ± 0.009	4
	X-MoE	78.25 ± 0.05	75.48 ± 0.05	2.890 ± 0.003	4
	MOESART	$\mathbf{73.68} \pm 0.02$	$\mathbf{78.03} \pm 0.03$	$\mathbf{2.641} \pm 0.003$	4
MovieLens (TW=(0.1,0.9))	Softmax	73.96 ± 0.02	86.02 ± 0.03	0.7701 ± 0.0002	16
	Top-k	77.72 ± 0.06	86.08 ± 0.06	0.8121 ± 0.0007	2
	V-MoE	75.67 ± 0.04	86.79 ± 0.03	0.7904 ± 0.0005	2
	S-MoE	79.47 ± 0.07	85.36 ± 0.06	0.8303 ± 0.0008	2
	Expert Choice	81.14 ± 0.04	84.65 ± 0.05	0.8477 ± 0.0005	2
	X-MoE	77.59 ± 0.12	86.86 ± 0.08	0.8119 ± 0.0013	2
	MOESART	$\mathbf{73.60} \pm 0.02$	$\mathbf{87.33} \pm 0.03$	$\mathbf{0.7684} \pm 0.0002$	2
MovieLens (TW=(0.9,0.1))	Softmax	41.93 ± 0.02	91.06 ± 0.01	0.7569 ± 0.0002	16
	Top-k	41.90 ± 0.02	91.17 ± 0.01	0.7616 ± 0.0004	2
	V-MoE	42.11 ± 0.03	91.16 ± 0.01	0.7605 ± 0.0006	2
	S-MoE	43.08 ± 0.06	90.86 ± 0.03	0.7917 ± 0.0009	2
	Expert Choice	44.94 ± 0.04	89.88 ± 0.02	0.8216 ± 0.0006	2
	X-MoE	44.66 ± 0.04	89.80 ± 0.02	0.7908 ± 0.0006	2
	MOESART	$\mathbf{40.89} \pm 0.02$	$\mathbf{91.61} \pm 0.01$	$\mathbf{0.7430} \pm 0.0003$	2
Image Tasks					
Dataset	Gate	Test Loss ($\times 10^{-2}$) $\downarrow$	Task-1 Accuracy $\uparrow$	Task-2 Accuracy $\uparrow$	$k/s\downarrow$
Multi-MNIST (TW=(0.5,0.5))	Softmax	7.16 ± 0.05	98.16 ± 0.02	97.57 ± 0.02	8
	Top-k	7.15 ± 0.05	98.12 ± 0.02	97.58 ± 0.02	4
	V-MoE	6.98 ± 0.04	98.16 ± 0.02	97.68 ± 0.02	4
	S-MoE	6.90 ± 0.05	98.18 ± 0.02	97.69 ± 0.02	4
	Expert Choice	8.57 ± 0.06	97.80 ± 0.02	97.22 ± 0.02	4
	X-MoE	7.02 ± 0.06	98.21 ± 0.02	97.63 ± 0.03	4
	MOESART	$\mathbf{5.86} \pm 0.03$	$\mathbf{98.40} \pm 0.02$	$\mathbf{97.92} \pm 0.02$	4
Multi-FMNIST (TW=(0.5,0.5))	Softmax	35.01 ± 0.09	88.10 ± 0.05	87.46 ± 0.05	5
	Top-k	34.96 ± 0.09	88.06 ± 0.05	87.45 ± 0.05	2
	V-MoE	34.43 ± 0.09	88.04 ± 0.05	87.53 ± 0.05	2
	S-MoE	34.68 ± 0.09	88.06 ± 0.04	87.52 ± 0.06	2
	Expert Choice	36.41 ± 0.11	87.50 ± 0.08	87.03 ± 0.06	2
	X-MoE	33.84 ± 0.10	88.05 ± 0.08	87.84 ± 0.08	2
	MOESART	$\mathbf{32.85} \pm 0.11$	$\mathbf{88.56} \pm 0.06$	$\mathbf{88.02} \pm 0.07$	2

improvement in ROC AUC over existing sparse gates on Books dataset, when the models are trained with $(0.1, 0.9)$ task weights combination. We also include comparisons with some differentiable gates in Appendix Section 8.6.2.7. *MOESART* appears to outperform even differentiable gates across many tasks. Additionally, in Appendix Section 8.6.2.8, we show *MOESART* consistently outperforms S-MoE(S).

8.4.2 Experiments on NLP Tasks

In this section, we study the performance of *MOESART* in the context of large language models (LLMs) on natural language processing tasks. In Section 8.4.2.1, we consider a distillation of a pre-trained BERT model into an MoE variant and present results on natural language understanding and question answering tasks. In Section 8.4.2.2, we consider a 1 billion MoE-Transformer for machine translation tasks.

8.4.2.1 Distillation on Language Understanding and Question Answering Tasks

In this section, we consider a setting where a pretrained LLM (non-MoE based) is distilled into an MoE based variant for more efficient inference while preserving or improving the performance. Zuo et al. [298] distilled BERT [54] into its Sparse-MoE based variant termed as MoEBERT. Specifically, the feedforward layers are replaced with MoE layers — this can result in a smaller number of (effective) parameters with per-input gating, thus allowing for more efficient inference. In the MoE layers of MoEBERT model, we consider Top-k gate [233] and our proposed *MOESART* gate during distillation and evaluate the performance on the GLUE [257] and SQuAD benchmarks [218]. We used $k = 2$ for both gates. More details about the setup are summarized in Appendix Section 8.6.3.

Results. We report the performance metrics in Table 8.4.2 for 7 GLUE and 2 SQuAD benchmarks. *MOESART* consistently outperforms Top-k on all GLUE and SQuAD benchmarks. On average, *MOESART* improves over Top-k in prediction performance metrics by 0.5% (up to 1.5%).

Table 8.4.2: *Performance metrics for distillation of BERT into MoEBERT with different gates on the GLUE and SQuAD development sets.*

	GLUE							SQuAD	
	RTE (Acc ↑)	CoLA (Mcc ↑)	MRPC (F1 ↑)	SST-2 (Acc ↑)	QNLI (Acc ↑)	QQP (F1 ↑)	MNLI (m/mm ↑)	v1.1 (F1 ↑)	v2.0 (F1 ↑)
Top-k	68.95	57.86	86.76	92.78	91.63	88.10	85.14	88.38	78.73
MOESART	**70.40**	**58.20**	**88.24**	**93.12**	**91.95**	**88.27**	**85.21**	**88.43**	**79.02**

8.4.2.2 Machine Translation Tasks

In this section, we consider two machine translation tasks: (i) Low-resource: German → English (De → En) from IWSLT'14 with 160K training sentence pairs, 7.3K validation sentence pairs, and 6.8K test sentence pairs. (ii) English → French (En → Fr) from WMT'14 with 35.8M training sentence pairs, 26.9K validation sentence pairs and 3K test sentence pairs. We use Fairseq [202] to implement our models. We follow the pre-processing steps in Ott et al. [202]. All experiments are run on 8 NVIDIA 40GB A100 GPUs.

We consider a 1 billion MoE-Transformer i.e., MoE variant of the Transformer architecture [254]), which is trained from scratch. We use 6 encoder and 6 decoder layers. We use multi-head attention layers with 16 heads. We set embedding dimension as 1024 and the intermediate dimension in FFN as 4096. We replace every alternate FFN with an MoE layer with 16 FFN experts. Dropout is set to 0.1. We set $k = 2$ for both Top-k and *MOESART* gates. We set trimmed lasso penalty as 0.001 for *MOESART*.

Our models are trained such that BLEU scores are maximized on the validation set. We use Adam [136] as the optimizer with $\beta_1 = 0.9, \beta_2 = 0.98$. For learning rate scheduler, we use an inverse sqrt scheduler with 4000 warm-up steps with a peak learning rate of 0.001. We set the batch size to $32K$ tokens, i.e., if we have eight GPUs, each GPU processes $4K$

Table 8.4.3: *Test BLEU scores for 1 billion MoE-Transformer with different gates on machine translation tasks.*

	De → En (IWSLT'14)	En → Fr (WMT'14)
Top-k	30.87	42.44
MOESART	**32.96**	**42.65**

and accumulate gradients for two steps. We trained on En-De for 100 epochs ($\sim 15,000$ steps) and on En-Fr for 4 epochs ($\sim 100,000$ steps). For inference, we use a beam size 5 for En-De and 10 for En-Fr, with a length penalty of 1.0. For final evaluation, we evaluate the model in terms of BLEU score on test set. Given the model sizes and constraints on compute resources, we did not perform any hyperparameter tuning on these tasks.

Results We report the performance numbers for the two gating approaches in Table 8.4.3. We can observe *MOESART* improves over Top-k by 2.1 BLEU points on De $\rightarrow$ En and performs comparably on En $\rightarrow$ Fr.

8.4.3 Ablation Studies for *MOESART*

We perform multiple ablation studies to show different elements of *MOESART*: (i) Effect of sampling strategies during training on out-of-sample generalization on SVHN. (ii) Spatial structure of learnt embeddings on MovieLens. Additional ablation studies studying (a-b) bias and performance of different adjustment strategies, (c) effect of varying k, (d) effect of trimmed lasso regularization on performance, and (e) performance under additional load balancing requirements, and (f) sensitivity of *MOESART* to hyperparameters, are included in Appendix Section 8.6.2.

Effect of sampling strategy. We first study the effect of different sampling strategies on performance of Sparse MoE. We consider two options: (a) sampling with replacement (b) sampling without replacement. For this ablation study, we consider *MOESART* with $k = 2$ on SVHN dataset. We show the evolution of out-of-sample loss during the course of training in Figure 8.4.1. We can observe that sampling without replacement appears to be more stable and achieves better out-of-sample loss. We hypothesize that the sub-optimal performance of sampling with replacement can be attributed to two reasons: (a) The variance is smaller for sampling without replacement, (b) For sampling with replacement, with $k = 2$, if the same expert gets sampled twice for an input, this example is unable to contribute to the gradient update of the gate parameters.

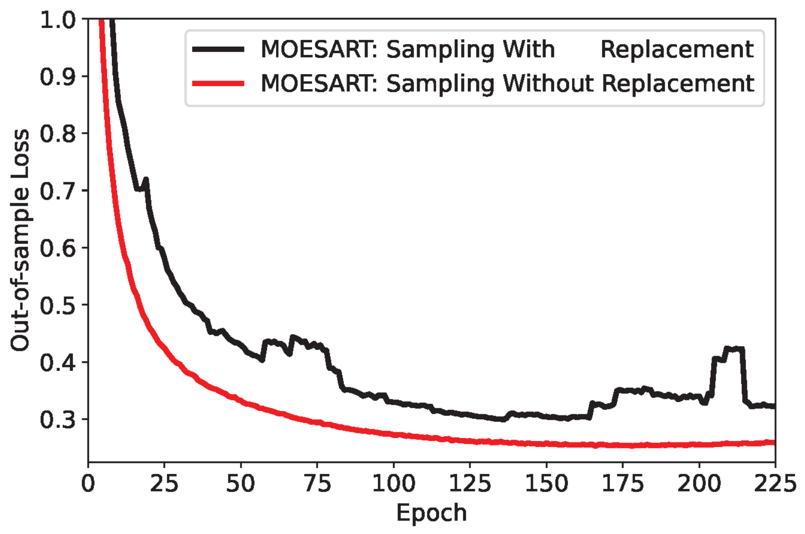
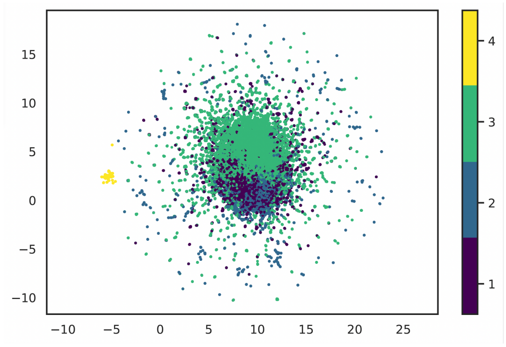
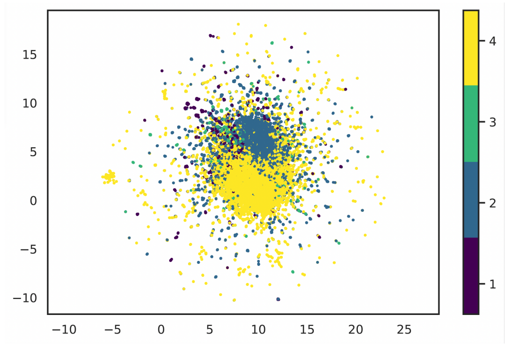


Figure 8.4.1: *Out-of-sample loss for MOESART for different sampling strategies on SVHN.*

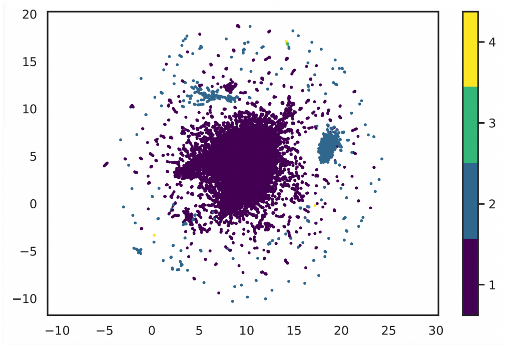
Spatial structure of learnt embeddings. We consider MovieLens dataset and the same multi-task MoE-based architecture as detailed in Section 8.6.2.9 — however, we use 4 experts for this exercise. The architecture has user and movie learnable embedding layers, which are concatenated and fed into an MoE layer with 4 experts, followed by a task-specific head for classification task and regression task. We set $k = 2$ for Top-k and *MOESART* . We optimize with Adam with 5×10^{-5} learning rate with a 512 batch size. We visualize the embeddings learnt by Top-k and *MOESART* gates in Figure 8.4.2. We use Uniform Manifold Approximation and Projection (UMAP) [184] to project the concatenated embeddings of each input to a two-dimensional space. Each data point represents an input to be routed. Each color stands for the top expert that each input is assigned to. *MOESART* seems to provide better specialization of experts, where the distinct distribution handled by each expert is much clearer (8.4.2c,8.4.2d) in comparison to that for Top-k gate (8.4.2a,8.4.2b). The learnt embeddings are more disentangled for *MOESART* in comparison to Top-k.



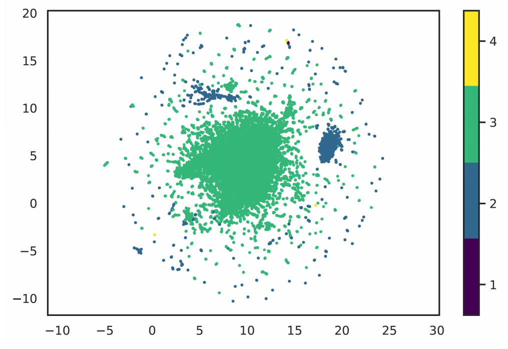
(a) *Top-k (Task-1)*



(b) *Top-k (Task-2)*



(c) *MOESART (Task-1)*



(d) *MOESART (Task-2)*

Figure 8.4.2: *UMAP projection of user/movie embeddings learnt by the different gates on (multi-task) MovieLens. Each data point denotes an embedding input to be routed. Color denotes expert index. MOESART seems to provide better specialization of experts, where the distribution handled by each expert is much clearer (8.4.2c,8.4.2d) in comparison to that for Top-k gate (8.4.2a,8.4.2b).*

8.5 Conclusion

We proposed a sampling-based gating mechanism with *MOESART*. Our approach aims to learn a sparse approximation of the softmax based classical MoE. We achieve this through sampling and novel expert reweighting strategies. The sparse approximation learnt by our approach appears to be substantially better than that learnt by Top-k gate and its variants. *MOESART* allows conditional training as it is k-sparse during training similar to Top-k style gating strategies. We performed large-scale experiments on 14 datasets from various domains. On standard vision and recommender systems, *MOESART* achieves up to 16% (relative) smaller out-of-sample loss and up to 15% (relative) improvement in ROC AUC over state-of-the-art k-sparse gates, e.g., Top-k, V-MoE, Expert Choice and X-MoE. Moreover, *MOESART* gate can consistently outperform Top-k gate in natural language processing tasks.

8.6 Appendix

8.6.1 Proof for $\tilde{g}_i$ in (iii) being an unbiased estimator of g_i

In this section, we study the biasness of adjustment strategy ((iii) in the main paper), as given by

$$\tilde{o}_i := \begin{cases} o_i + \log(r_i) - \log(kg_i) & i \in s, \\ -\infty & i \notin s, \end{cases} \tag{8.6.1}$$

Recall that $\tilde{g}_i = \exp(\tilde{o}_i)/\sum_{l\in[m]}\exp(\tilde{o}_l)$. We present the following theorem:

Theorem 8.6.1. *For each $i \in [m]$, if sampling (with replacement) is performed from the softmax probability, $g_i \propto exp(o_i)$, then $\tilde{g}_i$ (corresponding to $\tilde{o}_i$ defined in (8.6.1) is an unbiased estimator of g_i, i.e., $\mathbb{E}[\tilde{g}_i] = g_i$.*

Before, we can show the proof for Theorem 8.6.1, we present the following result:

Lemma 8.6.1. *For $\tilde{o}_i$ defined as in (8.6.1), the following equality holds:*

$$\sum_{j\in[m]} exp(\tilde{o}_j) = \sum_{l\in[m]} exp(o_l). \tag{8.6.2}$$

Proof for Lemma 8.6.1:

$$\sum_{j\in[m]} \exp(\tilde{o}_j) = \sum_{j\in s} \exp(\tilde{o}_j) + \sum_{j\notin s} \exp(\tilde{o}_j) = \sum_{j\in s} \exp(\tilde{o}_j) \tag{8.6.3}$$

$$= \sum_{j\in s} \exp(o_j - \log(kg_j) + \log(r_j)) = \sum_{j\in s} \frac{r_j \exp(o_j)}{kg_j} \tag{8.6.4}$$

$$= \sum_{j\in s} \frac{r_j \exp(o_j)}{k} \frac{\sum_{l=1}^{m} \exp(o_l)}{\exp(o_j)} \tag{8.6.5}$$

$$= \frac{1}{k}\left(\sum_{j\in s} r_j\right)\left(\sum_{l=1}^{m} \exp(o_l)\right) = \frac{1}{k}(k)\left(\sum_{l=1}^{m} \exp(o_l)\right) \tag{8.6.6}$$

$$= \sum_{l=1}^{m} \exp(o_l). \quad \blacksquare \tag{8.6.7}$$

Proof for Theorem 8.6.1:

$$\mathbb{E}_r\left[\tilde{g}_j\right] = \mathbb{E}_r\left[\frac{\exp(\tilde{o}_i)}{\sum_{l\in[m]} \exp(\tilde{o}_l)}\right] \tag{8.6.8}$$

$$= \mathbb{E}_r\left[\frac{\exp(\tilde{o}_i)}{\sum_{l=1}^{m} \exp(o_l)}\right] \quad \text{using Lemma 8.6.1} \tag{8.6.9}$$

$$= \mathbb{E}_r\left[\frac{\exp(o_i - \log(kg_i) + \log(r_i))}{\sum_{l=1}^{m} \exp(o_l)}\right] \tag{8.6.10}$$

$$= \frac{1}{k}\mathbb{E}_r\left[\frac{r_i}{g_i}\frac{\exp(o_i)}{\sum_{l=1}^{m} \exp(o_l)}\right] \tag{8.6.11}$$

$$= \frac{1}{k}\mathbb{E}_r[r_i] = \frac{1}{k}(kg_i) = g_i. \quad \blacksquare \tag{8.6.12}$$

8.6.2 Additional Ablation Studies

In this section, we show four ablation studies:

(a) Bias of different adjustment strategies in (i)-(ii) and proposed strategy in (8.3.3).

(b) Performance comparison of *MOESART* with different adjustment strategies in (i)-(ii) and (8.3.3) for training of Sparse-MoE on Image datasets.

(c) Effect of varying k.

(d) Effect of trimmed lasso regularization on performance of *MOESART* on Recommender Systems.

(e) Performance of *MOESART* under additional load balancing requirements.

(f) Sensitivity of *MOESART* to hyperparameter tuning.

(g) Comparison with differentiable gates.

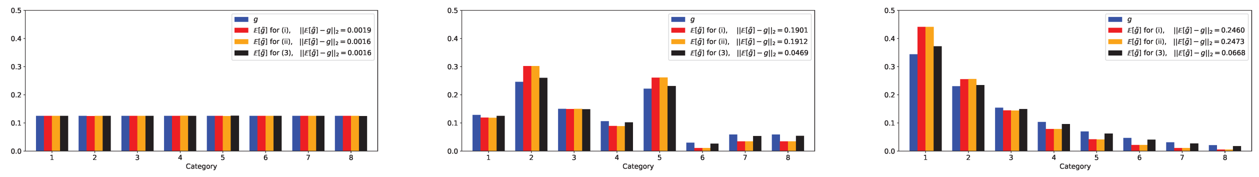
(h) Comparison with S-MoE(S) gate.

8.6.2.1 Bias of different adjustment strategies in (i)-(ii) and proposed strategy in (8.3.3)

In this ablation study, we empirically study the bias of different adjustment strategies in (i)-(ii) and the proposed strategy in (8.3.3). For this study, we consider different choices for g: uniform, random, decaying. We measure the bias with the metric $\|\mathbb{E}[\tilde{g}] - g\|_2$. To be precise, this metric $\|\mathbb{E}[\tilde{g}] - g\|_2$ is defined as:

$$\|(\mathbb{E}[\tilde{g}_1], \cdots, \mathbb{E}[\tilde{g}_m]) - (g_1, \cdots, g_m)\|_2 \tag{8.6.13}$$

We show the metric in Fig. 8.6.1 for different distribution shapes for g. For uniform setting, we observe the bias to be very similar across different adjustment strategies. However, for sufficiently non-uniform distributions, we can observe that there can be a significant gap in the bias for proposed strategy in (8.3.3) and the adjustment strategies in (i)-(ii). In Sparse MoE, the distribution $g(x)$ can be very different across different inputs, hence our proposed strategy is expected to have a lower bias overall across samples. A smaller bias can lead to improved learning. We observe this to be the case in Sparse MoE, which we show in the next ablation study.



(a) *Uniform g* (b) *Random g* (c) *Exponentially Decaying g*

Figure 8.6.1: *Comparison of bias of different adjustment strategies (i)-(iii) and the proposed strategy in (8.3.3) for different g.*

8.6.2.2 Performance comparison of our proposed adjustment in (8.3.3) versus (i)-(ii) on Image datasets.

In this ablation study, we compare the performance of *MOESART* under different adjustment strategies for computing $\tilde{g}$. In particular, we compare the proposed strategy in (8.3.3) without replacement against the strategies in (i)-(ii) without replacement. Note that (i) and (ii) are equivalent when sampling without replacement.

We show results for different adjustment strategies in Table 8.6.1 on multi-task image datasets. We also include Top-k for comparison. We can see our proposed strategy in (8.3.3) can be significantly better than the strategies in (i)-(ii). We can also observe that both sampling-based strategies (i-ii) and (8.3.3) significantly outperform Top-k gating. This potentially highlights that Top-k greedy gating has a large selection bias, which is reduced by sampling based approaches.

Table 8.6.1: *Comparison of test loss and task-specific metrics for MOESART with the proposed adjustment strategy in (8.3.3) against other strategies in (i)-(ii). We also include Top-k for reference.*

Image Tasks					
Dataset	Model	Test Loss ($\times 10^{-2}$) $\downarrow$	Task-1 Accuracy $\uparrow$	Task-2 Accuracy $\uparrow$	Training k/s $\downarrow$
Multi-MNIST (TW=(0.5,0.5))	Top-k	7.15 ± 0.05	98.12 ± 0.02	97.58 ± 0.02	4
	MOESART with (i),(ii)	6.15 ± 0.05	98.33 ± 0.02	$\mathbf{97.92} \pm 0.02$	4
	MOESART with (8.3.3)	$\mathbf{5.86} \pm 0.03$	$\mathbf{98.40} \pm 0.02$	$\mathbf{97.92} \pm 0.02$	4
Multi-FMNIST (TW=(0.5,0.5))	Top-k	34.96 ± 0.09	88.06 ± 0.05	87.45 ± 0.05	2
	MOESART with (i),(ii)	34.15 ± 0.09	88.19 ± 0.05	87.71 ± 0.05	2
	MOESART with (8.3.3)	$\mathbf{32.85} \pm 0.11$	$\mathbf{88.56} \pm 0.06$	$\mathbf{88.02} \pm 0.07$	2

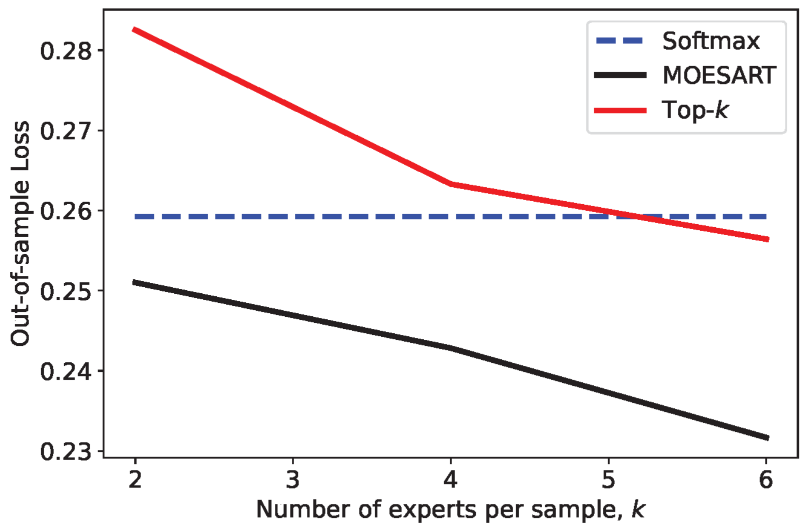


Figure 8.6.2: *Out-of-sample loss achieved by MOESART and Top-k for different k on SVHN.*

8.6.2.3 Effect of varying k

Here, we study the effect of k on MoE performance. Note that number of experts is fixed to 8. We train Sparse MoE models for different values of $k = \{2, 4, 6\}$ and visualize the generalization performance in Figure 8.6.2. For comparison, we visualize the results for both Top-k and *MOESART*. We also show the performance of softmax gate. We observe that both sparse gates improve in performance as k is increased. Notably, we consistently see a significant gap in performance between Top-k and *MOESART* for each k setting.

8.6.2.4 Effect of Trimmed Lasso regularization on performance on Recommender Systems

In this section, we perform ablation study for the trimmed lasso regularization on recommender systems datasets i.e., MovieLens and Books. We had performed a large set of 500 tuning trials, which tuned over multiple hyperparameters including the trimmed lasso regularization λ in the set $\{0, 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0\}$. We identify the best tuning trial for both cases $\lambda = 0$ and $\lambda > 0$ based on the validation set and report the test performance for the two cases in Table 8.6.2. We can observe that for both datasets, trimmed lasso regularization can provide some performance gain over the case without trimmed lasso regularization.

Table 8.6.2: *Comparison of test loss for best trial for MOESART without and with Trimmed Lasso Regularization on Recommender system datasets.*

Dataset	Model	Test Loss ($\times 10^{-2}$) $\downarrow$
MovieLens	*MOESART* without Trimmed Lasso	73.85
	MOESART with Trimmed Lasso	**73.40**
Books	*MOESART* without Trimmed Lasso	246.50
	MOESART with Trimmed Lasso	**243.02**

Use of trimmed lasso for other strategies e.g., Top-k Top-k style gates are sparse by construction and the use of trimmed lasso does not seem appropriate. For example, when $g(x)$ is parameterized as softmax($Topk(Ax + b, k)$) for Top-k, the trimmed lasso operator sorts the elements of g in descending order such that the trimmed lasso penalty penalizes the smallest $m - k$ elements of g per-input x. Given that the smallest $m - k$ entries of the above $g(x)$ are 0, the trimmed lasso penalty has no effect. However, we may consider the following formulation when adapting Trimmed Lasso regularization to Top-k:

$$(b) \min_{\{f_i\},g} \hat{\mathbb{E}} \left[\ell \left(y, \sum_{i \in [m]} f_i(x) g(x)_i \right) \right] + \lambda \sum_{j>k} T(\text{softmax}(Ax + b))_j, \tag{8.6.14}$$

where $g(x)$ is parameterized as softmax($Topk(Ax + b, k)$). We ran experiments to test this formulation on Books dataset. We did not observe any performance gain by imposing the trimmed lasso regularization when considering Top-k gate. We show the results below: Interestingly, we observe a decrease in performance. We hypothesize that in this formulation of

Table 8.6.3: *Comparison of test loss for best trial for Top-k without/with Trimmed Lasso Regularization on Books dataset.*

Dataset	Model	Test Loss ($\times 10^{-2}$) $\downarrow$
Books	Top-k without Trimmed Lasso	**247.00**
	Top-k with Trimmed Lasso	250.48
	MOESART without Trimmed Lasso	246.50
	MOESART with Trimmed Lasso	**243.02**

Top-k with trimmed lasso, the impact of trimmed lasso can further accelerate the probability mass into the top-k elements that are being already selected by the router — this may reduce the exploration capability of Top-k, affecting performance. These experiments further confirm the novelty and usefulness of imposing Trimmed Lasso regularization for our sampling-based approach.

8.6.2.5 Performance under Load balancing

Load balancing is also another important consideration for efficiency in Sparse MoE models. Load balancing requires similar number of examples to be routed to each expert. Traditionally, an auxiliary loss is added explicity to achieve load balancing. Shazeer et al. [233] and Fedus et al. [72] proposed two different auxiliary losses to achieve load balancing. Many follow-up works on Top-k based gating [48, 271, 295] impose one of these auxiliary losses. We add an auxiliary loss as the one imposed by Fedus et al. [72] to Top-k gate and *MOESART* gate and compare the performance of these methods when the load balancing regularization is designed to achieve 99% load balancing. We also compare against the Expert Choice by Zhou et al. [290], which is designed to achieve 100% load balancing across experts. We again consider (multi-task) MovieLens dataset. *MOESART* with load balancing achieves the best performance as shown in Table 8.6.4.

Table 8.6.4: *Test loss with load balancing on MovieLens.*

Model	Test Loss ($\times 10^{-2}$) $\downarrow$
Top-k	77.72 ± 0.03
Expert Choice	81.14 ± 0.04
MOESART	$\mathbf{74.04} \pm 0.03$

8.6.2.6 Sensitivity of *MOESART* to hyperparameter tuning

Here, we study the sensitivity of *MOESART* to hyperparameter tuning and show that gate can be beneficial in terms of hyperparameter tuning over Top-k style gates. We perform a large set of tuning trials and perform a bootstrapping procedure to see whether *MOESART* helps in reducing the hyperparameter tuning overload.

We visualize the impact of number of trials on performance in Figure 8.6.3. *MOESART* can achieve the same level of performance as Top-k gate with much lesser number of hyperparameter trials. This indicates that *MOESART* is not too heavily dependent on a very restricted set of hyperparameter values. We visualize this for various datasets in Fig. 8.6.3. We see tuning reduction by a factor of 100× for *MOESART* over Top-k. Alternatively, even a small number of trials e.g., 2-10 can show a significant gain over Top-k gate.

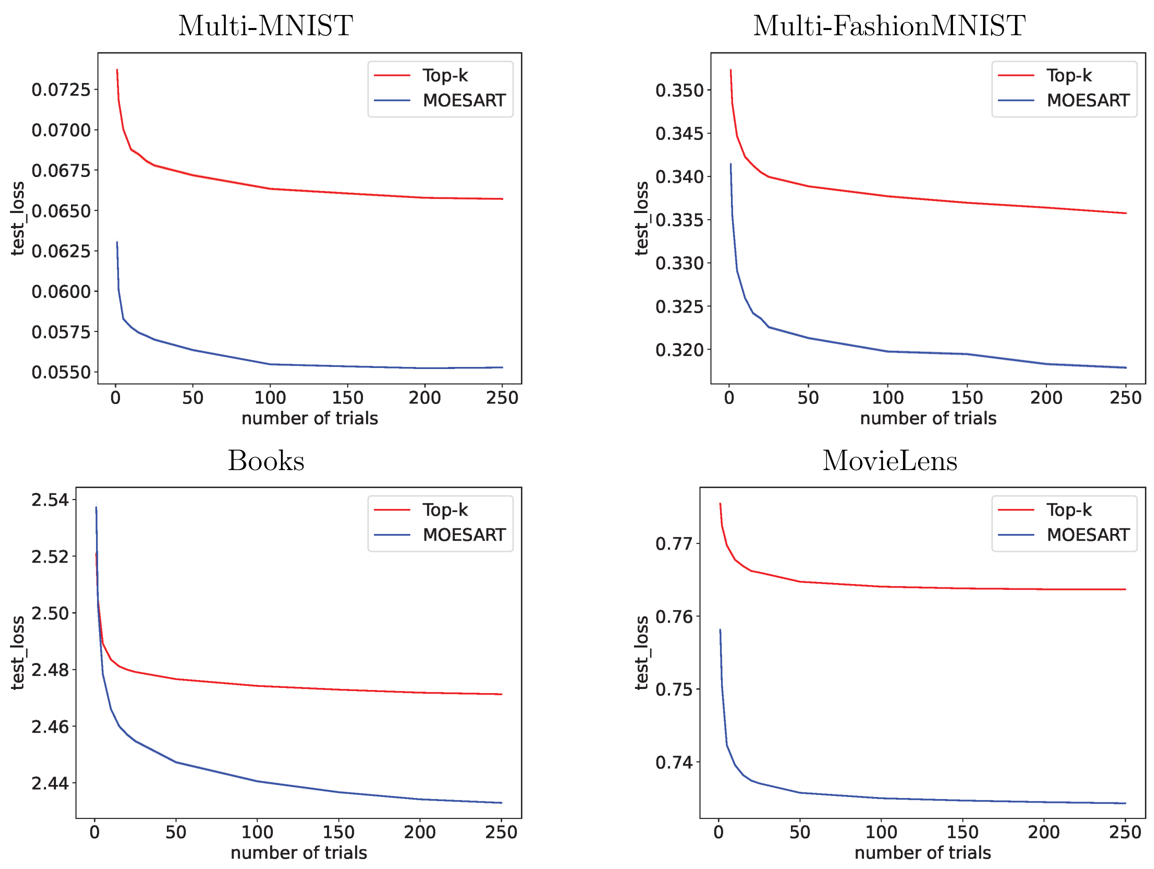


Figure 8.6.3: *Sensitivity of MOESART to hyperparameter tuning. MOESART* can achieve the same level of performance as Top-k with significantly lesser number of hyperparameter trials. We see tuning reduction by 100× for *MOESART* over Top-k.

8.6.2.7 Comparison of *MOESART* with Differentiable Gates

In this section, we compare our gating approach with some state-of-the-art differentiable gates. In particular, we compare with two differentiable gates: (i) DSelect-k [101], (ii) COMET [117].

Table 8.6.5: *Comparison of test loss and task-specific metrics for MOESART against differentiable gating methods: (1) DSelect-k [101], (2) COMET [117], across various datasets. Note that these differentiable gates can only support conditional training partially with customized implementations, hence they are more expensive during training.*

Recommender Systems				
Dataset	Model	Test Loss ($\times 10^{-2}$) $\downarrow$	Task-1 AUC $\uparrow$	Task-2 MSE $\downarrow$
Books (TW=(0.1,0.9))	Softmax (Dense)	248.96 ± 0.11	55.31 ± 0.08	2.697 ± 0.001
	DSelect-k	241.64 ± 0.14	59.05 ± 0.14	2.615 ± 0.002
	COMET	**241.13** ± 0.11	**66.13** ± 0.09	**2.612** ± 0.001
	MOESART	242.47 ± 0.09	64.99 ± 0.13	2.626 ± 0.001
Books (TW=(0.9,0.1))	Softmax (Dense)	74.69 ± 0.04	77.48 ± 0.04	2.697 ± 0.002
	DSelect-k	74.22 ± 0.03	77.61 ± 0.02	**2.632** ± 0.002
	COMET	74.29 ± 0.03	77.30 ± 0.02	2.636 ± 0.002
	MOESART	**73.68** ± 0.02	**78.03** ± 0.03	2.641 ± 0.003
MovieLens (TW=(0.1,0.9))	Softmax (Dense)	73.96 ± 0.02	86.02 ± 0.03	0.7701 ± 0.0002
	DSelect-k	73.81 ± 0.03	**87.79** ± 0.03	0.7715 ± 0.0003
	COMET	74.19 ± 0.03	87.67 ± 0.07	0.7756 ± 0.0004
	MOESART	**73.60** ± 0.02	87.33 ± 0.03	**0.7684** ± 0.0002
MovieLens (TW=(0.9,0.1))	Softmax (Dense)	41.93 ± 0.02	91.06 ± 0.01	0.7569 ± 0.0002
	DSelect-k	41.40 ± 0.03	91.44 ± 0.01	0.7582 ± 0.0005
	COMET	41.25 ± 0.04	91.45 ± 0.02	0.7563 ± 0.0008
	MOESART	**40.89** ± 0.02	**91.61** ± 0.01	**0.7430** ± 0.0003
Image Tasks				
Dataset	Model	Test Loss ($\times 10^{-2}$) $\downarrow$	Task-1 Accuracy $\uparrow$	Task-2 Accuracy $\uparrow$
Multi-MNIST (TW=(0.5,0.5))	Softmax (Dense)	7.16 ± 0.05	98.16 ± 0.02	97.57 ± 0.02
	DSelect-k	6.93 ± 0.06	98.14 ± 0.02	97.68 ± 0.03
	COMET	6.83 ± 0.06	98.21 ± 0.02	97.63 ± 0.03
	MOESART	**5.86** ± 0.03	**98.40** ± 0.02	**97.92** ± 0.02
Multi-FMNIST (TW=(0.5,0.5))	Softmax (Dense)	35.01 ± 0.09	88.10 ± 0.05	87.46 ± 0.05
	DSelect-k	36.88 ± 0.21	87.37 ± 0.07	86.61 ± 0.09
	COMET	34.88 ± 0.13	87.97 ± 0.06	87.42 ± 0.06
	MOESART	**32.85** ± 0.11	**88.56** ± 0.06	**88.02** ± 0.07

We would like to remind the reader that although these gates can allow conditional inference, they are less appealing in terms of conditional training. These gates are dense-to-sparse gates, and hence can only partially allow for conditional training (during a later stage of training with customized implementations). Hence, these gates are more expensive as during the dense phase of training they require computing gradients with respect to a larger set of experts. In contrast, our proposed gating approach *MOESART* completely allows conditional training and conditional inference.

We tuned these differentiable gates for their respective hyperparameters with random search over 500 tuning trials and report the averages across 50 runs for their best hyperparameters. We compare across various datasets and report the performance metrics in Table

8.6.5. *MOESART* appears to be quite competitive with differentiable gates. Surprisingly, *MOESART* can even outperform differentiable gates across many tasks. We hypothesize that this maybe due to the choice of parameterization. These gates rely on a particular activation function (Smooth-Step function [100]) to achieve binary state for sparse inference. The binary gates snap into place across samples reaching a permanent state of 0 or 1 as training progresses. This is understandably a desired property of Smooth-Step activation for conditional computation, however it means that the model cannot update decisions regarding the choice of expert once this permanent state is reached. In contrast, our sampling-based approach can allow for exploration throughout training, which can be beneficial.

8.6.2.8 Comparison of *MOESART* with S-MoE(S)

Fedus et al. [72] considered a sampling version of S-MoE in their Appendix for one single experiment. In this experiment, they showed that the sampling variant of S-MoE can degrade performance compared to Top-k based S-MoE. We have run new experiments to compare against this variant of S-MoE (denoted by S-MoE(S)) and we provide the results in Table 8.6.6. We observe a mixed trend, where S-MoE(S) sometimes outperforms S-MoE. In comparison, *MOESART* consistently outperforms S-MoE and S-MoE(S)

Next we discuss differences in S-MoE(S) and *MOESART*. Although both approaches i.e., *MOESART* and S-MoE(S) perform sampling, there are key differences in parameterization, weighting and regularization:

1. S-MoE(S) considers the following parameterization: $g(x) = g(x) \odot \mathbf{1}_S$ where $\mathbf{1}_S$ denotes a vector such that only k sampled indices are non-zero that are in the set S. This does not obey the simplex constraint. In contrast, *MOESART* is based on weight adjustment outlined in (8.3.3) and these adjusted weights always lie on the sparse simplex. The bias of S-MoE(S) is higher than that for *MOESART*, which can degrade performance of S-MoE(S).

2. S-MoE(S) is stochastic during training as well as inference. In contrast, *MOESART* uses

Table 8.6.6: *Comparison of test loss and task-specific metrics for MOESART against sampling-base variant S-MoE(S) across various datasets.*

Recommender Systems				
Dataset	Model	Test Loss ($\times 10^{-2}$) $\downarrow$	Task-1 AUC $\uparrow$	Task-2 MSE $\downarrow$
Books (TW=(0.1,0.9))	S-MoE	249.88 ± 0.13	56.68 ± 0.09	2.707 ± 0.001
	S-MoE(S)	251.69 ± 0.17	53.86 ± 0.07	2.727 ± 0.002
	MOESART	$\mathbf{242.47} \pm 0.09$	$\mathbf{64.99} \pm 0.13$	$\mathbf{2.626} \pm 0.001$
Books (TW=(0.9,0.1))	S-MoE	75.30 ± 0.05	77.13 ± 0.05	2.718 ± 0.002
	S-MoE(S)	75.88 ± 0.06	77.63 ± 0.06	2.824 ± 0.003
	MOESART	$\mathbf{73.68} \pm 0.02$	$\mathbf{78.03} \pm 0.03$	2.641 ± 0.003
MovieLens (TW=(0.1,0.9))	S-MoE	79.47 ± 0.07	85.36 ± 0.06	0.8303 ± 0.0008
	S-MoE(S)	73.89 ± 0.02	85.89 ± 0.02	0.7691 ± 0.0002
	MOESART	$\mathbf{73.60} \pm 0.02$	87.33 ± 0.03	$\mathbf{0.7684} \pm 0.0002$
MovieLens (TW=(0.9,0.1))	S-MoE	43.08 ± 0.06	90.86 ± 0.03	0.7917 ± 0.0009
	S-MoE(S)	41.17 ± 0.02	91.58 ± 0.01	0.7543 ± 0.0004
	MOESART	$\mathbf{40.89} \pm 0.02$	$\mathbf{91.61} \pm 0.01$	$\mathbf{0.7430} \pm 0.0003$
Image Tasks				
Dataset	Model	Test Loss ($\times 10^{-2}$) $\downarrow$	Task-1 Accuracy $\uparrow$	Task-2 Accuracy $\uparrow$
Multi-MNIST (TW=(0.5,0.5))	S-MoE	6.90 ± 0.05	98.18 ± 0.02	97.69 ± 0.02
	S-MoE(S)	7.19 ± 0.09	98.11 ± 0.04	97.45 ± 0.05
	MOESART	$\mathbf{5.86} \pm 0.03$	$\mathbf{98.40} \pm 0.02$	$\mathbf{97.92} \pm 0.02$
Multi-FMNIST (TW=(0.5,0.5))	S-MoE	34.68 ± 0.09	88.06 ± 0.04	87.52 ± 0.06
	S-MoE(S)	34.39 ± 0.10	88.06 ± 0.08	87.82 ± 0.08
	MOESART	$\mathbf{32.85} \pm 0.11$	$\mathbf{88.56} \pm 0.06$	$\mathbf{88.02} \pm 0.07$

top-k indices at inference.

3. There is no trimmed lasso regularization in S-MoE(S). Trimmed lasso regularization can boost performance as we show in ablation study in Table 8.6.2 in Appendix Section 8.6.2.4.

8.6.2.9 Architectures

SVHN. We use an architecture with an MoE-layer followed by a stack of 3 dense layers: the first two have 50 ReLU-activated units and the third has 10 units followed by a softmax. The MoE layer consists of 8 experts, each of which is a CNN that is composed (in order) of: (i) convolutional layer 1 (kernel size = 5, #filters = 10, ReLU-activated) followed by max pooling, (ii) convolutional layer 2 (kernel size=5, #filters = 20, ReLU-activated) followed by max pooling, and (iii) a sequence of 2 ReLU-activated dense layers with 50 units each.

8.6.2.10 Hyperparameters and Tuning

We performed 500 tuning trials for each gate with a random search over the hyperparameter space described below (for each dataset). For each gate, we tune the optimization and gate-specific hyperparameters and use the validation loss as the tuning metric. After tuning, we train each model for 50 repetitions (using random initializations) and report the averaged results along with the standard errors in Table 8.4.1.

SVHN.

- Learning Rates: 1×10^{-4} for Adam.
- Batch-size: 512.
- Epochs: 250 with early stopping (patience=50) based on validation set.
- Trimmed Lasso, λ: 0.0 for *MOESART*.
- m (number of experts): 8.
- k: 2 for all sparse (trainable) gates.
- Number of tuning trials per gate: 1

8.6.3 Additional Details for Section 8.4.2

8.6.3.1 Tuning Procedure for MoEBERT

Following Zuo et al. [298], we followed the 3-step process as outlined in the MoEBERT codebase[1]:

- Finetune BERT on downstream task. We used finetuned BERT from HuggingFace on each downstream task.
- Compute importance weights in FFN layers to construct an MoEBERT model, where FFN layers are replaced with MoE layers with the weight assignment strategy in [298].

[1]https://github.com/SimiaoZuo/MoEBERT

- Distill BERT into MoEBERT on the downstream task with a combination of cross-entropy loss and layer-wise discrepancy loss. For MoEBERT with Top-k and *MOESART*, we performed a tuning procedure and picked the best results based on development datasets. We performed a grid search over the following sets of hyperparameters:
  - Learning Rate: We used same learning rates as the optimal ones reported for each dataset in Table 7 of Zuo et al. [298].
  - Batch size: We used same batch sizes as the optimal ones reported for each dataset in Table 7 of Zuo et al. [298].
  - Weight Decay: Discrete uniform over the set $\{0, 0.01, 0.1\}$
  - Distillation Regularization ($\lambda_{distill}$ in [298]): Discrete uniform over the set $\{1, 2, 3, 4, 5\}$.
  - k: 2
  - τ (for *MOESART*): 1.0.
  - λ (for Trimmed Lasso regularization for *MOESART*): Discrete uniform over the set $\{0.0001, 0.001, 0.01, 0.1, 1.0, 10.0\}$.
  - Epochs: 10. Best model was recovered at best checkpoint based on development set.

Chapter 9

Pruning Large Vision and Language Models

In a departure from the previous chapters on ensemble learning, I also advised two junior PhD students (Xiang Meng, Gabriel I. Afriat) on post-training pruning of large vision and language models for improving inference efficiency of these models. These settings bring up interesting structural constraints. The next two sections provide a brief summary of these works.

9.1 Structured Pruning in Vision and Language Models with Combinatorial Optimization

Structured pruning [151, 261] reduces model size by removing entire subcomponents, e.g., channels in convolutional layers, neurons in dense layers and heads in multi-head attention. Structured pruning offers a practical solution to improve inference latency on standard hardware in contrast to unstructured pruning [87, 89, 152], which requires specialized hardware and software.

Although there are direct computational benefits from structured pruning, there is a significant challenge. Models can be highly sensitive to structured pruning (with a large drop in utility) and most existing methods [107, 159, 175, 189, 280] rely on gradual pruning or iterative retraining, where the model is finetuned on the original loss after every pruning stage to allow the model to recover accuracy. Such finetuning may not be desirable for large datasets and models under resource constraints. For example, finetuning an LLM on a standard GPU (A100) may not be possible beyond a few billion parameters [178]. In this context, recent works [144, 146] have focused on the challenging task of post-training one-shot structured pruning, in which a model must be compressed (without retraining) based on a small amount of calibration data, without significant loss in accuracy.

Despite impressive advances, state-of-the-art methods appear to face challenges in terms of increased computation time, memory usage, and balancing utility with structured sparsity. To address these challenges, we propose a novel optimization-based framework for one-shot structured pruning, which is highly scalable and can achieve good utility-sparsity tradeoffs. We employ a layer-wise reconstruction objective and a careful structure-aware reformulation of the optimization formulation to enable more scalability. To obtain good solutions to the problem, we propose a local combinatorial optimization algorithm, which leverages problem structure to perform efficient local search. By integrating these algorithmic components, we demonstrate that our method is capable of handling large vision and language models, up to 30 billion parameters using a single 32GB V100 GPU — an improvement over prior approaches that can handle up to 340 million parameters. The full paper is given in Meng et al. [187].

9.2 Multi-Objective Pruning with Non-Uniform Layer-wise Sparsity Allocation in Vision Transformers

Contemporary vision models, have huge parameter counts [59, 106], incurring significant computational costs during the inference phase. Pruning is a common strategy for compressing large neural networks, including vision transformers. The aim of pruning is to remove weights (set them to zero) in the network.

Many pruning techniques have been proposed for vision models [110]. These versions aim to maintain high accuracy (close to that of the dense model) while achieving maximum compression. However, despite notable advancements, the compression process often proves to be very time-consuming and costly: due to significant drops in model accuracy after compression, expensive retraining becomes necessary. In this context, post-training one-shot pruning can be of great interest [163]. Here, the goal is to prune a fully trained, dense model using a limited calibration dataset in *one* pruning step. In this paper we focus on one-shot pruning approaches: as they do not require retraining, they are computationally attractive and are particularly relevant for real-world applications.

Popular one-shot methods decide on what to prune by using a single objective criterion. There are two common objective functions that are used. One line of work, introduced by [90, 152] through the Optimal Brain Surgeon (OBS) framework, uses a local quadratic approximation of the original training loss. Various methods [15, 75, 145, 234, 281] have demonstrated the efficacy of OBS in achieving state-of-the-art one-shot pruning, by strategically selecting weights for removal from a trained neural network. The second line of work, based on a layer-wise OBS strategy, uses a data fidelity error based on the reconstruction of layer outputs. Here pruning task is broken into layer-wise subproblems, and numerical methods are used to compute a set of sparse weights for each layer [58, 76]. The former i.e., OBS objective, uses global information from the pre-trained neural network for the pruning problem. However, for scalability, block-level approximations are used for pruning modern architectures. On the other hand, the second approach i.e., layer-wise reconstruction loss uses more localized information, preventing large deviations in the embedding spaces; this can prevent catastrophic forgetting with the full OBS objective when using a small calibration dataset[1]. In our experiments, we observed that different objective functions can

[1]A large calibration set would lead to higher computation costs, which are undesirable for an efficient post-training one-shot pruning algorithm.

result in different accuracy-sparsity tradeoffs for different models: suggesting that usefulness of both these objective functions for pruning purposes. Motivated by this observation, we consider a novel multi-objective optimization problem, where both objectives are considered simultaneously. Interestingly, by simultnaeously considering both these objectives (as a part of our multi-objective optimization formulation), we observe that that we are able to obtain significantly better accuracy-sparsity tradeoffs—one that outperforms earlier approaches that choose an individual objective function.

Additionally, it has been observed previously [74, 76, 108] that allocating sparsity budgets non-uniformly across layers can substantially improve the accuracy-sparsity tradeoff compared to uniformly allocating sparsity budget across layers. This leads to the question: Can one optimally allocate sparsity budgets across the layers? The optimal allocation of sparsity across layers is itself a hard combinatorial problem with a large search space: e.g., with L layers and K sparsity levels, there are $\mathcal{O}(K^L)$ different configurations. In the context of one-shot pruning, hand-crafted heuristics as well as reinforcement learning based approaches that require finetuning [108] are not well-suited because of high tuning complexity. [74, 76] propose an interesting approach: they consider a proxy objective (for the sparsity budget allocation problem), constructed with only forward passes and then a dynamic program is used to obtain the sparsity budgets. This approach is feasible for the problem sizes we consider. However, the proxy objective considered by this method does not consider any interactions between different layers. We propose an optimization approach to perform sparsity budget allocation that considers interactions between different layers. To optimizie our non-differentiable, discrete optimization problem, we propose a novel coordinate-descent style algorithm. Our approach leads to high-quality sparsity budget allocations and accuracy-sparsity tradeoffs, without compromising on computational efficiency compared to earlier state-of-the-art approaches. The full paper is given in Afriat et al. [3].

Chapter 10

Conclusion

This thesis presented large-scale optimization approaches for several ensemble learning methods under structural constraints. In particular, in Chapters 2 and 3, we consider additive models with interactions under component selection and other structural constraints such as hierarchy. We discuss efficient algorithms based on sparse backpropagation and partially greedy coordinate descent algorithms to handle different formulations. The structural constraints aid global interpretability and allow insights into real-world applications. We consider a large-scale case study of predicting Census Survey Response Prediction in Chapter 4. Next, in Chapters 5 and 6, we propose efficient tensor-based formulation for tree ensembles, and consider flexible modeling for customized loss functions, multi-task learning and end-to-end feature selection for tree ensembles. Our methods can improve over existing tree ensemble toolkits. Finally, in Chapters 7 and 8, we consider multiple novel approaches for Sparse Mixture of Experts, with per-input sparsity constraints. We optimize these formulations to achieve efficient training and/or sparse inference. We show our methods can scale to large language models and improve over existing sparse routing approaches in terms of statistical performance.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[2] Ryan Prescott Adams and Richard S. Zemel. Ranking via sinkhorn propagation, 2011. URL https://arxiv.org/abs/1106.1925.

[3] Gabriel I. Afriat, Xiang Meng, Shibal Ibrahim, Hussein Hazimeh, and Rahul Mazumder. Moonshot: Pruning with non-uniform layer- wise sparsity allocation, 2024.

[4] Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey Hinton. Neural additive models: Interpretable machine learning with neural nets. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=wHkKTW2wrmm.

[5] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, et al. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[6] Uchenna Akujuobi and Xiangliang Zhang. Delve: A dataset-driven scholarly search and analysis system. *SIGKDD Explor. Newsl.*, 19(2):36–46, nov 2017. ISSN 1931-0145.

[7] Genevera I. Allen. Automatic feature selection via weighted kernels and regularization. *Journal of Computational and Graphical Statistics*, 22(2):284–299, 04 2013.

[8] Mikel Artetxe, Shruti Bhosale, Naman Goyal, et al. Efficient large scale language

modeling with mixtures of experts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, dec 2022.

[9] Susan Athey, Julie Tibshirani, and Stefan Wager. Generalized random forests. *The Annals of Statistics*, 2019.

[10] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Structured sparsity through convex optimization. *Statistical Science*, 27(4):450–468, November 2012. doi:10.1214/12-sts394.

[11] Jonathan T. Barron. A general and adaptive robust loss function. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4326–4334, 2019.

[12] Nancy Bates and Mary H. Mulry. Using a geographic segmentation to understand, predict, and plan for census and survey mail nonresponse. *Journal of Official Statistics*, 27(4):601–618, July 2011.

[13] R. Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, 5(4):537–550, 1994. doi:10.1109/72.298224.

[14] Amir Beck and Yonina C. Eldar. Sparsity constrained nonlinear optimization: Optimality conditions and algorithms. *SIAM Journal on Optimization*, 23(3):1480–1509, 2013. doi:10.1137/120869778. URL https://doi.org/10.1137/120869778.

[15] Riade Benbaki, Wenyu Chen, Xiang Meng, Hussein Hazimeh, Natalia Ponomareva, Zhe Zhao, and Rahul Mazumder. Fast as CHITA: Neural network pruning with combinatorial optimization. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 2031–2049. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/benbaki23a.html.

[16] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. In *International Conference on Learning Representations Workshop Tract*, 2016. URL https://openreview.net/forum?id=B1ckMDqlg.

[17] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

[18] K.P. Bennett. *Decision Tree Construction Via Linear Programming*. Number no. 1067 in Computer sciences technical report. University of Wisconsin-Madison, Computer Sciences Department, 1992.

[19] Kristin P. Bennett and Jennifer A. Blue. Optimal decision trees. Technical report, R.P.I. Math Report No. 214, Rensselaer Polytechnic Institute, 1996.

[20] James Bergstra, Brent Komer, Chris Eliasmith, et al. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

[21] D. Bertsimas and J. Dunn. *Machine Learning Under a Modern Optimization Lens*. Dynamic Ideas LLC, 2019.

[22] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106:1039–1082, 2017.

[23] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[24] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, April 2016. doi:10.1214/15-aos1388.

[25] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *Annals of Statistics*, 44(2):813–852, 2016.

[26] Jacob Bien, Jonathan Taylor, and Robert Tibshirani. A lasso for hierarchical interactions. *The Annals of Statistics*, 41(3):1111–1141, June 2013. doi:10.1214/13-aos1096.

[27] Rafael Blanquero, Emilio Carrizosa, Cristina Molero-Río, et al. Optimal randomized classification trees. *Computers & Operations Research*, 132:105281, 2021. ISSN 0305-0548.

[28] A.-L. Boulesteix, A. Bender, J. Lorenzo Bermejo, and C. Strobl. Random forest gini importance favours SNPs with large minor allele frequency: impact, sources and recommendations. *Briefings in Bioinformatics*, 13(3):292–304, September 2011. doi:10.1093/bib/bbr053. URL https://doi.org/10.1093/bib/bbr053.

[29] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984. ISBN 9780412048418.

[30] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[31] Antonio Bruce, J Gregory Robinson, and Monique V Sanders. Hard-to-count scores and broad demographic groups associated with patterns of response rates in census 2000. In *Joint Statistical Meeting*, 2001.

[32] Peter Buhlmann and Sara van de Geer. *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2011. ISBN 3642201911.

[33] Lars Buitinck, Gilles Louppe, Mathieu Blondel, et al. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[34] Nadia Burkart and Marco F. Huber. A survey on the explainability of supervised machine learning. *J. Artif. Intell. Res.*, 70:245–317, 2020.

[35] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. Feature selection in machine learning: A new perspective. *Neurocomputing*, 300:70–79, 2018.

[36] A. Colin Cameron and Pravin K. Trivedi. *Regression Analysis of Count Data*.

Econometric Society Monographs. Cambridge University Press, 2 edition, 2013. doi:10.1017/CBO9781139013567.

[37] Miguel A. Carreira-Perpinan and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In S. Bengio, H. Wallach, H. Larochelle, et al., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[38] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[39] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi:10.18653/v1/S17-2001. URL https://aclanthology.org/S17-2001.

[40] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014. ISSN 0045-7906. doi:https://doi.org/10.1016/j.compeleceng.2013.11.024. 40th-year commemorative issue.

[41] Chun-Hao Chang, Rich Caruana, and Anna Goldenberg. NODE-GAM: Neural generalized additive model for interpretable deep learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=g8NJR6fCCl8.

[42] Olivier Chapelle, Pannagadatta K. Shivaswamy, Srinivas Vadrevu, et al. Boosted multi-task learning. *Machine Learning*, 85:149–173, 2010.

[43] Jianbo Chen, Mitchell Stern, Martin J Wainwright, and Michael I Jordan. Kernel feature selection via conditional covariance minimization. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[44] Jianbo Chen, Le Song, Martin J. Wainwright, et al. Learning to explain: An information-theoretic perspective on model interpretation. In *International Conference on Machine*

Learning, 2018.

[45] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi:10.1145/2939672.2939785.

[46] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322.

[47] Yao Chen, Qingyi Gao, Faming Liang, et al. Nonlinear variable selection via deep neural networks. *Journal of Computational and Graphical Statistics*, 30(2):484–492, 2021.

[48] Zewen Chi, Li Dong, Shaohan Huang, et al. On the representation collapse of sparse mixture of experts. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[49] Hugh Chipman. Bayesian variable selection with related predictors. *The Canadian Journal of Statistics / La Revue Canadienne de Statistique*, 24(1):17–36, 1996. ISSN 03195724. URL http://www.jstor.org/stable/3315687.

[50] Aidan Clark, Diego De Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George Bm Van Den Driessche, Eliza Rutherford, Tom Hennigan, Matthew J Johnson, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Marc'Aurelio Ranzato, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine*

Learning, volume 162 of *Proceedings of Machine Learning Research*, pages 4057–4086. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/clark22a.html.

[51] Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *ArXiv*, abs/2009.09796, 2020.

[52] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[53] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[54] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL http://arxiv.org/abs/1810.04805. cite arxiv:1810.04805Comment: 13 pages.

[55] Joshua V. Dillon, Ian Langmore, Dustin Tran, et al. Tensorflow distributions. *CoRR*, abs/1711.10604, 2017.

[56] V. Dinh and L. S. T. Ho. Consistent feature selection for analytic deep neural networks. In *NIPS'20*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

[57] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. URL https://aclanthology.org/I05-5002.

[58] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Asso-

ciates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/c5dc3e08849bec07e33ca353de62ea04-Paper.pdf.

[59] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[60] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient scaling of language models with mixture-of-experts. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/du22c.html.

[61] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[62] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[63] Abhimanyu Dubey, Filip Radenovic, and Dhruv Mahajan. Scalable interpretability via polynomials. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=TwuColwZAVj.

[64] Paul H. C. Eilers and Brian D. Marx. Flexible smoothing with b-splines and penalties. *Statistical Science*, 11:89–121, 1996.

[65] Paul H.C. Eilers and Brian D. Marx. Multivariate calibration with temperature interaction using two-dimensional penalized signal regression. *Chemometrics and Intelligent*

Laboratory Systems, 66(2):159–174, June 2003. doi:10.1016/s0169-7439(03)00029-7.

[66] James Enouen and Yan Liu. Sparse interaction additive networks via feature interaction detection and sparse selection. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=Q6DJ12oQjrp.

[67] C. Erdman and Nancy J. Bates. The u.s. census bureau mail return rate challenge: Crowdsourcing to develop a hard-to-count score. 2014.

[68] Chandra Erdman and Nancy Bates. The low response score (LRS). *Public Opinion Quarterly*, 81(1):144–156, December 2016. doi:10.1093/poq/nfw040.

[69] Pablo A. Estevez, Michel Tesmer, Claudio A. Perez, and Jacek M. Zurada. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2): 189–201, 2009. doi:10.1109/TNN.2008.2005601.

[70] Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456): 1348–1360, 2001. doi:10.1198/016214501753382273. URL https://doi.org/10.1198/016214501753382273.

[71] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, jun 2008. ISSN 1532-4435.

[72] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[73] Jean Feng and Noah Simon. Sparse-input neural networks for high-dimensional nonparametric regression and classification. *arXiv: Methodology*, 2017.

[74] Elias Frantar and Dan Alistarh. SPDY: accurate pruning with speedup guarantees. *CoRR*, abs/2201.13096, 2022. URL https://arxiv.org/abs/2201.13096.

[75] Elias Frantar, Eldar Kurtic, and Dan Alistarh. M-fac: Efficient matrix-free approximations of second-order information. *Advances in Neural Information Processing Systems*, 34:14873–14886, 2021.

[76] Elias Frantar, Sidak Pal Singh, and Dan Alistarh. Optimal Brain Compression: a framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 36, 2022.

[77] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 2010. doi:10.18637/jss.v033.i01.

[78] Jerome H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1):1 – 67, 1991. doi:10.1214/aos/1176347963. URL https://doi.org/10.1214/aos/1176347963.

[79] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree, 2017.

[80] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[81] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3681–3688, Jul. 2019.

[82] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[83] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.

[84] Peter Hall and J. D. Opsomer. Theory for penalised spline regression. *Biometrika*, 92

(1):105–118, March 2005. doi:10.1093/biomet/92.1.105.

[85] M. Hamada and C. F. J. Wu. Analysis of designed experiments with complex aliasing. *Journal of Quality Technology*, 24(3):130–137, 1992.

[86] Lei Han and Yu Zhang. Learning tree structure in multi-task learning. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, page 397–406, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642.

[87] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

[88] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015. ISSN 2160-6455. doi:10.1145/2827872. URL https://doi.org/10.1145/2827872.

[89] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

[90] Babak Hassibi, David G.Stork, and Gregory Wolff. Optimal brain surgeon and general network pruning. pages 293 – 299 vol.1, 02 1993. ISBN 0-7803-0999-5. doi:10.1109/ICNN.1993.298572.

[91] T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models*. Chapman and Hall, 1990.

[92] T. J. Hastie, R. J. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2 edition, 2009.

[93] Trevor Hastie and Robert Tibshirani. Generalized additive models: some applications. *Journal of the American Statistical Association*, 82(398):371–386, 1987.

[94] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[95] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall/CRC, 2015. ISBN 1498712169, 9781498712163.

[96] H. Hazimeh, N. Ponomareva, P. Mol, et al. The tree ensemble layer: Differentiability meets conditional computation. In Hal Daumé III and Aarti Singh, editors, *ICML'20*, volume 119 of *Proceedings of Machine Learning Research*, pages 4138–4148. PMLR, 13–18 Jul 2020.

[97] Hussein Hazimeh and Rahul Mazumder. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Oper. Res.*, 68(5):1517–1537, September 2020. ISSN 0030-364X. doi:10.1287/opre.2019.1919.

[98] Hussein Hazimeh and Rahul Mazumder. Learning hierarchical interactions at scale: A convex optimization approach. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1833–1843. PMLR, 26–28 Aug 2020.

[99] Hussein Hazimeh and Rahul Mazumder. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Oper. Res.*, 68(5):1517–1537, September 2020. ISSN 0030-364X. doi:10.1287/opre.2019.1919.

[100] Hussein Hazimeh, Natalia Ponomareva, Petros Mol, et al. The tree ensemble layer: Differentiability meets conditional computation. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4138–4148. PMLR, 13–18 Jul 2020.

[101] Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed Chi. DSelect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=tKlYQJLYN8v.

[102] Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed Chi. DSelect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=tKlYQJLYN8v.

[103] Hussein Hazimeh, Rahul Mazumder, and Ali Saab. Sparse regression at scale: Branch-and-bound rooted in first-order optimization. *Mathematical Programming*, 196(1-2): 347–388, 2022.

[104] Hussein Hazimeh, Rahul Mazumder, and Peter Radchenko. Grouped variable selection with discrete optimization: Computational and statistical perspectives. *The Annals of Statistics*, 51(1):1–32, 2023.

[105] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi:10.1109/CVPR.2016.90.

[106] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[107] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society. doi:10.1109/ICCV.2017.155. URL https://doi.ieeecomputersociety.org/10.1109/ICCV.2017.155.

[108] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*, 2018.

[109] Thomas M. Hehn, Julian F. P. Kooij, and Fred A. Hamprecht. End-to-end learning of decision trees and forests. *International Journal of Computer Vision*, 128:997–1011, 2019.

[110] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021. URL http://jmlr.org/papers/v22/21-0366.html.

[111] J. Huang, J.L. Horowitz, and F. Wei. Variable selection in nonparametric additive models. *The Annals of Statistics*, 38:2282–2313, 2010.

[112] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976. ISSN 0020-0190.

[113] Shibal Ibrahim, Rahul Mazumder, Peter Radchenko, and Emanuel Ben-David. Predicting census survey response rates with parsimonious additive models and structured interactions. *arXiv*, abs/2108.11328, 2021.

[114] Shibal Ibrahim, Rahul Mazumder, Peter Radchenko, and Emanuel Ben-David. Predicting census survey response rates with parsimonious additive models and structured interactions. *arXiv*, abs/2108.11328, 2021.

[115] Shibal Ibrahim, Hussein Hazimeh, and Rahul Mazumder. Flexible modeling and multitask learning using differentiable tree ensembles. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 666–675, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi:10.1145/3534678.3539412. URL https://doi.org/10.1145/3534678.3539412.

[116] Shibal Ibrahim, Gabriel Afriat, Kayhan Behdin, and Rahul Mazumder. GRAND-

SLAMIN' interpretable additive modeling with structural constraints. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=F5DYsAc7Rt.

[117] Shibal Ibrahim, Wenyu Chen, Hussein Hazimeh, Natalia Ponomareva, Zhe Zhao, and Rahul Mazumder. Comet: Learning cardinality constrained mixture of experts with trees and local search. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 832–844, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701030. doi:10.1145/3580305.3599278. URL https://doi.org/10.1145/3580305.3599278.

[118] Shibal Ibrahim, Kayhan Behdin, and Rahul Mazumder. End-to-end feature selection approach for learning skinny trees. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research. PMLR, 25–27 Apr 2024.

[119] Yani Ioannou, Duncan P. Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *CoRR*, abs/1603.01250, 2016. URL http://arxiv.org/abs/1603.01250.

[120] Ozan Irsoy, O. T. Yildiz, and Ethem Alpaydin. Soft decision trees. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1819–1822, 2012.

[121] A. A. Ismail, S. Ö. Arik, J. Yoon, et al. Interpretable mixture of experts for structured data. *ArXiv*, abs/2206.02107, 2022.

[122] Robert A. Jacobs. Bias/variance analyses of mixtures-of-experts architectures. *Neural Computation*, 9(2):369–383, 1997. doi:10.1162/neco.1997.9.2.369.

[123] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

doi:10.1162/neco.1991.3.1.79.

[124] Wura Jacobs, Ehikowoicho Idoko, LaTrice Montgomery, et al. Concurrent e-cigarette and marijuana use and health-risk behaviors among u.s. high school students. *Preventive Medicine*, 145:106429, 2021. ISSN 0091-7435.

[125] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

[126] Alan Jeffares, Tennison Liu, Jonathan Crabbé, and Mihaela van der Schaar. Joint training of deep ensembles fails due to learner collusion. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=WpGLxnOWhn.

[127] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.

[128] W. Jiang and M.A. Tanner. On the identifiability of mixtures-of-experts. *Neural Networks*, 12(9):1253–1258, 1999. ISSN 0893-6080. doi:https://doi.org/10.1016/S0893-6080(99)00066-0. URL https://www.sciencedirect.com/science/article/pii/S0893608099000660.

[129] Xiaojie Jin, Xiaotong Yuan, Jiashi Feng, and Shuicheng Yan. Training skinny deep neural networks with iterative hard thresholding methods, 2016.

[130] M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the em algorithm. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 2, pages 1339–1344 vol.2, 1993. doi:10.1109/IJCNN.1993.716791.

[131] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em

algorithm. *Neural Comput.*, 6(2):181–214, mar 1994. ISSN 0899-7667.

[132] E. E. Kammann and M. P. Wand. Geoadditive models. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 52(1):1–18, January 2003. doi:10.1111/1467-9876.00385.

[133] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960. ISSN 00251909, 15265501.

[134] Guolin Ke, Qi Meng, Thomas Finley, et al. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[135] Rajiv Khanna, Ethan Elenberg, Alex Dimakis, Sahand Negahban, and Joydeep Ghosh. Scalable Greedy Feature Selection via Weak Submodularity. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1560–1568. PMLR, 20–22 Apr 2017. URL https://proceedings.mlr.press/v54/khanna17b.html.

[136] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[137] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[138] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324, 1997.

[139] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, et al. Deep neural decision forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages

1467–1475, 2015.

[140] Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. Beyond distillation: Task-level mixture-of-experts for efficient inference. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3577–3599, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.findings-emnlp.304. URL https://aclanthology.org/2021.findings-emnlp.304.

[141] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi:https://doi.org/10.1002/nav.3800020109. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

[142] Robert Kulzick, Laura Kail, Shawnna Mullenax, et al. 2020 census predictive models and audience segmentation report. June 2019.

[143] Abhishek Kumar and Hal Daumé. Learning task grouping and overlap in multi-task learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1723–1730, Madison, WI, USA, 2012. Omnipress. ISBN 9781450312851.

[144] Eldar Kurtić, Elias Frantar, and Dan Alistarh. Ziplm: Inference-aware structured pruning of language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=d8j3lsBWpV.

[145] Denis Kuznedelev, Eldar Kurtic, Elias Frantar, and Dan Alistarh. CAP: Correlation-aware pruning for highly-accurate sparse vision models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Xy7DoWSNZX.

[146] Woosuk Kwon, Sehoon Kim, Michael W. Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. A fast post-training pruning framework for transformers. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in*

Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=0GRBKLBjJE.

[147] Himabindu Lakkaraju and Osbert Bastani. "how do i fool you?": Manipulating user trust via misleading black box explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, AIES '20, page 79–85, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371100. doi:10.1145/3375627.3375833. URL https://doi.org/10.1145/3375627.3375833.

[148] Diane Lambert. Zero-inflated poisson regression, with an application to defects in manufacturing. *Technometrics*, 34(1):1–14, feb 1992. ISSN 0040-1706.

[149] Guanghui Lan. An optimal method for stochastic composite optimization. *Mathematical Programming*, 133(1):365–397, 2012.

[150] Nathan Lay, Adam P. Harrison, Sharon Schreiber, et al. Random hinge forest for differentiable learning. *CoRR*, abs/1802.03882, 2018.

[151] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2554–2564, 2016. doi:10.1109/CVPR.2016.280.

[152] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[153] Simon C. K. Lee. Addressing imbalanced insurance data through zero-inflated poisson regression with boosting. *ASTIN Bulletin*, 51:27 – 55, 2020.

[154] Ismael Lemhadri, Feng Ruan, and Rob Tibshirani. Lassonet: Neural networks with feature sparsity. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 10–18. PMLR, 13–15 Apr 2021.

[155] Benjamin J. Lengerich, S. Tan, Chun-Hao Kingsley Chang, Giles Hooker, and Rich Caruana. Purifying interaction effects with the functional anova: An efficient algorithm for recovering identifiable additive models. In *International Conference on Artificial Intelligence and Statistics*, 2019.

[156] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

[157] Hector J. Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, page 552–561. AAAI Press, 2012. ISBN 9781577355601.

[158] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, 2021.

[159] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

[160] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Comput. Surv.*, 50 (6), dec 2017. ISSN 0360-0300. doi:10.1145/3136625.

[161] K.C. Li, H.H. Lue, and C.H. Chen. Interactive tree-structured regression via principal hessian direction. *Journal of the American Statistical Association*, 95:547–560, 2000.

[162] Yingxing Li and David Ruppert. On the asymptotics of penalized splines. *Biometrika*, 95(2):415–436, 2008. ISSN 00063444.

[163] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461: 370–403, 2021.

[164] Michael Lim and Trevor Hastie. Learning interactions via hierarchical group-lasso regularization. *Journal of Computational and Graphical Statistics*, 24(3):627–654, July 2015. doi:10.1080/10618600.2014.938812.

[165] Yi Lin and Hao Helen Zhang. Component selection and smoothing in multivariate nonparametric regression. *The Annals of Statistics*, 34(5):2272–2297, October 2006. doi:10.1214/009053606000000722.

[166] Henrik Linusson. Multi-output random forests. 2013.

[167] Zachary C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, jun 2018. ISSN 1542-7730. doi:10.1145/3236386.3241340. URL https://doi.org/10.1145/3236386.3241340.

[168] Brian Liu, Miaolan Xie, and Madeleine Udell. Controlburn: Feature selection by sparse forests. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '21, page 1045–1054, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi:10.1145/3447548.3467387.

[169] Tianlin Liu, Joan Puigcerver, and Mathieu Blondel. Sparsity-constrained optimal transport. In *The Eleventh International Conference on Learning Representations*, 2023.

[170] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[171] Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. In *Proceedings of the 18th ACM SIGKDD International Con-*

ference on Knowledge Discovery and Data Mining, KDD '12, page 150–158, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450314626. doi:10.1145/2339530.2339556. URL https://doi.org/10.1145/2339530.2339556.

[172] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 623–631, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321747. doi:10.1145/2487575.2487579. URL https://doi.org/10.1145/2487575.2487579.

[173] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. In *International Conference on Learning Representations*, 2018.

[174] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[175] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 5068–5076. IEEE Computer Society, 2017. doi:10.1109/ICCV.2017.541.

[176] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1930–1939, 2018.

[177] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

[178] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Vota6rFhBQ.

[179] Enno Mammen and Byeong U. Park. A simple smooth backfitting method for additive models. *The Annals of Statistics*, 34(5):2252 – 2271, 2006.

[180] Enno Mammen, Oliver Linton, and Janni Nielsen. The existence and asymptotic properties of a backfitting projection algorithm under weak conditions. *The Annals of Statistics*, 27(5):1443–1490, 1999.

[181] Rahul Mazumder and Peter Radchenko. The Discrete Dantzig Selector: Estimating sparse linear models via mixed integer linear optimization. *IEEE Transactions on Information Theory*, 63 (5):3053 – 3075, 2017.

[182] Rahul Mazumder, Peter Radchenko, and Antoine Dedieu. Subset selection with shrinkage: Sparse linear modeling when the snr is low. *arXiv preprint arXiv:1708.03288 (Operations Research, to appear)*, 2017.

[183] Peter McCullagh and John A Nelder. *Generalized linear models*, volume 37. CRC press, 1989.

[184] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, February 2018.

[185] Lukas Meier, Sara van de Geer, and Peter Buhlmann. High-dimensional additive modeling. *The Annals of Statistics*, 37(6B):3779 – 3821, 2009. doi:10.1214/09-AOS692.

[186] Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Byt3oJ-0W.

[187] Xiang Meng, Shibal Ibrahim, Kayhan Behdin, Hussein Hazimeh, Natalia Ponomareva,

and Rahul Mazumder. Osscar: One-shot structured pruning in vision and language models with combinatorial optimization, 2024.

[188] Fan Min, Qinghua Hu, and William Zhu. Feature selection with test cost constraint. *International Journal of Approximate Reasoning*, 55(1, Part 2):167–179, 2014. ISSN 0888-613X. doi:https://doi.org/10.1016/j.ijar.2013.04.003. Special issue on Decision-Theoretic Rough Sets.

[189] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJGCiw5gl.

[190] Md. Monirul Kabir, Md. Monirul Islam, and Kazuyuki Murase. A new wrapper feature selection approach using neural network. *Neurocomputing*, 73(16):3273–3283, 2010.

[191] Nuno Moniz and Luís Torgo. Multi-source social feedback of online news feeds. *ArXiv*, abs/1801.07055, 2018.

[192] Tom Mule. 2010 census coverage measurement estimation report: Summary of estimates of coverage for persons in the united states. Dssd 2010 census coverage measurement memorandum series no. 2010-g-01, US Census Bureau, May 2012.

[193] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *J. Artif. Int. Res.*, 2(1):1–32, aug 1994. ISSN 1076-9757.

[194] Basil Mustafa, Carlos Riquelme Ruiz, Joan Puigcerver, Rodolphe Jenatton, and Neil Houlsby. Multimodal contrastive learning with LIMoe: the language-image mixture of experts. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=Qy1D9JyMBg0.

[195] J. A. Nelder. A reformulation of linear models. *Journal of the Royal Statistical Society. Series A (General)*, 140(1):48–77, 1977. ISSN 00359238. URL http://www.jstor.org/stable/2344517.

[196] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

[197] Yuval Netzer, Tao Wang, Adam Coates, A. Bissacco, Bo Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[198] Xiaonan Nie, Xupeng Miao, Shijie Cao, Lingxiao Ma, Qibin Liu, Jilong Xue, Youshan Miao, Yi Liu, Zhi Yang, and Bin Cui. Evomoe: An evolutional mixture-of-experts training framework via dense-to-sparse gate, 2021. URL https://arxiv.org/abs/2112.14397.

[199] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. Interpretml: A unified framework for machine learning interpretability. *ArXiv*, abs/1909.09223, 2019.

[200] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, 2017.

[201] V. Onnia, M. Tico, and J. Saarinen. Feature selection method using neural network. In *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*, volume 1, pages 513–516 vol.1, 2001. doi:10.1109/ICIP.2001.959066.

[202] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi:10.18653/v1/N19-4009. URL https://aclanthology.org/N19-4009.

[203] Zohreh Ovaisi, Ragib Ahsan, Yifan Zhang, et al. Correcting for selection bias in learning-to-rank systems. In *Proceedings of The Web Conference 2020*, WWW '20, page 1863–1873, New York, NY, USA, 2020. Association for Computing Machinery.

ISBN 9781450370233. doi:10.1145/3366423.3380255.

[204] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[205] Max Paulus, Dami Choi, Daniel Tarlow, Andreas Krause, and Chris J Maddison. Gradient estimation with stochastic softmax tricks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5691–5704. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/3df80af53dce8435cf9ad6c3e7a403fd-Paper.pdf.

[206] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(null):2825–2830, November 2011. ISSN 1532-4435.

[207] Julio L. Peixoto. Hierarchical variable selection in polynomial regression models. *The American Statistician*, 41(4):311–313, 1987. ISSN 00031305. URL http://www.jstor.org/stable/2684752.

[208] Hanchuan Peng, Fuhui Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005. doi:10.1109/TPAMI.2005.159.

[209] Alexandra Peste, Eugenia Iofinova, Adrian Vladu, and Dan Alistarh. AC/DC: Alternating compressed/decompressed training of deep neural networks. In A. Beygelzimer,

Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=T3_AJr9-R5g.

[210] Ben Peters, Vlad Niculae, and André F. T. Martins. Sparse sequence-to-sequence models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1504–1519, Florence, Italy, July 2019. Association for Computational Linguistics. doi:10.18653/v1/P19-1146. URL https://aclanthology.org/P19-1146.

[211] Gabriel Peyré and Marco Cuturi. Computational optimal transport, 2020.

[212] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1eiu2VtwH.

[213] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, et al. Catboost: Unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6639–6649, Red Hook, NY, USA, 2018. Curran Associates Inc.

[214] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1558602380.

[215] Peter Radchenko and Gareth M. James. Variable selection using adaptive nonlinear interaction structures in high dimensions. *Journal of the American Statistical Association*, 105(492):1541–1553, December 2010. doi:10.1198/jasa.2010.tm10130.

[216] Filip Radenovic, Abhimanyu Dubey, and Dhruv Mahajan. Neural basis models for interpretability. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=fpfDusqKZF.

[217] Colin Raffel, Noam Shazeer, Adam Roberts, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), jan 2020. ISSN 1532-4435.

[218] 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*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi:10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

[219] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi:10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124.

[220] Pradeep Ravikumar, John Lafferty, Han Liu, et al. Sparse additive models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(5):1009–1030, November 2009. doi:10.1111/j.1467-9868.2009.00718.x.

[221] Juha Reunanen. Overfitting in making comparisons between variable selection methods. *J. Mach. Learn. Res.*, 3(null):1371–1382, mar 2003. ISSN 1532-4435.

[222] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

[223] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi:10.1145/2939672.2939778.

[224] Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason E Weston. Hash layers for large sparse models. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=lMgDDWb1ULW.

[225] Debaditya Roy, K. Sri Rama Murty, and C. Krishna Mohan. Feature selection using deep neural networks. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2015. doi:10.1109/IJCNN.2015.7280626.

[226] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *ArXiv*, abs/1706.05098, 2017.

[227] Carlos Riquelme Ruiz, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=NGPmH3vbAA_.

[228] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3859–3869, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[229] Michael E. Sander, Joan Puigcerver, Josip Djolonga, et al. Fast, differentiable and sparse top-k: a convex analysis perspective. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[230] Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

[231] Robert E. Schapire and Yoav Freund. *Boosting: Foundations and Algorithms*. The MIT Press, 2012. ISBN 0262017180.

[232] Shubham Sharma, Sanghamitra Dutta, Emanuele Albini, Freddy Lecue, Daniele Mag-

azzeni, and Manuela Veloso. Refresh: Responsible and efficient feature reselection guided by shap values. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '23, page 443–453, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702310. doi:10.1145/3600211.3604706. URL https://doi.org/10.1145/3600211.3604706.

[233] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

[234] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109, 2020.

[235] Dylan Z Slack, Sophie Hilgard, Sameer Singh, and Himabindu Lakkaraju. Reliable post hoc explanations: Modeling uncertainty in explainability. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=rqfq0CYIekd.

[236] Tom W. Smith. Hard-to-survey populations in comparative perspective. In *Hard-to-Survey Populations*, pages 21–36. Cambridge University Press, Cambridge, 2014. ISBN 9781139381635. doi:10.1017/CBO9781139381635.004.

[237] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1170.

[238] Le Song, Alex Smola, Arthur Gretton, Karsten M. Borgwardt, and Justin Bedo. Su-

pervised feature selection via dependence estimation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 823–830, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595937933. doi:10.1145/1273496.1273600.

[239] Le Song, Alex Smola, Arthur Gretton, Justin Bedo, and Karsten Borgwardt. Feature selection via dependence maximization, 2012.

[240] Gary Stein, Bing Chen, Annie S. Wu, and Kien A. Hua. Decision tree classifier for network intrusion detection with ga-based feature selection. In *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 2*, ACM-SE 43, page 136–141, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930590. doi:10.1145/1167253.1167288.

[241] Charles J Stone. The dimensionality reduction principle for generalized additive models. *The Annals of Statistics*, pages 590–606, 1986.

[242] Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(1):25, 2007.

[243] Zhiqiang Tan and Cun-Hui Zhang. Doubly penalized estimation in additive regression with high-dimensional data. *The Annals of Statistics*, 47(5):2567–2600, 2019.

[244] Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, et al. Adaptive neural trees. *ArXiv*, abs/1807.06699, 2019.

[245] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 2022/05/03 1996.

[246] Roger Tourangeau, Brad Edwards, Timothy P Johnson, et al. *Hard-to-survey populations*. Cambridge University Press, 2014.

[247] Michael Tsang, Sirisha Rambhatla, and Yan Liu. How does this interaction affect me?

interpretable attribution for feature interactions. In *Advances in Neural Information Processing Systems*, 2020.

[248] US Census Bureau. 2010 census integrated communications campaign plan: The success of the census is in our hands. November 2009.

[249] US Census Bureau. American community survey data suppression. September 2016.

[250] US Census Bureau. Foundational communications models 1.0: the updated low response score (lrs) and an internet lrs. 2017.

[251] US Census Bureau. Planning database, June 2019.

[252] US Census Bureau. Planning database, Jun 2021.

[253] US Census Bureau. Planning database, October 2022.

[254] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[255] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression : An o(n) algorithm for incremental real time learning in high dimensional space. 2000.

[256] Grace Wahba. *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics, January 1990. doi:10.1137/1.9781611970128.

[257] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJ4km2R5t7.

[258] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[259] Yuyan Wang, Zhe Zhao, Bo Dai, Christopher Fifty, Dong Lin, Lichan Hong, and Ed H. Chi. Small towers make big differences. *ArXiv*, abs/2008.05808, 2020.

[260] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural Network Acceptability Judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 09 2019. ISSN 2307-387X. doi:10.1162/tacl_a_00290. URL https://doi.org/10.1162/tacl_a_00290.

[261] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 2082–2090, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

[262] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi:10.18653/v1/N18-1101. URL https://aclanthology.org/N18-1101.

[263] Thomas Wolf, Lysandre Debut, Victor Sanh, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

[264] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

[265] Simon N. Wood. Low-rank scale-invariant tensor product smooths for generalized additive mixed models. *Biometrics*, 62(4):1025–1036, May 2006. doi:10.1111/j.1541-0420.2006.00574.x.

[266] Simon N. Wood, Zheyuan Li, Gavin Shaddick, et al. Generalized additive models for gigadata: Modeling the u.k. black smoke network daily data. *Journal of the American Statistical Association*, 112(519):1199–1210, April 2017. doi:10.1080/01621459.2016.1195744.

[267] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1): 3–34, 2015.

[268] Lemeng Wu, Mengchen Liu, Yinpeng Chen, Dongdong Chen, Xiyang Dai, and Lu Yuan. Residual mixture of experts. *ArXiv*, abs/2204.09636, 2022.

[269] Julia D. Wulfkuhle, Lance A. Liotta, and Emanuel F. Petricoin. Proteomic applications for the early detection of cancer. *Nature Reviews Cancer*, 3(4):267–275, 2003.

[270] Sang Michael Xie and Stefano Ermon. Reparameterizable subset sampling via continuous relaxations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 3919–3925. AAAI Press, 2019. ISBN 9780999241141.

[271] Yuan Xie, Shaohan Huang, Tianyu Chen, et al. Moec: Mixture of expert clusters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(11):13807–13815, Jun. 2023. doi:10.1609/aaai.v37i11.26617.

[272] Eleftherios Spyromitros Xioufis, Grigorios Tsoumakas, William Groves, et al. Multi-target regression via input space expansion: treating targets as inputs. *Machine Learning*, 104:55–98, 2016.

[273] Shibiao Xu, Zhiqi Bu, Pratik Chaudhari, and Ian Barnett. Sparse neural additive model: Interpretable deep learning with feature selection via group sparsity. *ArXiv*, abs/2202.12482, 2022.

[274] Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Feature selection using stochastic gates. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10648–10659. PMLR, 13–18 Jul 2020.

[275] Xiaohan Yan and Jacob Bien. Hierarchical sparse modeling: A choice of two group lasso

formulations. *Statistical Science*, 32(4):531–560, November 2017. doi:10.1214/17-sts622.

[276] Zebin Yang, Aijun Zhang, and A. Sudjianto. Gami-net: An explainable neural network based on generalized additive models with structured interactions. *Pattern Recognit.*, 120:108192, 2021.

[277] Ashenafi A. Yirga, Sileshi F. Melesse, Henry G. Mwambi, et al. Negative binomial mixed models for analyzing longitudinal cd4 count data. *Scientific Reports*, 10(1):16742, 2020.

[278] Derek S. Young, Andrew M. Raim, and Nancy R. Johnson. Zero-inflated modelling for characterizing coverage errors of extracts from the us census bureau's master address file. *Journal of The Royal Statistical Society Series A-statistics in Society*, 180:73–97, 2017.

[279] Guo Yu, Daniela Witten, and Jacob Bien. Controlling costs: Feature selection on a budget. *Stat*, 11(1):e427, 2022.

[280] R. Yu, A. Li, C. Chen, J. Lai, V. I. Morariu, X. Han, M. Gao, C. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9194–9203, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society. doi:10.1109/CVPR.2018.00958. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00958.

[281] Xin Yu, Thiago Serra, Srikumar Ramalingam, and Shandian Zhe. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *International Conference on Machine Learning*, pages 25668–25683. PMLR, 2022.

[282] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society, Series B*, 68:49–67, 2006.

[283] Ming Yuan and Ding-Xuan Zhou. Minimax optimal rates of estimation in high dimensional additive models. *The Annals of Statistics*, 44(6):2564–2593, 2016.

[284] Cun-Hui Zhang. Nearly unbiased variable selection under minimax concave penalty.

The Annals of Statistics, 38(2):894–942, 2010. ISSN 00905364, 21688966. URL http://www.jstor.org/stable/25662264.

[285] Shichao Zhang. Cost-sensitive classification with respect to waiting cost. *Know.-Based Syst.*, 23(5):369–378, jul 2010. ISSN 0950-7051. doi:10.1016/j.knosys.2010.01.008. URL https://doi.org/10.1016/j.knosys.2010.01.008.

[286] Xiaohui Zhang, Daniel Povey, and Sanjeev Khudanpur. A diversity-penalizing ensemble training method for deep learning. In *Proc. Interspeech 2015*, pages 3590–3594, 2015. doi:10.21437/Interspeech.2015-712.

[287] Tuo Zhao and Han Liu. Sparse additive machine. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 1435–1443, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

[288] Arman Zharmagambetov, Suryabhan Singh Hada, and Miguel Á. Carreira-Perpiñán. An experimental comparison of old and new decision tree algorithms. *CoRR*, abs/1911.03054, 2019.

[289] Arman S. Zharmagambetov and Miguel Á. Carreira-Perpiñán. Smaller, more accurate regression forests using tree alternating optimization. In *ICML*, 2020.

[290] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Y Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V Le, and James Laudon. Mixture-of-experts with expert choice routing. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=jdJo1HIVinI.

[291] Zhengze Zhou and Giles Hooker. Unbiased measurement of feature importance in tree-based methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15:1 – 21, 2021.

[292] Haoran Zhu, Pavankumar Murali, Dzung Phan, et al. A scalable mip-based method for

learning optimal multivariate decision trees. In H. Larochelle, M. Ranzato, R. Hadsell, et al., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1771–1781. Curran Associates, Inc., 2020.

[293] Zexuan Zhu, Yew-Soon Ong, and Manoranjan Dash. Wrapper–filter feature selection algorithm using a memetic framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1):70–76, 2007. doi:10.1109/TSMCB.2006.883267.

[294] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, page 22–32, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930469. doi:10.1145/1060745.1060754. URL https://doi.org/10.1145/1060745.1060754.

[295] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam M. Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. 2022.

[296] Patrick; Weinzierl Zschech, Sven; Hambauer, Nico; Zilker, Sandra;, and Mathias Kraus. Gam(e) changer or not? An evaluation of interpretable machine learning models based on additive model constraints. *(2022). ECIS 2022 Research Papers*, 106., 2022.

[297] Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Jianfeng Gao, and Tuo Zhao. Taming sparsely activated transformer with stochastic experts. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=B72HXs80q4.

[298] Simiao Zuo, Qingru Zhang, Chen Liang, Pengcheng He, Tuo Zhao, and Weizhu Chen. MoEBERT: from BERT to mixture-of-experts via importance-guided adaptation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1610–1623, Seattle, United States, July 2022. Association for Computational Linguistics.

doi:10.18653/v1/2022.naacl-main.116. URL https://aclanthology.org/2022.naacl-main.116.