Anytime Prediction and Learning for the Balance between Computation and Accuracy

Hanzhang Hu

April, 2019
CMU-ML-19-106

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the Office of Naval Research award numbers N000140911052 and N000141512365, the US Army Research Laboratory award number W911NF1020016, and a gift from Verizon Media.

Abstract

When choosing machine learning algorithms, one often has to balance between two opposing factors, the computational speed and the accuracy of predictors. The trade-off during testing is often difficult to balance, because the test-time computational budget may be agnostic at training, and the budget may vary during testing. Analogously, given a novel data-set, one often lacks prior knowledge in the appropriate predictor complexity and training computation, and furthermore, may want to interrupt or prolong the training based on training results.

In this work, we address these trade-offs between computation and accuracy via *anytime* prediction and learning, which are algorithms that can be interrupted at any time and still produce valid solutions. Furthermore, the quality of the results improves with the consumed computation before interruption. With the versatility to adjust to any budget, anytime algorithms automatically utilize any agnostic computational budget to the maximum extent.

To address the test-time trade-off, we study anytime predictors, whose prediction computation can be interrupted during testing. We start with developing provably near-optimal anytime linear predictors, and derive a theoretical performance limitation for anytime predictors that are based on ensemble methods. Then we develop practical anytime predictions within individual neural networks via multi-objective optimization. Furthermore, leveraging these anytime predictors as weak learners, we circumvent the performance limitation on ensemble-based anytime predictors.

For the train-time trade-off, we consider the neural architecture search problem, where one seeks the optimal neural network structure for a data-set. We draw a parallel between this bi-level combinatorial optimization problem and the feature selection problem for linear prediction, and develop an iterative network growth algorithm that is inspired by a forward selection algorithm. We also consider the problem of training on large data-sets, and develop no-regret gradient boosting algorithms for stochastic data streams.

Acknowledgements

I would like to thank J. Andrew Bagnell, Martial Hebert, Ruslan Salakhutdinov, and Rich Caruana, for serving on my thesis committee. I appreciate them for setting aside time for me, and giving me invaluable discussions and suggestions.

I am happy to have been working with Drew and Martial for the past years. They have given me countless help, despite of their busy schedules. They gave me incredible freedom in exploring topics of my choice, and were patient with me to allow me gradually formalize and mature the research ideas. This experience of research and development has given me many invaluable lessons, and I am grateful to have Drew and Martial to guide me through them, like lighthouses in a dark ocean.

I have been fortunate to have many collaborators over the years, in roughly chronological order, Daniel Munoz, Alexander Grubb, Allie Del Giorno, Wen Sun, Arun Venkatraman, Debadeepta Dey, and John Langford. They have taught me many things both in research and in life. Without their help, this work would not have come to be. I especially want to thank Dey for setting up my experiments on his computational resources, and he has run thousands of my scripts over the two years.

- Chapter 3 is based on the Ph.D. thesis of Alexander Grubb, who made an excellent introduction to gradient boosting and its application to anytime prediction for me. In fact the main contributions of this chapter are improvements of unpublished results of his thesis.
- Chapter 4 involves a large number of experiments. Debadeepta Dey kindly offered his computational resources in the middle of this project, and without his help on running thousands of scripts, this work will never come to fruition.
- Chapter 5 came out of a paper reading session of the lab. The careful analysis of Wen Sun and the amazing intuition of Drew led to our initial exploration of this topic. We later also extended previous work of Alexander Grubb on gradient boosting with non-smooth losses to enable streaming gradient boosting to do so as well.
- Chapter 6 started as an internship project at Microsoft Research under the supervision of Debadeepta Dey. Initially this project came out as a bag of engineer hacks that works somehow. Thanks to the inputs from John Langford and Rich Caruana, we were able to steer the project into a more motivated direction that happens to fit into this thesis nicely.

Contents

1 Introduction **1**
- 1.1 Motivations and Problem Settings . . . 1
- 1.2 Approach . . . 2
- 1.3 Overview of Chapters and Their Contributions . . . 4

2 Preliminaries and Background **7**
- 2.1 Anytime Prediction . . . 7
- 2.2 Related Works to the Trade-off Between Computation and Accuracy . . . 8
  - 2.2.1 Anytime Prediction . . . 8
  - 2.2.2 Model Compression . . . 9
  - 2.2.3 Budgeted Prediction . . . 9

3 Anytime Linear Prediction via Feature Group Sequencing **11**
- 3.1 Introduction . . . 11
- 3.2 Computation-Aware Greedy Methods . . . 13
  - 3.2.1 Preliminaries . . . 13
  - 3.2.2 Anytime Prediction at Test-time . . . 14
  - 3.2.3 Computation-Aware Group Orthogonal Matching Pursuit(CS-G-OMP) . 14
  - 3.2.4 Computation-Aware Group Forward Regression (CS-G-FR) . . . 16
- 3.3 Near-Optimality at Features Selection . . . 16
- 3.4 Bi-criteria Analysis at Any Budget . . . 18
- 3.5 Experiments . . . 22
  - 3.5.1 Data-sets and Set-ups . . . 22
  - 3.5.2 Evaluation Metric and Approximated Oracle . . . 23
  - 3.5.3 Importance of Feature Cost . . . 24
  - 3.5.4 Group Whitening . . . 25
  - 3.5.5 Other Selection Criteria Variants . . . 25
- 3.6 Additional Proof Details . . . 27
  - 3.6.1 Functional Boosting View of Feature Selection . . . 27
  - 3.6.2 Proof of Lemma 3.3.3 and Lemma 3.3.4 . . . 28
  - 3.6.3 Proof of Main Theorem . . . 30
- 3.7 Extension to Generalized Linear Models . . . 31
  - 3.7.1 EXAMPLE EXPERIMENTS ON GLM . . . 31

4 Anytime Neural Network via Adaptive Loss Balancing 35
4.1 Introduction 35
4.2 Related Works 37
4.3 Optimizing Anytime Predictors in Networks 38
4.4 Ensemble of Exponentially Deepening Networks 40
4.5 Experiments 41
4.5.1 Data-sets and Training Details 41
4.5.2 Weight Scheme Comparisons 42
4.5.3 EANN: Closing Early Performance Gaps by Delaying Final Predictions. . 45
4.5.4 Data-set Difficulty versus Adaptive Weights 46
4.6 Conclusion and Discussion 47
4.7 Proof of Propostion 4.4.1 47
4.8 Implementation Details of ANNs 49
4.9 Ablation Study for AdaLoss Parameters 50
4.9.1 Weight Regularization 50
4.9.2 Ablation Study of AdaLoss parameters on CIFAR 50

5 Training Gradient Boosting on Stochastic Data Streams 53
5.1 Introduction 53
5.2 Related Works 54
5.3 Preliminaries 55
5.3.1 Online Boosting Setup 55
5.4 Weak Online Learning 56
5.4.1 Why Weak Learner Edge is Reasonable? 57
5.5 Algorithm 58
5.5.1 Smooth Loss Functions 58
5.5.2 Non-smooth Loss Functions 60
5.6 Experiments 62
5.6.1 Experimental Analysis of Regret Bounds 62
5.6.2 Batch Boosting vs. Streaming Boosting 64
5.7 Conclusion 65
5.8 Supplementary Details for Gradient Boosting on Stochastic Data Streams 66
5.8.1 Proof of Proposition 5.4.3 66
5.8.2 Proof of Theorem 5.5.1 68
5.8.3 Proof of Theorem 5.5.2 71
5.8.4 Counter Example for Alg. 6 74
5.8.5 Details of Implementation 75
5.8.6 Binary Classification 75
5.8.7 Proof of Proposition 5.4.3 76
5.8.8 Proof of Proposition 5.4.3 78

6 Anytime Learning via Forward Architecture Search 81
6.1 Introduction . . . 81
6.2 Background and References . . . 83
6.3 Neural Architecture Search as Optimization . . . 84
6.3.1 Connection to Feature Selection . . . 84
6.4 A NAS Approach from Gradient Boosting . . . 85
6.4.1 Gradient Boosting . . . 85
6.4.2 Gradient-Boosting-Inspired NAS . . . 85
6.4.3 Search Space . . . 86
6.4.4 Joint Weak Learning . . . 88
6.4.5 Weak Learner Finalization . . . 90
6.4.6 Utilizing Parallel Workers . . . 91
6.5 Selected Empirical Highlights . . . 92
6.5.1 Search Results on CIFAR10 . . . 92
6.5.2 Transfer to ImageNet . . . 92
6.5.3 Search Space: Direct versus Proxy . . . 94
6.5.4 Weak Learner Space: Weighted Sum versus Concatenation-Projection . . 95
6.5.5 Weak Learner Space: Number of Merged Operations . . . 95
6.5.6 Weaker Learner Training: Joint versus Isolated training with Parent Model 96
6.6 Discussion . . . 97
6.7 Conclusion . . . 97

7 Discussion and Conclusion 99
7.1 Discussion and Future Works . . . 99
7.1.1 Dynamic Models with Data-Dependent Computational Graphs . . . 99
7.1.2 Game Theoretical Approach to Training Anytime Predictors . . . 99
7.1.3 Determine When to Grow Models in Anytime Learning . . . 100
7.2 Conclusion . . . 100

List of Figures

3.1 Doubling Algorithm (b) has better anytime behaviors than greedy algorithm with no cost constraints (a). 20
3.2 The training time vs. the number of feature groups selected with two algorithms: CS-G-FR and CS-G-OMP. CS-G-OMP achieves a 8x and 20x overall training time speed-up on AGRICULTURAL and YAHOO! LTR. 23
3.3 (a) Explained Variance vs. Cost curve of CS-G-OMP in YAHOO! LTR. Vertical lines mark different α-stopping costs. (b) Explained Variance vs. Cost curve of CS-G-OMP and G-OMP on YAHOO! LTR set 1 with individual group size $s = 10$, stopped at 0.97-stop cost. 23
3.4 Explained Variance vs. Feature Cost curves on AGRICULTURAL (a) and YAHOO! LTR (b) comparing group whitening with no group whitening. The curves stop at 0.97-stopping cost. 25
3.5 (a),(b): Explained Variance vs. Feature Cost curves on AGRICULTURAL and YAHOO! LTR(group-size=10), using CS-G-OMP, CS-G-FR and their Single variants. Curves stop at 0.97 and 0.98 stopping costs. (c),(d): Same curve with the natural objectives of the data-sets: accuracy and NDCG@5. 26
3.6 CS-G-OMP test-time performance on MNIST. We note that CS-G-FR cannot be computed easily in this case and is omitted. 33

4.1 (a) The common ANN training strategy increases final errors from the optimal (green vs. blue), which decreases exponentially slowly. By learning to focus more on the final auxiliary losses, the proposed adaptive loss weights make a small ANN (orange) to outperform a large one (green) that has non-adaptive weights. (b) Anytime neural networks contain auxiliary predictions and losses, $\hat{y}_i$ and ℓ_i, for intermediate feature unit f_i. 36
4.2 (a) Relative Percentage Increase in Training Loss vs. depths (lower is better). CONST scheme is increasingly worse than the optimal at deep layers. AdaLoss performs about equally well on all layers in comparison to the OPT. (b)Ensemble of exponentially deepening anytime neural network (EANN) computes its ANNs in order of their depths. An anytime result is used if it is better than all previous ones on a validation set (layers in light blue). 39
4.3 Comparing small networks with AdaLoss versus big ones using CONST on CIFAR10 and CIFAR100. 43
4.4 Comparing small networks with AdaLoss versus big ones using CONST on SVHN. 43

4.5 Comparing small networks with AdaLoss versus big ones using CONST on ILSVRC with ResANNs and MSDNet. 44
4.6 ANNs performance are mostly decided by underlying models, but AdaLoss is beneficial regardless models. 45
4.7 **(a)** EANN performs better if the ANNs use AdaLoss instead of CONST. **(b)** EANN outperforms linear ensembles of DNNs on ILSVRC. **(c)** The learned adaptive weights of the same model on three data-sets. 46

5.1 Average regret of SGB with regression trees with various depths on SLICE and A9A datasets. 63
5.2 Log-log plots of test-time loss vs. computation complexity on various datasets. The x-axis represents computation complexity measured by number of weak leaner predictions; the y-axis measures square loss for regression tasks (ABALONE, SLICE and YEAR), and classification error for A9A and MNIST. . 64

6.1 (a) Cell-search applies found cells to a predefined outer structure. (b) Macro-search allows any connection. 86
6.2 An example weak learner x_c from the search space $\mathcal{H}_k$. 87
6.3 Training of a weak learner x_c, so that it can (a) and cannot (b) affect the current model. 89
6.4 Weighted sum is replaced with concat-projection, when the top operations are chosen. Any sf or sg are also removed. 91
6.5 The performance convex hull of the found models by Petridish on ILSVRC. Petridish models are of parameter $N = 6$ and $F = 44$. 95

List of Tables

3.1 Test time 0.97-Timeliness measurement of different methods on AGRICULTURAL. We break the methods into OMP, FR and Oracle family: e.g., “Single” in the G-CS-OMP family means G-CS-OMP-Single, and “FR” in the Oracle family means the oracle curve derived from G-FR. 22
3.2 Test time 0.99-Timeliness measurement of different methods on YAHOO! LTR. . 22

4.1 Average relative percentage increase in error from the OPT on CIFAR and SVHN at 1/4, 1/2, 3/4 and 1 of the total cost. E.g., the bottom right entry means that if OPT has a 10% final error rate, then AdaLoss has about 10.27%. 42
4.2 Test error rates at different fraction of the total costs on ResANN50, DenseANN169, and MSDNet38 on ILSVRC. The post-fix +C and +A stand for CONST and AdaLoss respectively. Published results of MSDNet38 Huang et al. (2018b) uses CONST. 42
4.3 Relative percentage increase in error rate by switching from $\gamma = 0$. (lower is better.) A small amount of $\gamma = 0.5$ drastically improves early predictions without increasing late error rate much. 50
4.4 Relative percentage increase in error rate by switching from $m = 0.9$. (lower is better.) The two options essentially result in the same error rates. 50
4.5 Relative percentage increase in error rate by switching from $e = 0$. (lower is better.) The options are essentially the same on CIFAR10 and CIFAR100. 50

5.1 Average test-time loss: square error for regression, and error rate for classification. 66

6.1 Comparison against state-of-the-art recognition results on CIFAR-10. Results marked with † are not trained with cutout. The first block represents approaches for macro-search. The second block represents approaches for cell-search. . . . 93
6.2 ILSVRC2012 transfer results. Petridish uses Isolated and the concat-projection (CP) modification by default. 94
6.3 Search space comparison between the direct space of $N = 6$ and $F = 32$ and the proxy space of $N = 3$ and $F = 16$ by evaluating their best mobile setting models on ILSVRC. 95
6.4 ILSVRC2012 transfer results. Ablation study on the choice of weighted-sum (WS), concat-projection at the end (CP-end), or the Petridish default merge operation in finalized weak learners. The searches were done with parameter initial channel $F = 32$ and s number of regular cells per resolution of $N = 6$. . . 96

6.5 Test error rates on CIFAR-10 by models found with different weak learner complexities. 96
6.6 ILSVRC2012 transfer results. Ablation study on the choice of Joint and Isolated for training the weak learners. The search were with parameter initial channel $F = 32$ and number of regular cell per resolution $N = 6$. 97

Chapter 1

Introduction

When evaluating a predictor for an application, one often needs to consider two critical aspects of algorithms: the accuracy of the prediction, and the computational cost of the predictor. These two factors are often opposed to each other: practitioners typically have to choose between predictors that are accurate but slow and ones that are fast but inaccurate. This trade-off between computational cost and accuracy is inherently difficult to manage, and is the focus of this work.

1.1 Motivations and Problem Settings

Machine learning algorithms typically have computational budget limits during test-time. For applications on mobile devices and Internet of Things (IoTs), it is critical for the predictors to fit in these devices with low computational power and consume little computation during test-time. For robotic applications such as autonomous vehicle or drones, it is paramount to have low latency in visual detection for planning maneuvers. Web services such as Email spam filters also require low latency to maintain user satisfaction. For such applications, one often cannot deploy predictors that achieve the state-of-the-art accuracy, because those predictors often are associated with high computational costs. Instead, these applications often seek the most accurate models that are within their computational budgets.

Furthermore, the computational budget limits of many applications can also vary during test-time, or can be agnostic during training. For instance, robotic applications have varying test-time budget limits based on the speed of the robot and the complexity of the environments. Web servers may handle heavier query traffic during the day than during the night, but they are expected to maintain the same low latency. Mobile and low-computation devices may want a low-power mode in case of low battery. Hence, it is beneficial to consider the setting where we seek the most accurate models at each possible budget limit of the applications. We draw special attention to the fact that under this setting, we delay the decision of the budget limits to the test-time, allowing greater flexibility in the algorithms. Furthermore, when the budget limit becomes known, one can extract from the spectrum of cost-efficient models. Alternatively, if one wants to minimize the average test-time computational costs of the prediction given a target accuracy level, such as in budgeted prediction (Bolukbasi et al., 2017; Guan et al., 2017), one can combine the spectrum of models with early-stopping policies in order to reduce computation on clear decisions and prolong computation on ambiguous ones.

While the above examples and problem settings focus on the balance between model accuracy and test-time computation, practitioners may often be concerned with their train-time computational costs. One reason for this concern is that the data-sets are becoming larger and are continuously updated due to improved data storage and collection. Specifically, fields such as finance, information retrieval, computer vision, text and vocal language processing have accumulated data from decades of practice. Training state-of-the-art models against the decades worth of data becomes increasingly challenging. Hence, it is crucial for modern machine learning models to be able to handle large data-sets that may not be present on the same machine or at even the same time. Furthermore, the models ideally should be able to be improved as more data becomes available or more train-time computational resource is allocated.

Another key reason for the rising train-time computation is the increased model complexity. As neural networks become the dominant methods for computer vision tasks and natural language processing, practitioners often rely on experts to find optimal network architectures via trial-and-error. However, such experimental process can be vastly expensive, taking thousands of GPU-days (Zoph and Le, 2017). Facing such complex and combinatorial hyper-parameter space of model architectures, we need guidance to search in a cost-efficient manner. In particular, practitioners may be interested in increasing the model complexity gradually: one starts with small architectures that can be trained and deployed easily; then as more train-time computation is allowed, the model complexity is gradually increased in a guided manner. Furthermore, ideally one would like the models to reusable and stable, so that new models can utilize previous found models and do not deviate from previous models too much to affect user experience.

In summary, the targeted problem settings of this work can be partitioned into two parts. The first focuses on the problem of finding accurate predictions at each possible test-time computational budgets, and the second focuses on enabling the previous solutions to handle the computational cost in training due to increased data-set sizes and increased complexity in model hyper-parameters.

1.2 Approach

For each of test-time and train-time trade-off between computation and accuracy, we develop both algorithms with theoretical performance guarantees and algorithms that work well on real-world data-sets. We summarize the main approaches that we take as follows.

One approach to have accurate predictions at each possible test-time computational budget limit is to first produce crude predictions early, and then continuously improve them. Such algorithms are called *anytime* algorithms, and they automatically adjust to the varying or agnostic test-time budget limits because the algorithm utilize the computational resources until the budget limit. One common approach to achieve anytime prediction is through ensembles of weak predictors (Brubaker et al., 2008; Cai et al., 2015; Grubb and Bagnell, 2012b; Lefakis and Fleuret, 2010; Reyzin, 2011; Sochman and Matas, 2005; Xu et al., 2014), so that at test-time, one computes the predictors iteratively and then reports the partial ensemble result as the intermediate or anytime results. Indeed, the first anytime predictor of this work follows this idea of combining weak predictors in a generalized linear prediction setting, and utilizes submodular optimization results (Das and Kempe, 2011) to bound the performance of the predictions. Furthermore, we prove a limitation of any anytime algorithm that stems from an ensemble of weak predictors, proving that in general such an ensemble of cost B can only achieve comparable rewards of the optimal ensemble of cost $B/4$.

Such limitations lead us to consider an alternative approach to anytime prediction, where we train a single model to produce multiple intermediate results for anytime predictions, and we optimize all the anytime results jointly. Viewing anytime prediction as a multi-objective optimization, we develop an adaptive method to balance the weights of the objectives, and improve anytime prediction quality on multiple neural networks and data-sets. By exploiting anytime neural networks as weak learners, we can form ensembles of anytime predictors for anytime prediction, and interestingly, by making weak learners to be anytime predictors, we can circumvent the previous theoretical limitation on ensemble-based anytime predictors.

For the train-time trade-off between computation and accuracy, we specifically target problems that are often accompanied by our anytime predictors. Since anytime models often stem from ensembles of weaker models that are trained sequentially, they can be difficult to scale to large data-sets, especially on data-sets that may be expensive to loop through. To address this weakness, we develop gradient boosting algorithms for stochastic data streams so that we can train all weak learners jointly. Gradient boosting can be considered as approximated gradient descent in the functional space, and can be analyzed via gradient descent (Grubb and Bagnell, 2011; Hazan et al., 2007). Combining gradient boosting with analysis from online learning (Beygelzimer et al., 2015b; Cesa-Bianchi et al., 2004), we analyze the proposed algorithms under stochastic data streams. We bound the regrets of the proposed algorithm, showing that it achieves no regret in prediction losses against any competitor under strongly convex losses and under the assumption that each weak online learner can predict better than random by a margin.

We also address the increased complexity of hyper parameters in the specific setting of neural architecture search (Elsken et al., 2018b; Zoph and Le, 2017), where one seeks the optimal architecture given a data-set and an optimization objective. We first formulate the problem as a bi-level optimization and show its connections to the earlier anytime linear prediction problem. Inspired by forward selection approach in the linear prediction setting, we propose to expand existing neural networks models iteratively guided by gradient boosting. Each step of the model expansion is determined by fitting potential short-cut connections against the gradient of the loss with respect to intermediate layers.

In summary, the thesis statement of this work is as follows.

> **Thesis Statement:** Modern machine learning applications often have to address the trade-off between computational cost and predictive power. This work addresses the trade-off between computational speed and prediction accuracy at both test-time and training-time, providing theoretical performance guarantees and practical experimental results. Specifically, for dynamically trading speed and accuracy at test-time, we leverage cost-greedy methods to achieve near-optimal anytime linear predictions, and we also derive an anytime performance upper bound on such ensemble-based methods in general. However, utilizing multi-objective optimization, we show that this upper bound can be avoided via ensemble of anytime weak learners. To address the rising problem of training computation, we propose to adapt ensemble meth-

ods to stochastic data-streams. Furthermore, we draw connection between anytime prediction and neural architecture search, and develop practical algorithms to expand network architectures iteratively in order to explore the vast space of networks efficiently during training.

1.3 Overview of Chapters and Their Contributions

Chapter 3 covers the anytime linear prediction under the setting where features are computed in groups and feature groups have costs. Under this setting, we learn an ordering of the features, in which the features should be computed at the test-time. Whenever a new feature becomes available at test-time, we update the latest linear prediction. In Section 3.2, we extend feature selection methods Forward Regression (FR) and Orthogonal Matching Pursuit (OMP) to handle feature groups that have costs. We then provide a theoretical analysis of these two cost-greedy algorithms in Section 3.3, utilizing spectral analysis and sub-modular optimization. We first prove that both algorithms achieve near-optimal linear predictions in terms of explained variance, at test-time budgets where the algorithms just finish computing new feature groups. Then we show that these bounds are inadequate for bounding performance for all test-time budgets. Instead we propose a simple modification, called doubling, to the previous cost-greedy procedure in Section 3.4, and provide theoretically analysis that the modified algorithms is near-optimal at any test-time budget B in comparison to the optimal at budget $B/4$ in Theorem 3.4.3. We further show that the constant $B/4$ is tight in Theorem 3.4.1, which shows that it is impossible in general for anytime algorithms via ensembles to compare against the optimal of a budget that is more than $B/4$. The contribution of this chapter is summarized as follows.

1. We cast the problem of anytime linear prediction as a feature group sequencing problem and propose extensions to Forward Regression (FR) and Orthogonal Matching Pursuit (OMP) under the setting where features are in groups that have costs.
2. We theoretically analyze our extensions to FR and OMP and show that they both achieve $(1 - e^{-\lambda^*})$ near-optimal explained variance with linear predictions at budgets when they choose feature groups, where λ^* is a constant related to how correlated the features groups are to each other.
3. We develop the first anytime algorithm with provable performance guarantees at *any* budget limit B, by relating the prediction performance to that of the optimal of cost $B/4$.
4. We further show that the fraction $1/4$ is tight, as in that it is impossible to achieve multiplicative bounds of the prediction performance at B against any optimal of cost greater than $B/4$.
5. The previous pair of theoretical results present a tight bound on anytime predictions based on ensemble of weak predictors.

As Chapter 3 seals the fate of anytime predictors via ensembles of weak learners, we move on in Chapter 4 to develop anytime predictors within neural networks. We pose the training of anytime predictors within a single network as a multi-objective optimization, and propose to balance the weights of the anytime objectives adaptively in a weighted sum in Section 4.3. The adaptive weight balancing intuitively normalizes the losses so that they have the same scale. This simple modification can be derived from three theoretical considerations, including maximum likelihood models, optimization with log-barriers, and optimization of geometric mean of the expected anytime losses. We also show experimentally in Section 4.5 that the proposed weight balancing leads to better anytime predictions within the networks across multiple architectures and data-sets. The anytime neural networks also allow us to revisit the limitation of anytime predictors via ensembles. In fact, we show in Section 4.4 that an ensemble of anytime neural networks can circumvent the previous hard example, so that the performance at a test-time budget B is comparable to the optimal at budget B/C, where the constant C can be smaller than 4. This suggests that future anytime predictors should combine weak learners that are anytime predictors on their own, instead of regular weak predictors. The contribution of this chapter is summarized as follows.

6. For training anytime predictions within neural networks, we derive an adaptive weight scheme for anytime losses from multiple theoretical considerations, and show that experimentally this scheme achieves near-optimal final accuracy *and* competitive anytime ones on multiple data-sets and models.
7. We assemble anytime neural networks of exponentially increasing depths to achieve near-optimal anytime predictions at every budget at the cost of a constant fraction of additional consumed budget, under the assumption that each anytime neural network is near-optimal in its later fraction of depths. We verify that this assumption holds practically in current state-of-the-art networks.
8. The near-optimal guarantee of ensemble of anytime neural networks breaks the earlier hardness result of anytime predictors from ensemble of regular predictors by increasing the constant $1/4$ to $1/2.91$.

Starting from Chapter 5, we switch the topic from test-time balance of computation and accuracy to the training-time cost-effectiveness. Chapter 5 covers how to train an ensemble for gradient boosting given a stochastic stream of data samples. Gradient boosting is a common way to form ensemble of weak learners, and each weak learner is trained to match the functional gradient of the loss with respect to the current predictor. We set up these preliminary details in Section 5.3. Such boosting can suffer on large data-sets, because it trains the weak learners sequentially and loop the data many passes. To address this weakness of gradient boosting, we propose a modification to handle stochastic data streams in Section 5.5, so that all weak learners are online learners and are trained and optimized jointly. Combining theoretical analysis of convex optimization for gradient boosting and that of online learning for handling stochastic streams, we prove in Theorem 5.5.1 and Theorem 5.5.2 that the proposed algorithms can achieve no regret against any competitor under convex losses and under the assumptions that the weak online learners are better than random by a margin. The contribution of this chapter is summarized as follows.

9. Assuming a non-trivial edge can be achieved by each deployed weak online learner, we develop gradient boosting algorithms to handle smooth or non-smooth loss functions on stochastic data streams.
10. The theoretical analysis show that under the smooth losses, the proposed algorithms achieves

exponential decay on the average regret with respect to the number of weak learners.

11. Under non-smooth but strongly convex losses, we show that the proposed streaming gradient boosting can instead achieve $O(\ln N/N)$ average regret with respect to the number of weak learners N.

Chapter 6 considers on the search through the hyper-parameter space, and focuses on the problem of neural architecture search. Traditionally, practitioners tune their architecture via trial and error, and it can take massive computational resources. Recent works have automated this procedure via reinforcement learning and evolutionary algorithms, but the training computational cost is still demanding. In Section 6.3, we draw a connection from the architecture search to learning anytime predictions with ensemble methods, showing that they both solve a bi-level optimization problem where the outer level searches for the discrete choice of architecture or weak learners, and the inner level optimizes the parameters of architecture or the weak learners. In Section 6.4, we develop a greedy search procedure that adds shortcut connections to existing network architectures iteratively. The added connections are chosen by matching candidate connections to the gradient of the loss with respect to intermediate layers, similar to gradient boosting of weak learners. To estimate the gradients efficiently, we initialize a large number of potential shortcut connections and train them jointly, and we utilize feature selection to keep only the most important ones. We show experimentally in Section 6.5 that such greedy procedure can find cost-efficient models that are at the state-of-the-art level. The contribution of this chapter is summarized as follows.

12. We propose an approach to increase complexity of neural networks iteratively during training. We alternate between two phases. The first expands the model with potential shortcut connections and train them jointly. The second phase trims the previous potential connections using feature selection and continue training the model.

13. The proposed approach can be applied to both improve a small repeatable pattern, called cell, and improve the macro network architecture directly, unlike most popular approaches that only focus on cells. This opens up neural architecture search to fields where no domain knowledge of the macro structure exists.

14. On cell-search, the proposed method finds a model that achieves 2.61% error rate on CIFAR10 using 2.9M parameters within 5 GPU-days.

15. On macro-search, the proposed method finds a model that achieves 2.83% error rate on CIFAR10 using 2.2M parameters within 5 GPU-days.

16. The proposed approach can warm start from existing networks, leveraging previous training results. Furthermore, it directly expands models on the lower convex hull of error rate vs. test-time computation, and is hence able to naturally produce a gallery of cost-effective models for applications to choose.

Chapter 2

Preliminaries and Background

2.1 Anytime Prediction

We formally introduce anytime prediction in this section, since most of this work is based on this idea. Anytime predictors output valid results if they are interrupted at any point during testing. Furthermore, the results improve with more resources spent. Such idea of partial computation is exploited by many algorithms, not just for prediction. For instance, bisection method for finding square root of a real number is an example where the longer the computation, the more precise the approximation becomes.

Formally, we consider anytime prediction as a multi-objective optimization problem. An anytime predictor $\hat{y}$ takes an input x, and produces a sequence of partial results until an agnostic interruption happens. Let the parameters of the predictor be θ We denote $\hat{y}_t(x;\theta)$ to be the the latest prediction at computational budget limit $t \in \mathbb{R}$. Let y be the target prediction, and the loss function be ℓ. Then the predictor at time t suffers the expected loss $\ell_t(\theta) := \mathbb{E}_{x,y\sim D}\ell(\hat{y}_t(x;\theta), y)$, where D is the stochastic distribution of the data. Then an ideal anytime predictor simultaneously optimizes the expectation ℓ_t for all $t \in \mathbb{R}$, i.e., finding the optimal θ^* that is simultaneously optimal for all budgets t,

$$\theta^* \in \cap_{t\in\mathbb{R}}\{\theta' : \theta' = \arg\min_{\theta} \ell_t(\theta)\}. \tag{2.1}$$

The multi-objective optimization in Eq. 2.1 often cannot be solved, because not only there are infinitely many objectives ℓ_t, but also these objectives are in general in conflict with each other. Hence, varies approximation have to be made for optimizing for Eq. 2.1. One common approximation is to only consider ℓ_t if a new prediction becomes available at t, i.e.,

$$\theta^* \in \cap_{t\in A}\{\theta' : \theta' = \arg\min_{\theta} \ell_t(\theta)\}, \tag{2.2}$$

where A is the set of time where $\hat{y}$ makes new predictions. This is often used in practice, because we often know roughly which t to focus on and design the predictor to output at those locations specifically. However, by only focusing on the budgets where new predictions are made, this approximation can overestimate its performance at other time budgets. An extreme example is to focus only on the final prediction and produce no early results, i.e., a non-anytime predictor.

In fact, in both Chapter 3 and Chapter 4 we apply this approximation first, and then convert the solutions for the general budgets $t \in \mathbb{R}$.

The multi-objective minimization in Eq. 2.2 is also more special than general multi-objective problems, since the predictions happen in the order of computation. Hence, beside typical multi-objective approaches such as weighted sum and game-theoretical min-max optimization, one can instead add complexity to the anytime predictor iteratively, and each addition triggers a new prediction. This approach is appealing and is often used in anytime prediction literature, because it replaces the difficult multi-objective problem with an iterative optimization problem. Furthermore, the theoretical analysis on the iterations can often be translated to performance at all budgets at which the predictions are made.

2.2 Related Works to the Trade-off Between Computation and Accuracy

There are a wide array of works that address the trade-off between computation and accuracy. Here we provide a brief summary of these approaches to establish a background for this work.

2.2.1 Anytime Prediction

There are many ways to generate anytime predictions within predictors. Some predictors innately have structures or procedures that can provide anytime predictions. For instance, a decision tree can naturally provide exit the prediction at any depth. In stacked recurrent models or iterative inference procedures, one can stop early without finishing all iterations. In feed-forward neural networks, auxiliary predictors that leverage early feature layers can be trained to produce early predictions. In fact, as deep neural networks (DNNs) have become the backbone of many modern machine learning applications, many works have studied DNNs with auxiliary predictors. Larsson et al. (2017a); Lee et al. (2015); Szegedy et al. (2017); Zhao et al. (2017) use auxiliary prediction to regularize the networks for faster and better convergence. Bengio et al. (2009); Zamir et al. (2017) set the auxiliary predictions from easy to hard for curriculum learning. Chen and Koltun (2017); Xie and Tu (2015) make pixel level predictions in images, and find learning early predictions in coarse scales also improve the fine resolution predictions. Huang et al. (2018b) shows the crucial importance of maintaining multi-scale features for high quality early classifications.

Anytime predictors can also be built iteratively from weak predictors. In (Weinberger et al., 2009; Xu et al., 2012; 2013a), feature manipulations such as polynomials on the existing features are iteratively tried and selected to add to the overall linear predictor. (Reyzin, 2011) train a boosted ensemble of weak learners, and then at test-time, sample the weak learners to run according to their weights in the ensemble. (Grubb and Bagnell, 2012b) adjust gradient boosting to account for computational costs of weak learners during weak learner selection, and compute the weak learners sequentially during test-time to update the outputs.

2.2.2 Model Compression

A wide range of works improve the trade-off between computation and accuracy by compressing the model.

The most rudimentary form of model compression is perhaps the feature selection in linear prediction, where one seeks the most important features of the model. There are two typical approaches to feature selection, sparse optimization and iterative selection (or elimination). In sparse optimization, one optimizes the completely model while having some constraints or regularization to induce sparsity in the selected features. $L1$-regularization, or Lasso (Tibshirani, 1994) is typically used for selecting individual feature dimensions. When there are feature groups, where grouped features are computed together, Group Lasso regularization (Yuan and Lin, 2006) is often used. The most common approaches to iterative approach is through greedy algorithms, which are classified by their greedy criteria. In particular, forward regression enumerates all possible selections and compute the marginal change in the objectives. Alternatively, Orthogonal Matching Pursuit (Pati et al., 1993) and Least-angle Regression(LARS) (Efron et al., 2004) can be considered as approximation to forward regression via gradient boosting: a feature is selected, if it is to best to represent the gradient of the loss with respect to the prediction.

Neural network compression has become a common problem due to the growing network sizes and the limited GPU memory. Huang et al. (2017a); Li et al. (2017); Liu et al. (2017b) prune network weights and connections based on their magnitudes. Hubara et al. (2016); Iandola et al. (2016); Rastegari et al. (2016) quantize weights within networks to reduce computation and memory footprint. A closely related topic is knowledge distillation Ba and Caruana (2014); Hinton et al. (2014), where the training target of the target network is replaced with the predicted logits of the source network.

2.2.3 Budgeted Prediction

We note that anytime prediction is related to but different from budgeted prediction, which aims to minimize **average test-time computational cost** without sacrificing average accuracy. Specifically, in anytime prediction, the budget t determines the computational cost for all samples x, whereas in budgeted prediction, the predictor has the freedom to choose when to exit for each sample x, provided the expected prediction accuracy meets some threshold, and the expected computational cost becomes a minimization objective. As a result, a budgeted predictor may not have early predictions for a particular data sample, and the predictor can also exit early on the sample, so that the result on the sample is not improved if more computational budget is given. At the same time, an anytime predictor tries to optimize the result on the sample at multiple budget limits, and this may leads to worse accuracy at a specific budget limit. Hence, we consider the two problems orthogonal.

A typical approach to budgeted prediction is through cascaded predictors (Brubaker et al., 2008; Cai et al., 2015; Lefakis and Fleuret, 2010; Sochman and Matas, 2005; Viola and Jones, 2001b; Xu et al., 2014), where a sequence of predictions are trained along side with a policy that determines the exit point of each sample on the sequence. As a result, data samples with easy decisions take early-exits, while the difficult decisions can take longer computation. Overall, this often results in a reduction in computation at a small increase of error rates. Cascaded prediction can be also considered as a combination between anytime predictors and the early-exit policy.

Cascaded predictors and budgeted prediction has also been applied to neural networks. Bolukbasi et al. (2017); Veit and Belongie (2017); Wang et al. (2017) dynamically skip network computation based on samples, and the early-exit policies are trained through reinforcement learning or iterative optimization.

Chapter 3

Anytime Linear Prediction via Feature Group Sequencing

3.1 Introduction

In this work, we consider anytime predictions under the common machine learning setting, where features are computed in groups with associated costs. We further assume that the cost of prediction is dominated by feature computation. Hence, we can achieve anytime predictions by computing feature groups in a specific order and outputting linear predictions using only computed features at interruption.

Formally, we are given n samples (x^i, y^i) from a feature matrix $X \in \mathbb{R}^{n \times D}$ and a response vector $Y \in \mathbb{R}^n$. We also have a partition of the D feature dimensions into J feature groups, $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_J$, and an associated cost of each group $c(\mathcal{G}_j)$. Our anytime prediction approach learns a sequencing of the feature groups, $G = g_1, g_2, ..., g_J$. For each budget limit B, the computed groups at cost B is a prefix of the sequencing, $G_{\langle B \rangle} = g_1, g_2, .., g_{J_{\langle B \rangle}}$, where $J_{\langle B \rangle} = \max\{j \leq J | \sum_{i \leq j} c(g_i) \leq B\}$ indexes the last group within the budget B. An ideal anytime algorithm seeks a sequencing G to minimize risk at all budgets B:

$$R(G_{\langle B \rangle}) := \min_w \frac{1}{2n} \|Y - X_{G_{\langle B \rangle}} w\|_2^2 + \frac{\lambda}{2} \|w\|_2^2, \tag{3.1}$$

where $X_{G_{\langle B \rangle}}$ contains features in $G_{\langle B \rangle}$, w is the associated linear predictor coefficient, and λ is a regularizing constant. Equivalently, if we assume that the y^i's have unit variance and zero mean by normalization, we can maximize the explained variance,

$$F(G_{\langle B \rangle}) := \frac{1}{2n} Y^T Y - R(G_{\langle B \rangle}) \tag{3.2}$$

$$= \frac{1}{2n} Y^T Y - \min_w \left(\frac{1}{2n} \|Y - X_{G_{\langle B \rangle}} w\|_2^2 + \frac{\lambda}{2} \|w\|_2^2 \right) \tag{3.3}$$

The above optimization problem is closest to the problem of subset selection for regression (Das and Kempe, 2011), which selects at most k features to optimize a linear regression. The problem is also similar to that of sparse model recovery (Tibshirani, 1994), which recovers coefficients of a true linear model. One common approach to these two problems is to select the features greedily via Forward Regression (FR) (Miller, 1984) or Orthogonal Matching Pursuit (OMP) (Pati et al., 1993). Forward Regression greedily selects features that maximize the marginal increase in explained variance at each step. Orthogonal Matching Pursuit selects features as follows. The linear model coefficients of the unselected features are set to zero. At each step, the feature whose model coefficient has the largest gradient of the risk is selected. In this work, we extend FR and OMP to the setting where features are in groups that have costs. The extension to FR is intuitive: we only need to select feature groups using their marginal gain in objective per unit cost instead of using just the marginal gain. However, we have two notes about the extension to OMP. First, to incorporate feature costs, we need to evaluate a feature based on the squared norm of the associated weight vector gradient per unit cost instead of just the gradient norm. Second, when we compute the gradient norm for a feature group, ∇_g, we have to use the norm $\nabla_g^T(X_g^TX_g)^{-1}\nabla_g$, which is $\|\nabla_g\|_2^2$ if and only if each feature group g is whitened, which is an assumption in group OMP analysis by Lozano et al. (2009; 2011). Our analysis sheds light on why this assumption is important in a group setting. Like previous analyses of greedy algorithms by Streeter and Golovin (2008), our analysis guarantees that our methods produce near-optimal linear predictions, measured by explained variance, at budgets where feature groups are selected. Thus, they exhibit the desired anytime behavior at those budgets. Finally, we extend our algorithm to account for *all* budgets and show a novel anytime result: for any budget B, if *OPT* is the optimal explained variance of cost B, then our proposed sequencing can approximate within a factor of *OPT* with cost at most $4B$. Furthermore, with a cost less than $4B$, a fixed sequence of predictors cannot approximate *OPT* in general. To our knowledge, these are the first anytime performance bounds at all budgets.

In previous works, both FR and OMP are theoretically analyzed for both the problem of subset selection and model recovery. Das and Kempe (2011) cast the subset selection problem as a submodular maximization that selects a set S with $|S| \leq k$ to maximize the explained variance and prove that FR and OMP achieve $(1 - e^{-\lambda^*})$ and $(1 - e^{-\lambda^{*2}})$ near-optimal explained variance, where λ^* is the minimum eigenvalue of the sample covariance, $\frac{1}{n}X^TX$. We can adopt these previous analyses to our extensions to FR and OMP under the group setting with costs and produce the same near-optimal results. We also present a novel analysis of OMP that leads to the same near-optimal factor $(1 - e^{-\lambda^*})$ as that of FR. Works on model recovery have also analyzed FR and OMP. Zhang (2009) proves that OMP discovers the true linear model coefficients, if they exist. This result was then extended by (Lozano et al., 2009; 2011) to the setting of feature groups using generalized linear models. However, we note that these theoretical analyses of model recovery assume that a true model exists. They focus on recovering model coefficients rather than directly analyzing prediction performance.

Besides greedy selection, another family of approaches to find the optimal subset S that minimizes $R(S)$ is to relax the NP-hard selection problem as a convex optimization. Lasso (Tibshirani, 1994), a well-known method, uses L_1 regularization to force sparsity in the linear model. To get an ordering of the features, compute the Lasso solution path by varying the L_1 regularization constant. Group Lasso (Yuan and Lin, 2006) extends Lasso to the group setting, replacing the L_1 norm with the sum of L_2 norms of feature groups. Group Lasso can also incorporate feature costs by scaling the L_2 norms of feature groups. Lasso-based methods are generally analyzed for model recovery, not prediction performance. We demonstrate experimentally that our greedy methods achieve better prediction performance than cost-weighted Group Lasso.

Various works have addressed anytime prediction previously. The most well-known family of approaches use *cascades* (Viola and Jones, 2001b), which achieve anytime prediction by filtering out samples with a sequence of classifiers of increasing complexity and feature costs. At each stage, cascade methods (Brubaker et al., 2008; Cai et al., 2015; Lefakis and Fleuret, 2010; Sochman and Matas, 2005; Xu et al., 2014) typically achieve a target accuracy and assign a portion of samples with their final predictions. While this design frees up computation for the more difficult samples, it prevents recovery from early mistakes. Most cascade methods select features of each stage before being trained. Although the more recent works start to learn feature sequencing, the learned sequences are the same as those of cost-weighted Group Lasso (Chen et al., 2012a) and greedy methods (Cai et al., 2015) when they are restricted to linear prediction. Hence our study of anytime linear prediction can help cascade methods choose features and learn cascades. Another branch of anytime prediction methods uses boosting. It outputs as results partial sums of the ensemble (Grubb and Bagnell, 2012b) or averages of randomly sampled weak learners (Reyzin, 2011). Our greedy methods can be viewed as a gradient boosting scheme by treating each feature as a weak learner. Some works approach anytime prediction with feature transformations (Xu et al., 2012; 2013a) and learn computation-aware, non-linear transformation of features for linear classification. Similarly, Weinberger et al. (2009) hashes high dimensional features to low dimensional subspaces. These approaches operate on readily-computed features, which is orthogonal to our problem setting. Karayev et al. (2012) models the anytime prediction as a Markov Decision Process and learns a policy of applying intermediate learners and computing features through reinforcement learning.

Contributions

- We cast the problem of anytime linear prediction as a feature group sequencing problem and propose extensions to FR and OMP under the setting where features are in groups that have costs.

- We theoretically analyze our extensions to FR and OMP and show that they both achieve $(1 - e^{-\lambda^*})$ near-optimal explained variance with linear predictions at budgets when they choose feature groups.

- We develop the first anytime algorithm that provably approximates the optimal performance of *all* budgets B with cost of $4B$; we also prove it impossible to achieve a constant-factor approximation with cost less than $4B$.

3.2 Computation-Aware Greedy Methods

3.2.1 Preliminaries

Before introducing our greedy methods for forming cost-efficient anytime predictors, we first formally state our assumptions and define some terminology.

We assume that all feature dimensions and responses are normalized to have zero mean and unit variance, i.e., we assume each column of X has zero mean and unit variance. We also assume the feature group costs $c(g)$ dominates the costs of computing linear predictions using the features.

We define the regularized feature covariance matrix as $C := \frac{1}{n}X^TX + \lambda I_D$. Let C_{st} be the sub-matrix that selects rows from s and columns from t. Let C_S be short for C_{SS}. Given a non-empty union of selected feature groups S, the maximum explained variance $F(S)$ is achieved with the regularized optimal coefficient

$$w(S) = \frac{1}{n}(\frac{1}{n}X_S^TX_S + \lambda I)^{-1}(X_S^TY) \tag{3.4}$$

$$= \frac{1}{n}C_S^{-1}X_S^TY. \tag{3.5}$$

When we take gradient of $F(S)$ with respect to the coefficient of a feature group g, if $g \subseteq S$ then the gradient is

$$\nabla_g F(S) = \frac{1}{n}X_g^T(Y - X_S w(S)) - \lambda w(S)_g. \tag{3.6}$$

If $g \cap S = \emptyset$ then we can extend $w(S)$ to dimensions of g, setting $w(S)_g = 0$, and then take the gradient to have $\nabla_g F(S) = \frac{1}{n}X_g^T(Y - X_S w(S))$. In both cases, we have

$$\nabla_g F(S) = \frac{1}{n}X_g^TY - C_{gS}w(S). \tag{3.7}$$

We further shorten the notations by defining $b_g^S = \nabla_g F(S)$. If S is empty, we assume that coefficient $w(\emptyset)$ has zero for all features so that $F(\emptyset) = 0$. When $S = [s_1, s_2, ...,]$ is a sequence of feature groups, we define S_j to be the prefix sequence $[s_1, s_2, ..., s_j]$. We overload notations of a sequence S so that S also represents the set of features contained in the union of $s_1, s_2, ...,$ in notations such as $F(S)$, $w(S)$, C_S and b_S^S, where we need the selected features in S for evaluation and the ordering does not affect the computation.

3.2.2 Anytime Prediction at Test-time

Algorithm 1 describes anytime linear prediction at test-time. Given a learned ordering S for computation the features, the predictor compute them in order and update the linear prediction $\hat{Y}$ whenever a new feature becomes available. We can update predictions frequently, because we assume that the linear prediction computation is dominated by its feature computation. At interruption or termination of the feature computation, we report the latest linear prediction. Hence, to produce anytime linear predictions, we need to learn a sequencing of the features groups.

3.2.3 Computation-Aware Group Orthogonal Matching Pursuit(CS-G-OMP)

In Algorithm 2, we present Computation-Aware Group Orthogonal Matching Pursuit (CS-G-OMP), which learns a near-optimal sequencing of the feature groups for anytime linear predictions.

Algorithm 1 Anytime Linear Prediction at Test-time

```
1: Input: An ordering of features $S = s1, s2, ....$ The linear prediction weight $w(S_j)$ for each
   $j = 1, 2, ...,$. Input feature matrix $X$.
2: Output: Linear prediction on $X$.
3: Initialize $\hat{Y}_0 = \vec{0}$.
4: Initialize $\hat{Y} = \hat{Y}_0$.
5: for $j = 1, 2, ...$ do
6:     Compute feature group $s_j$.
7:     Compute predictions $\hat{Y}_j = Xw(S_j)$.
8:     Update $\hat{Y} = \hat{Y}_j$
9: end for
10: Return $\hat{Y}$.
```

The feature groups are selected greedily. At the j^{th} selection step $(*)$, we have chosen $j-1$ groups, $G_{j-1} = g_1, g_2, ..., g_{j-1}$, and have computed the best model using G_{j-1}, $w(G_{j-1})$. To evaluate a feature group g, we first compute the gradient $b_g = \nabla_g F(G_{j-1})$ of the explained variance F with respect to the coefficients of g. Then, we evaluate it with the whitened gradient L_2-norm square per unit cost, $\frac{b_g^T (X_g^T X_g)^{-1} b_g}{c(g)}$. We select the group g that maximizes this value as g_j, and continue until all groups are depleted.

Before providing performance guarantees with formal theoretical analysis of Algorithm 2 in Section 3.3, we first provide some intuition on why we introduce a group-whitening at line 8 in Algorithm 2. If there are no feature groups, OMP greedily selects features whose coefficients have the largest gradients of the objective function. In linear regression, the gradient for a feature g is the inner-product of X_g and the prediction residual $Y - \hat{Y}$. Hence OMP selects features that best reconstruct the residual. From this perspective, OMP under group setting should seek the feature group whose span contains the largest projection of the residual. Let the projection to feature group g be $P_g = X_g(X_g^T X_g)^{-1} X_g^T$ and recall projection matrices are idempotent. We observe that the criterion for CS-G-OMP selection step is $\frac{\|P_g(Y-\hat{Y})\|_2^2}{c(g)}$, i.e, a cost-weighted norm square of the projection of the residual onto a feature group. The name group whitening is chosen because the criterion is $\frac{\|b_g\|_2^2}{c(g)}$ if and only if feature groups are whitened. We assume *feature groups are whitened* in our formal analysis, and we will reflect on the theoretical effects of not group-whitening during the analysis.

Besides group-whitening, one may suggest other approaches to evaluate gradient vectors b_g for group g. For example, L_2 norm and L_∞ norm can be used to achieve greedy criteria $\frac{\|b_g\|_2^2}{c(g)}$ and $\frac{\|b_g\|_\infty^2}{c(g)}$, respectively. The former criterion forgoes group whitening, so we call it *no-whiten*. Thus, it overestimates a feature group that has correlated but effective features, an extreme example of which is a feature group of identical but effective features. The latter criterion evaluates only the best feature of each feature group, so we call it *single*. Thus, it underestimates a feature group that has a descriptive feature span but no top-performing individual feature dimensions. We will show in experiments that no-whiten and single are indeed inferior to our CS-G-OMP choice.

Algorithm 2 Cost Sensitive Group Orthogonal Matching Pursuit (CS-G-OMP)

```
1: Input: The normalized feature matrix $X \in \mathbb{R}^{n \times D}$. The normalized response vector $Y \in \mathbb{R}^n$,
   which has a zero mean and unit variance. Feature groups $\mathcal{G}_1, ...\mathcal{G}_J$ that partition $\{1, .., D\}$,
   and group costs $c(\mathcal{G}_j)$. Regularization constant $\lambda$.
2: Output: A sequence $G = g_1, g_2, ..., g_J$ of feature groups. For each $j \leq J$, a coefficient
   $w(G_j)$ for the features $G_j = g_1, ..., g_j$.
3: Set $G_0 = \emptyset$ to be an empty sequence.
4: Set $w(G_0) = \vec{0}$ to be a zero vector of zero length.
5: Compute $C = X^T X$.
6: for $j = 1, 2, ..., J$ do
7:     for $g \notin G_{j-1}$ do
8:         Compute $b_g = \nabla_g F(G_{j-1}) = \frac{1}{n} X_g^T (Y - X_{G_{j-1}} w(G_{j-1}))$.
9:     end for
10:    $g_j = \underset{g = \mathcal{G}_1, ..., \mathcal{G}_J, g \notin G_{j-1}}{\arg\max} \frac{b_g^T (X_g^T X_g)^{-1} b_g}{c(g)}$.
11:    Append $g_j$ to the sequence: $G_j = G_{j-1} \oplus g_j$.
12:    Compute $w(G_j) = \frac{1}{n} C_{G_j}^{-1} X_{G_j}^T Y$.
13: end for
```

3.2.4 Computation-Aware Group Forward Regression (CS-G-FR)

The learning procedure extending from Forward Regression is similar to Algorithm 2, as stated in Algorithm 3: we compute the linear models $w(G_{j-1} \oplus g)$ at line 4 instead of the gradients b_g and replace the selection criterion $\frac{b_g^T (X_g^T X_g)^{-1} b_g}{c(g)}$ at line 8 with the marginal gain in explained variance per unit cost, $\frac{F(G_{j-1} \oplus g) - F(G_{j-1})}{c(g)}$. We call this cost-sensitive FR extension as CS-G-FR.

3.3 Near-Optimality at Features Selection

This section proves that CS-G-FR and CS-G-OMP produce near-optimal explained variance F at budgets where features are selected. The main challenge of our analysis is to prove Lemma 3.3.1, which is a common stepping stone in submodular maximization analysis, e.g., Equation 8 in (Krause and Golovin, 2012). The main Theorem 3.3.2 follows from the lemma by standard techniques, which we defer to the Section 3.6.

Lemma 3.3.1 (main). *Let G_j be the first j feature groups selected by our greedy algorithm. There exists a constant $\gamma = \frac{\lambda^* + \lambda}{1 + \lambda} > 0$ such that for any sequence S, total cost K, and indices $j = 1, 2, ..., J$, $F(S_{\langle K \rangle}) - F(G_{j-1}) \leq \frac{K}{\gamma}[\frac{F(G_j) - F(G_{j-1})}{c(g_j)}]$.*

Theorem 3.3.2. *Let $B = \sum_{i=1}^{L} c(g_i)$ for some L. There exists a constant $\gamma = \frac{\lambda^* + \lambda}{1 + \lambda}$, such that for any sequence S and total cost K, $F(G_{\langle B \rangle}) > (1 - e^{-\gamma \frac{B}{K}}) F(S_{\langle K \rangle})$.*

Before delving into the proof of Lemma 3.3.1, we first discuss some implications of Theorem 3.3.2, which argues that the explained variance of greedily selected features of cost B is within $(1 - e^{\gamma \frac{B}{K}})$-factor of that of any competing feature sequence of cost K. If we apply

Algorithm 3 Cost Sensitive Group Forward Regression (CS-G-OMP)

```
1: Input: The normalized feature matrix $X \in \mathbb{R}^{n \times D}$. The normalized response vector $Y \in \mathbb{R}^n$,
   which has a zero mean and unit variance. Feature groups $\mathcal{G}_1, ...\mathcal{G}_J$ that partition $\{1, .., D\}$,
   and group costs $c(\mathcal{G}_j)$. Regularization constant $\lambda$.
2: Output: A sequence $G = g_1, g_2, ..., g_J$ of feature groups. For each $j \leq J$, a coefficient
   $w(G_j)$ for the features $G_j = g_1, ..., g_j$.
3: Set $G_0 = \emptyset$ to be an empty sequence.
4: Set $w(G_0) = \vec{0}$ to be a zero vector of zero length.
5: Compute $C = X^T X$.
6: for $j = 1, 2, ..., J$ do
7:    for $g \notin G_{j-1}$ do
8:       Compute $w = w(G_{j-1} \oplus g) = \frac{1}{n} C^{-1}_{G_{j-1} \oplus g} X^T_{G_{j-1} \oplus g} Y$.
9:       Compute $F(G_{j-1} \oplus g) = \frac{1}{2n}(\|Y\|^2 - \|Y - X_{G_{j-1} \oplus g} w\|^2) - \frac{\lambda}{2}\|w\|^2$.
10:   end for
11:   $g_j = \underset{g = \mathcal{G}_1, ..., \mathcal{G}_J, g \notin G_{j-1}}{\arg\max} \frac{F(G_{j-1} \oplus g) - F(G_{j-1})}{c(g)}$.
12:   Append $g_j$ to the sequence: $G_j = G_{j-1} \oplus g_j$.
13:   Record $w(G_j) = \frac{1}{n} C^{-1}_{G_j} X^T_{G_j} Y$.
14: end for
```

minimum regularization ($\lambda \to 0$), then the constant γ approaches λ^*. The resulting bound factor $(1 - e^{-\lambda^* \frac{B}{K}})$ is the bound for FR by Das and Kempe (2011). However, we achieve the same bound for OMP, improving theoretical guarantees of OMP. We also note that less-correlated features lead to a higher λ^* and a stronger bound.

Lemma 3.3.1 for CS-G-FR is standard if we follow proofs in (Streeter and Golovin, 2008) and (Das and Kempe, 2011) because the objective F is γ-approximately submodular. However, we present a proof of Lemma 3.3.1 for CS-G-OMP without approximate submodularity to achieve the same constant γ. This proof in turn uses Lemma 3.3.3 and Lemma 3.3.4, whose proofs are based on the Taylor expansions of the regularized risk $\mathcal{R}[f_S] = R(S)$, a M-strongly smooth and m-strongly convex loss functional of predictors $f(x) = w^T x$. We defer these two proofs to the additional details in Section 3.6 and note that $M = m$ with our choice of R.

Lemma 3.3.3 (Using Smoothness)**.** *Let S and G be some fixed sequences. Then $F(S) - F(G) \leq \frac{1}{2m}\langle b^G_{G \oplus S}, C^{-1}_{G \oplus S} b$*

Lemma 3.3.4 (Using Convexity)**.** *For $j = 1, 2, ..., J$, $F(G_j) - F(G_{j-1}) \geq \frac{1}{2M}\langle b^{G_{j-1}}_{g_j}, C^{-1}_{g_j} b^{G_{j-1}}_{g_j}\rangle$.*

Note that in Lemma 3.3.4, since we assume feature groups are whitened, then $C_{g_j} = (1 + \lambda)I$. The bound of the lemma becomes $F(G_j) - F(G_{j-1}) \geq \frac{1}{2M(1+\lambda)}\langle b^{G_{j-1}}_{g_j}, b^{G_{j-1}}_{g_j}\rangle$. If feature groups are not whitened, the constant $(1 + \lambda)$ can be scaled up to $(|\mathcal{G}_j| + \lambda)$, which detriments the strength of Theorem 3.3.2 especially when feature groups are large.

Proof. (of Lemma 3.3.1, using Lemma 3.3.3 and Lemma 3.3.4)
Using Lemma 3.3.3, on $S_{\langle K \rangle}$ and G_{j-1}, we have:

$$F(S_{\langle K \rangle}) - F(G_{j-1})$$

$$\leq \frac{1}{2m} \langle b^{G_{j-1}}_{G_{j-1} \oplus S_{\langle K \rangle}}, C^{G}_{G_{j-1} \oplus S_{\langle K \rangle}} b^{G_{j-1}}_{G_{j-1} \oplus S_{\langle K \rangle}} \rangle \tag{3.8}$$

Note that the gradient $b^{G_{j-1}}_{G_{j-1}}$ equals 0, because $F(G_{j-1})$ is achieved by the linear model $w(G_{j-1})$. Then, using block matrix inverse formula, we have:

$$F(S_{\langle K \rangle}) - F(G_{j-1}) \leq \frac{1}{2m} \langle b^{G_{j-1}}_{S_{\langle K \rangle}}, C^{G}_{S_{\langle K \rangle}} b^{G_{j-1}}_{S_{\langle K \rangle}} \rangle \tag{3.9}$$

where $C^{G}_{S_{\langle K \rangle}} = C_{S_{\langle K \rangle}} - C_{S_{\langle K \rangle} G} C^{-1}_{S_{\langle K \rangle}} C_{G S_{\langle K \rangle}}$. Using spectral techniques in Lemmas 2.5 and 2.6 in (Das and Kempe, 2011) and noting that the minimum eigenvalue of C, $\lambda_{min}(C)$, is $\lambda^* + \lambda$, we have

$$\frac{1}{2m} \langle b^{G_{j-1}}_{S_{\langle K \rangle}}, C^{G}_{S_{\langle K \rangle}} b^{G_{j-1}}_{S_{\langle K \rangle}} \rangle \leq \frac{1}{2m(\lambda^* + \lambda)} \langle b^{G_{j-1}}_{S_{\langle K \rangle}}, b^{G_{j-1}}_{S_{\langle K \rangle}} \rangle. \tag{3.10}$$

Expanding $S_{\langle K \rangle}$ into individual groups s_i, we continue:

$$= \frac{1}{2m(\lambda^* + \lambda)} \sum_{s_i \in S_{\langle K \rangle}} \langle b^{G_{j-1}}_{s_i}, b^{G_{j-1}}_{s_i} \rangle \tag{3.11}$$

$$\leq \frac{1}{2m(\lambda^* + \lambda)} \sum_{s_i \in S_{\langle K \rangle}} c(s_i) \max_{g} \frac{\langle b^{G_{j-1}}_{g}, b^{G_{j-1}}_{g} \rangle}{c(g)} \tag{3.12}$$

$$= \frac{1}{2m(\lambda^* + \lambda)} \sum_{s_i \in S_{\langle K \rangle}} c(s_i) \frac{\langle b^{G_{j-1}}_{g_j}, b^{G_{j-1}}_{g_j} \rangle}{c(g_j)} \tag{3.13}$$

$$\leq \frac{M(1 + \lambda)}{m(\lambda^* + \lambda)} \sum_{s_i \in S_{\langle K \rangle}} c(s_i) \frac{F(G_j) - F(G_{j-1})}{c(g_j)}. \tag{3.14}$$

The last equality follows from the greedy selection step of Algorithm 2 when feature groups are whitened. The last inequality is given by Lemma 3.3.4. The theorem then follows from $\gamma = (\frac{m}{M}) \frac{\lambda^* + \lambda}{1 + \lambda} = \frac{\lambda^* + \lambda}{1 + \lambda}$. □

3.4 Bi-criteria Analysis at Any Budget

Our analysis so far only bounds algorithm performance at budgets when new items are selected. However, an ideal analysis should apply to all budgets. As illustrated in Figure 3.1a, previous methods may choose expensive features early; until they are computed, we have no bounds. Figure 3.1b illustrates our proposed fix: each new item g_{j+1} cannot be more costly than the current sequence G_j.

This section proves two theorems of anytime prediction at *any* budget. Theorem 3.4.1 shows that to approximate the optimal explained variance of cost B within a constant factor, an anytime algorithm must cost at least $4B$. We then motivate and formalize our fix in Algorithm 4, which is shown in Theorem 3.4.3 to achieve this *bi-criteria approximation* bound for both budget and objective with the form: $F(G_{\langle B\rangle}) > (1 - e^{-\frac{\gamma^2}{1+\gamma}})F(S_{\langle \frac{B}{4}\rangle})$, where γ is the approximate submodular ratio, i.e., the maximum constant $\gamma \leq 1$ such that for all sets $A' \subseteq A$ and all element x,

$$\gamma(F(A \cup \{x\}) - F(A)) \leq F(A' \cup \{x\}) - F(A'). \tag{3.15}$$

We first illustrate the inherent difficulty in generating single sequences that are competitive at arbitrary budgets B by using the following budgeted maximization problem:

$$X = \{1, 2, \ldots\}, \;\; c(x) = x, \;\; F(S) = \sum_{x \in S} e^x. \tag{3.16}$$

The above problem originates from fitting the linear model $Y = \sum_{i=1}^{D} e^i X_i$, where X_i's are i.i.d. and X_i costs i.

Theorem 3.4.1. *Let $\mathcal{A}$ be any algorithm for selecting sequences $A = (a_1, \ldots)$. The best bi-criteria approximation that $\mathcal{A}$ can satisfy must be at least a 4-approximation in cost for the sequence described in Equation (3.16). That is, there does not exist a $C < 4$, and a $c_1 \in [0, 1)$, such that for any budget B and any sequence S,*

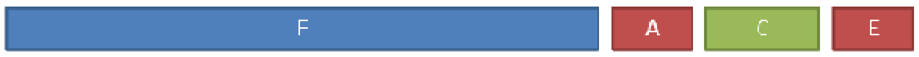
$$F(A_{\langle B\rangle}) > (1 - c_1)\, F(S_{\langle \frac{B}{C}\rangle}).$$

Proof. For any budget B, it is clear that the optimal selection contains a single item, B, whose value is e^B. For any budget B, let $m(B)$ denote the item of the maximum cost that is selected by the algorithm. If the bi-criteria bound holds, then $\sum_{k=1}^{m(B)} e^k \geq F(A_{\langle B\rangle}) > (1 - c_1)\, F(S_{\langle \frac{B}{C}\rangle})$. Taking the log of both sides and rearranging terms, we have $m(B) \geq \lfloor \frac{B}{C} \rfloor + \ln(1 - c_1) + \ln(e - 1) - 2$. Since $3 - \ln(1 - c_1) - \ln(e - 1) > 0$, we have for B large enough: $C \geq \frac{B}{m(B)}$. Hence, we need to minimize $\frac{B}{m(B)}$ for all B to minimize C. We can assume a_j to be increasing because otherwise we could remove the violating a_j from the sequence and decrease the ratio $\frac{B}{m(B)}$ for all subsequent j.

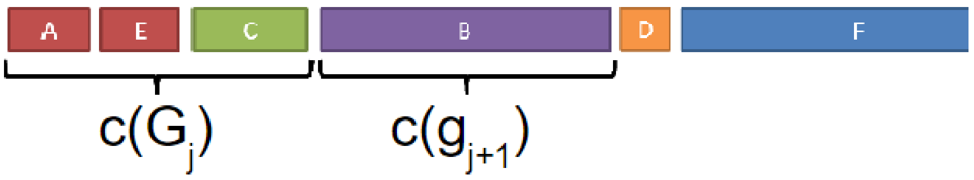
Let $b_j := c(A_j)$ and $\alpha_j := \frac{c(a_j)}{b_{j-1}}$. Then immediately before a_j is available, $\frac{B}{m(B)} \to \frac{c(A_j)}{c(a_{j-1})} \geq \frac{(1+\alpha_j)b_{j-1}}{b_{j-1}} = 1 + \alpha_j$. If we can bound $\frac{B}{m(B)} \leq C$ for all B, then there exists α_{max} such that $\alpha_j < \alpha_{max}$ for all j large enough. Immediately after a new a_j is selected, $\frac{B}{m(B)} = \frac{c(A_j)}{c(a_j)} = \frac{1+\alpha_j}{\alpha_j}$. For $\frac{B}{m(B)}$ to be bounded, there must exist some $\alpha_{min} > 0$ such that $\alpha_j > \alpha_{min}$ for large enough j. Now we consider the ratio $\frac{B}{m(B)}$ right before a_{j+1} is selected:

$$\frac{c(A_{j+1})}{c(a_j)} = \frac{b_j(1 + \alpha_{j+1})}{b_j \frac{\alpha_j}{1+\alpha_j}} = 1 + \frac{\alpha_{j+1}}{\alpha_j} + \alpha_{j+1} + \frac{1}{\alpha_j}. \tag{3.17}$$

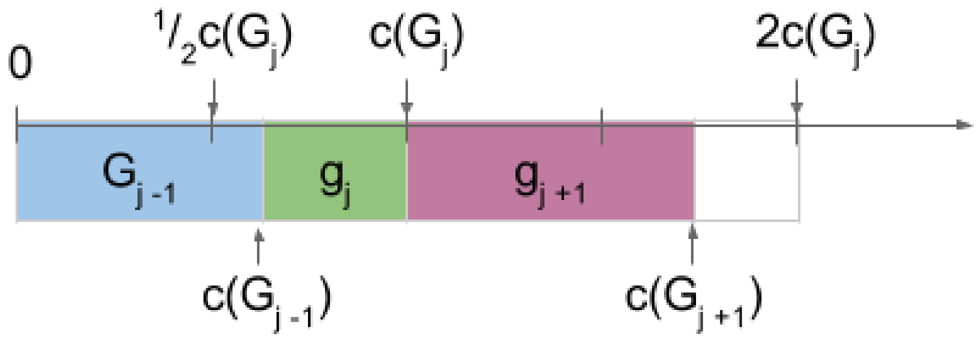
Assume for seek of contradiction that $\frac{c(A_{j+1})}{c(a_j)}$ is bounded above by z for some $z \in (1, 4)$. Let $y := \frac{\alpha_{j+1}}{\alpha_j}$. Then we have: $z \geq 1 + y + y\alpha_j + \frac{1}{\alpha_j} \geq 1 + y + 2\sqrt{y} = (\sqrt{y} + 1)^2$. Hence $y \leq (\sqrt{z} - 1)^2 < 1$. So $a_{j+1} \leq (\sqrt{z} - 1)^2 a_j$, which implies that a_j converges to 0 and we have a contradiction. So $C \geq \frac{B}{m(B)} \to \frac{c(A_{j+1})}{c(a_j)} \geq 4$ for large j. $\square$



(a) Before F is computed, we have no output or bounds.



(b) Our constraint $c(g_{j+1}) \leq c(G_j)$ induces a smoother cost increase.



(c) Illustration of Doubling Algorithm Cost Constraint

Figure 3.1: Doubling Algorithm (b) has better anytime behaviors than greedy algorithm with no cost constraints (a).

The above proof lower bounds the cost approximation ratio C by Eq. 3.17, which is shown to be at least 4 for $C < \infty$. We note that $Eq.$ 3.17 equals 4 if $\forall j, \alpha_j = 1$, which means the sequence total cost is doubled at each selection step. This observation leads to *Doubling Algorithm* (Alg. 4): we perform greedy selection in the same way as CS-G-FR, except that the total cost can be at most doubled at each step (illustrated in Figure 3.1c). The advantage of Doubling Algorithm over CS-G-FR is that the former prevents early computation of expensive features and induces a smoother increase of total cost; in most real-world data-sets, the two are identical after few steps because feature costs are often in a narrow range. We will analyze Doubling Algorithm with the following assumption, called *doubling capable*.

Definition 3.4.2. *Let $G = (g_1, \ldots)$ be the sequence selected by the doubling algorithm. The set X and function F are doubling capable if, at every iteration j, the following set is non-empty: $\{x \mid x \in X \setminus G_{j-1},\ c(x) \leq c(G_{j-1})\}$*

Theorem 3.4.3. *Let $G = (g_1, \ldots)$ be the sequence selected by the doubling algorithm (Algorithm 4). Fix some $B > c_{min}$. Let F be γ-approximately submodular as in Definition 3.15. For any sequence S,*

$$F(G_{\langle B \rangle}) > \left(1 - e^{-\frac{\gamma^2}{1+\gamma}}\right) F(S_{\langle \frac{B}{4} \rangle}).$$

Proof. Doubling capable easily leads to the observation that for all budgets B, there exists an index j such that $\frac{B}{2} \leq c(G_j) < B$. Choose K and k to be the largest integers such that $\frac{B}{2} \leq c(G_K) < B$

Algorithm 4 Forward Regression with Doubling Modification

```
1: **Input:** Objective function $F$, elements $X$, minimum cost $c_{\text{min}}$.
2: **Output:** A sequence $G = g_1, g_2, ..., g_J$ of elements. For each $j \leq J$, a parameter $w(G_j)$ for
   the elements $G_j = g_1, ..., g_j$ for maximizing $F$.
3: Set $g_1 = \underset{x \in X,\ c(x) \leq c_{\text{min}}}{\arg\max} \frac{F(\{x\})}{c(x)}$.
4: Set $G_1 = [g_1]$ as a one-element sequence.
5: Set $w(G_1)$ be the parameter associated with $g_1$ to optimize $F$.
6: **for** $j = 2, ..., J$ **do**
7:     **for** $g \notin G_{j-1}$, $c(g) \leq c(G_{j-1})$ **do**
8:         Compute $F(G_{j-1} \oplus g)$ and the associated parameter $w(G_{j-1} \oplus g)$.
9:     **end for**
10:    $g_j = \underset{g = \mathcal{G}_1, ..., \mathcal{G}_J, g \notin G_{j-1}, c(g) \leq c(G_{j-1})}{\arg\max} \frac{F(G_{j-1} \oplus g) - F(G_{j-1})}{c(g)}$.
11:    Append $g_j$ to the sequence: $G_j = G_{j-1} \oplus g_j$.
12:    Record $w(G_j) = w(G_{j-1} \oplus g)$.
13: **end for**
```

and $\frac{B}{8} \leq c(G_k) < \frac{B}{4}$. Since at each step we at most double the total cost and $4c(G_k) < B$, we observe $K \geq k + 2$. For each j, define $s_j = \frac{F(G_{j+1}) - F(G_j)}{c(g_{j+1})}$ as the best rate of improvement among the items Doubling Algorithm is allowed to consider after choosing G_j. Consider the item x in sequence $S_{\langle \frac{B}{4} \rangle}$ of the maximum cost.

(Case 1) If $c(x) \leq c(G_k)$, then every item in $S_{\langle \frac{B}{4} \rangle}$ was a candidate for g_j for all $j = k+1, ..., K$. So by approximate submodularity from Equation 3.15, we have

$$F(S_{\langle \frac{B}{4} \rangle}) \leq F(S_{\langle \frac{B}{4} \rangle} \cup G_j) \leq F(G_j) + \frac{Bs_j}{4\gamma}. \tag{3.18}$$

Then using the standard submodular maximization proof technique, we define $\Delta_j = F(S_{\langle \frac{B}{4} \rangle}) - F(G_j)$. Applying $s_j = \frac{\Delta_j - \Delta_{j+1}}{c(g_{j+1})}$ in the above inequality, we have $\Delta_{k+j} \leq \Delta_k \prod_{j=k+1}^{k+j}(1 - \gamma\frac{4c(g_j)}{B})$. Maximizing the inequality by setting $c(g_j) = \frac{B}{K-k} \leq \frac{c(G_K) - c(G_k)}{4(K-k)}$, and using $(1 - z/l)^l < e^{-z}$, we have $F(G_K) > (1 - e^{-\gamma})F(S_{\langle \frac{B}{4} \rangle})$.

From now on, we assume that $c(x) > c(G_k)$ and consider two cases by comparing $c(g_{k+2})$ and $c(G_k)$.

(Case 2.1) If $c(g_{k+2}) \geq c(G_k)$, then $c(G_K) - c(G_{k+1}) \geq c(g_{k+2}) \geq c(G_k)$. Since $c(G_{k+1}) \leq 2c(G_k)$ and $c(x) > c(G_k)$, we have $c(G_K) - c(G_{k+1}) \geq \frac{B}{2} - 2c(G_k)$.
So $c(G_K) - c(G_{k+1}) \geq \max(c(G_k), \frac{B}{2} - 2c(G_k)) \geq \frac{B}{6}$. Thus, using the same proof techniques as in case 1, we can analyze the ratio between Δ_{k+1} and Δ_K, and have: $F(G_K) > (1 - e^{-\frac{2}{3}\gamma})F(S_{\langle \frac{B}{4} \rangle})$.

(Case 2.2) Finally, if $c(g_{k+2}) < c(G_k) < c(x) < c(G_{k+1})$, g_{k+2} was a candidate for g_{k+1}, and x was a candidate for g_{k+2}. For an item y, let $r(y^j) = \frac{F(G_j \cup \{y\}) - F(G_j)}{c(y)}$ be the improvement rate of item y at G_j. Then we have $r(g_{k+1}^k) > r(g_{k+2}^k)$ and $r(g_{k+2}^{k+1}) > r(x^{k+1})$. Since the objective function is increasing, we have
$r(x^k)c(x) \leq r(x^{k+1})c(x) + r(g_{k+1}^k)c(g_{k+1})$, so that $r(x^k) \leq r(x^{k+1}) + r(g_{k+1}^k)\frac{c(g_{k+1})}{c(x)}$. Then by

Table 3.1: Test time 0.97-Timeliness measurement of different methods on AGRICULTURAL. We break the methods into OMP, FR and Oracle family: e.g., "Single" in the G-CS-OMP family means G-CS-OMP-Single, and "FR" in the Oracle family means the oracle curve derived from G-FR.

CS-G-OMP-Variants				CS-G-FR	Oracles		Sparse
CS-G-OMP	Single	No-Whiten	G-OMP		FR Oracle	OMP Oracle	
0.4406	0.4086	0.4340	0.4073	**0.4525**	**0.4551**	0.4508	0.3997

Table 3.2: Test time 0.99-Timeliness measurement of different methods on YAHOO! LTR.

Group	CS-G-OMP-Variants				CS-G-FR	Oracles		Sparse
Size	CS-G-OMP	Single	No-Whiten	G-OMP		FR	OMP	
5	**0.3188**	0.3039	0.3111	0.2985	**0.3222**	**0.3225**	0.3211	0.2934
10	**0.3142**	0.3117	0.3079	0.2909	**0.3205**	**0.3207**	0.3164	0.2858
15	**0.3165**	0.3159	0.3116	0.2892	**0.3213**	**0.3213**	0.3177	0.2952
20	**0.3161**	0.3124	0.3065	0.2875	**0.3180**	**0.3180**	0.3163	0.2895

the definition of γ in Equation 3.15, we have $\gamma r(g_{k+2}^{k+1}) \leq r(g_{k+2}^{k})$. Hence we have $\gamma r(x^{k+1}) \leq r(g_{k+1}^{k})$, which leads to $r(x^k) \leq r(g_{k+1}^{k})(\frac{1}{\gamma} + \frac{c(g_{k+1})}{c(x)}) \leq r(g_{k+1}^{k})(1 + \frac{1}{\gamma})$. Then inequality (3.18) holds with a coefficient adjustment and becomes $F(S_{\langle \frac{B}{4} \rangle}) \leq F(G_k) + \frac{B s_k (1+\gamma)}{4\gamma^2}$. Noting that the above inequality holds for all $j = k+1, ..., K$, we can replace the constant γ in the proof of case 1 with $\frac{\gamma^2}{1+\gamma}$ and have the following bound: $F(G_K) > (1 - e^{-\frac{\gamma^2}{1+\gamma}})F(S_{\langle \frac{B}{4} \rangle})$.

$\square$

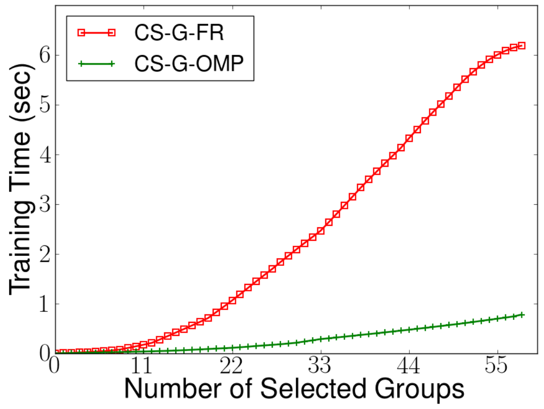
3.5 Experiments

3.5.1 Data-sets and Set-ups

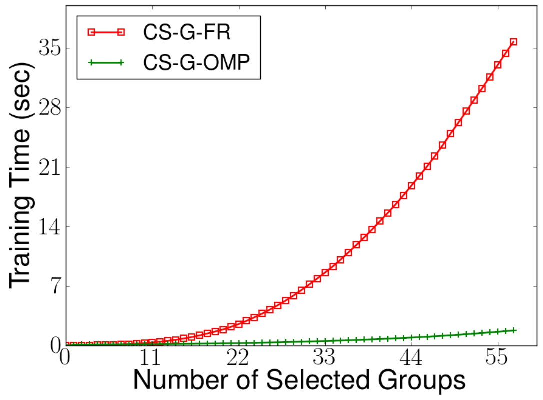
We experiment our methods for anytime linear prediction on two real-world data-sets, each of which has a significant number of feature groups with associated costs.

- **Yahoo! Learning to Rank Challenge** (Chapelle and Chang, 2011) contains 883k web documents, each of which has a relevance score in $\{0, 1, 2, 3, 4\}$. Each of the 501 document features has an associated computational cost in $\{1, 5, 20, 50, 100, 150, 200\}$; the total feature cost is around 17K. The original data-set has no feature group structures, so we generated random group structures by grouping features of the same cost into groups of a given size s.[1]
- **Agriculture** is a proprietary data-set that contains 510k data samples, 328 features, and 57 feature groups. Each sample has a binary label in $\{1, 2\}$. Each feature group has an associated

[1]We experiment on group sizes $s \in \{5, 10, 15, 20\}$. We choose regularizer $\lambda = 10^{-5}$ based on validation. We use $s = 10$ for qualitative results such as plots and illustrations, but we report quantitative results for all group size s. For our quantitative results, we report the average test performance. The initial risk is $R(\emptyset) = 0.85$.



(a) Training Time OMP vs. FR (AGRICULTURAL)

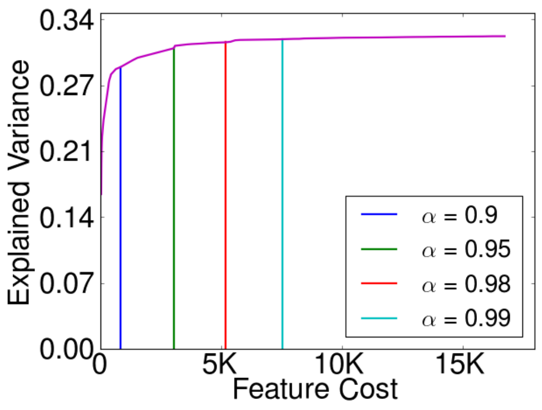


(b) Training Time OMP vs. FR (YAHOO! LTR)

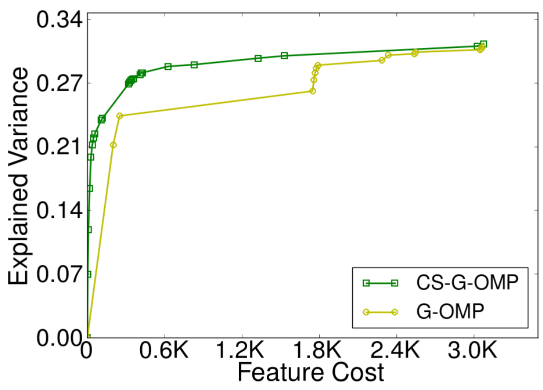
Figure 3.2: The training time vs. the number of feature groups selected with two algorithms: CS-G-FR and CS-G-OMP. CS-G-OMP achieves a 8x and 20x overall training time speed-up on AGRICULTURAL and YAHOO! LTR.

cost measured in its average computation time.[2]

3.5.2 Evaluation Metric and Approximated Oracle



(a) Plateau Effect and α-Stopping Costs



(b) Importance of Costs (CS-G-OMP vs. G-OMP)

Figure 3.3: (a) Explained Variance vs. Cost curve of CS-G-OMP in YAHOO! LTR. Vertical lines mark different α-stopping costs. (b) Explained Variance vs. Cost curve of CS-G-OMP and G-OMP on YAHOO! LTR set 1 with individual group size $s = 10$, stopped at 0.97-stop cost.

Following the practice of Karayev et al. (2012), we use the area under the maximization objective F (explained variance) vs. cost curve normalized by the total area as the *timeliness*

[2] There are 6 groups of size 32; the other groups have sizes between 1 and 6. The cost of each group is its expected computation time in seconds, ranging between 0.0005 and 0.0088; the total feature cost is 0.111. We choose regularizer $\lambda = 10^{-7}$. The data-set is split into five 100k sets, and the remaining 10k are used for validation. We report the cross validation results on the five 100K sets as the test results. The initial risk is $R(\emptyset) = 0.091$.

measurement of the anytime performance of an algorithm. In our data-sets, the performance of linear predictors plateaus much before all features are used, e.g., Figure 3.3a demonstrates this effect in YAHOO! LTR, where the last one percent of total improvement is bought by half of the total feature cost. Hence the majority of the timeliness measurement is from the plateau performance of linear predictors. The difference between timeliness of different anytime algorithms diminishes due to the plateau effect. Furthermore, the difference vanishes as we include additional redundant high cost features. To account for this effect, we stop the curve when it reaches the plateau. We define an *α-stopping cost* for parameter α in $[0, 1]$ as the cost at which our CS-G-OMP achieves α of the final objective value in training and ignore the objective vs. cost curve after the α-stopping cost. We call the timeliness measure on the shortened curve as *α-timeliness*; 1-timeliness equals the normalized area under the full curve and 0-timeliness is zero. If a curve does not pick a group at α-stopping cost, we linearly interpolate the objective value at the stopping cost to computr timeliness. We say an objective vs. cost curve has reached its final plateau if at least 95% of the total objective has been achieved and the next 1% requires more than 20% feature costs. (If the plateau does not exist, we use $\alpha = 1$.) Following this rule, we choose $\alpha = 0.97$ for AGRICULTURAL and $\alpha = 0.99$ for YAHOO! LTR.

Since an exhaustive search for the best feature sequencing is intractable, we approximate with the **Oracle** anytime performance following the approach of Karayev et al. (2012). Given an objective vs. cost curve of a sequencing, we reorder the feature groups in descending order of their marginal benefit per unit cost, assuming that the marginal benefits stay the same after reordering. We specify which sequencing is used for creating **Oracle** in Section 3.5.5. For baseline performance, we use cost-weighted Group Lasso (Yuan and Lin, 2006), which scales the regularization constant of each group with the cost of the group. We note that the cascade design by Chen et al. (2012a) can be reduced to this baseline if we enforce linear prediction. More specifically, the baseline solves the following minimization problem: $\min_{w \in \mathbb{R}^D} \|Y - Xw\|_2^2 + \lambda \sum_{j=1}^{J} c(\mathcal{G}_j) \|w_{\mathcal{G}_j}\|_2$, and we vary value of regularization constant λ to obtain lasso paths. We call this baseline algorithm **Sparse**[3].

3.5.3 Importance of Feature Cost

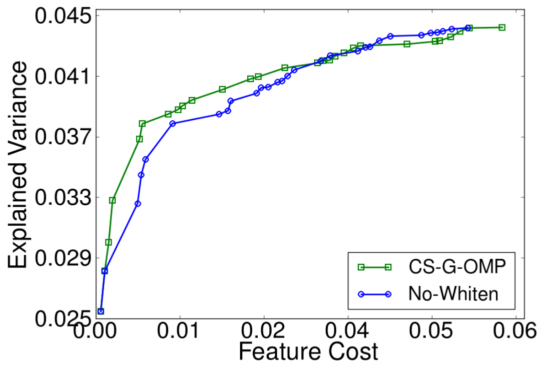
Our proposed CS-G-OMP differs from Group Orthogonal Matching Pursuit (G-OMP) (Lozano et al., 2009) in that G-OMP does not consider feature costs when evaluating features. We show that this difference is crucial for anytime linear prediction. In Figure 3.3b, we compare the objective vs. costs curves of CS-G-OMP and G-OMP that are stopped at 0.97-stopping cost on YAHOO! LTR. As expected, CS-G-OMP achieves a better overall prediction at every budget, qualitatively demonstrating the importance of incorporating feature costs. Table 3.1 and Table 3.2 quantify this effect, showing that CS-G-OMP achieves a better timeliness measure than regular G-OMP.

[2]Karayev et al. (2012) define *timeliness* as the area under the average precision vs. time curve

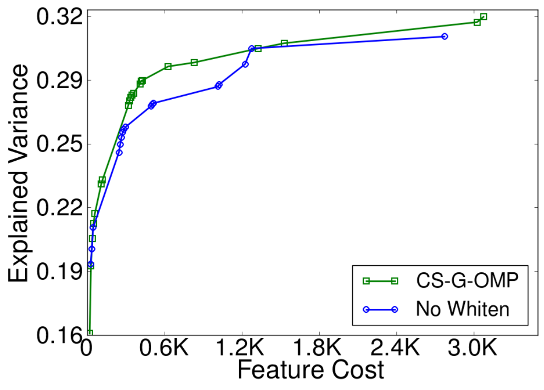
[3]We use an off-the-shelf software, SPAMS (SPArse Modeling Software (Jenatton et al., 2010)), to solve the optimization.

3.5.4 Group Whitening

We provide experimental evidence that Group whitening, i.e., $X_g^T X_g = I_{D_g}$ for each group g, is a key assumption of both this work and previous feature group selection literature by Lozano et al. (2009; 2011). In Figure 3.4, we compare anytime prediction performances using group whitened data against those using the common normalization scheme where each feature dimension is individually normalized to have zero mean and unit variance. The objective vs. cost curve qualitatively shows that group whitening consistently results in the better predictions. This behavior is expected from data-sets whose feature groups contain correlated features, e.g., group whitening effectively prevents selection step $(*)$ from overestimating the predictive power of feature groups of repeated good features. Table 3.1 and Table 3.2 demonstrate quantitatively the consistent better timeliness performance of CS-G-OMP over that of CS-G-OMP-no-whiten.



(a) Group Whiten vs. No-Whiten (AGRICULTURAL)



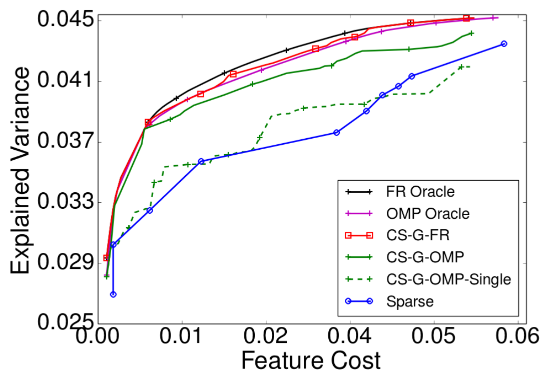
(b) Group Whiten vs. No-Whiten (YAHOO! LTR)

Figure 3.4: Explained Variance vs. Feature Cost curves on AGRICULTURAL (a) and YAHOO! LTR (b) comparing group whitening with no group whitening. The curves stop at 0.97-stopping cost.

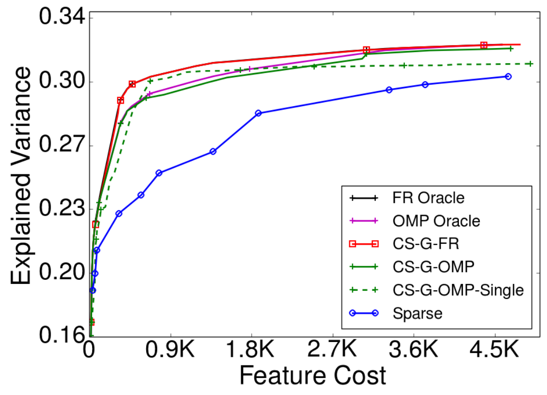
3.5.5 Other Selection Criteria Variants

This section compares CS-G-OMP and CS-G-FR, along with variants of these two methods and the baseline, Sparse. We formulated the variant of CS-G-OMP, *single*, in Section 3.2 and it intuitively chooses feature groups of the best single feature dimension per group cost. Our experiments show that this modification degrades prediction performance of CS-G-OMP. Since FR directly optimizes the objective at each step, we expect CS-G-FR to perform the best and use its curve to compute the **Oracle** curve as an approximate to the best achievable performance.

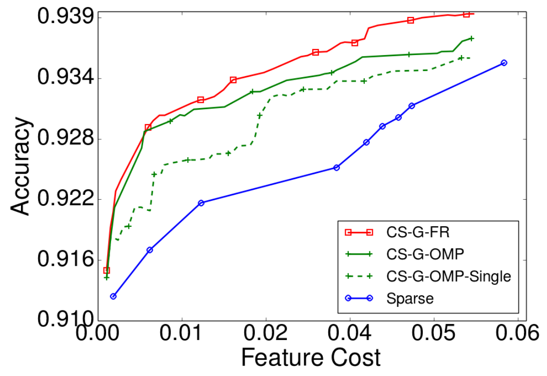
In Figure 3.5, we evaluate CS-G-FR, CS-G-OMP and CS-G-OMP-single based on the objective in Theorem 3.3.2, i.e., explained variance vs. feature cost curves. CS-G-FR, as expected, outperforms all other methods. CS-G-OMP outperforms the baseline method, Sparse, and the CS-G-OMP-Single variant. The performance advantage of CS-G-OMP over CS-G-OMP-Single is much clearer in the AGRICULTURAL data-set than in the YAHOO! LTR data-set. AGRICULTURAL has a natural group structure which may contain correlated features in each group. YAHOO! LTR has a randomly generated group structure whose features were filtered by feature selection before the data-set was published (Chapelle and Chang, 2011). CS-G-FR and CS-G-OMP outperform the baseline algorithm, Sparse. We speculate that linearly scaling group regularization constants by group costs did not enforce Group-Lasso to choose the most cost-efficient features early. The test-time timeliness measures of each of the methods are recorded in Table 3.1 and Table 3.2, and quantitatively confirm the analysis above. Since AGRICULTURAL and YAHOO! LTR are originally a classification and a ranking data-set, respectively, we also report in Figure 3.5 the performance using classification accuracy and NDCG@5. This demonstrates the same qualitatively results as using explained variants.



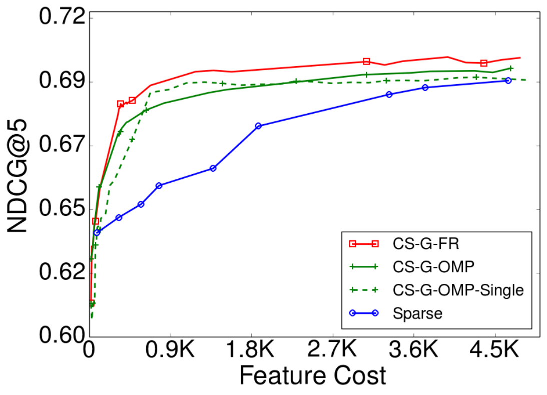
(a) FR vs. OMP vs. Sparse (AGRICULTURAL)



(b) FR vs. OMP vs. Sparse (YAHOO! LTR)



(c) FR vs. OMP vs. Sparse (AGRICULTURAL)



(d) FR vs. OMP vs. Sparse (YAHOO! LTR)

Figure 3.5: (a),(b): Explained Variance vs. Feature Cost curves on AGRICULTURAL and YAHOO! LTR(group-size=10), using CS-G-OMP, CS-G-FR and their Single variants. Curves stop at 0.97 and 0.98 stopping costs. (c),(d): Same curve with the natural objectives of the data-sets: accuracy and NDCG@5.

As expected, when compared against CS-G-OMP, CS-G-FR consistently chooses more cost-efficient features at the cost of a longer training time. In the context of linear regression, let us assume that the group sizes are bounded by a constant when we are to select the number K feature group. We can then compute a new model of K groups in $O(K^2N)$ using Woodbury's matrix inversion lemma, evaluate it in $O(KN)$, and compute the gradients with respect to the weights of unselected groups in $O(N(J-K))$. Thus, CS-G-OMP requires $O(K^2N+JN)$ at step $K=1,2,3,...,J$ and CS-G-FR requires $O((J-K)K^2N)$, so the total training complexities for CS-G-OMP and CS-G-FR are $O(J^3N)$ and $O(J^4N)$, using $\sum_{K=1}^{J} K^2 = \frac{1}{6}J(J+1)(2J+1)$ and $\sum_{K=1}^{J} K^3 = \frac{1}{4}J^2(J+1)^2$. We also show this training complexity gap empirically in Figure 3.2, which plots the curves of training time vs. number of feature groups selected. When all feature groups are selected, CS-G-OMP achieves a 8x speed-up in AGRICULTURAL over CS-G-FR. In YAHOO! LTR, CS-G-OMP achieves a speed-up factor between 10 and 20; the smaller the sizes of the groups, the larger speed-up due to the increase in the number of groups. Both greedy methods are much faster than the Lasso path computation using SPAMS, however.

3.6 Additional Proof Details

This section describes a functional boosting view of selecting features for generalized linear models of one-dimensional response. We then prove Lemma 3.3.3 and Lemma 3.3.4 for this more general setting. These more general results in turn extend Theorem 3.3.2 to generalized linear models.

3.6.1 Functional Boosting View of Feature Selection

We view each feature f as a function h_f that maps sample x to x_f. We define $f_S : \mathbb{R}^D \to \mathbb{R}$ to be the best linear predictor using features in S, i.e., $f_S(x) \triangleq w(S)^T x_S$. For each feature dimension $d \in D$, the coefficient of d is in $w(S)$ is $w(S)_d = f_S(e_d)$, where e_d is the d^{th} dimensional unit vector. So $\|w(S)\|_2^2 = \sum_{d=1}^{D} \|f_S(e_d)\|_2^2$. Given a generalized linear model with link function $\nabla\Phi$, the predictor is $E[y|x] = \nabla\Phi(w^T x)$ for some w and the calibrated loss is $r(w) = \sum_{i=1}^{n}(\Phi(w^T x_i) - y_i w^T x_i)$. Replacing $f_S(x_i) = w(S)^T x_i$, we have

$$r(w(S)) = \sum_{i=1}^{n}(\Phi(f_S(x_i)) - y_i f_S(x_i)). \tag{3.19}$$

Note that the risk function in Equation 3.1 can be rewritten as the following to resemble Equation 3.19:

$$\begin{aligned} R(S) = \mathcal{R}[f_S] =& \frac{1}{n}\sum_{i=1}^{n}(\Phi(f_S(x_i)) - y_i^T f_S(x_i)) \\ &+ \frac{\lambda}{2}\sum_{d=1}^{D} \|f_S(e_d)\|_2^2 + A, \end{aligned} \tag{3.20}$$

where $\phi(x) = \frac{1}{2}x^2$ for linear predictions and constant $A = \frac{1}{2n}\sum_{i=1}^{n} y_i^2$. Next we define the inner product between two functions $f, h : \mathbb{R}^D \to \mathbb{R}$ over the training set to be:

$$\langle f, h \rangle \triangleq \frac{1}{n}\sum_{i=1}^{n} f(x_i)h(x_i) + \frac{\lambda}{2}\sum_{d=1}^{D} f(e_d)h(e_d). \tag{3.21}$$

With this definition of inner product, we can compute the derivative of $\mathcal{R}$:

$$\nabla \mathcal{R}[f] = \sum_{i=1}^{n} (\nabla \Phi(f(x_i)) - y_i)\delta_{x_i} + \sum_{d=1}^{D} f(e_d)\delta_{e_d}, \tag{3.22}$$

where $\nabla \phi(x) = x$ for linear predictions, and δ_x is an indicator function for x. Then the gradient of objective $F(S)$ w.r.t coefficient w_f of a feature dimension d can be written as:

$$\begin{aligned} b_d^S &= -\frac{1}{n}\sum_{i=1}^{n} (\nabla \Phi_p(w(S)^T x^i) - y^i)x_d^i - \lambda w(S)_d & (3.23)\\ &= -\langle \nabla \mathcal{R}[f_S], h_d \rangle. & (3.24)\end{aligned}$$

In addition, the regularized covariance matrix of features C satisfies,

$$C_{ij} = \frac{1}{n} X_i^T X_j + \lambda I(i = j) = \langle h_i, h_j \rangle, \tag{3.25}$$

for all $i, j = 1, 2, ..., D$. So in this functional boosting view, Algorithm 2 greedily chooses group g that maximizes, with a slight abuse of notation of $\langle\ ,\ \rangle$, $\|\langle h_g, \nabla \mathcal{R}[f_S]\rangle\|_2^2 / c(g)$, i.e., the ratio between similarity of a feature group and the functional gradient, measured in sum of square of inner products, and the cost of the group

3.6.2 Proof of Lemma 3.3.3 and Lemma 3.3.4

The more general version of Lemma 3.3.3 and Lemma 3.3.4 assumes that the objective functional $\mathcal{R}$ is m-strongly smooth and M-strongly convex using our proposed inner product rule. M-strong convexity is a reasonable assumption, because the regularization term $\|w\|_2^2 = \sum_{d=1}^{D} \|f_S(e_d)\|_2^2$ ensures that all loss functional $\mathcal{R}$ with a convex Φ strongly convex. In the linear prediction case, both m and M equals 1.

The following two lemmas are the more general versions of Lemma 3.3.3 and Lemma 3.3.4.

Lemma 3.6.1. *Let $\mathcal{R}$ be an m-strongly smooth functional with respect to our definition of inner products. Let S and G be some fixed sequences. Then*

$$F(S) - F(G) \leq \frac{1}{2m} \langle b_{G\oplus S}^G, C_{G\oplus S}^{-1} b_{G\oplus S}^G \rangle$$

Proof. First we optimize over the weights in S.

$$\begin{aligned} &F(S) - F(G)\\ &= \mathcal{R}[f_G] - \mathcal{R}[f_S] = \mathcal{R}[f_G] - \mathcal{R}[\sum_{s\in S} \alpha_s^T h_s]\\ &\leq \mathcal{R}[f_G] - \min_{w: w_i^T \in \mathbb{R}^{d_{s_i}}, s_i \in S} \mathcal{R}[\sum_{s_i\in S} w_{s_i}^T h_{s_i}]\end{aligned}$$

Adding dimensions in G will not increase the risk, we have:

$$\le \mathcal{R}[f_G] - \min_{w:w_i\in\mathbb{R}^{d_{s_i}},s_i\in G\oplus S} \mathcal{R}[\sum_{s_i\in G\oplus S} w_{s_i}h_{s_i}]$$

Since $f_G = \sum_{g_i\in G}\alpha_i h_{g_i}$, we have:

$$\le \mathcal{R}[f_G] - \min_w \mathcal{R}[f_G + \sum_{s_i\in G\oplus S} w_i^T h_{s_i}]$$

Expanding using strong smoothness around f_G, we have:

$$\begin{aligned}&\le \mathcal{R}[f_G] - \min_w(\mathcal{R}[f_G] + \langle\nabla\mathcal{R}[f_G], \sum_{s_i\in G\oplus S} w_i^T h_{s_i}\rangle \\ &\quad + \frac{m}{2}\|\sum_{s_i\in G\oplus S} w_i^T h_{s_i}\|_2^2) \\ &= \max_w -\langle\nabla\mathcal{R}[f_G], \sum_{s_i\in G\oplus S} w_i^T h_{s_i}\rangle - \frac{m}{2}\|\sum_{s_i\in G\oplus S} w_i^T h_{s_i}\|_2^2 \\ &= \max_w \langle b_{G\oplus S}^G, w\rangle - \frac{m}{2}\langle w, C_{G\oplus S}w\rangle\end{aligned}$$

Solving w directly we have:

$$F(S) - F(G) \le \frac{1}{2m}\langle b_{G\oplus S}^G, C_{G\oplus S}^{-1} b_{G\oplus S}^G\rangle$$

□

Lemma 3.6.2. *Let $\mathcal{R}$ be a M-strongly convex functional with respect to our definition of inner products. Then*

$$F(G_j) - F(G_{j-1}) \ge \frac{1}{2M(1+\lambda)}\langle b_{g_j}^{G_{j-1}}, b_{g_j}^{G_{j-1}}\rangle \tag{3.26}$$

Proof. After the greedy algorithm chooses some group g_j at step j, we form $f_{G_j} = \sum_{\alpha_i}\alpha_i^T h_{g_i}$, such that

$$\mathcal{R}[f_G] = \min_{\alpha_i\in\mathbb{R}^{d_{g_i}}} \mathcal{R}[\sum_{g_i\in G_j}\alpha_i^T h_{g_i}] \le \min_{\beta\in\mathbb{R}^{d_{g_j}}} \mathcal{R}[f_{G_{j-1}} + \beta h_{g_j}]$$

Setting $\beta = \arg\min_{\beta\in\mathbb{R}^{d_{g_j}}} \mathcal{R}[f_{G_{j-1}} + \beta h_{g_j}]$, using the strongly convex condition at $f_{G_{j-1}}$, we have:

$$\begin{aligned}&F(G_j) - F(G_{j-1}) \\ &= \mathcal{R}[f_{G_{j-1}}] - \mathcal{R}[f_{G_j}] \ge \mathcal{R}[f_{G_{j-1}}] - \mathcal{R}[f_{G_{j-1}} + \beta h_{g_j}] \\ &\ge \mathcal{R}[f_{G_{j-1}}] - (\mathcal{R}[f_{G_{j-1}}] + \langle\nabla\mathcal{R}[f_{G_{j-1}}], \beta h_{g_j}\rangle\end{aligned}$$

$$
\begin{aligned}
&+ \frac{M}{2}\|\beta h_{g_j}\|_2^2) \\
&= -\langle \nabla \mathcal{R}[f_{G_{j-1}}], \beta h_{g_j}\rangle - \frac{M}{2}\|\beta h_{g_j}\|_2^2 \\
&= \langle b_{g_j}^{G_{j-1}}, \beta\rangle - \frac{M}{2}\langle \beta, C_{g_j}\beta\rangle \\
&\geq \frac{1}{2M}\langle b_{g_j}^{G_{j-1}}, C_{g_j}^{-1} b_{g_j}^{G_{j-1}}\rangle \\
&= \frac{1}{2M(1+\lambda)}\langle b_{g_j}^{G_{j-1}}, b_{g_j}^{G_{j-1}}\rangle
\end{aligned}
$$

The last equality holds because each group is whitened, so that $C_{g_j} = (1+\lambda)I$. □

Note that the $(1+\lambda)$ constant is a result of group whitening, without which the constant can be as large as $(D_{g_j} + \lambda)$ for the worst case where all the D_{g_j} number of features are the same.

The proofs above for Lemma 3.6.1 and 3.6.2 are for one-dimensional output responses. They can be easily generalized to multi-dimensional responses by replacing 2-norms with Frobenius norms and vector inner-products with “Frobenius products”, i.e., the sum of the products of all elements.

3.6.3 Proof of Main Theorem

Given Lemma 3.6.1 and Lemma 3.6.2, the proof of Lemma 3.3.1 holds with the same analysis with a more general constant $\gamma = \frac{m\lambda_{min}(C)}{M(1+\lambda)}$. The following prove our main theorem 3.3.2.

Proof. (of Theorem 3.3.2, given Lemma 3.3.1) Define $\Delta_j = F(S_{\langle K\rangle}) - F(G_{j-1})$. Then we have $\Delta_j - \Delta_{j+1} = F(G_j) - F(G_{j-1})$. By Lemma 3.3.1, we have:

$$
\begin{aligned}
\Delta_j &= F(S_{\langle K\rangle}) - F(G_{j-1}) \\
&\leq \frac{K}{\gamma}[\frac{F(G_j) - F(G_{j-1})}{c(g_j)}] = \frac{K}{\gamma}[\frac{\Delta_j - \Delta_{j+1}}{c(g_j)}]
\end{aligned}
$$

Rearranging we get $\Delta_{j+1} \leq \Delta_j(1 - \frac{\gamma c(g_j)}{K})$. Unroll we get:

$$
\begin{aligned}
\Delta_{L+1} &\leq \Delta_1 \prod_{j=1}^{L}(1 - \frac{\gamma c(g_j)}{K}) \leq \Delta_1(\frac{1}{L}\sum_{j=1}^{L}(1 - \frac{\gamma c(g_j)}{K}))^L \\
&= \Delta_1(1 - \frac{B\gamma}{LK})^L < \Delta_1 e^{-\gamma\frac{B}{K}}
\end{aligned}
$$

By definition of Δ_1 and Δ_{L+1}, we have:

$$
F(S_{\langle K\rangle}) - F(G_{\langle B\rangle}) < F(S_{\langle K\rangle})e^{-\gamma\frac{B}{K}}
$$

The theorem follows and linear prediction is the special case that $m = M$. □

3.7 Extension to Generalized Linear Models

While we only formulated the feature group sequencing problem in linear prediction setting previously, we can extend our algorithm for generalized linear modelsMcCullagh and Nelder (1989) and multi-dimensional responses. In general, we assume that we have P dimensional responses, and predictions are of the form $E[y|x] = \nabla\phi(Wx)$, for some known convex function $\phi : \mathbb{R}^P \to \mathbb{R}$, and an unknown coefficient $P \times D$ matrix, W. Thus, the generalized linear prediction problem is to minimize over coefficient matrix $W : P \times D$:

$$\mathbf{r}(W) = \frac{1}{n}\sum_{i=1}^{n}(\phi(Wx^i) - {y^i}^T Wx^i) + \frac{\lambda}{2}\|W\|_F^2, \tag{3.27}$$

where λ is the regularization constant for Frobenius norm of the coefficient matrix. In particular, we have $\phi(x) = \frac{1}{2}x^2$ for linear prediction. The risk of a collection of features, S, is then $R(S) = \min_{W:\forall g \notin S W_g = \mathbf{0}} \mathbf{r}(W)$. To extend CS-G-OMP to feature sequencing in this general setting, we again, at each step, take gradient of the objective $\mathbf{r}$ w.r.t. W, and choose the feature group that has the largest ratio of group gradient Frobenius norm square to group cost. More specifically, after choosing groups in G, we have a best coefficient matrix restricted to G, $W(G)$. Then we compute the gradient w.r.t. W at $W(G)$ (we keep the convention that unselected groups have zero coefficients) as:

$$\nabla\mathbf{r}(W) = \frac{1}{n}\sum_{i=1}^{n}(\nabla\phi(Wx^i) - y^i){x^i}^T + \lambda W; \tag{3.28}$$

we then evaluate $\|\mathbf{r}(W)_g\|_F^2/c(g)$ for each feature group g, and add the maximizer to the selected groups to create new models. Algorithm 5 demonstrates the procedure.

Our theoretical result Theorem 3.3.2 can also be proven in this general setting. Proofs of Lemma 3.3.3 and 3.3.4) in appendix are readily for generalized linear models[4]. Given these two lemmas, our proofs of Lemma 3.3.1 and Theorem 3.3.2 hold as they are.

3.7.1 EXAMPLE EXPERIMENTS ON GLM

We present here experimental results of CS-G-OMP with generalized linear models on a MNIST database of handwritten digit classification (LeCun et al., 2001). We generate features from raw digit pixels following the recent development of Karampatziakis and Mineiro (2014). It generates about 11,000 dimensional features via generalized eigenvectors of pairs of second moments of the raw pixel values of different classes, and achieves one-percent error rate with logistic regressions. We partition the generated features into 54 equal-sized random feature groups, and apply CS-G-OMP with multi-class-logistic regression, targeting the one-hot encodings of sample labels. Mathematically, we choose our mean function of generalized linear model, $\nabla\phi : \mathbb{R}^P \to \mathbb{R}^P$, as $\nabla\phi(x) = \frac{exp(x)}{\sum_p^P exp(x)_p}$ for Algorithm 5, where $exp(x)$ is an element-wise exponential function.

[4]Inner products, $\langle \bullet, \bullet \rangle$, in Lemma 3.3.3 and 3.3.4 now represent Frobenius products, which are sums of element-wise products of matrices.

Algorithm 5 Cost Sensitive Group Orthogonal Matching Pursuit For Generalized Linear Model

```
1:  Input: The data matrix $\mathbf{X} = [\mathbf{f}_1, ..., \mathbf{f}_D] \in \mathbb{R}^{n \times D}$, with group structures, such that for each
    group $g$, $\mathbf{X}_g^T \mathbf{X}_g = I_{D_g}$. The cost $c(g)$ of each group $g$. The response matrix $\mathbf{Y} \in \{0,1\}^{n \times P}$.
    The link function $\nabla\phi$. Regularization constant $\lambda$.
2:  Output: A sequence $((G_j, W_j))_j$, where $G_j = (g_1, g_2, ..., g_j)$ is the sequence of first $j$
    selected feature groups, $g_1, g_2, ..., g_j$, and $W_j : P \times D$ restricted to features in $G_j$ is the
    associated coefficient matrix.
3:  Set $G_0 = \emptyset$ to be an empty sequence.
4:  Set $w(G_0) = \vec{0}$ to be a zero matrix of zero input size and $P$ output size.
5:  Compute $C = X^T X$.
6:  for $j = 1, 2, ..., J$ do
7:      Set $W = W(G_{j-1})$.
8:      for $g \notin G_{j-1}$ do
9:          Compute with Eq. 3.28: $\mathbf{r}' = \nabla \mathbf{r}(W) = \frac{1}{n}\sum_{i=1}^{n}(\nabla\phi(Wx^i) - y^i)x^{iT} + \lambda W$.
10:     end for
11:     $g_j = \operatorname*{arg\,max}_{g=\mathcal{G}_1,...,\mathcal{G}_J, g \notin G_{j-1}} \frac{\|\mathbf{r}'_g\|_F^2}{c(g)}$.
12:     Append $g_j$ to the sequence: $G_j = G_{j-1} \oplus g_j$.
13:     Compute $W(G_j) = \operatorname*{arg\,min}_{W : \forall g \notin G_j W_g = \mathbf{0}} R(W)$.
14: end for
```

As shown in Figure 5.2e, the test-time accuracy improves greatly at start, quickly reducing the number of mistakes below 150 (i.e., 98.5% accuracy with the 10K test samples of MNIST) with 2200 out of the 11k total features, and plateaus between 105 and 100 mistakes with 6k features and beyond. The peak performance is 99 mistakes, and the final result has 101 mistakes. Since logistic regression with 11K features and 60K training samples takes non-trivial time to train, the runtime gap between CS-G-OMP and CS-G-FR further widens: CS-G-OMP is able to finish the sequencing in 12 hours, while CS-G-FR takes days to progress. This is because the model training time is orders of magnitudes longer than that of computing gradient w.r.t. the coefficient matrix. In fact, one selection of CS-G-FR takes longer than the full run of CS-G-OMP. As a result, we do not report CS-G-FR result on this data-set.

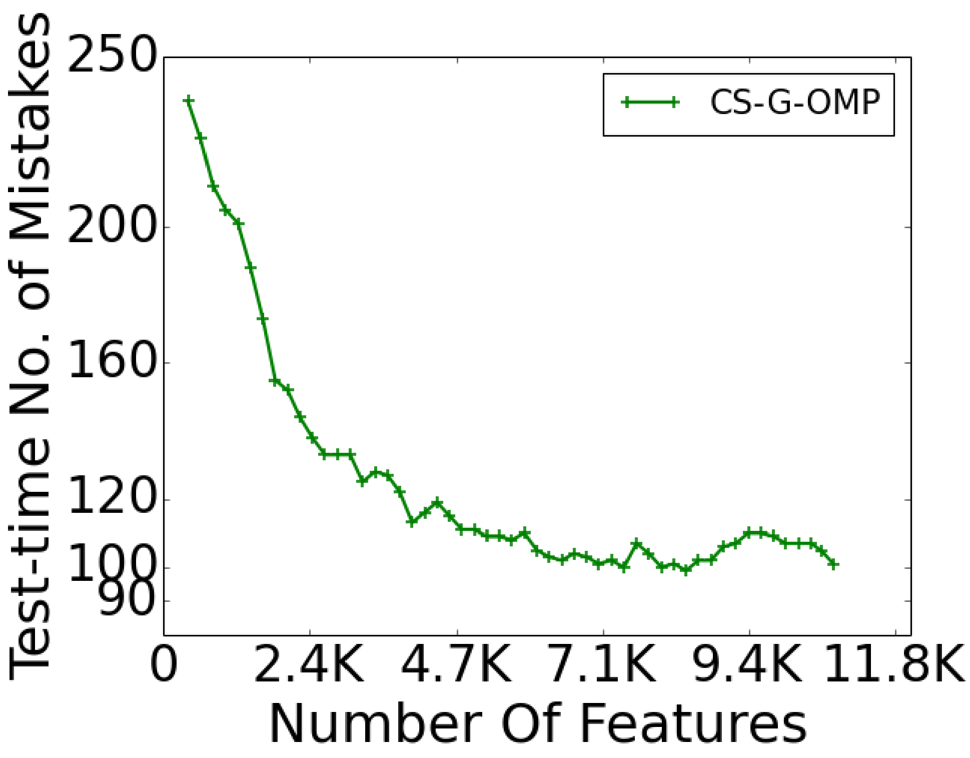


Figure 3.6: CS-G-OMP test-time performance on MNIST. We note that CS-G-FR cannot be computed easily in this case and is omitted.

Chapter 4

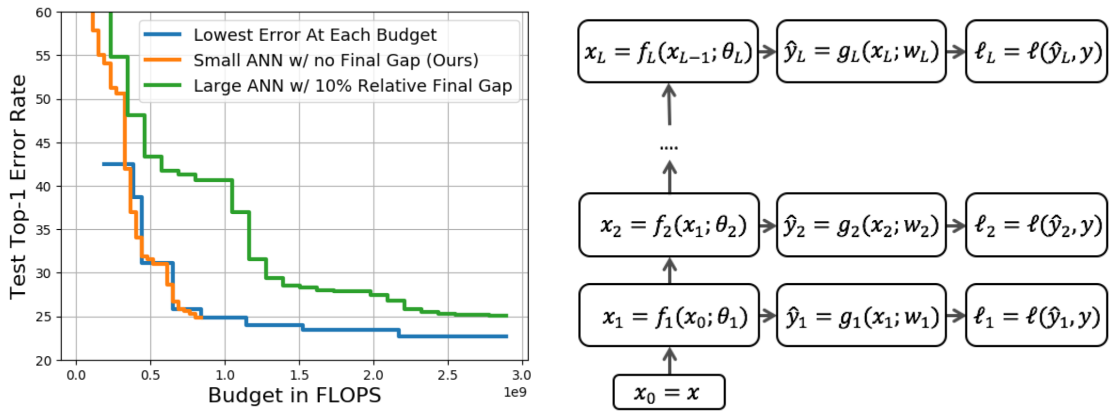
Anytime Neural Network via Adaptive Loss Balancing

4.1 Introduction

Recent years have seen advancement in visual recognition tasks by increasingly accurate convolutional neural networks, from AlexNet Krizhevsky et al. (2012) and VGG Simonyan and Zisserman (2015), to ResNet He et al. (2016), ResNeXt Xie et al. (2017), and DenseNet Huang et al. (2017b). As models become more accurate and computationally expensive, it becomes more difficult for applications to choose between slow predictors with high accuracy and fast predictors with low accuracy. Some applications also desire multiple trade-offs between computation and accuracy, because they have computational budgets that may vary at test time. E.g., web servers for facial recognition or spam filtering may have higher load during the afternoon than at midnight. Autonomous vehicles need faster object detection when moving rapidly than when it is stationary. Furthermore, real-time and latency sensitive applications may desire fast predictions on easy samples and slow but accurate predictions on difficult ones.

An **anytime predictor** Boddy and Dean (1989); Grass and Zilberstein (1996); Grubb and Bagnell (2012b); Horvitz (1987); Huang et al. (2018b) can automatically trade off between computation and accuracy. For each test sample, an anytime predictor produces a fast and crude initial prediction and continues to refine it as budget allows, so that at any test-time budget, the anytime predictor has a valid result for the sample, and the more budget is spent, the better the prediction. Anytime predictors are different from cascaded predictors Bolukbasi et al. (2017); Cai et al. (2015); Guan et al. (2017); Viola and Jones (2001b); Xu et al. (2014) for **budgeted prediction**, which aim to minimize **average test-time computational cost** without sacrificing average accuracy: a different task (with relation to anytime prediction). Cascades achieve this by early exiting on easy samples to save computation for difficult ones, but cascades cannot incrementally improve individual samples after an exit. Furthermore, early exit policy of cascades can be combined with existing anytime predictors Bolukbasi et al. (2017); Guan et al. (2017). Hence, we consider cascades to be orthogonal to anytime predictions.

This work studies how to convert well-known DNN architectures to produce competitive anytime predictions. We form anytime neural networks (ANNs) by appending auxiliary predictions



(a) (b)

Figure 4.1: (a) The common ANN training strategy increases final errors from the optimal (green vs. blue), which decreases exponentially slowly. By learning to focus more on the final auxiliary losses, the proposed adaptive loss weights make a small ANN (orange) to outperform a large one (green) that has non-adaptive weights. (b) Anytime neural networks contain auxiliary predictions and losses, $\hat{y}_i$ and ℓ_i, for intermediate feature unit f_i.

and losses to DNNs, as we will detail in Sec. 4.3 and Fig. 4.1b. Inference-time prediction then can be stopped at the latest prediction layer that is within the budget. Note that this work deals with the case where it is **not known apriori** where the interrupt during inference time will occur. We define the optimal at each auxiliary loss as the result from training the ANN only for that loss to convergence. Then our objective is to have near-optimal final predictions and competitive early ones. Near-optimal final accuracy is imperative for anytime predictors, because, as demonstrated in Fig. 4.1a, accuracy gains are often exponentially more expensive as model sizes grow, so that reducing 1% error rate could take 50% extra computation. Unfortunately, existing anytime predictors often optimize the anytime losses in static weighted sums Huang et al. (2018b); Lee et al. (2015); Zamir et al. (2017) that poorly optimize final predictions, as we will show in Sec. 4.3 and Sec. 4.5.

Instead, we optimize the losses in an **adaptive** weighted sum, where the weight of a loss is inversely proportional to the empirical mean of the loss on the training set. Intuitively, this normalizes losses to have the same scale, so that the optimization leads each loss to be about the same relative to its optimal. We provide multiple theoretical considerations to motivate such weights. First of all, when the losses are mean square errors, our approach is maximizing the likelihood of a model where the prediction targets have Gaussian noises. Secondly, inspired by the maximum likelihood estimation, we optimize the model parameters and the loss weights jointly, with log-barriers on the weights to avoid the trivial solution of zero weights. Finally, we find the joint optimization equivalent to optimizing the geometric mean of the expected training losses, an objective that treats the relative improvement of each loss equally. Empirically, we show on multiple models and visual recognition data-sets that the proposed adaptive weights outperform natural, non-adaptive weighting schemes as follows. We compare small ANNs using our adaptive weights against ANNs that are $50 \sim 100\%$ larger but use non-adaptive weights. The small ANNs can reach the same final accuracy as the larger ones, and reach each accuracy level faster.

Early and late accuracy in an ANN are often anti-correlated (e.g., Fig. 7 in Huang et al. (2018b) shows ANNs with better final predictions have worse early ones). To mitigate this *fundamental* issue we propose to assemble ANNs of exponentially increasing depths. If ANNs are near-optimal in a late fraction of their layers, the exponential ensemble only pays a constant fraction of additional computation to be near-optimal at every test-time budget. In addition, exponential ensembles outperform linear ensembles of networks, which are commonly used baselines for existing works Huang et al. (2018b); Zamir et al. (2017). In summary our contributions are:

- We derive an adaptive weight scheme for training losses in ANNs from multiple theoretical considerations, and show that experimentally this scheme achieves near-optimal final accuracy *and* competitive anytime ones on multiple data-sets and models.
- We assemble ANNs of exponentially increasing depths to achieve near-optimal anytime predictions at every budget at the cost of a constant fraction of additional consumed budget.

4.2 Related Works

Meta-algorithms for anytime and budgeted prediction. Anytime and budgeted prediction has a rich history in learning literature. Weinberger et al. (2009); Xu et al. (2012; 2013a) sequentially generate features to empower the final predictor. Grubb and Bagnell (2012b); Hu et al. (2016); Reyzin (2011) apply boosting and greedy methods to order feature and predictor computation. Karayev et al. (2012); Odena et al. (2017) form Markov Decision Processes for computation of weak predictors and features, and learn policies to order them. However, these meta-algorithms are not easily compatible with complex and accurate predictors like DNNs, because the anytime predictions without DNNs are inaccurate, and there are no intermediate results during the computation of the DNNs. Cascade designs for budgeted prediction Bolukbasi et al. (2017); Cai et al. (2015); Chen et al. (2012a); Guan et al. (2017); Lefakis and Fleuret (2010); Nan and Saligrama (2017); Viola and Jones (2001b); Xu et al. (2014) reduce the average test-time computation by early exiting on easy samples and saving computation for difficult ones. As cascades build upon existing anytime predictors, or combine multiple predictors, they are orthogonal to learning ANNs end-to-end.

Neural networks with early auxiliary predictions. Multiple works have addressed training DNNs with early auxiliary predictions for various purposes. Larsson et al. (2017a); Lee et al. (2015); Szegedy et al. (2017); Zhao et al. (2017) use them to regularize the networks for faster and better convergence. Bengio et al. (2009); Zamir et al. (2017) set the auxiliary predictions from easy to hard for curriculum learning. Chen and Koltun (2017); Xie and Tu (2015) make pixel level predictions in images, and find learning early predictions in coarse scales also improve the fine resolution predictions. Huang et al. (2018b) shows the crucial importance of maintaining multi-scale features for high quality early classifications. The above works use manually-tuned static weights to combine the auxiliary losses, or change the weights only once Chen and Koltun (2017). This work proposes adaptive weights to balance the losses to the same scales online, and provides multiple theoretical motivations. We empirically show adaptive losses induce better ANNs on multiple models, including the state-of-the-art anytime predictor for image recognition, MSDNet Huang et al. (2018b).

Model compression. Many works have studied how to compress neural networks. Li et al. (2017); Liu et al. (2017b) prune network weights and connections. Hubara et al. (2016); Iandola et al. (2016); Rastegari et al. (2016) quantize weights within networks to reduce computation and memory footprint. Veit and Belongie (2017); Wang et al. (2017) dynamically skip network computation based on samples. Ba and Caruana (2014); Hinton et al. (2014) transfer knowledge of deep networks into shallow ones by changing the training target of shallow networks. These works are orthogonal to ours, because they train a separate model for each trade-off between computation and accuracy, but we train a single model to handle all possible trade-offs.

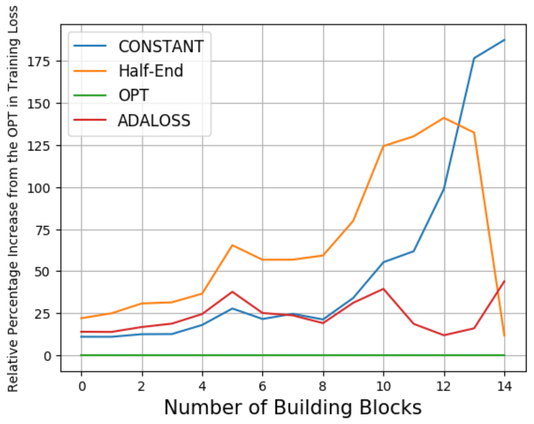
4.3 Optimizing Anytime Predictors in Networks

As illustrated in Fig. 4.1b, a feed-forward network consists of a sequence of transformations $f_1, ..., f_L$ of feature maps. Starting with the input feature map x_0, each subsequent feature map is generated by $x_i = f_i(x_{i-1})$. Typical DNNs use the final feature map x_L to produce predictions, and hence require the completion of the whole network for results. Anytime neural networks (ANNs) instead introduce auxiliary predictions and losses using the intermediate feature maps $x_1, ..., x_{L-1}$, and thus, have early predictions that are improving with computation.

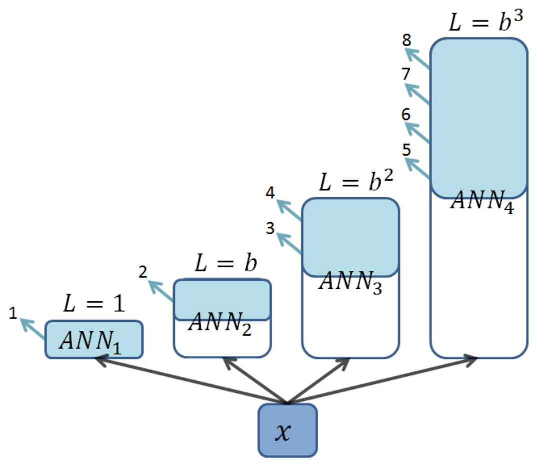
Weighted sum objective. Let the intermediate predictions be $\hat{y}_i = g_i(x_i)$ for some function g_i, and let the corresponding expected loss be $\ell_i = E_{(x_0,y)\sim\mathcal{D}}[\ell(y, \hat{y}_i)]$, where $\mathcal{D}$ is the distribution of the data, and ℓ is some loss such as cross-entropy. Let θ be the parameter of the ANN, and define the optimal loss at prediction $\hat{y}_i$ to be $\ell_i * = \min_\theta \ell_i(\theta)$. Then the goal of anytime prediction is to seek a universal $\theta^* \in \cap_{i=1}^{L}\{\theta' : \theta' = \arg\min_\theta \ell_i(\theta)\}$. Such an ideal θ^* does not exist in general as this is a multi-objective optimization, which only has Pareto front, a set containing all solutions such that improving one ℓ_i necessitates degrading others. Finding all solutions in the Pareto front for ANNs is not practical or useful, since this requires training multiple models, but each ANN only runs one. Hence, following previous works on anytime models Huang et al. (2018b); Lee et al. (2015); Zamir et al. (2017), we optimize the losses in a weighted sum $\min_\theta \sum_{i=1}^{L} B_i \ell_i(\theta)$, where B_i is the weight of the loss ℓ_i. We call the choices of B_i *weight schemes*.

Static weight schemes. Previous works often use static weight schemes as part of their formulation. Huang et al. (2018b); Lee et al. (2015); Xie and Tu (2015) use CONST scheme that sets $B_i = 1$ for all i. Zamir et al. (2017) use LINEAR scheme that sets B_1 to B_L to linearly increase from 0.25 to 1. However, as we will show in Sec. 4.5.2, these static schemes not only cannot adjust weights in a data and model-dependent manner, but also may significantly degrade predictions at later layers.

Qualitative weight scheme comparison. Before we formally introduce our proposed adaptive weights, we first shed light on how existing static weights suffer. We experiment with a ResNet of 15 basic residual blocks on CIFAR100 Krizhevsky et al. (2009) data-set (See Sec. 4.5 for data-set details). An anytime predictor is attached to each residual block, and we estimate the optimal performance (OPT) in training cross entropy of predictor i by training a network that has weight only on ℓ_i to convergence. Then for each weight scheme we train an ANN to measure the relative increase in training loss at each depth i from the OPT. In Fig. 4.2a, we observe that the intuitive CONST scheme has high relative losses in late layers. This indicates that there is not enough weights in the late layers, though losses have the same B_i. We also note that balancing



(a)



(b)

Figure 4.2: (a) Relative Percentage Increase in Training Loss vs. depths (lower is better). CONST scheme is increasingly worse than the optimal at deep layers. AdaLoss performs about equally well on all layers in comparison to the OPT. (b)Ensemble of exponentially deepening anytime neural network (EANN) computes its ANNs in order of their depths. An anytime result is used if it is better than all previous ones on a validation set (layers in light blue).

the weights is non-trivial. For instance, if we put half of the total weights in the final layer and distribute the other half evenly, we get the "Half-End" scheme. As expected, the final loss is improved, but this is at the cost of significant increases of early training losses. In contrast, the adaptive weight scheme that we propose next (AdaLoss), achieves roughly even relative increases in training losses automatically, and is much better than the CONST scheme in the late layers.

Adaptive Loss Balancing (AdaLoss). Given all losses are of the same form (cross-entropy), it may be surprising that better performance is achieved with differing weights. Because early features typically have less predictive power than later ones, early losses are naturally on a larger scale and possess larger gradients. Hence, if we weigh losses equally, early losses and gradients often dominate later ones, and the optimization becomes focused on the early losses. To automatically balance the weights among the losses of different scales, we propose an adaptive loss balancing scheme (AdaLoss). Specifically, we keep an exponential average of each loss $\hat{\ell}_i$ during training, and set $B_i \propto \frac{1}{\hat{\ell}_i}$. This is inspired by Chen and Koltun (2017), which scales the losses to the same scale *only once* during training, and provides a brief intuitive argument: the adaptive weights set the losses to be on the same scale. We next present multiple theoretical justifications for AdaLoss.

Before considering general cases, we first consider a simple example, where the loss function $\ell(y, \hat{y}) = \|y - \hat{y}\|_2^2$ is the square loss. For this example, we model each $y|x$ to be sampled from the multiplication of L independent Gaussian distributions, $\mathcal{N}(\hat{y}_i, \sigma_i^2 I)$ for $i = 1, ..., L$, where $\hat{y}_i(x;\theta)$ is the i^{th} prediction, and $\sigma_i^2 \in \mathbb{R}^+$, i.e., $Pr(y|x;\theta,\sigma_1^2,...,\sigma_L^2) \propto \prod_{i=1}^{L} \frac{1}{\sqrt{\sigma_i^2}} \exp(-\frac{\|y-\hat{y}_i\|_2^2}{2\sigma_i^2})$. Then we compute the empirical expected log-likelihood for a maximum likelihood estimator (MLE):

$$\hat{E}\big[\ln(Pr(y|x))\big] \propto \hat{E}\big[\sum_{i=1}^{L}(-\frac{\|y-\hat{y}_i\|_2^2}{\sigma_i^2} - \ln\sigma_i^2)\big] \tag{4.1}$$

$$= \sum_{i=1}^{L}(-\frac{\tilde{\ell}_i}{\sigma_i^2} - \ln\sigma_i^2), \tag{4.2}$$

where $\hat{E}$ is averaging over samples, and $\tilde{\ell}_i$ is the empirical estimate of ℓ_i. If we fix θ and optimize over σ_i^2, we get $\sigma_i^2 = \tilde{\ell}_i$. As computing the empirical means is expensive over large data-sets, AdaLoss replaces $\tilde{\ell}_i$ with $\hat{\ell}_i$, the exponential moving average of the losses, and sets $B_i \propto \hat{\ell}_i^{-1} \approx \sigma_i^{-2}$ so as to solve the MLE online by jointly updating θ and B_i. We note that the naturally appeared $\ln\sigma_i^2$ terms in Eq. 4.2 are log-barriers preventing $B_i = 0$.

Inspired by this observation, we form the following joint optimization over θ and B_i for general losses without probability models:

$$\min_{\theta, B_1,\ldots,B_L} \sum_{i=1}^{L}(B_i\ell_i(\theta) - \lambda\ln B_i), \tag{4.3}$$

where $\lambda > 0$ is a hyper parameter to balance between the log-barriers and weighted losses. Under the optimal condition, $B_i = \frac{\lambda}{\ell_i}$. AdaLoss estimates this with $B_i \propto \hat{\ell}_i(\theta)^{-1}$. We can also eliminate B_i from Eq. 4.3 under the optimal condition, and we transform Eq. 4.3 to the following problem:

$$\min_{\theta} \sum_{i=1}^{L} \ln\ell_i(\theta). \tag{4.4}$$

This is equivalent to minimizing the geometric mean of the expected training losses, and it differs from minimizing the expected geometric mean of losses, as ln and expectation are not commutable. Eq. 4.4 discards any constant scaling of losses automatically discarded as constant offsets, so that the scale difference between the early and late losses are automatically reconciled. Geometric mean is also known as the canonical mean for multiple positive quantities of various scales. AdaLoss optimizes Eq. 4.4, since the objective gradient is $\sum_{i=1}^{L}\frac{\nabla\ell_i(\theta)}{\ell_i(\theta)}$. AdaLoss wants to weigh each $\ell_i(\theta)$ by exactly $\frac{1}{\ell_i(\theta)}$, and estimates the weight by $\frac{1}{\hat{\ell}_i(\theta)}$. This concludes our theoretical considerations for AdaLoss.

4.4 Ensemble of Exponentially Deepening Networks

In practice, we often observe ANNs using AdaLoss to be much more competitive in their later half than the early half on validation sets, such as in Table. 4.1 of Sec. 4.5.2. Fortunately, we can leverage this effect to form competitive anytime predictors at every budget, with a constant fraction of additional computation. Specifically, we assemble ANNs whose depths grow exponentially. Each ANN only starts computing if the smaller ones are finished, and its predictions are used if they are better than the best existing ones in validation. We call this ensemble an **EANN**, as illustrated in Fig. 4.2b. An EANN only delays the computation of any large ANN by at most a constant fraction of computation, because the earlier networks are exponentially smaller. Hence, if each ANN is near-optimal in later predictions, then we can achieve near-optimal accuracy at any test-time interruption, with the extra computation. Formally, the following proposition characterizes the exponential base and the increased computational cost.

Proposition 4.4.1. *Let $b > 1$. Assume for any L, any ANN of depth L has competitive anytime prediction at depth $i > \frac{L}{b}$ against the optimal of depth i. Then after B layers of computation, EANN produces anytime predictions that are competitive against the optimal of depth $\frac{B}{C}$ for some $C > 1$, such that $\sup_B C = 2 + \frac{1}{b-1}$, and C has expectation $E_{B \sim uniform(1,L)}[C] \leq 1 - \frac{1}{2b} + \frac{1+\ln(b)}{b-1}$.*

This proposition says that an EANN is competitive at any budget B against the optimal of the cost $\frac{B}{C}$. Furthermore, the stronger each anytime model is, i.e., the larger b becomes, the smaller the computation inflation, C, is: as b approaches ∞, $\sup_B C$, shrinks to 2, and $E[C]$, shrinks to 1. Moreover, if we have M number of parallel workers instead of one, we can speed up EANNs by computing ANNs in parallel in a first-in-first-out schedule, so that we effectively increase the constant b to b^M for computing C. It is also worth noting that if we form the sequence using regular networks instead of ANNs, then we will lose the ability to output frequently, since at budget B, we only produce $\Theta(\log(B))$ intermediate predictions instead of the $\Theta(B)$ predictions in an EANN. We will further have a larger cost inflation, C, such that $\sup_B C \geq 4$ and $E[C] \geq 1.5 + \sqrt{2} \approx 2.91$, so that the average cost inflation is at least about 2.91. We defer the proofs to the appendix.

4.5 Experiments

We list the key questions that our experiments aim to answer.

- How do anytime predictions trained with adaptive weights compare against those trained with static constant weights (over different architectures)? (Sec. 4.5.2)
- How do underlying DNN architectures affect ANNs? (Sec. 4.5.2)
- How can sub-par early predictions in ANNs be mitigated by ANN ensembles? (Sec. 4.5.3)
- How does data-set difficulty affect the adaptive weights scheme? (Sec. 4.5.4)

4.5.1 Data-sets and Training Details

Data-sets. We experiment on CIFAR10, CIFAR100 Krizhevsky et al. (2009), SVHN Netzer et al. (2011)[1] and ILSVRC Russakovsky et al. (2015)[2].

[1]Both CIFAR data-sets consist of 32x32 colored images. CIFAR10 and CIFAR100 have 10 and 100 classes, and each have 50000 training and 10000 testing images. We held out the last 5000 training samples in CIFAR10 and CIFAR100 for validation; the same parameters are then used in other models. We adopt the standard augmentation from He et al. (2016); Lee et al. (2015). SVHN contains around 600000 training and around 26032 testing 32x32 images of numeric digits from the Google Street Views. We adopt the same pad-and-crop augmentations of CIFAR for SVHN, and also add Gaussian blur.

[2] ILSVRC2012 Russakovsky et al. (2015) is a visual recognition data-set containing around 1.2 million natural and 50000 validation images for 1000 classes. We report the top-1 error rates on the validation set using a single-crop of size 224x224, after scaling the smaller side of the image to 256, following He et al. (2016).

	1/4	1/2	3/4	1
OPT	0.00	0.00	0.00	0.00
CONST	**15.07**	16.40	18.76	18.90
LINEAR	25.67	13.02	12.97	12.65
ADALOSS	32.99	**9.97**	**3.96**	**2.73**

Table 4.1: Average relative percentage increase in error from the OPT on CIFAR and SVHN at 1/4, 1/2, 3/4 and 1 of the total cost. E.g., the bottom right entry means that if OPT has a 10% final error rate, then AdaLoss has about 10.27%.

	1/4	1/2	3/4	1
ResANN50+C	**54.34**	35.61	27.23	25.14
ResANN50+A	54.98	**34.92**	**26.59**	**24.42**
DenseANN169+C	48.15	45.00	29.09	25.60
DenseANN169+A	**47.17**	**44.64**	**28.22**	**24.07**
MSDNet38	**33.9**	28.0	25.7	24.3
MSDNet38+A	35.75	28.04	25.82	**23.99**

Table 4.2: Test error rates at different fraction of the total costs on ResANN50, DenseANN169, and MSDNet38 on ILSVRC. The post-fix +C and +A stand for CONST and AdaLoss respectively. Published results of MSDNet38 Huang et al. (2018b) uses CONST.

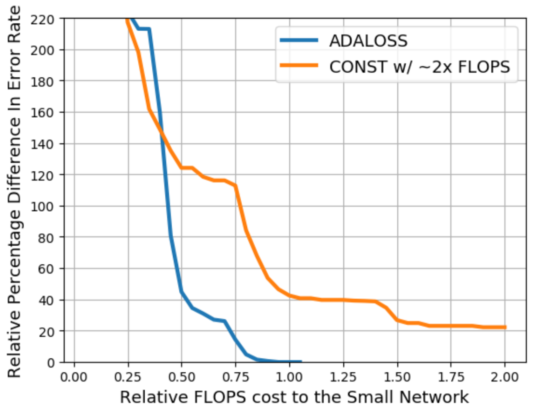
Training details. We optimize the models using stochastic gradient descent, with initial learning rate of 0.1, momentum of 0.9 and a weight decay of 1e-4. On CIFAR and SVHN, we divide the learning rate by 10 at 1/2 and 3/4 of the total epochs. We train for 300 epochs on CIFAR and 60 epochs on SVHN. On ILSVRC, we train for 90 epochs, and divide the learning rate by 10 at epoch 30 and 60. We evaluate test error using single-crop.

Base models. We compare our proposed AdaLoss weights against the intuitive CONST weights. On CIFAR and SVHN, we also compare AdaLoss against LINEAR and OPT, defined in Sec. 4.3. We evaluate the weights on multiple models including ResNet He et al. (2016) and DenseNet Huang et al. (2017b), and MSDNet Huang et al. (2018b). For ResNet and DenseNet, we augment them with auxiliary predictors and losses, and call the resulting models ResANN and DenseANN, and defer the details of these models to the appendix Sec. 4.8.

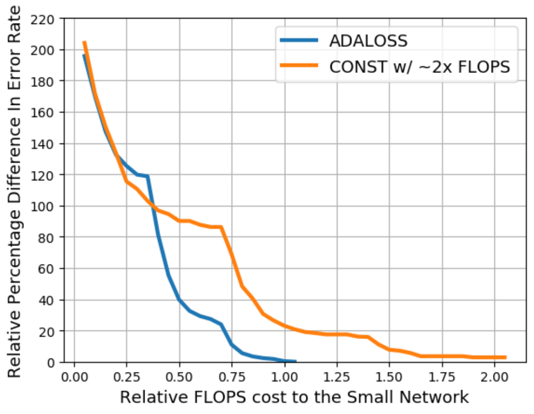
4.5.2 Weight Scheme Comparisons

AdaLoss vs. CONST on the same models. Table 4.1 presents the average relative test error rate increase from OPT on 12 ResANNs on CIFAR10, CIFAR100 and SVHN[3]. As training an OPT for each depth is too expensive, we instead report the average relative comparison at 1/4, 1/2, 3/4, and 1 of the total ANN costs. We observe that the CONST scheme makes $15 \sim 18\%$ more errors than the OPT, and the relative gap widens at later layers. The LINEAR scheme also has about 13%

[3]The 12 models are named by (n, c) drawn from $\{7, 9, 13, 17, 25\} \times \{16, 32\}$ and $\{(9, 64), (9, 128)\}$, where n represents the number of residual units in each of the three blocks of the network, and c is the filter size of the first convolution.

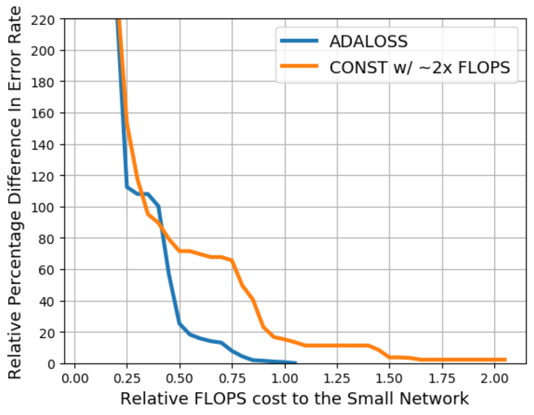


(a) ResANNs on CIFAR10



(b) ResANNs on CIFAR100

Figure 4.3: Comparing small networks with AdaLoss versus big ones using CONST on CIFAR10 and CIFAR100.

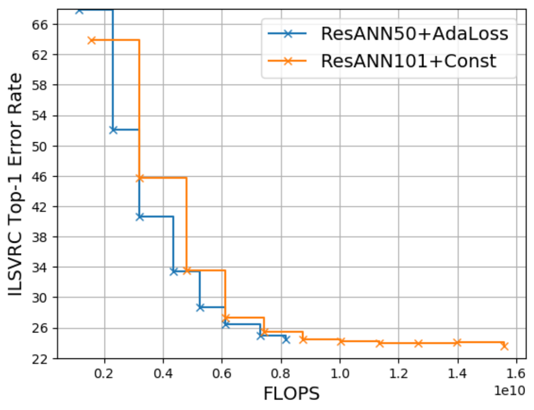


(a) ResANNs on SVHN

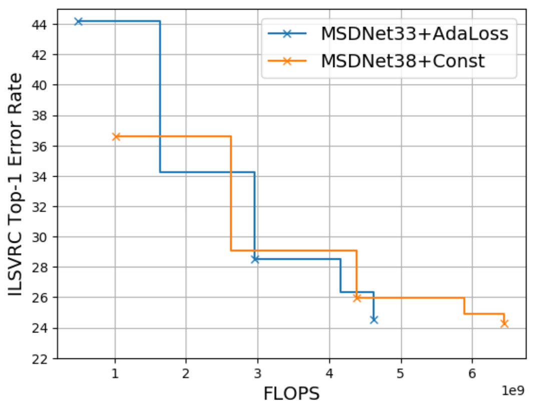
Figure 4.4: Comparing small networks with AdaLoss versus big ones using CONST on SVHN.

relative gap in later layers. In contrast, AdaLoss enjoys small performance gaps in the later half of layers. On ILSVRC, we compare AdaLoss against CONST on ResANN50, DenseANN169, and MSDNet38, which have similar final errors and total computational costs (See Fig. 4.6a). In Table 4.2, we observe the trade-offs between early and late accuracy on ResANN50 and MSDNet38. Furthermore, DenseANN169 performs *uniformly* better with AdaLoss than with CONST. Since comparing the weight schemes requires evaluating ANNs at multiple budget limits, and AdaLoss and CONST outperform each other at a significant fraction of depths on most of our experiments, we consider the two schemes *incomparable on the same model*.

Small networks with AdaLoss vs. large ones with CONST. Our previous comparison between AdaLoss and CONST on the same models is not fully conclusive, since each scheme can outperform the other at a significant portion of the total cost. To address this, we set the final error rate, model architecture type, and the filter size c as constants, and vary the model depths so that



(a) ResANNs on ILSVRC



(b) MSDNet on ILSVRC

Figure 4.5: Comparing small networks with AdaLoss versus big ones using CONST on ILSVRC with ResANNs and MSDNet.

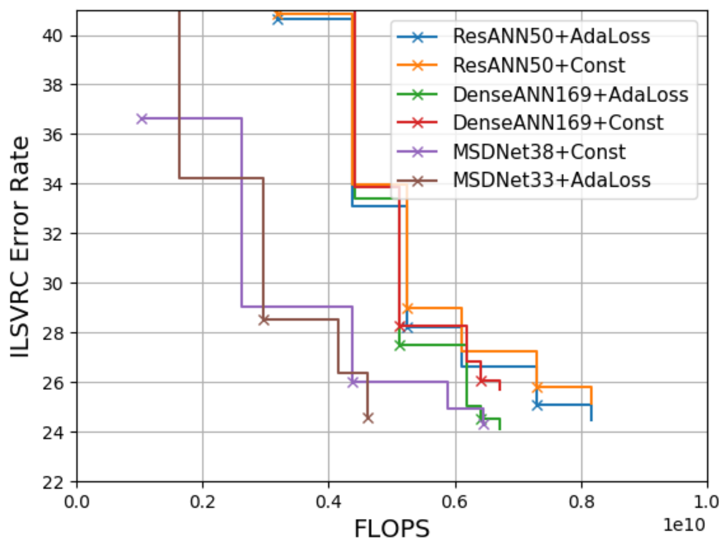
AdaLoss and CONST reach the target final error rate. Then we compare the early predictions and the costs of models. On each of CIFAR10, 100 and SVHN, we compare six pairs of ResANNs, where the CONST uses twice the computation as AdaLoss[4]. Fig. 4.3a, 4.3b, and 4.4a show the averaged relative comparisons[5], and they show that the small ANNs with AdaLoss are better anytime predictors than the large ones with CONST, because both models have the same final accuracy (on CIFAR10, the small ones are even better), and the small models reach the same error rates faster than the large ones. We have similar observations on ILSVRC using ResANNs and MSDNets in Fig. 4.5a and Fig. 4.5b. For instance, MSDNet Huang et al. (2018b) is the state-of-the-art anytime predictor. The published MSDNet38 uses CONST, and has 24.3% error rate using 6.6e9 total FLOPS in convolutions. By switching to AdaLoss, we improve a much smaller MSDNet33 (details in the appendix), which costs 4.5e9 FLOPS, to reach 24.5% final error. The two models also have similar early errors.

AdaLoss can reach the same accuracies with similar or smaller costs than CONST, because in practice, a linear decrease in final error rate may often require an exponential increase in total computation, and CONST degrades the final performances significantly (Table 4.1). Since AdaLoss requires much smaller models than CONST to reach the same final errors, and with a fixed final error rate, AdaLoss reaches each early error rate with less or similar cost, we conclude that AdaLoss is the better scheme for anytime predictions.

Various base networks on ILSVRC. We compare ResANNs, DenseANNs and MSDNets that have final error rate of near 24% in Fig. 4.6a, and observe that the anytime performance is mostly decided by the specific underlying model. MSDNets are more cost-effective than DenseANNs, which in turn are better than ResANNs. However, AdaLoss is helpful regardless of underlying model. Both ResANN50 and DenseANN169 see improvements switching from CONST to AdaLoss, which is also shown in Table 4.2. Thanks to AdaLoss, DenseANN169

[4]AdaLoss takes (n, c) from $\{7, 9, 13\} \times \{16, 32\}$, and CONST takes (n, c) from $\{13, 17, 25\} \times \{16, 32\}$.

[5]The relative plots pivot at the final predictor from AdaLoss, e.g., the location (0.5, 200) means having half the computation and 200% extra relative errors than the final predictor from AdaLoss



(a) ANNs comparison on ILSVRC

Figure 4.6: ANNs performance are mostly decided by underlying models, but AdaLoss is beneficial regardless models.

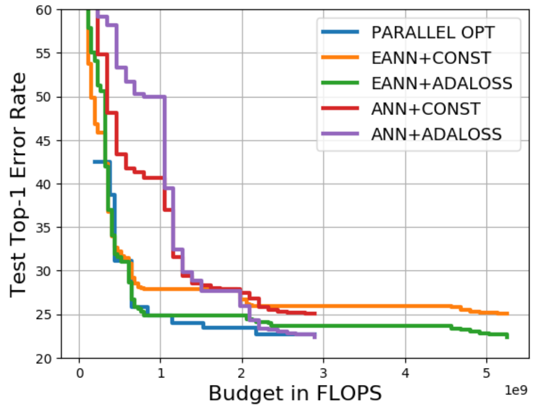
achieves the same final error using similar FLOPS as the original published results of MSDNet38 Huang et al. (2018b). This suggests that Huang et al. (2018b) improve over DenseANNs by having better early predictions without sacrificing the final cost efficiency via impressive architecture insight. AdaLoss brings a complementary improvement to MSDNets, as it enables smaller MSDNets to reach the final error rates of bigger MSDNets, while having similar or better early predictions.

4.5.3 EANN: Closing Early Performance Gaps by Delaying Final Predictions.

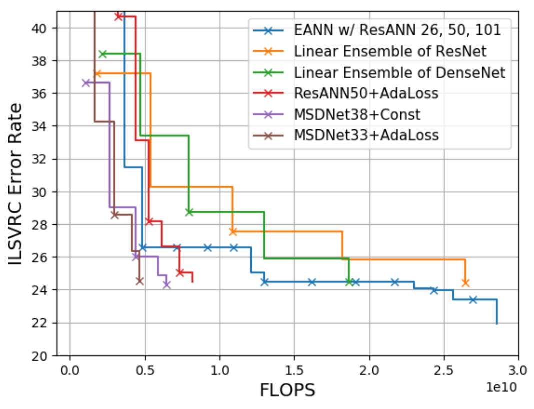
EANNs on CIFAR100. In Fig. 4.7a, we assemble ResANNs to form EANNs[6] on CIFAR100 and make three observations. First, EANNs are better than the ANN in early computation, because the ensembles dedicate early predictions to small networks. Even though CONST has the best early predictions as in Table 4.1, it is still better to deploy small networks. Second, because the final prediction of each network is kept for a long period, AdaLoss leads to significantly better EANNs than CONST does, thanks to the superior final predictions from AdaLoss. Finally, though EANNs delay computation of large networks, it actually appears closer to the OPT, because of accuracy saturation. Hence, EANNs should be considered when performance saturation is severe.

EANN on ILSVRC. Huang et al. (2018b) and Zamir et al. (2017) use ensembles of networks of linearly growing sizes as baseline anytime predictors. However, in Fig. 4.7b, an EANN using ResANNs of depths 26, 50 and 101 outperforms the linear ensembles of ResNets and DenseNets

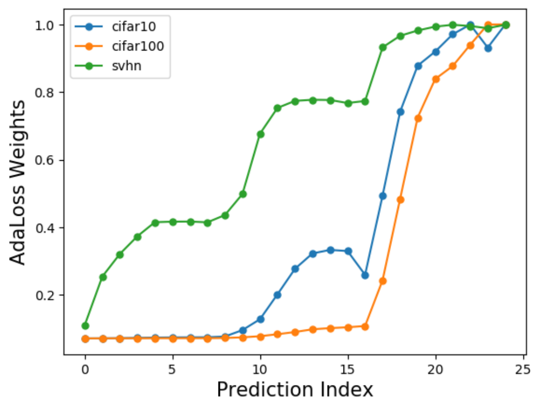
[6]The ResANNs have $c = 32$ and $n = 7, 13, 25$, so that they form an EANN with an exponential base $b \approx 2$. By proposition 4.4.1, the average cost inflation is $E[C] \approx 2.44$ for $b = 2$, so that the EANN should compete against the OPT of $n = 20$, using 2.44 times of original costs.



(a) EANNs on CIFAR100



(b) EANN on ILSVRC



(c) AdaLoss Weights on three data-sets

Figure 4.7: **(a)** EANN performs better if the ANNs use AdaLoss instead of CONST. **(b)** EANN outperforms linear ensembles of DNNs on ILSVRC. **(c)** The learned adaptive weights of the same model on three data-sets.

significantly on ILSVRC. In particular, this drastically reduces the gap between ensembles and the state-of-the-art anytime predictor MSDNet Huang et al. (2018b). Comparing ResANN 50 and the EANN, we note that the EANN achieves better early accuracy but delays final predictions. As the accuracy is not saturated by ResANN 26, the delay appears significant. Hence, EANNs may not be the best when the performance is not saturated or when the constant fraction of extra cost is critical.

4.5.4 Data-set Difficulty versus Adaptive Weights

In Fig. 4.7c, we plot the final AdaLoss weights of the same ResANN model (25,32) on CIFAR10, CIFAR100, and SVHN to study the effects of the data-sets on the weights. We observe that from the easiest data-set, SVHN, to the hardest, CIFAR100, the weights are more concentrated on the final layers. This suggests that AdaLoss can automatically decide that harder data-sets need more concentrated final weights to have near-optimal final performance, whereas on easy data-sets, more efforts are directed to early predictions. Hence, AdaLoss weights may provide information for practitioners to design and choose models based on data-sets.

4.6 Conclusion and Discussion

This work devises simple adaptive weights, AdaLoss, for training anytime predictions in DNNs. We provide multiple theoretical motivations for such weights, and show experimentally that adaptive weights enable small ANNs to outperform large ANNs with the commonly used non-adaptive constant weights. Future works on adaptive weights includes examining AdaLoss for multi-task problems and investigating its "first-order" variants that normalize the losses by individual gradient norms to address unknown offsets of losses as well as the unknown scales. We also note that this work can be combined with orthogonal works in early-exit budgeted predictions Bolukbasi et al. (2017); Guan et al. (2017) for saving average test computation.

4.7 Proof of Propostion 4.4.1

Proof. For each budget consumed x, we compute the cost x' of the optimal that EANN is competitive against. The goal is then to analyze the ratio $C = \frac{x}{x'}$. The first ANN in EANN has depth 1. The optimal and the result of EANN are the same. Now assume EANN is on depth z of ANN number $n+1$ for $n \geq 0$, which has depth b^n.
(Case 1) For $z \leq b^{n-1}$, EANN reuse the result from the end of ANN number n. The cost spent is $x = z + \sum_{i=0}^{n-1} b^i = z + \frac{b^n - 1}{b-1}$. The optimal we compete has cost of the last ANN, which is b^{n-1} The ratio satisfies:

$$\begin{aligned} C = x/x' &= \frac{z}{b^{n-1}} + 1 + \frac{1}{b-1} - \frac{1}{b^{n-1}(b-1)} \\ &\leq 2 + \frac{1}{b-1} - \frac{1}{b^{n-1}(b-1)} < 2 + \frac{1}{b-1}. \end{aligned}$$

Furthermore, since C increases with z,

$$\begin{aligned} &E_{z \sim Uniform(0, b^{n-1})}[C] \\ &\leq b^{1-n} \int_0^{b^{n-1}} z b^{1-n} + 1 + \frac{1}{b-1} \, dz \\ &= 1.5 + \frac{1}{b-1}. \end{aligned}$$

(Case 2) For $b^{n-1} < z \leq b^n$, EANN outputs anytime results from ANN number $n+1$ at depth z. The cost is still $x = z + \frac{b^n - 1}{b-1}$. The optimal competitor has cost $x' = z$. Hence the ratio is

$$C = x/x' = 1 + \frac{b^n - 1}{z(b-1)}$$

$$\leq 2 + \frac{1}{b-1} - \frac{1}{b^{n-1}(b-1)} < 2 + \frac{1}{b-1}.$$

Furthermore, since C decreases with z,

$$\begin{aligned}
&E_{z\sim Uniform(b^{n-1},b^n)}[C] \\
&\leq 1 + \frac{1}{b^n - b^{n-1}} \int_{b^{n-1}}^{b^n} \frac{b^n - 1}{z(b-1)}\, dz \\
&= 1 + \frac{(b - b^{1-n}) \ln b}{(b-1)^2} \\
&< 1 + \frac{b \ln b}{(b-1)^2}
\end{aligned}$$

Finally, since case 1 and case 2 happen with probability $\frac{1}{b}$ and $(1 - \frac{1}{b})$, we have

$$\sup_B C = 2 + \frac{1}{b-1} \tag{4.5}$$

and

$$E_{B\sim Uniform(0,L)}[C] \leq 1 - \frac{1}{2b} + \frac{1}{b-1} + \frac{\ln b}{b-1}. \tag{4.6}$$

We also note that with large b, $\sup_B C \to 2$ and $E[C] \to 1$ from above. $\square$

If we form a sequence of regular networks that grow exponentially in depth instead of ANN, then the worst case happen right before a new prediction is produced. Hence the ratio between the consumed budget and the cost of the optimal that the current anytime prediction can compete, C, right before the number $n+1$ network is completed, is

$$\frac{\sum_{i=1}^{n} b^i}{b^{n-1}} \xrightarrow{n\to\infty} \frac{b^2}{b-1} = 2 + (b-1) + \frac{1}{b-1} \geq 4.$$

Note that $(b-1) + \frac{1}{b-1} \geq 2$ and the inequality is tight at $b = 2$. Hence we know $\sup_B C$ is at least 4. Furthermore, the expected value of C, assume B is uniformly sampled such that the interruption happens on the $(n+1)^{th}$ network, is:

$$\begin{aligned}
E[C] &= \frac{1}{b^n} \int_0^{b^n} \frac{x + \frac{b^n - 1}{b-1}}{b^{n-1}}\, dx \\
&\xrightarrow{n\to\infty} 1.5 + \frac{b-1}{2} + \frac{1}{b-1} \geq 1.5 + \sqrt{2} \approx 2.91.
\end{aligned}$$

The inequality is tight at $b = 1 + \sqrt{2}$. With large n, since almost all budgets are consumed by the last few networks, we know the overall expectation $E_{B\sim Uniform(0,L)}[C]$ approaches $1.5 + \frac{b-1}{2} + \frac{1}{b-1}$, which is at least $1.5 + \sqrt{2}$.

4.8 Implementation Details of ANNs

CIFAR and SVHN ResANNs. For CIFAR10, CIFAR100 Krizhevsky et al. (2009), and SVHN Netzer et al. (2011), ResANN follow He et al. (2016) to have three blocks, each of which has n residual units. Each of such basic residual units consists of two 3x3 convolutions, which are interleaved by BN-ReLU. A pre-activation (BN-ReLU) is applied to the input of the residual units. The result of the second 3x3 conv and the initial input are added together as the output of the unit. The auxiliary predictors each applies a BN-ReLU and a global average pooling on its input feature map, and applies a linear prediction. The auxiliary loss is the cross-entropy loss, treating the linear prediction results as logits. For each (n, c) pair such that $n < 25$, we set the anytime prediction period s to be 1, i.e., every residual block leads to an auxiliary prediction. We set the prediction period $s = 3$ for $n = 25$.

ResANNs on ILSVRC. Residual blocks for ILSVRC are bottleneck blocks, which consists of a chain of 1x1 conv, 3x3 conv and 1x1 conv. These convolutions are interleaved by BN-ReLU, and pre-activation BN-ReLU is also applied. Again, the output of the unit is the sum of the input feature map and the result of the final conv. ResANN50 and 101 are augmented from ResNet50 and 101 He et al. (2016), where we add BN-ReLU, global pooling and linear prediction to every two bottleneck residual units for ResNet50, and every three for ResNet101. We create ResANN26 for creating EANN on ILSVRC, and ResANN26 has four blocks, each of which has two bottleneck residual units. The prediction period is every two units, using the same linear predictors.

DenseANNs on ILSVRC. We augment DenseNet169 Huang et al. (2017b) to create DenseANN 169. DenseNet169 has 82 dense layers, each of which has a 1x1 conv that project concatenation of previous features to $4k$ channels, where k is the growth rate Huang et al. (2017b), followed by a 3x3 conv to generate k channels of features for the dense layer. The two convs are interleaved by BN-ReLU, and a pre-activation BN-ReLU is used for each layer. The 82 layers are organized into four blocks of size 6, 12, 32 and 32. Between each neighboring blocks, a 1x1 conv followed by BN-ReLU-2x2-average-pooling is applied to shrink the existing feature maps by half in the hight, width, and channel dimensions. We add linear anytime predictions every 14 dense layers, starting from layer 12 (1-based indexing). The original DenseNet paper Huang et al. (2017b) mentioned that they use drop-out with keep rate 0.9 after each conv in CIFAR and SVHN, but we found drop-out to be detrimental to performance on ILSVRC.

MSDNet on ILSVRC. MSDNet38 is described in the appendix of Huang et al. (2018b). We set the four blocks to have 10, 9, 10 and 9 layers, and drop the feature maps of the finest resolution after each block as suggest in the original paper. We successfully reproduced the published results to 24.3% error rate on ILSVRC using our Tensorflow implementation. We used the original published results for MSDNet38+CONST in the main text. We use MSDNet33, which has four blocks of 8, 8, 8 and 9 layers, for the small network that uses AdaLoss. We predict using MSDNet33 every seven layers, starting at the fifth layer (1-based indexing).

γ	1/4	1/2	3/4	1	sum
0.0	0.00	0.00	0.00	0.00	0.00
0.05	-20.08	-2.15	2.22	2.43	-17.59
0.15	-23.88	-0.20	5.18	5.17	-13.72

Table 4.3: Relative percentage increase in error rate by switching from $\gamma = 0$. (lower is better.) A small amount of $\gamma = 0.5$ drastically improves early predictions without increasing late error rate much.

EMA m	1/4	1/2	3/4	1
0.9	0.00	0.00	0.00	0.00
0.99	-0.29	0.25	0.05	0.15

Table 4.4: Relative percentage increase in error rate by switching from $m = 0.9$. (lower is better.) The two options essentially result in the same error rates.

4.9 Ablation Study for AdaLoss Parameters

4.9.1 Weight Regularization

In practice, some expected loss ℓ_i could be much larger than the other losses, so that AdaLoss may assign such ℓ_i too small a weight for it to receive enough optimization to recover. To prevent this, we mix the uniform constant weight with AdaLoss as a form of regularization as follows in Eq. 4.7. Such mixture prevents the weight of ℓ_i from being too close to zero.

$$\min_{\theta} \sum_{i=1}^{L} \big(\alpha(1-\gamma)\ln \ell_i(\theta) + \gamma \ell_i(\theta)\big), \tag{4.7}$$

where $\alpha > 0$ and $\gamma > 0$ are hyper parameters. In practice, since DNNs often have elaborate learning rate schedules that assume $B_L = 1$, we choose $\alpha = \min_i \hat{\ell}_i(\theta)$ at each iteration to scale the max weight to 1. We choose $\gamma = 0.05$ from validation sets on CIFAR10 and CIFAR100 from the set $\{0, 0.05, 0.15\}$.

4.9.2 Ablation Study of AdaLoss parameters on CIFAR

Update period e	1/4	1/2	3/4	1
1	0.00	0.00	0.00	0.00
100	0.71	0.23	0.24	0.45

Table 4.5: Relative percentage increase in error rate by switching from $e = 0$. (lower is better.) The options are essentially the same on CIFAR10 and CIFAR100.

We conduct ablation studies for the parameters of AdaLoss: (1) γ in Eq. 4.7, which is the mixture weight of the uniform static weighting, (2) the exponential moving average (EMA) momentum, m, for updating the expected loss $\hat{\ell}_i$ at each stochastic gradient descent (SGD) step, and (3) the number of SGD steps e to wait between updating AdaLoss weights B_i using the learned $\hat{\ell}_i$. We choose $\gamma \in \{0, 0.05, 0.15\}$, $m \in \{0.9, 0.99\}$, and $e \in \{1, 100\}$, and evaluate them on CIFAR10 and CIFAR100 ResANNs whose $n \in \{9, 17, 25\}$ and $c = 32$. Over the 72 experiments, we found the effects of m, and e are almost negligible, as they generate $< 0.5\%$ of relative difference in error rates on average, which translates to 0.1% absolute error difference on CIFAR100. These comparisons are in Table 4.4 and Table 4.5. In the experiment sections, we choose $m = 0.9$ and $e = 1$.

However, γ does affect the performance significantly, as show in Table 4.3. $\gamma = 0$ means pure AdaLoss and $\gamma = 1$ means CONST. We observe that with $\gamma = 0.05$, the small amount of uniform static weight reduces the error rate at 1/4 of the total cost by 20% relatively, but at the cost of minor 2.5% relative increase in late predictions. Increasing γ further to 0.15 has only marginal benefits to early predictions, but has the same negative impact to late accuracy. This suggests that while a small γ helps, we should only use a small amount. Throughout the experiment sections in the main text, we choose $\gamma = 0.05$.

Chapter 5

Training Gradient Boosting on Stochastic Data Streams

5.1 Introduction

Boosting (Freund and Schapire, 1995) is a popular method that leverages simple learning models (e.g., decision stumps) to generate powerful learners. Boosting has been used to great effect and trump other learning algorithms in a variety of applications. In computer vision, boosting was made popular by the seminal Viola-Jones Cascade (Viola and Jones, 2001a) and is still used to generate state-of-the-art results in pedestrian detection (Nam et al., 2014; Yang et al., 2015; Zhu and Peng, 2016). Boosting has also found success in domains ranging from document relevance ranking (Chapelle et al., 2011) and transportation (Zhang and Haghani, 2015) to medical inference (Atkinson et al., 2012). Finally, boosting yields an anytime property at test time, which allows it to work with varying computation budgets (Grubb and Bagnell, 2012a) for use in real-time applications such as controls and robotics.

The advent of large-scale data-sets has driven the need for adapting boosting from the traditional batch setting, where the optimization is done over the whole dataset, to the online setting where the weak learners (models) can be updated with streaming data. In fact, online boosting has received tremendous attention so far. For classification, (Beygelzimer et al., 2015b; Chen et al., 2012b; Oza and Russell, 2001) proposed online boosting algorithms along with theoretical justifications. Recent work by Beygelzimer et al. (2015a), addressed the regression task through the introduction of *Online Gradient Boosting* (OGB). We build upon on the developments in (Beygelzimer et al., 2015a) to devise a new set of algorithms presented below.

In this work, we develop streaming boosting algorithms for regression with strong theoretical guarantees under stochastic setting, where at each round the data are i.i.d sampled from some unknown fixed distribution. In particular, our algorithms are streaming extension to the classic gradient boosting (Friedman, 2001), where weak predictors are trained in a stage-wise fashion to approximate the functional gradient of the loss with respect to the previous ensemble prediction, a procedure that is shown by Mason et al. (2000) to be functional gradient descent of the loss in the space of predictors. Since the weak learners cannot match the gradients of the loss exactly, we measure the error of approximation by redefining of *edge* of online weak learners (Beygelzimer et al., 2015b) for online regression setting.

Assuming a non-trivial edge can be achieved by each deployed weak online learner, we develop algorithms to handle smooth or non-smooth loss functions, and theoretically analyze the convergence rates of our streaming boosting algorithms. Our first algorithm targets strongly convex and smooth loss functions and achieves exponential decay on the average regret with respect to the number of weak learners. We show the ratio of the decay depends on the edge and also the condition number of the loss function. The second algorithm, designed for strongly convex but non-smooth loss functions, extends from the batch residual gradient boosting algorithm from (Grubb and Bagnell, 2011). We show that the algorithm achieves $O(\ln N/N)$ convergence rate with respect to the number of weak learners N, which matches the online gradient descent (OGD)'s no-regret rate for strongly convex loss (Hazan et al., 2007). Both of our algorithms promise that as T (the number of samples) and N go to infinity, the average regret converges to zero. Our analysis leverages Online-to-Batch reduction (Cesa-Bianchi et al., 2004; Hazan and Kale, 2014), hence our results naturally extends to adversarial online learning setting as long as the weak online learning edge holds in adversarial setting, a harsher setting than stochastic setting. We conclude with some proof-of-concept experiments to support our analysis. We demonstrate that our algorithm significantly boosts the performance of weak learners and converges to the performance of classic gradient boosting with less computation.

5.2 Related Works

Online boosting algorithms have been evolving since their batch counterparts are introduced. Oza and Russell (2001) developed some of the first online boosting algorithm, and their work are applied to online feature selection (Grabner and Bischof, 2006) and online semi-supervised learning (Grabner et al., 2008). Leistner et al. (2009) introduced online gradient boosting for the classification setting albeit without a theoretical analysis. Chen et al. (2012b) developed the first convergence guarantees of online boosting for classification. Then Beygelzimer et al. (2015b) presented two online classification boosting algorithms that are proved to be respectively optimal and adaptive.

Our work is most related to (Beygelzimer et al., 2015a), which extends gradient boosting for regression to the online setting under a smooth loss: each weak online learner is trained by minimizing a linear loss, and weak learners are combined using Frank-Wolfe (Frank and Wolfe, 1956) fashioned updates. Their analysis generalizes those of batch boosting for regression (Zhang and Yu, 2005). In particular, these proofs forgo edge assumptions of the weak learners. Though Frank-Wolfe is a nice projection-free algorithm, it has relatively slow convergence and usually is restricted to smooth loss functions. In our work, each weak learner instead minimizes the squared loss between its prediction and the gradient, which allows us to treat weak learners as approximations of the gradients thanks to the weak learner edge assumption. Hence we can mimic classic gradient boosting and use a gradient descent approach to combine the weak learners' predictions. These differences enable our algorithms to handle non-smooth convex losses, such as hinge and L_1-losses, and result in convergence bounds that is more analogous to the bounds of classic batch boosting algorithms. This work also differs from (Beygelzimer et al., 2015a) in that we assume an online weak learner edge exists, a common assumption in the classic boosting literature (Freund and Schapire, 1995; 1999) that is extended to the online boosting for classification by (Beygelzimer et al., 2015b; Chen et al., 2012b). With this assumption, we analyze online gradient boosting using techniques from gradient descent for convex losses (Hazan et al., 2007).

5.3 Preliminaries

In the classic online learning setting, at every time step t, the learner $\mathcal{A}$ first makes a prediction (i.e., picks a predictor $f_t \in \mathcal{F}$, where $\mathcal{F}$ is a pre-defined class of predictors) on the input $x_t \in \mathbb{R}^d$, then receives a loss $\ell_t(f_t(x_t))$. The learner then updates f_t to f_{t+1}. The samples (ℓ_t, x_t) could be generated by an adversary, but this work mainly focuses on the setting where $(\ell_t, x_t) \sim D$ are i.i.d sampled from a distribution D. The regret $R_{\mathcal{A}}(T)$ of the learner is defined as the difference between the total loss from the learner and the total loss from the best hypothesis in hindsight under the sequence of samples $\{(\ell_t, x_t)\}_t$:

$$R_{\mathcal{A}}(T) = \sum_{t=1}^{T} \ell_t(f_t(x_t)) - \min_{f^* \in \mathcal{F}} \sum_{t=1}^{T} \ell_t(f^*(x_t)). \tag{5.1}$$

We say the online learner is *no-regret* if and only if $R_{\mathcal{A}}(T)$ is $o(T)$. That is, time averaged, the online learner predictor f_t is doing as well as the best hypothesis f^* in hindsight. We define *risk* of a hypothesis f as $\mathbb{E}_{(\ell,x)\sim D}[\ell(f(x))]$. Our analysis of the risk leverages the classic Online-to-Batch reduction (Cesa-Bianchi et al., 2004; Hazan and Kale, 2014). The online-to-batch reduction first analyzes regret without the stochastic assumption on the sequence of loss ℓ, and it then relates regret to risk using concentration of measure.

Throughout the paper we will use the concepts of strong convexity and smoothness. A function $\ell(x)$ is said to be λ-strongly convex and β-smooth with respect to norm $\|\cdot\|$ if and only if for any pair x_1 and x_2:

$$\begin{aligned}&\frac{\lambda}{2}\|x_1 - x_2\|^2 \leq \ell(x_1) - \ell(x_2) - \nabla\ell(x_2)(x_1 - x_2)\\ &\leq \frac{\beta}{2}\|x_1 - x_2\|^2,\end{aligned} \tag{5.2}$$

where $\nabla\ell(x)$ denotes the gradient of function ℓ with respect to x.

5.3.1 Online Boosting Setup

Our online boosting setup is similar to (Beygelzimer et al., 2015b) and (Beygelzimer et al., 2015a). At each time step $t = 1, .., T$, the environment picks loss $\ell_t : \mathbb{R}^m \to \mathbb{R}$. The online boosting learner makes a prediction $y_t \in \mathbb{R}^m$ without knowing ℓ_t. Then the learner suffers loss $\ell_t(y_t)$. Throughout the paper we assume the loss is bounded as $|\ell_t(y)| \leq B, B \in \mathbb{R}^+, \forall t, y$. We also assume that the gradient of the loss $\nabla\ell_t(y)$ is also bounded as $\|\nabla\ell_t(y)\| \leq G, G \in \mathbb{R}^+, \forall t, y$.[1]

[1]Throughout the paper, the notation $\|x\|$ for any finite dimension vector x stands for the classic L2 norm.

The online boosting learner maintains a sequence of weak online learning algorithms $\mathcal{A}_1, ..., \mathcal{A}_N$. Each weak learner $\mathcal{A}_i$ can only use hypothesis from a restricted hypothesis class $\mathcal{H}$ to produce its prediction $\hat{y}_t^i = h_t^i(x_t)$ ($h : \mathbb{R}^d \to \mathbb{R}^m, \forall h \in \mathcal{H}$), where $h_t^i \in \mathcal{H}$. To make a prediction y_t at each iteration, each $\mathcal{A}_i$ will first make a prediction $\hat{y}_t^i \in \mathbb{R}^m$ where $\hat{y}_t^i = h_t^i(x_t)$. The online boosting learner combines all the weak learners' predictions to produce the final prediction y_t for sample x_t. The online learner then suffers loss $\ell_t(y_t)$ after the loss ℓ_t is revealed. As we will show later, with the loss ℓ_t, the online learner will pass a square loss to each weak learner. Each weak learner will then use its internal no-regret online update procedure to update its own weak hypothesis from h_t^i to h_{t+1}^i. In stochastic setting where ℓ_t and x_t are i.i.d samples from a fixed distribution, the online boosting learner will output a combination of the hypothesises that were generated by weak learners as the final boosted hypothesis for future testing.

By leveraging linear combination of weak learners, the goal of the online boosting learner is to boost the performance of a single online learner $\mathcal{A}_i$. Additionally, we ideally want the prediction error to decrease exponentially fast in the number N of weak learners, as is the result from classic batch gradient boosting (Grubb and Bagnell, 2011).

5.4 Weak Online Learning

We specifically consider the setting where each weak learner minimizes a square loss $\|y - h(x)\|^2$, where y is the regression target, and h is in the weak-learner hypothesis class $\mathcal{H}$. At each step t, a weak online learner $\mathcal{A}$ chooses a predictor $h_t \in \mathcal{H}$ to predict $h_t(x_t)$, receives the target y_t[2] and then suffers loss $\|y_t - h_t(x_t)\|^2$. With this, we now introduce the definition of Weak Online Learning Edge.

Definition 5.4.1. ***(Weak Online Learning Edge)*** *Given a restricted hypothesis class $\mathcal{H}$ and a sequence of square losses $\{\|y_t - h(x_t)\|^2\}_t$, the weak online learner predicts a sequence $\{h_t\}$ that has edge $\gamma \in (0, 1]$, such that with high probability $1 - \delta$:*

$$\sum_{t=1}^{T} \|y_t - h_t(x_t)\|^2 \leq (1-\gamma) \sum_{t=1}^{T} \|y_t\|^2 + R(T), \tag{5.3}$$

where $R(T) \in o(T)$ is usually known as the excess loss.

The high probability $1 - \delta$ comes from the possible randomness of the weak online learner and the sequence of examples. Usually the dependence of the high probability bound on δ is poly-logarithmic in $1/\delta$ that is included in the term $R(T)$. We will give a concrete example on this edge definition in next section where we will show what $R(T)$ consists of. Intuitively, a larger edge implies that the hypothesis is able to better explain the variance of the learning targets y. Our online weak learning definition is closely related to the one from (Beygelzimer et al., 2015b) in that our definition is an result of the following two assumptions: (1) the online learning problem is agnostic-learnable (i.e., the weak learner has $\frac{o(T)}{T} \to 0$ time-averaged regret against the best

[2]Abuse of notation: in Sec 5.4, $y_t \in \mathbb{R}^m$ simply stands for a regression target for the weak learner at step t, not the final prediction of the boosted learner defined in Sec. 5.3.1.

hypothesis $h \in \mathcal{H}$) with high probability:

$$\sum_{t=1}^{T} \|y_t - h_t(x_t)\|^2 \leq \min_{h\in\mathcal{H}} \sum_{t=1}^{T} \|y_t - h(x_t)\|^2 + o(T), \tag{5.4}$$

and (2) the restricted hypothesis class $\mathcal{H}$ is rich enough such that for any sequence of $\{y_t, x_t\}$ with high probability:

$$\min_{h\in\mathcal{H}} \sum_{t=1}^{T} \|y_t - h(x_t)\|^2 \leq (1-\gamma) \sum_{t=1}^{T} \|y_t\|^2 + o(T). \tag{5.5}$$

Our definition of online weak learning directly generalizes the batch weak learning definition in (Grubb and Bagnell, 2011) to the online setting by the additional agnostic learnability assumption as shown in Eqn. 5.4.

Note that we pick square losses (Eqn. 5.5) in our weak online learning definition. As we will show later, the goal is to enforce that the weak learners to accurately predict gradients, as was also originally used in the batch gradient boosting algorithm (Friedman, 2001). Least-squares losses are also shown to be important in streaming tasks by (Gao et al., 2016) for their superior computational and theoretical properties.

The above online weak learning edge definition immediately implies the following result, which is used in later proofs:

Lemma 5.4.2. *Given the sequence of losses $\|y_t - h(x_t)\|^2$, $1 \leq t \leq T$, the online weak learner generates a sequence of predictors $\{h_t\}_t$, such that:*

$$\sum_{t=1}^{T} 2y_t^T h_t(x_t) \geq \gamma \sum_{t=1}^{T} \|y_t\|^2 - R(T), \quad \gamma \in (0,1]. \tag{5.6}$$

The above lemma can be proved by expanding the square on the LHS of Eqn. 5.3, cancelling common terms and rearranging terms.

5.4.1 Why Weak Learner Edge is Reasonable?

We demonstrate here that the weak online learning edge assumption is reasonable. Let us consider the case that the hypothesis class $\mathcal{H}$ is closed under scaling (meaning if $h \in \mathcal{H}$, then for all $\alpha \in \mathbb{R}$, $\alpha h \in \mathcal{H}$) and let us assume $x \sim D$, and $y = f^*(x)$ for some unknown function f^*. We define the inner product $\langle h_1, h_2 \rangle$ of any two functions h_1, h_2 as $\mathbb{E}_{x\sim D}[h_1(x)^T h_2(x)]$ and the squared norm $\|h\|^2$ of any function h as $\langle h, h \rangle$. We assume f^* is bounded in a sense $\|f^*(x)\| \leq F \in \mathbb{R}^+$. The following proposition shows that as long as f^* is not perpendicular to the span of $\mathcal{H}$ ($f^* \not\perp span(\mathcal{H})$), i.e., $\exists h \in span(\mathcal{H})$ such that $\langle h, f^* \rangle \neq 0$, then we can achieve a non-zero edge:

Proposition 5.4.3. *Consider any sequence of pairs $\{x_t, y_t\}_{t=1}^T$, where x_t is i.i.d sampled from D, $y_t = f^*(x_t)$ and $f^* \not\perp span(\mathcal{H})$. Run any no-regret online algorithm $\mathcal{A}$ on sequence of losses*

Algorithm 6 Streaming Gradient Boosting (SGB)

```
1: Input: A restricted class $\mathcal{H}$. $N$ online weak learners $\{\mathcal{A}_i\}_{i=1}^N$. Learning rate $\eta$.
2: Each weak learner initlizes a hypothesis $h_i^1 \in \mathcal{H}, \forall 1 \le i \le N$.
3: for t = 1 to T do
4:     Receive $x_t$ and initialize $y_t^0 = y_0$ (e.g., $y_0 = 0$).
5:     for i = 1 to N do
6:         Set the partial sum $y_t^i = y_t^{i-1} - \eta h_i^t(x_t)$.
7:     end for
8:     Predict $y_t = y_t^N$.
9:     $\ell_t$ is revealed and learner suffers loss $\ell_t(y_t)$.
10:    for i = 1 to N do
11:        Compute gradient w.r.t partial sum: $\nabla_i^t = \nabla \ell_t(y_t^{i-1})$.
12:        Feed loss $\|\nabla_i^t - h_i^t(x_t)\|^2$ to $\mathcal{A}^i$.
13:        Weak learner $\mathcal{A}^i$ computes $h_i^{t+1}$ using its no-regret update procedure.
14:    end for
15: end for
16: Set $\bar{h}_i = \frac{1}{T}\sum_{t=1}^T h_i^t, \forall 1 \le i \le N$.
17: Return: $\{\bar{h}_1, ..., \bar{h}_N\}$.
```

$\{\|y_t - h(x_t)\|^2\}_t$ *and output a sequence of predictions* $\{h_t\}_t$*. With probability at least* $1-\delta$*, there exists a weak online learning edge* $\gamma \in (0, 1]$*, such that:*

$$\sum_{t=1}^{T} \|h_t(x_t) - y_t\|^2 \le (1-\gamma)\sum_{t=1}^{T}\|y_t\|^2 + R_{\mathcal{A}}(T) + (2-\gamma)O\Big(\sqrt{T\ln(1/\delta)}\Big),$$

where $R_{\mathcal{A}}(T)$ *is the regret of online algorithm* $\mathcal{A}$*.*

The proof of the above proposition can be found in Appendix. Matching to Eq. 5.3, we have $R(T) = R_{\mathcal{A}}(T) + (2-\gamma)O\Big(\sqrt{T\ln(1/\delta)}\Big) \in o(T)$. In addition, the contrapositive of the proposition implies that without a positive edge, $span(\mathcal{H})$ is orthogonal to f^* so that no linear boosted ensemble can approximate f^*. Hence having a positive online weak learner edge is necessary for online boosted algorithms.

5.5 Algorithm

5.5.1 Smooth Loss Functions

We first present Streaming Gradient Boosting (SGB), an algorithm (Alg. 6) that is designed for loss functions $\{\ell_t(y)\}$ that are λ-strongly convex and β-smooth. Alg. 6 is the online version of the classic batch gradient boosting algorithms (Friedman, 2001; Grubb and Bagnell, 2011). Alg. 6 maintains N weak learners. At each time step t, given example x_t, the algorithm predicts y_t by linearly combining the weak learners' predictions (Line 5). Then after receiving loss ℓ_t, for each weak learner, the algorithm computes the gradient of ℓ_t with respect to y evaluated at the *partial* sum y_t^{i-1} (Line 11) and feeds the square loss $l_t(h)$ with the computed gradient as the regression target to weak learner $\mathcal{A}^i$ (Line 12). The weak learner $\mathcal{A}^i$ then performs its own no-regret online update to compute h_i^{t+1} (Line 13).

Line 16 and 17 are needed for stochastic setting. We compute the average $\bar{h}_i$ for every weak learner $\mathcal{A}_i$ in Line 16. In testing time, given $x \sim D$, we predict y as:

$$y = y_0 - \eta \sum_{i=1}^{N} \bar{h}_i(x). \tag{5.7}$$

Since we penalize the weak learners by the squared deviation of its own prediction and the gradient from the previous partial sum, we essentially force weak learners to produce predictions that are close to the gradients (in a no-regret perspective). With this perspective, SGB can be understood as using the weak learners' predictions as N gradient descent steps where the gradient of each step i is approximated by a weak learner's prediction (Line 5). Let us define $\Delta_0 = \sum_{t=1}^{T}(\ell_t(y_t^0) - \ell_t(f^*(x_t)))$, for any $f^* \in \mathcal{F}$. Namely Δ_0 measures the performance of the initialization $\{y_t^0\}_t$. Under our assumption that the loss is bounded, $|\ell_t(x)| \leq B, \forall t, x$, we can simply upper bound Δ_0 as $\Delta_0 \leq 2BT$. Alg. 6 has the following performance guarantee:

Theorem 5.5.1. *Assume weak learner $\mathcal{A}_i, \forall i$ has weak online learning edge $\gamma \in (0, 1]$. Let $f^* = \arg\min_{f\in\mathcal{F}} \sum_t \ell_t(f(x_t))$. There exists a $\eta = \frac{\gamma}{\beta(8-4\gamma)}$, for λ-strongly convex and β-smooth loss functions, ℓ_t, such that when $T \to \infty$, Alg. 6 generates a sequence of predictions $\{y_t\}_t$ where:*

$$\frac{1}{T}[\sum_{t=1}^{T} \ell_t(y_t) - \sum_{t=1}^{T} \ell_t(f^*(x_t))] \leq 2B(1 - \frac{\gamma^2\lambda}{16\beta})^N. \tag{5.8}$$

For stochastic setting where $(x_t, \ell_t) \sim D$ independently, we have when $T \to \infty$:

$$\mathbb{E}\big[\ell\big(y_0 - \eta \sum_{i=1}^{N} \bar{h}_i(x)\big) - \ell(f^*(x))\big] \leq 2B(1 - \frac{\gamma^2\lambda}{16\beta})^N. \tag{5.9}$$

The expectation in Eqn. 5.9 of the above theorem is taken over the randomness of the sequence of pairs of loss and samples $\{\ell_t, x_t\}_{t=1}^T$ (note that $\bar{h}_i$ is dependent on $\ell_1, x_1, ..., \ell_T, x_T$) and ℓ, x. Theorem 5.5.1 shows that with infinite amount samples the average regret decreases exponentially as we increase the number of weak learners. This performance guarantee is very similar to classic batch boosting algorithms (Grubb and Bagnell, 2011; Schapire and Freund, 2012), where the empirical risk decreases exponentially with the number of algorithm iterations, i.e., the number of weak learners. Theorem 5.5.1 mirrors that of Theorem 1 in (Beygelzimer et al., 2015a), which bounds the regret of the Frank-Wolfe-based Online Gradient Boosting algorithm. Our results utilize the additional assumptions that the losses ℓ_t are strongly convex and that the weak learners have edge, allowing us to shrink the average regret exponentially with respect to N, while the average regret in (Beygelzimer et al., 2015a) shrinks in the order of $1/N$ (though this dependency on N is optimal under their setting).

Proof of Theorem 5.5.1, detailed in Appendix 5.8.2, weaves our additional assumptions into the proof framework of gradient descent on smooth losses. In particular, using weak learner edge assumption, we derive Lemma 5.4.2 and the Lemma 5.8.1 to relate parts of the strong smoothness expansion of the losses to the norm-squared of the gradients $\|\nabla \ell_t(y_t^i)\|^2$, which is an upper bound of $2\lambda(\ell_t(y_t^i) - \ell_t(f^*(x_t)))$ due to strong convexity. Using this observation, we can relate the total regret of the ensemble of the first i learners, $\Delta_i = \sum_{t=1}^{T}(\ell_t(y_t^i) - \ell_t(f^*(x_t)))$, with the regret from using $i+1$ learners, Δ_{i+1}, and show that Δ_{i+1} shrinks Δ_i by a constant fraction while only adding a small term $O(R(T)) \in o(T)$. Solving the recursion on the sequence of Δ_i, we arrive at the final exponentially decaying regret bound in the number of learners.

5.5.1.0.1 Remark Due to the weak online learning edge assumption, the regret bound shown in Eqn. 5.8 and the risk bound shown in Eqn. 5.9 are stronger than typical bounds in classic online learning, in a sense that we are competing against f^* that could potentially be much more powerful than any hypothesis from $\mathcal{H}$. For instance when the loss function is square loss $\ell(f(x)) = \|f(x) - z\|^2$, Theorem 5.5.1 essentially shows that the risk of the boosted hypothesis $\mathbb{E}[\|y_0 - \eta\sum_{i=1}^{N}\bar{h}_i(x) - z\|^2]$ approaches to zero as N approaches to infinity, under the assumption that $\mathcal{A}_i, \forall i$ have no-zero weak learning edge (e.g.,$f^* \in span(\mathcal{H})$). Note that this is analogous to the results of classification based batch boosting (Freund and Schapire, 1995; Grubb and Bagnell, 2011) and online boosting (Beygelzimer et al., 2015b): as number of weak learners increase, the average number of prediction mistakes approaches to zero. In other words, with the corresponding edge assumptions, these batch/online boosting classification algorithms can compete against any arbitrarily powerful classifier that always makes zero mistakes on any given training data.

5.5.2 Non-smooth Loss Functions

The regret bound shown in Theorem 5.5.1 only applies for strongly convex and smooth loss functions. In fact, one can show that Alg. 6 will fail for general non-smooth loss functions. We can construct a sequence of non-smooth loss functions and a special weak hypothesis class $\mathcal{H}$, which together show that the regret of Alg. 6 grows linearly in the number of samples, regardless of the number of weak learners. We refer readers to Appendix 5.8.4 for more details.

Our next algorithm, Alg. 7, extends SGB (Alg. 6) to handle strongly convex but non-smooth losses. Instead of training each weak learner to fit the subgradients of non-smooth loss with respect to current prediction, we instead keep track of a residual Δ_i[3] that accumulates the difference between the subgradients, ∇_k, and the fitted prediction $h_k(x_t)$, from $k = 1$ up to $i-1$. Instead of fitting the predictor h_{i+1} to match the subgradient ∇_{i+1}, we fit it to match the sum of the subgradient and the residuals, $\nabla_{i+1} + \Delta_i$. More specifically, in Line 13 of Alg. 7, for each weak learner $\mathcal{A}^i$, we feed a square loss with the sum of residual and the gradient as the regression target. Then Line 14 sets the new the residual Δ_i^t as the difference between the target $(\Delta_{i-1}^t + \nabla_i^t)$ and the weak learner $\mathcal{A}^i$'s prediction $h_i^t(x_t)$.

The last line of Alg. 7 is needed for stochastic setting where $(\ell_t, x_t) \sim D$ i.i.d. In test, given sample $x \sim D$, we predict y using $h_t^i, \forall i, t$ in procedure shown in Alg. 8. For notation simplicity,

[3]Note the abusive notation. For the non-smooth loss setting (Alg. 7), Δ_i does not refer to the regret of the ensemble's regret with the i-th as used in the analysis of Alg. 6

Algorithm 7 Streaming Gradient Boosting (SGB) for non-smooth loss (Residual Projection)

```
1: Input: A restricted class $\mathcal{H}$. $N$ online weak learners $\{\mathcal{A}_i\}_{i=1}^N$. Learning rate schedule $\{\eta_i\}_{i=1}^N$.
2: $\forall i, \mathcal{A}_i$ initializes a hypothesis $h_i^1 \in \mathcal{H}$.
3: for t = 1 to T do
4:     Receive $x_t$ and initialize $y_t^0 = y_0$ (e.g., $y_0 = 0$).
5:     for i = 1 to N do
6:         Set the projected partial sum $y_t^i = \Pi_{\mathcal{Y}}(y_t^{i-1} - \eta_i h_i^t(x_t))$.
7:     end for
8:     Predict $y_t = \frac{1}{N}\sum_{i=0}^N y_t^i$
9:     The loss $\ell_t$ is revealed and compute loss $\ell_t(y_t)$.
10:    Set initial residual $\Delta_0^t = 0$.
11:    for i = 1 to N do
12:        Compute subgradient w.r.t. partial sum: $\nabla_i^t = \nabla \ell_t(y_t^{i-1})$.
13:        Feed loss $\left\|(\Delta_{i-1}^t + \nabla_i^t) - h(x)\right\|^2$ to $\mathcal{A}^i$.
14:        Update residual: $\Delta_i^t = \Delta_{i-1}^t + \nabla_i^t - h_i^t(x_t)$.
15:        Weak learner $\mathcal{A}^i$ computes $h_i^{t+1}$ using its no-regret update procedure.
16:    end for
17: end for
18: Return: $h_t^i, 1 \le i \le N, 1 \le t \le T$.
```

we denote the testing procedure shown in Alg. 8 as $\mathcal{T}(x)$, which $\mathcal{T}$ explicitly depends on the returns $h_t^i, 1 \le i \le N, 1 \le t \le T$ from SGB (Residual Projection). Since it's impractical to store and apply all TN models, we follow a common stochastic learning technique which uses the final predictor at time T for testing (e.g., Johnson and Zhang (2013)) in the experiment section (i.e., simply set $t = T$ in Line 3 in Alg. 8). In practice, if the learners converge and T is large, the average and final predictions are close.

Intuitively, this approach prevents the weak learners from consistently failing to match a certain direction of the subgradient as the net error in the direction is stored in residual. By the assumption of weak learner edge, the directions will be approximated. We also note that if we assume the subgradients are bounded, then the residual magnitudes increase at most linearly

Algorithm 8 SGB (Residual Projection) for testing

```
1: Input: Test sample $x$ and $h_t^i, 1 \le i \le N, 1 \le t \le T$ from the output of Alg. 7.
2: for t = 1 to T do
3:     for i = 1 to N do
4:         $y_t^i = \Pi_{\mathcal{Y}}(y_t^{i-1} - \eta_i h_i^t(x))$.
5:     end for
6:     $y_t = \frac{1}{N}\sum_{i=0}^N y_t^i$.
7: end for
8: Predict: $y = \mathcal{T}(x) = \frac{1}{T}\sum_{t=1}^T y_t$.
```

in the number of weak learners. Simultaneously, each weak learner shrinks the residual by at least a constant factor due to the assumption of edge. Hence, we expect the residual to shrink exponentially in the number of learners. Utilizing this observation, we arrive at the following performance guarantee:

Theorem 5.5.2. *Assume the loss ℓ_t is λ-strongly convex for all t with bounded gradients, $\|\nabla \ell_t(y)\| \leq G$ for all y, and each weak learner $\mathcal{A}_i$ has edge $\gamma \in (0,1]$. Let $\mathcal{F}$ be a function space, and $\mathcal{H} \subset \mathcal{F}$ be a restriction of $\mathcal{F}$ Let $f^* = \arg\min_{f\in\mathcal{F}} \frac{1}{T}\sum_{t=1}^{T} \ell_t(f(x_t))$ be the optimal predictor in $\mathcal{F}$ in hindsight. Let $c = \frac{2}{\gamma} - 1$. Let step size be $\eta_i = \frac{1}{\lambda i}$. When $T \to \infty$, we have:*

$$\frac{1}{T}\sum_{t=1}^{T}(\ell_t(y_t) - \ell_t(f^*(x_t))) \leq \frac{4c^2G^2}{\lambda N}(1 + \ln N + \frac{1}{8N}). \tag{5.10}$$

For stochastic setting where $(x_t, \ell_t) \sim D$ independently, when $T \to \infty$ we have:

$$\mathbb{E}\big[\ell(\mathcal{T}(x)) - \ell(f^*(x))\big] \leq \frac{4c^2G^2}{\lambda N}(1 + \ln N + \frac{1}{8N}).$$

The above theorem shows that the average regret of Alg. 7 is $O(\ln N/N)$ with respect to the number N of weak learners, which matches the regret bounds of Online Gradient Descent for strongly convex loss. The key idea for proving Theorem 5.5.2 is to combine our online weak learning edge definition with the proof framework of Online Gradient Descent for strongly convex loss functions from (Hazan et al., 2007). The detailed proof can be found in Appendix 5.8.3.

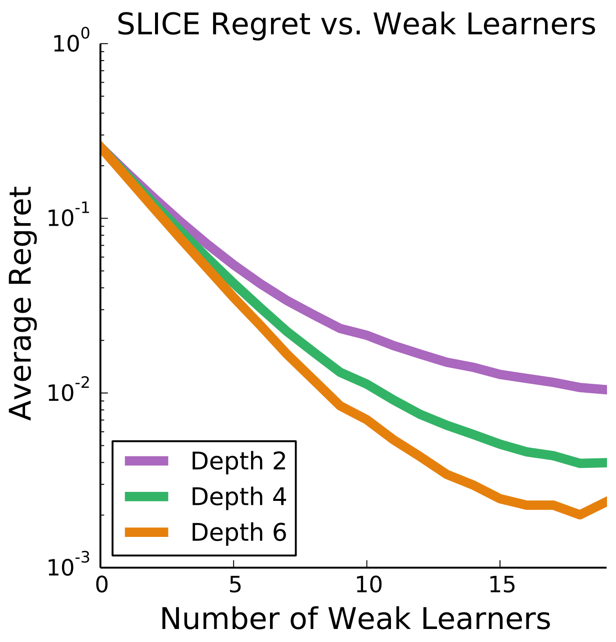
5.6 Experiments

We demonstrate the performance of our Streaming Gradient Boosting using the following UCI datasets (Lichman, 2013): YEAR, ABALONE, SLICE, and A9A (Kohavi and Becker, 1996) as well as the MNIST (LeCun et al., 2001) dataset. If available, we use the given train-test split of each data-set. Otherwise, we create a random 90%-10% train-test split.

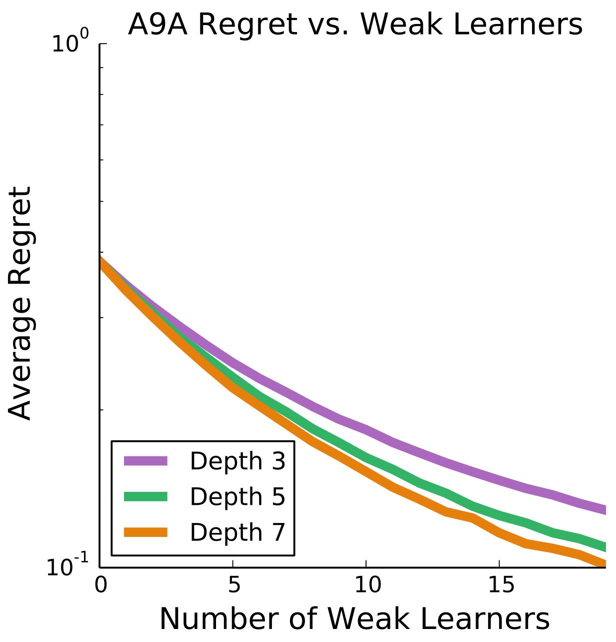
5.6.1 Experimental Analysis of Regret Bounds

We first demonstrate the relationships between the regret bounds shown in Eqn. 5.8 and the parameters including the number of weak learners, the number of samples and edge γ. We compute the regret of SGB with respect to a deep regression tree (depth$\geq$ 15), which plays the f^* in Eqn. 5.8. We use regression trees as the weak learners. We assume that deeper trees have higher edges γ because they empirically fit training data better. We show how the regret relates to the trees' depth, the number of weak learners N (Fig. 5.1b) and the number of samples T (Fig. 5.1d).

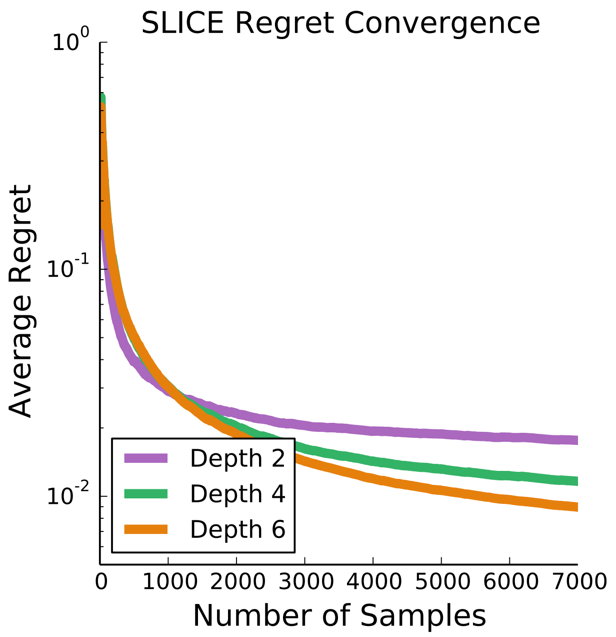
For the experimental results shown in Fig. 5.1, we used smooth loss functions with L_2 regularization (see Appendix 5.8.5 for more details). We use logistic loss and square loss for binary classification (A9A) and regression task (SLICE), respectively. For each regression tree weak learner, Follow The Regularized Leader (FTRL) (Shalev-Shwartz, 2011) was used as the no-regret online update algorithm with regularization posed as the depth of the tree. Fig. 5.1b



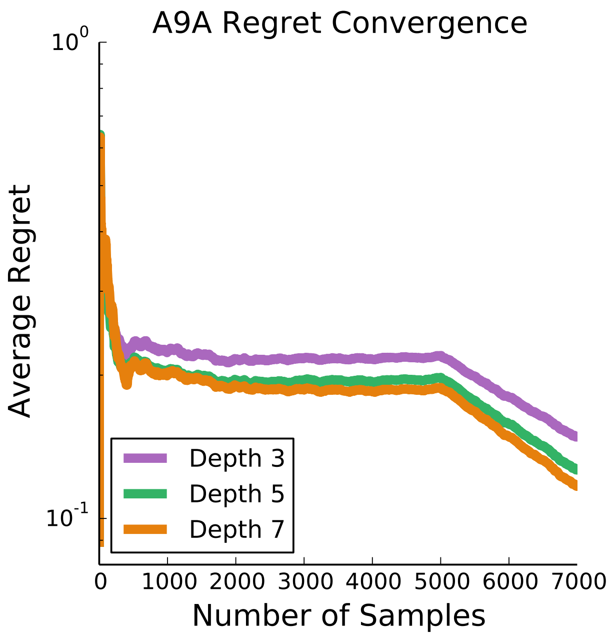
(a) a



(b) b



(c) c



(d) d

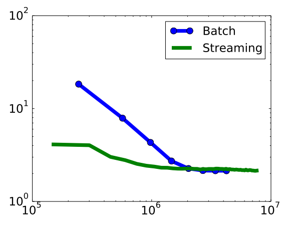
Figure 5.1: Average regret of SGB with regression trees with various depths on SLICE and A9A datasets.

shows the relationship between the number of weak learners and the average regret given a fixed total number of samples. The average regret decreases as we increase the number of weak learners. We note that the curves are close to linear at the beginning, matching our theoretical analysis that the average regret decays exponentially (note the y-axis is log scale) with respect to the number of weak learners. This shows that SGB can significantly boost the performance of a single weak learner.

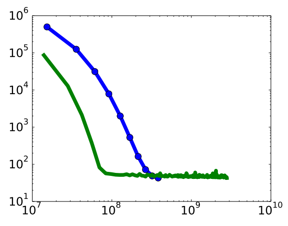
To investigate the effect of the edge parameter γ, we additionally compute the average regret in Fig. 5.1 as the depth of the regression tree is increased. The tree depth increases the model complexity of the base learner and should relate to a larger γ edge parameter. From this experiment, we see that the average regret shrinks as the depth of the trees increases.

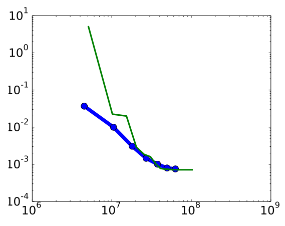
Finally, Fig. 5.1d shows the convergence of the average regret with respect to the number of samples. We see that more powerful weak learners (deeper regression trees) results in faster convergence of our algorithm. We ran Alg. 7 on A9A with hinge loss and SLICE with $L1$ (least absolute deviation) loss and observed very similar results as shown in Fig. 5.1.

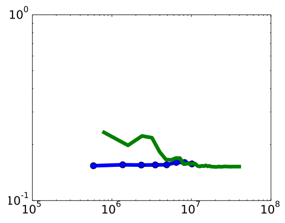
5.6.2 Batch Boosting vs. Streaming Boosting

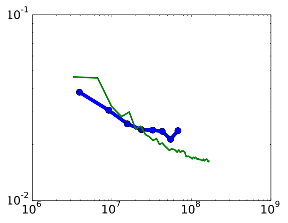


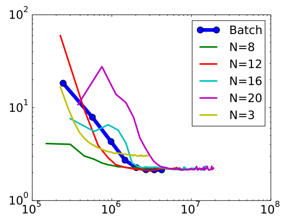
(a) a


(b) b


(c) c


(d) d


(e) e



(f) f

Figure 5.2: Log-log plots of test-time loss vs. computation complexity on various data-sets. The x-axis represents computation complexity measured by number of weak leaner predictions; the y-axis measures square loss for regression tasks (ABALONE, SLICE and YEAR), and classification error for A9A and MNIST.

We next compare batch boosting to SGB using two-layer neural networks as weak learners[4] and see that SGB reaches similar final performance as the batch boosting algorithm albeit with less training computation. As stated in Sec 5.5.2, we report h_T^i instead of $\bar{h}_i$ for SGB, since at convergence the average prediction is close to the final prediction, and the latter is impractical to compute. We implement our baseline, the classic batch gradient boosting (**GB**) (Friedman, 2001), by optimizing each weak learner until convergence in order. In both GB and SGB, we train weak learners using ADAM (Kingma and Ba, 2015) optimization and use the default random parameter initialization for NN.

[4]The number of hidden units by data-set: ABALONE, A9A: 1; YEAR, SLICE: 10; MNIST: 5x5 convolution with stride of 2 and 5 output channels. Sigmoid is used as the activation for all except SLICE, which uses leaky ReLU.

We analyze the complexity of training SGB and GB. We define the prediction complexity of one weak learner as the *unit cost*, since the training run-time complexity almost equates the total complexity of weak learner predictions and updates. Our choice of weak learner and update method (two-layer networks and ADAM) determines that updating a weak learner is about two units cost. In training using SGB, each of the T data samples triggers predictions and updates with all N of the weak learners. This results in a training computational complexity of $3TN = O(TN)$. For GB, let T_B be the samples needed for each weak learner to converge. Then the complexity of training GB is $T_B \sum_{i=1}^{N} i + 2T_BN \simeq \frac{1}{2}T_BN^2 = O(T_BN^2)$, because when training weak learner i, all previous $i-1$ weak learners must also predict for each data point[5]. Hence, SGB and GB will have the same training complexity if $T_B \simeq \frac{6T}{N} = \Theta(\frac{T}{N})$. In our experiments we observe weak learners typically converge less than $\frac{T}{N}$ samples, but our following experiment shows that SGB still can converge faster overall.

Fig. 5.2 plots the test-time loss versus training computation, measured by the unit cost. Blue dots highlights when the weak learners are added in GB. We first note that SGB successfully converges to the results of GB in all cases, supporting that SGB is a truly a streaming conversion of GB. As it takes many weak learners to achieve good performance on ABALONE and YEAR, we observe that SGB converges with less computation than GB. On A9A, however, GB is more computationally efficient than SGB, because the first weak learner in GB already performs well and learning a single weak learner for GB is faster than simultaneously optimizing all $N = 8$ weak learners with SGB. This suggests that if we initially set N too big, SGB could be less computationally efficient. In fact Fig. 5.2f shows that very larger N causes slower convergence to the same final error plateau. On the other hand, small N ($N = 3$) results in worse performance. We specify the chosen N for SGB in Fig. 5.2, and they are around the number of weak learners that GB requires to converge and achieve good performance. We also note that SGB has slower initial progress compared to GB on SLICE in Fig. 5.2c and MNIST in Fig. 5.2e. This is an understandable result as SGB has a much larger pool of parameters to optimize. Despite this initial disadvantage, SGB surpasses GB and converges faster overall, suggesting the advantage of updating all the weak learners together. In practice, if we do not have a good guess of N, we can still use SGB to add multiple weak learners at a time in GB to speed up convergence. Table 5.1 records the test error (square error for regression and error ratio for classification) of the neural network base learner, GB, and SGB. We observe that SGB achieves test errors that are competitive with GB in all cases.

5.7 Conclusion

In this paper, we present SGB for online convex programming. By introducing an online weak learning edge definition that naturally extends the edge definition from batch boosting to the online setting and by using square loss, we are able to boost the predictions from weak learners in a gradient descent fashion. Our SGB algorithm guarantees exponential regret shrinkage in the number N of weak learners for strongly convex and smooth loss functions. We additionally extend SGB for optimizing non-smooth loss function, which achieves $O(\ln N/N)$ no-regret rate.

[5]Saving previous predictions is disallowed, because data may not be revisited in an actual streaming setting.

	Base	**GB**	**SGB**
ABALONE (regression)	8.2848	2.1411	2.1532
YEAR (regression)	4.99×10^5	42.8976	43.0573
SLICE (regression)	0.036045	0.000755	0.000713
A9A (classification)	0.1547	0.1579	0.1523
MNIST (classification)	0.163280	0.019320	0.016320

Table 5.1: Average test-time loss: square error for regression, and error rate for classification.

Finally, experimental results support the theoretical analysis.

Though our SGB algorithm currently utilizes the procedure of gradient descent to combine the weak learners predictions, our online weak learning definition and the design of square loss for weak learners leave open the possibility to leverage other gradient-based update procedures such as accelerated gradient descent, mirror descent, and adaptive gradient descent for combining the weak learners' predictions.

5.8 Supplementary Details for Gradient Boosting on Stochastic Data Streams

5.8.1 Proof of Proposition 5.4.3

Proof. Given that a no-regret online learning algorithm $\mathcal{A}$ running on sequence of loss $\|h(x_t) - y_t\|^2$, we have can easily see that Eqn. 5.4 holds as:

$$\sum_{t=1}^{T} \|h_t(x_t) - y_t\|^2 \leq \min_{h\in\mathcal{H}} \sum_{t=1}^{T} \|h(x_t) - y_t\|^2 + R_{\mathcal{A}}(T), \tag{5.11}$$

where $R_{\mathcal{A}}(T)$ is the regret of $\mathcal{A}$ and is $o(T)$. To prove Proposition 5.4.3, we only need to show that Eqn. 5.5 holds for some $\gamma \in (0, 1]$. This is equivalent to showing that there exist a hypothesis $\tilde{h} \in \mathcal{H}$ ($\|\tilde{h}\| = 1$), such that $\langle \tilde{h}, f^*\rangle > 0$. To see this equivalence, let us assume that $\langle \tilde{h}, f^*/\|f^*\|\rangle = \epsilon > 0$. Let us set $h^* = \epsilon\|f^*\|\tilde{h}$. Using Pythagorean theorem, we can see that $\|h^* - f^*\|^2 = (1-\epsilon^2)\|f^*\|^2$. Hence we get γ is at least ϵ^2, which is in $(0, 1]$.

Now since we assume that $f^* \not\perp span(\mathcal{H})$, then there must exist $h' \in \mathcal{H}$, such that $\langle f^*, h'\rangle \neq 0$, otherwise $f^* \perp \mathcal{H}$. Consider the hypothesis $h'/\|h'\|$ and $-h'/\|h'\|$ (we assume $\mathcal{H}$ is closed under scale), we have that either $\langle h', f^*\rangle > 0$ or $\langle -h', f^*\rangle > 0$. Namely, we find at least one hypothesis h such that $\langle h, f^*\rangle > 0$ and $\|h\| = 1$. Hence if we pick $\tilde{h} = \arg\max_{h\in\mathcal{H},\|h\|=1}\langle h, f^*/\|f^*\|\rangle$, we must have $\langle \tilde{h}, f^*/\|f^*\|\rangle = \epsilon > 0$. Namely we can find a hypothesis $h^* \in \mathcal{H}$, which is $\epsilon\|f^*\|\tilde{h}$, such that there is non-zero $\gamma \in (0, 1]$:

$$\|h^* - f^*\|^2 \leq (1-\gamma)\|f^*\|^2. \tag{5.12}$$

To show that we can extend this γ to the finite sample case, we are going to use Hoeffding inequality to relate the norm $\|\cdot\|$ to its finite sample approximation.

Applying Hoeffding inequality, we get with probability at least $1-\delta/2$,

$$|\frac{1}{T}\sum_{t=1}^{T}\|y_t\|^2 - \langle f^*, f^*\rangle| \leq O\big(\sqrt{\frac{F^2}{T}\ln(4/\delta)}\big), \tag{5.13}$$

where based on assumption that $f^*(\cdot)$ is bounded as $\|f^*(\cdot)\| \leq F$. Similarly, we have with probability at least $1-\delta/2$:

$$|\frac{1}{T}\sum_{t=1}^{T}\|h^*(x_t) - f^*(x_t)\|^2 - \|h^* - f^*\|^2| \leq O\Big(\sqrt{\frac{F^2}{T}\ln(4/\delta)}\Big), \tag{5.14}$$

Apply union bound for the above two high probability statements, we get with probability at least $1-\delta$,

$$\begin{aligned}
&|\frac{1}{T}\sum_{t=1}^{T}y_t^2 - \langle f^*, f^*\rangle| \leq O\Big(\sqrt{\frac{F^2}{T}\ln(4/\delta)}\Big), \quad \textit{and},\\
&|\frac{1}{T}\sum_{t=1}^{T}(h^*(x_t) - f^*(x_t))^2 - \|h^* - f^*\|| \leq O\Big(\sqrt{\frac{F^2}{T}\ln(4/\delta)}\Big).
\end{aligned} \tag{5.15}$$

Now to prove the theorem, we proceed as follows:

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\|h^*(x_t) - f^*(x_t)\|^2\\
&\leq \|h^* - f^*\| + O\Big(\sqrt{\frac{F^2}{T}\ln(4/\delta)}\Big)\\
&\leq (1-\gamma)\|f^*\|^2 + O\Big(\sqrt{\frac{F^2}{T}\ln(4/\delta)}\Big)\\
&\leq (1-\gamma)\frac{1}{T}\sum_{t=1}^{T}y_t^2 + (1-\gamma)O\Big(\sqrt{\frac{F^2}{T}\ln(4/\delta)}\Big) + O\Big(\sqrt{\frac{F^2}{T}\ln(4/\delta)}\Big).
\end{aligned} \tag{5.16}$$

Hence we get with probability at least $1-\delta$:

$$\sum_{t=1}^{T}\|h^*(x_t) - f^*(x_t)\|^2 \leq \sum_{t=1}^{T}\|y_t\|^2 + (2-\gamma)O\Big(\sqrt{T\ln(1/\delta)}\Big). \tag{5.17}$$

Set $R(T) = R_{\mathcal{A}}(T) + (2-\gamma)O\Big(\sqrt{T\ln(1/\delta)}\Big)$, we prove the proposition. $\square$

5.8.2 Proof of Theorem 5.5.1

An important property of λ-strong convexity that we will use later in the proof is that for any x and $x^* = \arg\min_x l(x)$, we have:

$$\|\nabla l(x)\|^2 \geq 2\lambda(l(x) - l(x^*)). \tag{5.18}$$

We prove Eqn. 5.18 below.

From the λ-strong convexity of $l(x)$, we have:

$$l(y) \geq l(x) + \nabla l(x)(y-x) + \frac{\lambda}{2}\|y-x\|^2. \tag{5.19}$$

Replace y by x^* in the above equation, we have:

$$\begin{aligned}
&l(x^*) \geq l(x) + \nabla l(x)(x^*-x) + \frac{\lambda}{2}\|x^*-x\|^2\\
\Rightarrow & 2\lambda l(x^*) \geq 2\lambda l(x) + 2\lambda\nabla l(x)(x^*-x) + \lambda^2\|x^*-x\|^2\\
\Rightarrow & -2\lambda\nabla l(x)(x^*-x) - \lambda^2\|x^*-x\|^2 \geq 2\lambda(l(x)-l(x^*))\\
\Rightarrow & \|\nabla l(x)\|^2 - \|\nabla l(x)\|^2 - 2\lambda\nabla l(x)(x^*-x) - \lambda^2\|x^*-x\|^2 \geq 2\lambda(l(x)-l(x^*))\\
\Rightarrow & \|\nabla l(x)\|^2 - \|\nabla l(x) + \lambda(x^*-x)\|^2 \geq 2\lambda(l(x)-l(x^*))\\
\Rightarrow & \|\nabla l(x)\|^2 \geq 2\lambda(l(x)-l(x^*)).
\end{aligned} \tag{5.20}$$

5.8.2.1 Proofs for Lemma 5.4.2

Proof. Complete the square on the left hand side (LHS) of Eqn. 5.3, we have:

$$\sum \|y_t\|^2 - 2y_t^T h_t(x_t) + \|h_t(x_t)\|^2 \leq (1-\gamma)\sum_t \|y_t\|^2 + R(T). \tag{5.21}$$

Now let us cancel the $\sum y_t^2$ from both side of the above inequality, we have:

$$\sum -2y_t^T h_t(x_t) \leq \sum -2y_t^T h_t(x_t) + \|h_t(x_t)\|^2 \leq -\gamma\sum \|y_t\|^2 + R(T). \tag{5.22}$$

Rearrange, we have:

$$\sum 2y_t^T h_t(x_t) \geq \gamma\sum \|y_t\|^2 - R(T). \tag{5.23}$$

□

5.8.2.2 Proof of Theorem 5.5.1

We need another lemma for proving theorem 5.5.1:

Lemma 5.8.1. *For each weak learner $\mathcal{A}_i$, we have:*

$$\sum_t \|h_t^i(x_t)\|^2 \leq (4-2\gamma)\sum_t \|\nabla \ell_t(y_t^{i-1})\|^2 + 2R(T). \tag{5.24}$$

Proof of Lemma 5.8.1. For $\sum_t (h_t^i(x_t))^2$, we have:

$$\begin{aligned}
&\sum_t \|h_t^i(x_t)\|^2 = \sum_t \|h_t^i(x_t) - \nabla\ell_t(y_t^{i-1}) + \nabla\ell_t(y_t^{i-1})\|^2 \\
&\leq \sum_t \|h_t^i(x_t) - \nabla\ell_t(y_t^{i-1})\|^2 + \sum_t \|\nabla\ell_t y_t^{i-1}\|^2 + \sum_t 2(h_t^i(x_t) - \nabla\ell_t(y_t)^{i-1})^T\nabla\ell_t(y_t^{i-1}) \\
&\leq \sum_t 2\|h_t^i(x_t) - \nabla\ell_t(y_t^{i-1})\|^2 + \sum_t 2\|\nabla\ell_t(y_t^{i-1}\|^2 \\
&\leq 2(1-\gamma)\sum_t \|\nabla\ell_t(y_t^{i-1}\|^2 + 2R(T) + 2\sum_t \|\nabla\ell_t(y_t^{i-1}\|^2 \\
&\qquad \textit{(By Weak Onling Learning Definition)} \\
&\leq (4-2\gamma)\sum_t \|\nabla\ell_t(y_t^{i-1}\|^2 + 2R(T). \qquad (5.25)
\end{aligned}$$

1435 □

Proof of Theorem 5.5.1. For $1 \leq i \leq N$, let us define $\Delta_i = \sum_{t=1}^T (\ell_t(y_t^i) - \ell_t(f^*(x_t)))$. Following similar proof strategy as shown in (Beygelzimer et al., 2015a), we will link Δ_i to Δ_{i-1}. For Δ_i, we have:

$$\begin{aligned}
&\Delta_i = \sum_{t=1}^T (\ell_t(y_t^i) - \ell_t(f^*(x_t))) = \sum_t \ell_t(y_t^{i-1} - \eta h_t^i(x_t)) - \sum_t \ell_t(f^*(x_t)) \\
&\leq \sum_t \big[\ell_t(y_t^{i-1}) - \eta\nabla\ell_t(y_t^{i-1})^T h_t^i(x_t) + \frac{\beta\eta^2}{2}\|h_t^i(x_t)\|^2\big] - \sum_t \ell_t(f^*(x_t)) \\
&\qquad \textit{(By } \beta\textit{-smoothness of } \ell_t\textit{)} \\
&\leq \sum_t \big[\ell_t(y_t^{i-1}) - \frac{\eta\gamma}{2}\|\nabla\ell_t(y_t^{i-1})\|^2 + \frac{\eta R(T)}{2} + \frac{\beta\eta^2}{2}\|h_t^i(x_t)\|^2\big] - \sum_t \ell_t(f^*(x_t)) \\
&\qquad \textit{(By Lemma 5.4.2)} \\
&\leq \sum_t \big[\ell_t(y_t^{i-1}) - \frac{\eta\gamma}{2}\|\nabla\ell_t(y_t^{i-1})\|^2 + \frac{\eta R(T)}{2} + \beta\eta^2(2-\gamma)\|\nabla\ell_t(y_t^{i-1})\|^2 + \beta\eta^2 R(T) - \ell_t(f^*(x_t))\big] \\
&\qquad \textit{(By Lemma 5.8.1)} \\
&= \Delta_{i-1} - (\frac{\eta\gamma}{2} - \beta\eta^2(2-\gamma))\sum_t \|\nabla\ell_t(y_t^{i-1})\|^2 + (\frac{\eta}{2} + \beta\eta^2)R(T) \\
&\leq \Delta_{i-1} - (\eta\gamma\lambda - \beta\eta^2\lambda(4-2\gamma))\sum_t \big(\ell_t(y_t^{i-1}) - \ell_t(f^*(x_t))\big) + (\frac{\eta}{2} + \beta\eta^2)R(T) \\
&\qquad \textit{(By Eqn. 5.18)} \\
&= \Delta_{i-1}\big[1 - (\eta\gamma\lambda - \beta\eta^2\lambda(4-2\gamma))\big] + (\frac{\eta}{2} + \beta\eta^2)R(T) \qquad (5.26)
\end{aligned}$$

Due to the setting of η, we know that $0 < (1 - (\eta\gamma\lambda - \beta\eta^2\lambda(4-2\gamma))) < 1$. For notation simplicity, let us first define $C = 1 - (\eta\gamma\lambda - \beta\eta^2\lambda(4-2\gamma))$. Starting from Δ_0, keep applying the relationship between Δ_i and Δ_{i-1} N times, we have:

$$\begin{aligned}\Delta_N &= C^N\Delta_0 + (\frac{\eta}{2}+\beta\eta^2)R(T)\sum_{i=1}^{N}C^{i-1}\\ &= C^N\Delta_0 + (\frac{\eta}{2}+\beta\eta^2)R(T)\frac{1-C^N}{1-C}\\ &\leq C^N\Delta_0 + (\frac{\eta}{2}+\beta\eta^2)R(T)\frac{1}{1-C}.\end{aligned}$$

Now divide both sides by T, and take T to infinity, we have:

$$\frac{1}{T}\Delta_N = C^N\frac{1}{T}\Delta_0 \leq C^N 2B, \tag{5.27}$$

where we simply assume that $\ell_t(y)\in[-B,B], B\in\mathcal{R}^+$ for any t and y. Now let us go back to C, to minimize C, we can take the derivative of C with respect to η, set it to zero and solve for η, we will have:

$$\eta = \frac{\gamma}{\beta(8-4\gamma)}. \tag{5.28}$$

Substitute this η back to C, we have:

$$C = 1-\frac{\gamma^2\lambda}{\beta(16-8\gamma)} \geq 1-\frac{\lambda}{8\beta} \geq 1-\frac{1}{8} = \frac{7}{8}. \tag{5.29}$$

Hence, we can see that there exist a $\eta = \frac{\gamma}{\beta(8-4\gamma)}$, such that:

$$\frac{1}{T}\Delta_N \leq 2B(1-\frac{\gamma^2\lambda}{\beta(16-8\gamma)})^N \leq 2B(1-\frac{\gamma^2\lambda}{16\beta})^N. \tag{5.30}$$

Hence we prove the first part of the theorem regarding the regret. For the second part of the theorem where ℓ_t and x_t are i.i.d sampled from a fixed distribution, we proceed as follows.

Let us take expectation on both sides of the inequality 5.30. The left hand side of inequality 5.30 becomes:

$$\begin{aligned}\frac{1}{T}\mathbb{E}\Delta_N &= \mathbb{E}\frac{1}{T}\big[\sum_{t=1}^{T}(\ell_t(y_t^N)-\ell_t(f^*(x_t)))\big] \qquad (5.31)\\ &= \frac{1}{T}\mathbb{E}\big[\sum_{t=1}^{T}\ell_t(-\mu\sum_{i=1}^{N}h_t^i(x_t))\big]-\frac{1}{T}\mathbb{E}_{(\ell_t,x_t)\sim D}[\ell_t(f^*(x_t))]\\ &= \frac{1}{T}\sum_{i=1}^{T}\mathbb{E}_t\big[\ell_t(-\mu\sum_{i=1}^{N}h_t^i(x_t))\big]-\mathbb{E}_{(\ell,x)\sim D}\ell(f^*(x)), \qquad (5.32)\end{aligned}$$

where the expectation is taken over the randomness of x_t and ℓ_t. Note that h_t^i only depends on $x_1,\ell_1,...,x_{t-1},\ell_{t-1}$. We also define $\mathbb{E}_t$ as the expectation over the randomness of x_t and ℓ_t at step

t conditioned on $x_1, \ell_1, ..., x_{t-1}, \ell_{t-1}$. Since ℓ_t, x_t are sampled i.i.d from D, we can simply write $\mathbb{E}_t[\ell_t(-\mu \sum_{i=1}^N h_t^i(x_t))]$ as $\mathbb{E}_t[\ell(-\mu \sum_{i=1}^N h_t^i(x))]$. Now the above inequality can be simplied as:

$$\begin{aligned}
\frac{1}{T}\mathbb{E}\Delta_N &= \frac{1}{T}\sum_{t=1}^T \mathbb{E}_t[\ell(-\mu\sum_{i=1}^N h_t^i(x))] - \mathbb{E}_{(\ell,x)\sim D}\ell(f^*(x)) \\
&\geq \mathbb{E}\big[\ell(-\mu\sum_{i=1}^N \frac{1}{T}\sum_{t=1}^T h_t^i(x))\big] - \mathbb{E}_{(\ell,x)\sim D}\ell(f^*(x)) \\
&= \mathbb{E}\big[\ell(-\mu\sum_{i=1}^N \bar{h}_i(x))\big] - \mathbb{E}_{(\ell,x)\sim D}\ell(f^*(x)) \qquad (5.33)
\end{aligned}$$

1438 Now use the fact that $1/T\mathbb{E}\Delta_N \leq 2B(1-\frac{\gamma^2\lambda}{16\beta})^N$, we prove the theorem. □

1439 5.8.3 Proof of Theorem 5.5.2

Lemma 5.8.2. *In Alg. 7, if we assume the 2-norm of gradients of the loss w.r.t. partial sums by G (i.e., $\|\nabla_t^i\| = \|\nabla\ell_t(y_t^{i-1})\| \leq G$), and assume that each weak learner $\mathcal{A}_i$ has regret $R(T) = o(T)$, then we there exists a constant $c = \frac{1-\gamma+\sqrt{1-\gamma(1-\frac{R(T)}{TG^2})}}{\gamma} < \frac{2}{\gamma} - 1$ such that*

$$\sum_{t=1}^T \|\Delta_i^t\|^2 \leq c^2G^2T \quad \text{and} \quad \sum_{t=1}^T \|h_i^t(x_t)\|^2 \leq (4-2\gamma)(1+c)^2G^2T + 2R(T) \leq 4c^2G^2T. \qquad (5.34)$$

Proof. We prove the first inequality by induction on the weak learner index i. When $i = 0$, the claim is clearly true since $\Delta_0^t = 0$ for all t. Now we assume the claim is true for some $i \geq 0$, and prove it for $i+1$. We first note that by the inequality $\frac{1}{T}\sum_{t=1}^T a_t \leq \sqrt{\frac{\sum_t a_t^2}{T}}$ for all sequence $\{a_t\}_t$, we have

$$\begin{aligned}
&\frac{1}{T}(\sum_t \|\Delta_i^t\|)^2 \leq \sum_t \|\Delta_i^t\|^2 \leq c^2G^2T && (5.35)\\
\Rightarrow &(\sum_t \|\Delta_i^t\|)^2 \leq c^2G^2T^2 && (5.36)\\
\Rightarrow &\sum_t \|\Delta_i^t\| \leq cGT && (5.37)
\end{aligned}$$

Then by the assumption that weak learner $\mathcal{A}_i$ has an edge γ with regret $R(T)$, we have from step 14 of Alg. 7:

$$\begin{aligned}
\sum_t \|\Delta_{i+1}^t\|^2 = \sum_t \|\Delta_i^t + \nabla_{i+1}^t - h_{i+1}^t(x_t)\|^2 &\leq (1-\gamma)\sum_t \|\Delta_i^t + \nabla_{i+1}^t\|^2 + R(T) && (5.38)\\
&\leq (1-\gamma)\sum_t \big(\|\Delta_i^t\| + G\big)^2 + R(T) && (5.39)
\end{aligned}$$

$$\leq (1-\gamma)\left(\sum_t \|\Delta_i^t\|^2 + 2G\sum_t \|\Delta_i^t\| + G^2T\right) + R(T) \tag{5.40}$$

$$\leq (1-\gamma)(1+c)^2G^2T + R(T) \tag{5.41}$$

$$= c^2G^2T \tag{5.42}$$

We have the last equality because c is chosen as the positive root of the quadratic equation: $\gamma c^2+(2\gamma-2)c+(\gamma-1-\frac{R(T)}{TG^2}) = 0$, which is equivalent to $c^2G^2T = (1-\gamma)(c+1)^2G^2T+R(T)$.

The second inequality of the lemma can be derived from a similar argument of Lemma 5.8.1 by expanding $\|\left(\Delta_{i-1}^t + \nabla_i^t - h_i^t(x_t)\right) - \left(\Delta_{i-1}^t + \nabla_i^t\right)\|^2$ and then applying edge assumption. $\square$

We now use the above lemma to prove the performance guarantee of Alg. 7 as follows.

Proof of Theorem 5.5.2. We first define the intermediate predictors as: $f_0^t(x) := h_0(x)$, $\hat{f}_i^t(x) := f^{t-1}(x) - \eta_i h_i^t(x$ and $f_i^t(x) := P(\hat{f}_i^t(x))$. Then for all $i = 1, ..., N$ we have:

$$\|f_i^t(x_t) - f^*(x_t)\|^2 \leq \|\hat{f}_i^t(x_t) - f^*(x_t)\|^2 = \|f_{i-1}^t(x_t) - \eta_i h_i^t(x_t) - f^*(x_t)\|^2 \tag{5.43}$$

$$\begin{aligned}&= \|f_{i-1}^t(x_t) - f^*(x_t)\|^2 + \eta_i^2\|h_i^t(x_t)\|^2 - 2\eta_i\langle f_{i-1}^t(x_t) - f^*(x_t), h_i^t(x_t) - \Delta_{i-1}^t - \nabla_i^t\rangle \\ &\quad - 2\eta_i\langle f_{i-1}^t(x_t) - f^*(x_t), \Delta_{i-1}^t + \nabla_i^t\rangle\end{aligned} \tag{5.44}$$

Rearanging terms we have:

$$\langle f^*(x_t) - f_{i-1}^t(x_t), \nabla_i^t\rangle \tag{5.45}$$

$$\begin{aligned}\geq &\frac{1}{2\eta_i}\|f_i^t(x_t) - f^*(x_t)\|^2 - \frac{1}{2\eta_i}\|f_{i-1}^t(x_t) - f^*(x_t)\|^2 - \frac{\eta_i}{2}\|h_i^t(x_t)\|^2 \\ &- \langle f^*(x_t) - f_{i-1}^t(x_t), h_i^t(x_t) - \Delta_{i-1}^t - \nabla_i^t\rangle - \langle f^*(x_t) - f_{i-1}^t(x_t), \Delta_{i-1}^t\rangle\end{aligned} \tag{5.46}$$

Using λ-strongly convex of ℓ_t and applying the above equality and $\Delta_i^t = \Delta_{i-1}^t + \nabla_i^t - h_i^t(x_t)$, we have:

$$\ell_t(f^*(x_t)) \geq \ell_t(f_{i-1}^t(x_t)) + \langle f^*(x_t) - f_{i-1}^t(x_t), \nabla_i^t\rangle + \frac{\lambda}{2}\|f^*(x_t) - f_{i-1}^t(x_t)\|^2 \tag{5.47}$$

$$\begin{aligned}\geq &\ell_t(f_{i-1}^t(x_t)) + \frac{1}{2\eta_i}\|f_i^t(x_t) - f^*(x_t)\|^2 - \frac{1}{2\eta_i}\|f_{i-1}^t(x_t) - f^*(x_t)\|^2 - \frac{\eta_i}{2}\|h_i^t(x_t)\|^2 \\ &+ \langle f^*(x_t) - f_{i-1}^t(x_t), \Delta_i^t\rangle - \langle f^*(x_t) - f_{i-1}^t(x_t), \Delta_{i-1}^t\rangle + \frac{\lambda}{2}\|f^*(x_t) - f_{i-1}^t(x_t)\|^2\end{aligned} \tag{5.48}$$

Summing over $t = 1, ..., T$ and $i = 1, ..., N$ we have:

$$\begin{aligned}&N\sum_{t=1}^T \ell_t(f^*(x_t)) \\ \geq &\sum_{i=1}^N\sum_{t=1}^T\left[\ell_t(f_{i-1}^t(x_t)) + \langle f^*(x_t) - f_{i-1}^t(x_t), \nabla_i^t\rangle + \frac{\lambda}{2}\|f^*(x_t) - f_{i-1}^t(x_t)\|^2\right]\end{aligned} \tag{5.49}$$

$$
\begin{aligned}
=&\sum_{i=1}^{N}\sum_{t=1}^{T}\ell_t(f_{i-1}^t(x_t)) - \sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\eta_i}{2}\|h_i^t(x_t)\|^2\\
&+\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{1}{2\eta_i}\|f_i^t(x_t)-f^*(x_t)\|^2-\sum_{i=1}^{N}\sum_{t=1}^{T}(\frac{1}{2\eta_i}-\frac{\lambda}{2})\|f_{i-1}^t(x_t)-f^*(x_t)\|^2\\
&+\sum_{i=1}^{N}\sum_{t=1}^{T}\left\langle f^*(x_t)-f_{i-1}^t(x_t),\Delta_i^t\right\rangle-\sum_{i=1}^{N}\sum_{t=1}^{T}\left\langle f^*(x_t)-f_{i-1}^t(x_t),\Delta_{i-1}^t\right\rangle \qquad (5.50)\\
=&\sum_{i=1}^{N}\sum_{t=1}^{T}\ell_t(f_{i-1}^t(x_t)) - \sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\eta_i}{2}\|h_i^t(x_t)\|^2\\
&+\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{1}{2\eta_i}\|f_i^t(x_t)-f^*(x_t)\|^2-\sum_{i=0}^{N-1}\sum_{t=1}^{T}(\frac{1}{2\eta_{i+1}}-\frac{\lambda}{2})\|f_{i}^t(x_t)-f^*(x_t)\|^2\\
&+\sum_{i=1}^{N}\sum_{t=1}^{T}\left\langle f^*(x_t)-f_{i-1}^t(x_t),\Delta_i^t\right\rangle-\sum_{i=1}^{N-1}\sum_{t=1}^{T}\left\langle f^*(x_t)-(f_{i-1}^t(x_t)-\eta_i h_i^t(x_t)),\Delta_{i}^t\right\rangle\\
&-\sum_{t=1}^{T}\left\langle f^*(x_t)-f_0^t(x_t),\Delta_0^t\right\rangle \qquad \text{(We switched index and apply } \Delta_0^t=0 \text{ next.)} \qquad (5.51)\\
=&\sum_{i=1}^{N}\sum_{t=1}^{T}\ell_t(f_{i-1}^t(x_t)) - \sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\eta_i}{2}\|h_i^t(x_t)\|^2-\sum_{i=1}^{N-1}\sum_{t=1}^{T}\left\langle \eta_i h_i^t(x_t),\Delta_i^t\right\rangle\\
&+\sum_{i=1}^{N-1}\sum_{t=1}^{T}\frac{1}{2}\|f_i^t(x_t)-f^*(x_t)\|^2(\frac{1}{\eta_i}-\frac{1}{\eta_{i+1}}+\lambda)-\sum_{t=1}^{T}(\frac{1}{2\eta_1}-\frac{\lambda}{2})\|f_0^t(x_t)-f^*(x_t)\|^2\\
&+\sum_{t=1}^{T}\left[\left\langle f^*(x_t)-f_{N-1}^t(x_t),\Delta_N^t\right\rangle+\frac{1}{2\eta_N}\|f_{N-1}^t(x_t)-\eta_N h_N^t(x_t)-f^*(x_t)\|^2\right] \qquad (5.52)
\end{aligned}
$$

(We next apply $\eta_i = \dfrac{1}{\lambda i}$ and complete the squares for the last sum.)

$$
\begin{aligned}
=&\sum_{i=1}^{N}\sum_{t=1}^{T}\ell_t(f_{i-1}^t(x_t)) - \sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\eta_i}{2}\|h_i^t(x_t)\|^2-\sum_{i=1}^{N-1}\sum_{t=1}^{T}\left\langle \eta_i h_i^t(x_t),\Delta_i^t\right\rangle\\
&+\frac{1}{2\eta_N}\sum_{t=1}^{T}\|\left(f_{N-1}^t(x_t)-f^*(x_t)\right)+\eta_N(\Delta_N^t-h_N^t(x_t))\|^2\\
&-\frac{\eta_N}{2}\sum_{t=1}^{T}\left(\|\Delta_N^t-h_N^t(x_t)\|^2-\|h_N^t(x_t)\|^2\right) \qquad (5.53)
\end{aligned}
$$

(We next drop the completed square, and apply Cauchy-Schwarz)

$$
\geq\sum_{i=1}^{N}\sum_{t=1}^{T}\ell_t(f_{i-1}^t(x_t)) - \sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\eta_i}{2}\|h_i^t(x_t)\|^2-\sum_{i=1}^{N}\eta_i\sum_{t=1}^{T}\|h_i^t(x_t)\|\|\Delta_i^t\|-\frac{\eta_N}{2}\sum_{t=1}^{T}\|\Delta_N^t\|^2 \qquad (5.54)
$$

(We next apply Cauchy-Schwarz again.)

$$\geq \sum_{i=1}^{N}\sum_{t=1}^{T} \ell_t(f_{i-1}^t(x_t)) - \sum_{i=1}^{N}\frac{\eta_i}{2}\sum_{t=1}^{T}\|h_i^t(x_t)\|^2 - \frac{\eta_N}{2}\sum_{t=1}^{T}\|\Delta_N^t\|^2$$
$$-\sum_{i=1}^{N}\eta_i\sqrt{\sum_{t=1}^{T}\|h_i^t(x_t)\|^2\sum_{t=1}^{T}\|\Delta_i^t\|^2} \tag{5.55}$$

Now we apply Lemma 5.8.2 and replace the remaining $\eta_i = \frac{1}{\lambda i}$. Using $\sum_{i=1}^{N}\frac{1}{i} \leq 1 + \ln N$, we have:

$$N\sum_{t=1}^{T}\ell_t(f^*(x_t))$$
$$\geq \sum_{i=1}^{N}\sum_{t=1}^{T}\ell_t(f_{i-1}^t(x_t)) - \sum_{i=1}^{N}\frac{1}{2i\lambda}4c^2G^2T - \frac{1}{2N\lambda}c^2G^2T - \sum_{i=1}^{N}\frac{1}{i\lambda}2c^2G^2T \tag{5.56}$$
$$\geq \sum_{i=1}^{N}\sum_{t=1}^{T}\ell_t(f_{i-1}^t(x_t)) - \frac{4c^2G^2T}{\lambda}(1+\ln N) - \frac{c^2G^2T}{2N\lambda} \tag{5.57}$$

Dividing both sides by NT and rearrange terms, we get:

$$\frac{1}{TN}\sum_{i=1}^{N}\sum_{t=1}^{T}\left[\ell_t(y_t^i) - \ell_t(f^*(x_t))\right] \leq \frac{4c^2G^2}{N\lambda}(1+\ln N) + \frac{c^2G^2}{2N^2\lambda}.$$

Using Jensen's inequality for the LHS of the above inequality, we get:

$$\frac{1}{T}\sum_{t=1}^{T}\ell_t(\frac{1}{N}\sum_{i=1}^{N}y_t^i) - \ell_t(f^*(x_t)) \leq \frac{4c^2G^2}{N\lambda}(1+\ln N) + \frac{c^2G^2}{2N^2\lambda},$$

which proves the first part of the theorem.

For stochastic setting, we can prove it by using similar proof techniques (e.g., take expectation on both sides of Eqn. 5.58 and use Jensen inequality) that we used for proving theorem 5.5.1. □

5.8.4 Counter Example for Alg. 6

In this section, we provide an counter example where we show that Alg. 6 cannot guarantee to work for non-smooth loss. We set $y \in \mathbb{R}^2$, and design a loss function $\ell_t(y) = 2|y_{[1]}| + |y_{[2]}|$, where $y_{[i]}$ stands for the i'th entry of the vector y, for all time step t. The subgradient of this non-smooth loss is $[2,1]^T$, or $[2,-1]^T$, or $[-2,1]^T$, or $[-2,-1]^T$, depending on the position of y. We restricted the weak hypothesis class $\mathcal{H}$ to consist of only two types of hypothesis: hypothesis $h(x) = [\alpha, 0]^T$, or hypothesis $h(x) = [0,\alpha]^T$, where $\alpha \in [-2,2]$. We can show that given a sequence of training examples $\{(x_\tau, g_\tau)\}_{\tau=1}^{t}$, where g_t is the one of the gradient from the total four possible subgradient of ℓ_t, the hypothesis that minimizes the accumulated square loss $\sum_{\tau=1}^{t}(h(x_\tau) - g_\tau)^2$ is going to be the type of $h(x) = [\alpha, 0]^T$.

Now we consider using Follow the Leader (FTL) as a no-regret online learning algorithm for each weak learner. Based on the above analysis, we know that no matter what the sequence of training examples each weak learner has received as far, the weak leaners always choose the hypothesis with type $h(x) = [\alpha, 0]^T$ from $\mathcal{H}$. So, for every time step t, if we initialize $y_t^0 = [a, b]^T$, where $a > 0$ and $b > 0$, then the output y_t^N (computed from Line 8 in Alg.1) always have the form of $y_t^N = [\eta, b]$, where $\eta \in \mathbb{R}$. Namely, all weak learners' prediction only moves y_t horizontally and it will never be moved vertically. But note that the optimal solution is located at $[0, 0]^T$. Since for all t, $y^N_{t_{[2]}}$ is also b constant away from 0, the total regret accumulates linearly as bT, regardless of how many weak learners we have.

5.8.5 Details of Implementation

5.8.6 Binary Classification

For binary classification, following (Friedman, 2001), let us define feature $x \in \mathbb{R}^n$, label $u \in \{-1, 1\}$. With x_t and u_t, the loss function ℓ_t is defined as:

$$\ell_t(y) = \ln(1 + \exp(-u_t y)) + \lambda y^2. \tag{5.58}$$

where $y \in \mathbb{R}$. In this setting, we have $\mathcal{H} : \mathbb{R}^n \to \mathbb{R}$. The regularization is to avoid overfitting: we can set $y = \infty * sign(u_t)$ to make the loss close to zero.

The loss function $\ell_t(y)$ is twice differentiable with respect to y, and the second derivative is:

$$\nabla^2 \ell_t(y) = \frac{\exp(u_t y)}{(1 + \exp(u_t y))^2} \tag{5.59}$$

Note that we have:

$$\nabla^2 \ell_t(y) \leq \frac{1}{1/\exp(u_t y) + 2 + \exp(u_t y)} \leq \frac{1}{4}. \tag{5.60}$$

Hence, $\ell_t(y)$ is $1/4$-smooth.

Under the assumption that the output from hypothesis from $\mathcal{H}$ is bounded as $|y| \leq Y \in \mathbb{R}^+$, we also have:

$$\nabla^2 \ell_t(y) \geq \frac{1}{2 + 2\exp(Y)} \tag{5.61}$$

Hence, with boundness assumption, we can see that $\ell_t(y)$ is $1/(2 + 2\exp(Y))$-strongly convex and $(1/4)$-smooth.

The another loss we tried is the hinge loss:

$$\ell_t(y) = \max(0, 1 - u_t y) + \lambda y^2. \tag{5.62}$$

With the regularization, the loss $\ell_t(y)$ is still strongly convex, but no longer smooth.

5.8.6.1 Multi-class Classification

Follow the settings in (Friedman, 2001), for multi-class classification problem, let us define feature $x \in \mathbb{R}^n$, and label information $u \in \mathbb{R}^k$, as a one-hot representation, where $u[i] = 1$ ($u[i]$ is the i-th element of u), if the example is labelled by i, and $u[i] = 0$ otherwise. The loss function ℓ_t is defined as:

$$\ell_t(y) = -\sum_{i=1}^{k} u_t[i] \ln \frac{\exp(y[i])}{\sum_{j=1}^{k} \exp(y[j])}, \tag{5.63}$$

where $y \in \mathbb{R}^k$. In this setting, we let weak learner i pick hypothesis h from $\mathcal{H}$ that takes feature x_t as input, and output $\hat{y}_i \in \mathbb{R}^k$. The online boosting algorithm then linearly combines the weak learners' prediction to predict y.

5.8.7 Proof of Proposition 5.4.3

Proof. Given that a no-regret online learning algorithm $\mathcal{A}$ running on sequence of loss $(h(x_t) - y_t)^2$, we have can easily see that Eqn. 5.4 holds as:

$$\sum_{t=1}^{T} (h_t(x_t) - y_t)^2 \leq \min_{h \in \mathcal{H}} (h(x_t) - y_t)^2 + R_{\mathcal{A}}(T), \tag{5.64}$$

where $R_{\mathcal{A}}(T)$ is the regret of $\mathcal{A}$ and is $o(T)$. To prove Proposition 5.4.3, we only need to show that Eqn. 5.5 holds for some $\gamma \in (0, 1]$.

Consider $\sum_{t=1}^{T} y_t^2$, we have:

$$\frac{1}{T}\sum_{t=1}^{T} y_t^2 = \frac{1}{T}\sum_{t=1}^{T} (f^*(x_t))^2 = \frac{1}{T}\sum_{t=1}^{T} (\sum_{i=1}^{N} \alpha_i \hat{h}_i(x_t))^2. \tag{5.65}$$

Clearly $\frac{1}{T}\sum_{t=1}^{T}(\sum_{i=1}^{M} \alpha_i \hat{h}_t(x_t))$ is an unbiased estimate of $\mathbb{E}_{x\sim D}(\sum_{i=1}^{M} \alpha_i \hat{h}_t(x))^2$, which based on our definition of inner product, can be written as $\langle \sum_{i=1}^{M} \alpha_i \hat{h}_i, \sum_{i=1}^{M} \alpha_i \hat{h}_i \rangle$. Applying Hoeffding inequality here, we get with probability at least $1 - \delta$,

$$|\frac{1}{T}\sum_{t=1}^{t} y_t^2 - \langle \sum_{i=1}^{M} \alpha_i \hat{h}_i, \sum_{i=1}^{M} \alpha_i \hat{h}_i \rangle| \leq \sqrt{\frac{2D^2}{T} \ln(2/\delta)}, \tag{5.66}$$

where we assume that $f^*(\cdot)$ is bounded as $|f^*(\cdot)| \leq D$. Also, since $\hat{h}_i$ are basis of $\mathcal{H}$, we have:

$$\langle \sum_{i=1}^{M} \alpha_i \hat{h}_i, \sum_{i=1}^{M} \alpha_i \hat{h}_i \rangle = \sum_{i=1}^{M} \alpha_i^2. \tag{5.67}$$

Without loss of generality, we assume that $\alpha_1 = \arg\max_{\alpha_i} (\alpha_i)^2$ and $\alpha_1 > 0$. Since $\hat{h}_1$ is one of the basis of the span of $\mathcal{H}$, there must exist a hypothesis h (we assume $\|h\| = 1$ under the assumption that $\mathcal{H}$ is closed under scalar), such that $\langle h, \hat{h}_1 \rangle = \nu, \nu \in (0, 1]$. Let us define $\tilde{h} = (\alpha_1 \nu)h$. Using Pythagorean theorem, it is straightforward to verify that $\|\tilde{h} - \alpha_1 \hat{h}_1\|^2 = (1 - \nu^2)\alpha_1^2$.

Using the above results, we can show that for $\|\tilde{h} - \sum_{i=1}^M \alpha_i \hat{h}_i\|^2$, we have:

$$\begin{aligned}
&\|\tilde{h} - \sum_{i=1}^M \alpha_i \hat{h}_i\|^2 = \|\tilde{h} - \alpha_1 \hat{h}_1 + \alpha_1 \hat{h}_1 - \sum_{i=1}^M \alpha_i \hat{h}_i\|^2 \\
&\leq \|\tilde{h} - \alpha_1 \hat{h}_1\|^2 + \|\sum_{i=2}^M \alpha_i \hat{h}_i\|^2 \qquad \textit{(Triangular inequality)} \\
&= (1 - \nu^2)\alpha_1^2 + \sum_{i=2}^M \alpha_i^2 = (1 - \frac{\nu^2 \alpha_1^2}{\sum_{i=1}^M \alpha_i^2}) \sum_{i=1}^M \alpha_i^2 \\
&= (1 - \gamma)\langle \sum_{i=1}^M \alpha_i \hat{h}_i, \sum_{i=1}^M \alpha_i \hat{h}_i \rangle, \qquad (5.68)
\end{aligned}$$

where we define the edge $\gamma = \nu^2 \alpha_1^2 / (\alpha_1^2 + ... + \alpha_M^2) \in (0, 1]$.

For $\|\tilde{h} - \sum_{i=1}^M \alpha_i \hat{h}_i\|$, apply Hoeffding inequality again, we get:

$$\left|\frac{1}{T}\sum_{t=1}^T (\tilde{h}(x_t) - \sum_{i=1}^M \alpha_i \hat{h}_i(x_t))^2 - \|\tilde{h} - \sum_{i=1}^M \alpha_i \hat{h}_i\|^2\right| \leq \sqrt{\frac{2D^2}{T}\ln(2/\delta)}, \qquad (5.69)$$

with probability at least $1 - \delta$. Apply union bound on two Eqn. 5.77 and 5.69, we get with probability at least $1 - \delta$,

$$\begin{aligned}
&\left|\frac{1}{T}\sum_{t=1}^t y_t^2 - \langle \sum_{i=1}^M \alpha_i \hat{h}_i, \sum_{i=1}^M \alpha_i \hat{h}_i \rangle\right| \leq \sqrt{\frac{2D^2}{T}\ln(4/\delta)}, \quad \textit{and} \\
&\left|\frac{1}{T}\sum_{t=1}^T (\tilde{h}(x_t) - \sum_{i=1}^M \alpha_i \hat{h}_i(x_t))^2 - \|\tilde{h} - \sum_{i=1}^M \alpha_i \hat{h}_i\|^2\right| \leq \sqrt{\frac{2D^2}{T}\ln(4/\delta)}. \qquad (5.70)
\end{aligned}$$

Combine the above two inequalities together with the inequality shown in (5.68), we have with probability at least $1 - \delta$:

$$\begin{aligned}
&\sum_{t=1}^T (\tilde{h}(x_t) - \sum_{i=1}^M \alpha_i \hat{h}_i(x_t))^2 \leq T\|\tilde{h} - \sum_{i=1}^M \alpha_i \hat{h}_i\|^2 + \sqrt{2D^2 T \ln(4/\delta)} \\
&\leq (1 - \gamma)T\langle \sum_{i=1}^M \alpha_i \hat{h}_i, \sum_{i=1}^M \alpha_i \hat{h}_i \rangle + \sqrt{2D^2 T \ln(4/\delta)} \\
&\leq (1 - \gamma)\big(\sum_{t=1}^T y_t^2 + \sqrt{2D^2 T \ln(4/\delta)}\big) + \sqrt{2D^2 T \ln(4/\delta)} \\
&= (1 - \gamma)\sum_{t=1}^T y_t^2 + (2 - \gamma)\sqrt{2D^2 T \ln(4/\delta)}. \qquad (5.71)
\end{aligned}$$

Since we have $\min_{h\in\mathcal{H}}\sum_{t=1}^{T}(h(x)-y_t)^2 \le \sum_{t=1}^{T}(\tilde{h}(x_t)-\sum_{i=1}^{M}\alpha_i\hat{h}_i(x_t))^2$, combine the above inequality with Eqn. 5.64, we have with probability at least $1-\delta$:

$$\sum_{t=1}^{t}(h_t(x_t)-y_t)^2 \le (1-\gamma)\sum_{t=1}^{T}y_t^2 + R_{\mathcal{A}}(T) + (2-\gamma)\sqrt{2D^2T\ln(4/\delta)}$$
$$= (1-\gamma)\sum_{t=1}^{T}y_t^2 + R(T), \tag{5.72}$$

where we define $R(T) = R_{\mathcal{A}} + (2-\gamma)\sqrt{2D^2T\ln(4/\delta)}$, which is $o(T)$. Hence based on the construction of $\tilde{h}$, we can see there must exist an edge which is at least no smaller than the γ we defined here, which is $v^2\alpha_1/(\alpha_1^2+...+\alpha_M^2)$. □

5.8.8 Proof of Proposition 5.4.3

Proof. Given that a no-regret online learning algorithm $\mathcal{A}$ running on sequence of loss $(h(x_t)-y_t)^2$, we have can easily see that Eqn. 5.4 holds as:

$$\sum_{t=1}^{T}(h_t(x_t)-y_t)^2 \le \min_{h\in\mathcal{H}}(h(x_t)-y_t)^2 + R_{\mathcal{A}}(T), \tag{5.73}$$

where $R_{\mathcal{A}}(T)$ is the regret of $\mathcal{A}$ and is $o(T)$. To prove Proposition 5.4.3, we only need to show that Eqn. 5.5 holds for some $\gamma\in(0,1]$. This is equivalent to showing that there exist a hypothesis $\tilde{h}\in\mathcal{H}$ ($\|\tilde{h}\|=1$), such that $\langle\tilde{h},f^*\rangle>0$. To see this equivalence, let us assume that $\langle\tilde{h},f^*\rangle=\epsilon>0$. Let us set $h^*=\epsilon\tilde{h}$. Using Pythagorean theorem, we can see that $\|h^*-f^*\|^2=(1-\epsilon^2)\|f^*\|^2$. Hence we get γ is at least ϵ^2, which is in $(0,1]$.

Now since we assume that $f^*\not\perp span(\mathcal{H})$, then there must exist $h'\in\mathcal{H}$, such that $\langle f^*,h'\rangle\neq0$, otherwise $f^*\perp\mathcal{H}$. Consider the hypothesis $h'/\|h'\|$ and $-h'/\|h'\|$ (we assume $\mathcal{H}$ is closed under scale), we have that either $\langle h',f^*\rangle>0$ or $\langle -h',f^*\rangle>0$. Namely, we find at least one hypothesis h such that $\langle h,f^*\rangle>0$ and $\|h\|=1$. Hence if we pick $\tilde{h}=\arg\max_{h\in\mathcal{H},\|h\|=1}\langle h,f^*\rangle$, we must have $\langle\tilde{h},f^*\rangle=\epsilon>0$. In summary we can find a hypothesis $h^*\in\mathcal{H}$, which is $\epsilon\tilde{h}$, such that there is non-zero $\gamma\in(0,1]$:

$$\|h^*-f^*\|^2 \le (1-\gamma)\|f^*\|^2. \tag{5.74}$$

Another fact is that if $\|f^*\|=0$, we can set $\gamma=1$. Since we assume that $\mathcal{H}$ contains the hypothesis h_0 that always predicts zero, we must have $\|h_0-f^*\|=\|f^*\|=0=(1-1)\|f^*\|$. Hence we prove that λ could be set to 1.

Now we consider the case where $\|f^*\|\neq0$. To show that there exist such $\tilde{h}$, we use proof of by contradiction: assume for any $h\in\mathcal{H}$, we have $\langle h,f^*\rangle=0$. Let us define the matrix $H=[\hat{h}_1,\hat{h}_2,...,\hat{h}_M]$ and matrix G as $G=H^TH$, as $G_{i,j}=\langle h_i,h_j\rangle$. Since we assume that for any $h\in\mathcal{H}$ (including $\hat{h}_1,...,\hat{h}_M$), we have $\langle h,f^*\rangle=0$, this implies the following equation:

$$(H^TH)\alpha=0, \tag{5.75}$$

where $\alpha = [\alpha_1, ..., \alpha_M]^T$. Multiply α^T on the left hand side, we then have $\alpha^T H^T H\alpha = \|H\alpha\|^2 = 0$, which implies that $\|H\alpha\| = 0$. Note that based on the definition of f^*, we have $f^* = H\alpha$, hence $\|f^*\| = 0$. This contradicts the case that $\|f^*\| \neq 0$. Hence, if $\|f^*\| \neq 0$, there must exist a hypothesis $\tilde{h} \in \{\hat{h}_1, ..., \hat{h}_M\}$, such that $\langle \tilde{h}, f^* \rangle = \epsilon \geq 0$. As we showed above, in this case, λ will be equal to ϵ^2, which is in $(0, 1]$.

In summary, we can find a hypothesis $h^* \in \mathcal{H}$ such that there is a non-zero γ:

$$\|h^* - f^*\|^2 \leq (1-\gamma)\|f^*\|^2. \tag{5.76}$$

To show that we can extend this γ to the finite sample case, we are going to use Hoeffding inequality to relate the norm $\|\cdot\|$ to its finite sample approximation.

Applying Hoeffding inequality, we get with probability at least $1 - \delta/2$,

$$|\frac{1}{T}\sum_{t=1}^{T} y_t^2 - \langle f^*, f^* \rangle| \leq \sqrt{\frac{2D^2}{T}\ln(4/\delta)}, \tag{5.77}$$

where we assume that $f^*(\cdot)$ is bounded as $|f^*(\cdot)| \leq D$. Similarly, we have with probability at least $1 - \delta/2$:

$$|\frac{1}{T}\sum_{t=1}^{T}(h^*(x_t) - f^*(x_t)) - \|h^* - f^*\|| \leq \sqrt{\frac{2D^2}{T}\ln(4/\delta)}. \tag{5.78}$$

Apply union bound for the above two high probability statements, we get with probability at least $1 - \delta$,

$$\begin{aligned} &|\frac{1}{T}\sum_{t=1}^{T} y_t^2 - \langle f^*, f^* \rangle| \leq \sqrt{\frac{2D^2}{T}\ln(4/\delta)}, \quad and, \\ &|\frac{1}{T}\sum_{t=1}^{T}(h^*(x_t) - f^*(x_t))^2 - \|h^* - f^*\|| \leq \sqrt{\frac{2D^2}{T}\ln(4/\delta)}. \end{aligned} \tag{5.79}$$

Now to prove the theorem, we proceed as follows:

$$\begin{aligned} &\frac{1}{T}\sum_{t=1}^{T}(h^*(x_t) - f^*(x_t))^2 \\ &\leq \|h^* - f^*\| + \sqrt{\frac{2D^2}{T}\ln(4/\delta)} \\ &\leq (1-\gamma)\|f^*\|^2 + \sqrt{\frac{2D^2}{T}\ln(4/\delta)} \\ &\leq (1-\gamma)\frac{1}{T}\sum_{t=1}^{T} y_t^2 + (1-\gamma)\sqrt{\frac{2D^2}{T}\ln(4/\delta)} + \sqrt{\frac{2D^2}{T}\ln(4/\delta)}. \end{aligned} \tag{5.80}$$

Hence we get with probability at least $1 - \delta$:

$$\sum_{t=1}^{T}(h^*(x_t) - f^*(x_t))^2 \leq \sum_{t=1}^{T} y_t^2 + (2-\gamma)\sqrt{2D^2 T\ln(4\delta)}. \tag{5.81}$$

Set $R(T) = R_{\mathcal{A}}(T) + (2-\gamma)\sqrt{2D^2T\ln(4/\delta)}$, we prove the proposition. $\square$

Chapter 6

Anytime Learning via Forward Architecture Search

6.1 Introduction

Deep neural networks have achieved state-of-the-art performance on many large scale supervised learning tasks across many domains like computer vision, natural language processing and audio and speech-related tasks using architectures manually designed by skilled practitioners using domain knowledge with experimental trial and error. Can we make this work for less skilled practitioners? Is it possible to search amongst plausible architectures in an automated fashion to create a more automatic learning algorithm? Neural architecture search (NAS) (Zoph and Le, 2017) algorithms attempt to automatically find good architectures given data-sets.

We view NAS as a bi-level combinatorial optimization problem (as per (Liu et al., 2019)) where we seek both the optimal architecture and its associated optimal parameters. Interestingly, this formulation generalizes the well-studied feature selection problem for linear prediction. This observation permits us to draw parallels between NAS algorithms and feature selection algorithms.

In particular, a plethora of NAS works have leveraged sampling methods including reinforcement learning (Liu et al., 2018; Zoph and Le, 2017; Zoph et al., 2018), evolutionary algorithms (Elsken et al., 2018a; Real et al., 2017; 2018), and Bayesian optimization (Kandasamy et al., 2018) to enumerate all possible architectures in a guided manner. However, interestingly, we do not often see successes of these sampling methods for feature selection. Indeed, these sample-based NAS often take hundreds to thousands of GPU-days to find good architectures, and can be barely better than random search (Elsken et al., 2018b).

Another popular NAS approach is analogous to sparse optimization or backward elimination for feature selection, e.g., (Han Cai, 2019; Liu et al., 2019; Pham et al., 2018). The approach starts with a super-graph that is the union of all possible architectures, and learns to down-weight the unnecessary edges gradually via gradient descent or reinforcement learning. Such approaches drastically cut down the search time of NAS. However, these methods require some domain knowledge on the optimal network size and the super-graph must fit into the GPU for efficient training.

In this work, we instead take an approach that is analogous to a forward feature selection algorithm in order to iteratively grow existing networks. Although forward methods such as orthogonal matching pursuit and least-angle regression are popular in feature selection and can often result in performance guarantees, there are only few works in NAS (Liu et al., 2017a) that take analogous approaches. We are interested in forward NAS approaches for multiple reasons. From a deployment point of view, practitioners may want to expand their existing models when extra model complexity and training computation become viable. Forward methods can utilize such extra computational resource without rebooting the training as in backward methods and sparse optimization. Furthermore, the iterative growth naturally results in a spectrum of models of various complexity and accuracy for practitioners to choose from. Unlike backwards approaches, forward methods need not specify a finite search space up front making them more general and easily used.

Specifically, inspired by forward feature selection algorithms and early neural network growth work (Fahlman and Lebiere, 1990), we propose a method (Petridish) of growing networks from small to large, where we opportunistically add shortcut connections in a fashion that is analogous to applying gradient boosting to the intermediate feature layers. To select from the possible shortcut connections, we also exploit sparsity-inducing regularizaiton while we train the eligible shortcuts alongside the existing networks. We experiment with it for both the popular cell-search (Zoph et al., 2018), where we seek a shortcut connection pattern and repeat it using a manually designed skeleton network to form an architecture, and the less popular but more general macro-search, where shortcut connections can be freely formed. Experimental results show Petridish macro-search to be better than previous macro-search NAS works on vision tasks, and brings macro-search performance up to par with cell-search counter to popular belief from early NAS works (Pham et al., 2018; Zoph and Le, 2017) that macro-search is inferior than cell-search. Petridish cell-search also finds models that are more cost-efficient than those from (Liu et al., 2019), while using similar training computation. This indicates that forward selection methods, though currently rarely used by the NAS community, can be exploited by future NAS algorithms.

A key tool throughout our algorithm design is amortization, where we trade off computational costs of different operations so they are similar up to a constant factor so as to guarantee that our approach never wastes more than a constant factor of computation. As an example, training the network has a cost, as does training extensions to the network. By doing both simultaneously with each amortizing the other's computational complexity we avoid significant waste computation.

We summarize our contribution as follows.

- We propose an approach to increase the complexity of neural networks iteratively during training. We alternate between two phases. The first expands the model with potential shortcut connections and trains them jointly. The second phase trims the previous potential connections using feature selection and continues training the model.
- The proposed approach can be applied to improve a small repeatable pattern (cell), and improve the macro network architecture directly, unlike most popular approaches that only focus on cells. This opens up neural architecture search to fields where no domain knowledge of the macro structure exists.
- On cell-search, the proposed method finds a model that achieves 2.61% error rate on CIFAR10 using 2.9M parameters within 5 GPU-days.
- On macro-search, the proposed method finds a model that achieves 2.83% error rate on

CIFAR10 using 2.2M parameters within 5 GPU-days.

- The proposed approach can warm start from existing networks, leveraging previous training results. Furthermore, it directly expands models on the lower convex hull of error rate vs. test-time computation, and is hence able to naturally produce a gallery of cost-effective models for applications to choose.

6.2 Background and References

One of the earliest neural architecture growth was by Fahlman and Lebiere (1990) termed the "Cascade-Correlation Learning Architecture" (C2) which has inspired Petridish. In C2, neurons of a neural network are trained iteratively. Once existing neurons are converged, C2 considers adding a candidate hidden neuron. The candidate hidden neuron before insertion to the network is connected to the input neurons and all currently existing hidden neurons. The weights of the incoming connections to this shadow neuron are optimized such that the correlation between the activations of this shadow neuron and the error at the output neurons is maximized. Then the shadow neuron is inserted into the network and its incoming weights are frozen. Its outgoing weights are then trained in the usual way. This idea of gradually expanding existing networks was also studied in a recent context (Cortes et al., 2017; Huang et al., 2018a) through the view of boosting networks.

The work of (Zoph and Le, 2017; Zoph et al., 2018) renewed interest in NAS in recent times. Their method uses a recursive neural network (RNN) as a controller network which is used to sample architectures. Each of these architectures are trained on separate machines and their resulting accuracies are used to update the parameters of the controller network via policy gradients (Williams, 1992). The majority of the time is spent in training each of the sampled architectures in parallel on independent machines. The resulting search times are generally on the order of thousands of GPU hours (See Table 6.1).

Pham et al. (2018) introduced a much more efficient version of this algorithm termed as Efficient Neural Architecture Search (ENAS) where the controller samples network architectures from a large super-graph of all possible architectures but trains them all jointly where the weights of edges which are common amongst the sampled architectures are shared across all of them at training time. This leads to orders of magnitude improvement in search times but still has the restriction that a super-graph to sample from must be constructed apriori.

Liu et al. (2017a) proposed a method which instead of using policy gradients as in Zoph et al. (2018), trains predictors on the results of training a batch of architectures to predict top-K architectures which are likely to do well in subsequent rounds in a progressive manner and hence termed as Progressive Neural Architecture Search (PNAS).

Liu et al. (2019) proposed a novel method based on bilevel optimization (Colson et al., 2007) termed as Differentiable Architecture Search (DARTS) which relaxes the originally discrete optimization problem to a continuous one and maintains two sets of continuous parameters: 1. The (architecture) parameters over the layer types and 2. The regular parameters of the network itself for each layer type. This is optimized in an alternating fashion where first the architecture parameters are trained alternated by the parameters of the layers of each type. Discrete architectures are then backed out by just selecting the architecture parameters which have the maximum value and discarding others. DARTS achieves impressive results on cell-search space with short search times.

Cai et al. (2018); Elsken et al. (2018a) both speed up architecture searches by incrementally modifying models from existing cost-effective models using evolutionary algorithms. This work differs from them in how the network is grown. In particular, we guide the growth with gradient boosting on intermediate layers, instead of using evolutionary samples for significant computational savings.

6.3 Neural Architecture Search as Optimization

Given a data sample x with label y, a neural network architecture α with parameters w produces a prediction $\hat{y}(x;\alpha,w)$ and suffers a prediction loss $\ell(\hat{y}(x;\alpha,w),y)$. The expected loss is then

$$\mathcal{L}(\alpha,w)=\mathbb{E}_{x,y\sim\mathcal{D}}[\ell(\hat{y}(x;\alpha,w),y)]\approx\frac{1}{|\mathcal{D}_{\text{train}}|}\sum_{(x,y)\in\mathcal{D}_{\text{train}}}\ell(\hat{y}(x;\alpha,w),y), \tag{6.1}$$

where $\mathcal{D}$ is the true distribution of data samples, and in practice, the loss $\mathcal{L}$ is estimated on the empirical training data $\mathcal{D}_{\text{train}}$. The problem of neural architectures search can be formulated as a bi-level optimization (Colson et al., 2007) of both the network architecture α and the model parameters w under the expected training loss $\mathcal{L}$ as follows.

$$\min_{\alpha}\mathcal{L}(\alpha,w(\alpha)),\quad s.t.\quad w(\alpha)=\arg\min_{w}\mathcal{L}(\alpha,w)\quad and\quad c(\alpha)\leq K, \tag{6.2}$$

where $c(\alpha)$ represents the test-time computational cost of the architecture, and K is some constant.

We formalize α as a set of discrete decisions on which operations to include in an architecture. Let $x_1,x_2,...,$ be intermediate layers, and $x_0=x$ be the input. Each layer x_i is a function of the previous layers, i.e., $x_i=f_i(x_0,x_1,...,x_{i-1})$ for some function f_i, though it is not necessary for x_i to directly use each of the previous layers. Each shortcut connection is defined by a triplet (x_j,x_i,op), where x_j and x_i $(j<i)$ represent the input and output layers, and op is a unary operation such as conv 3x3 and max pooling 3x3. Such a shortcut results in a tensor $op(x_j)$ that can be used directly by x_i. Shortcuts to x_i are combined by a merge operation at x_i, such as averaging, summation, or concatenation in order to form x_i. In this work, we set the merge operations as summation, unless we specify otherwise using ablation studies. Instead, we focus on the choice of the shortcut connections implying each α is an unordered collection of (x_j,x_i,op).

6.3.1 Connection to Feature Selection

Before delving into a proposed approach, we first draw an interesting connection of Eq. 6.2 to a well studied problem, feature selection for linear predictions:

$$\min_{\alpha}\frac{1}{2n}\|Y-X_\alpha w(\alpha)\|^2+\frac{\lambda}{2}\|w\|^2 \tag{6.3}$$

$$s.t.\quad w(\alpha)=(\frac{1}{n}X_\alpha^TX_\alpha+\lambda I)^{-1}\frac{1}{n}X_\alpha Y\quad and\quad c(\alpha)\leq K, \tag{6.4}$$

where $X \in \mathcal{R}^{n \times d}$ is the feature matrix of the n samples of d-dimensional features, $Y \in \mathcal{R}^n$ is the regression targets, and X_α selects the features included in α. We note that Eq. 6.2 generalizes Eq. 6.4, since $w(\alpha)$ solves for the optimal coefficient given the selected features.

This observation permits us to translate existing NAS algorithms to feature selection algorithms as discussed in the introduction and related work. In contrast to most other work, ours is based on forward selection, where feature are iteratively selected, or their coefficients are gradually increased. Unfortunately, common algorithms such as Forward Regression (FR) and its approximation Orthogonal Matching Pursuit (OMP), cannot directly be applied to the NAS problem, because both methods require computing $w(\alpha)$ at each architecture, with such computations taking a GPU-day on its own. Instead, we have to consider methods that approximate $w(\alpha)$ and α at the same time. Fortunately, one such forward algorithm for feature selection is Least-angle regression (LARS) (Efron et al., 2004).

In LARS, we compute the correlation between the residual of linear prediction and each feature, and find the feature with the largest absolute correlation. Then we update the coefficient of this feature until its absolute correlation is no longer the largest. One practical approximation of LARS is to iteratively update the coefficients of the most correlated feature with small steps, so that we avoid computing the line search analytically. Under this modification, LARS can be viewed as gradient boosting with small step sizes. In Eq. 6.2, the gradient of the empirical loss with respect to the prediction is

$$\nabla_{\hat{y}} \mathcal{L}(\alpha, w) = \mathbb{E}_{x,y \sim \mathcal{D}}[\nabla_{\hat{y}} \ell(\hat{y}(x; \alpha, w), y)]. \tag{6.5}$$

Under linear prediction, this gradient becomes the residual up to a constant, $\nabla_{\hat{y}} \mathcal{L}(\alpha, w) = \frac{1}{n}(X_\alpha w(\alpha) - Y)$. Under linear predictions, features can be viewed as weak learners. Hence, the correlations between the features and the residual are the correlations between the weak learners and the functional gradient with respect to predictions. The selected weak learner is then the one that can match the gradient the most. In other words, LARS follows gradient boosting to select weak learners.

6.4 A NAS Approach from Gradient Boosting

6.4.1 Gradient Boosting

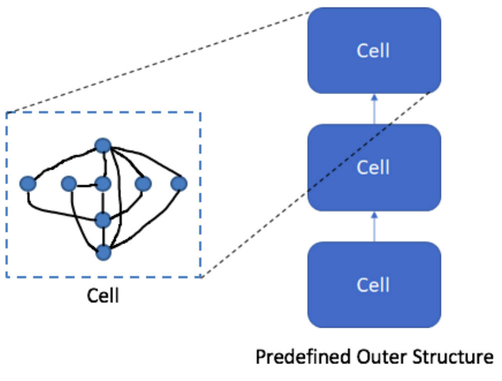
Let $\mathcal{H}$ be a space of weak learners. Gradient boosting matches weak learners $h \in \mathcal{H}$ to the functional gradient of the loss $\mathcal{L}$ with respect to the prediction $\hat{y}$, i.e., $\nabla_{\hat{y}} \mathcal{L}$ in Eq. 6.5. The weak learner that matches the negative gradient the best, h^*, is added to the ensemble of learners, i.e.,

$$h^* = \operatorname*{arg\,min}_{h \in \mathcal{H}} \langle \nabla_{\hat{y}} \mathcal{L}, h \rangle. \tag{6.6}$$

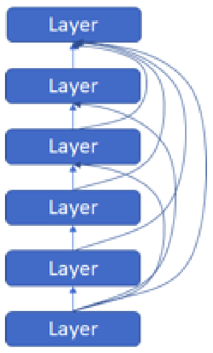
Then the predictor is updated to become $\hat{y} \leftarrow \hat{y} + \eta h^*$, where η is the learning rate.

6.4.2 Gradient-Boosting-Inspired NAS

Following gradient boosting strictly would limit the model growth to be only at the prediction of the network, $\hat{y}$. Instead, this work seeks to expand the expressiveness of the network at



(a)



(b)

Figure 6.1: (a) Cell-search applies found cells to a predefined outer structure. (b) Macro-search allows any connection.

intermediate layers, $x_1, x_2, ...$, jointly. Inspired by gradient boosting, we consider adding a weak learner $h_k \in \mathcal{H}_k$ at each x_k, where $\mathcal{H}_k$ (specified next) is the space of weak learners for layer x_k. h_k helps reduce the gradient of the loss $\mathcal{L}$ with respect to x_k, $\nabla_{x_k}\mathcal{L} = \mathbb{E}_{x,y\sim\mathcal{D}}[\nabla_{x_k}\ell(\hat{y}(x;\alpha,w),y)]$. In other words, we choose h_k with

$$h_k = \arg\min_{h\in\mathcal{H}_k}\langle h, \nabla_{x_k}\mathcal{L}(\alpha,w)\rangle = \arg\min_{h\in\mathcal{H}_k}\langle h, \mathbb{E}_{x,y\sim\mathcal{D}}[\nabla_{x_k}\ell(\hat{y}(x;\alpha,w),y)]\rangle. \tag{6.7}$$

Then we expand the model by adding a small step η in the direction of h_k to x_k. In other words, we replace each x_k with $x_k + \eta h_k$ in the original network. The next sections details the choice of the weak learner space, and how we learn h_k.

6.4.3 Search Space

Cell-search vs. Macro-search. The early architecture searches (Real et al., 2017; Zoph and Le, 2017) typically allow any layer to connect to any other layer. This is often referred to as macro-search. However, as a number of works (Pham et al., 2018; Real et al., 2018; Zoph et al., 2018) showcase that a more restricted search, cell-search, leads to better models, the community has almost abandoned macro-search. As illustrated in Fig. 6.1, in a cell-search, the search algorithm search for a local connection pattern called cell, such as the residual unit in a ResNet (He et al., 2016). The cells instruct how neighboring layers are connected, and we apply these patterns in a human defined outer structure to form the final network. For example, the outer structure may be a straight-forward feed-forward network that contains information of the total number of cells and where down-sampling happens. In contrast, in a macro-search, the search algorithm is allowed to connect any layer to another, so that there is no predefined outer structure, and there may not be repeatable patterns that can be considered as cells.

In this work, we revisit macro-search. For a fair comparison between macro-search and cell-search, we set the only difference between the two to be whether the connection pattern is repeated. Specifically, both start with the same initial seed model, which is a network built with

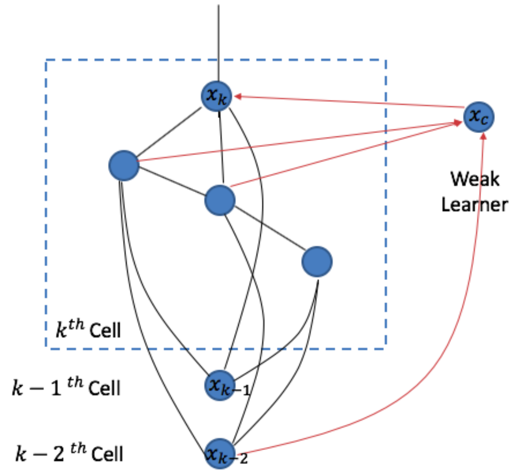


Figure 6.2: An example weak learner x_c from the search space $\mathcal{H}_k$.

simple cells. Both searches add weak learners at the same locations and at the same rate: one weak learner is always added to the end output of each cell per growth iteration. Cell-search adds the same connection pattern to each cell while macro-search allows different patterns. The space of the weak learners, which we detail next, is the same for both.

Weak Learner Space $\mathcal{H}$. Given an intermediate layer x_k to expand at, its associated weak learner space $\mathcal{H}_k$ is defined by four terms: the possible inputs, the possible unary operations on the inputs, a merge operation to combine the results, and the maximum number of inputs. Following (Liu et al., 2019; Real et al., 2018; Zoph et al., 2018), we limit weak learners to only take input from layers within the same cell or from the output layers of the previous two cells. The eligible unary operations are dependent on data-set. Following (Liu et al., 2019), seven operations are eligible for vision tasks: separable conv 3x3 and 5x5, dilated conv 3x3 and 5x5, max and average pooling 3x3, and identity. Following (Real et al., 2018; Zoph et al., 2018), the separable conv is repeated twice. The outputs of the unary operations are of the same shape as the output location x_k. Let the collection of eligible unary operations be $\texttt{Op}$. We determine through an ablation study in Sec. 6.5.4 how to merge the unary operations into a weak learner. For vision tasks, we found concatenation of the operations followed by a projection to reduce the filter size works the best. The maximum number of inputs is also data-set dependent, and for vision tasks, we set it to be $I_{max} = 3$, which we choose from ablation studies in experiments. Then the weak learner space $\mathcal{H}_k$ for a layer x_k is formally

$$\mathcal{H}_k = \{\texttt{cat\_proj}(op_1(z_1), ..., op_{I_{max}}(z_{I_{max}})) : z_1, ..., z_t \in \texttt{In}(x_k), op_1, ..., op_{I_{max}} \in \texttt{Op}\}, \tag{6.8}$$

where $\texttt{In}(x_k)$ is the collection of eligible input layers. Fig. 6.2 shows an example of a weak learner in the above space.

Additional Search Space Details. For the vision tasks, the initial model for both macro and cell-search is a modified ResNet (He et al., 2016), where we replace each 3x3 convolution with a 3x3 separable convolution. This is one of the simplest seeds within the search space of existing literature (Liu et al., 2019; Pham et al., 2018; Zoph et al., 2018). Following (Zoph et al., 2018),

Algorithm 9 Petridish.initialize_candidates

```
1: Input: (1) $L_x$, the list of layers in the current model (macro-search) or current cell (cell-search) in topological order; (2) is_out($x$), whether we can expand at $x$; (3) $\lambda$, hyper parameter for selection shortcut connections.
2: Output: (1) $L'_x$, the modified $L_x$ with weak learners $x_c$; (2) $L_c$, the list of $x_c$ created; (3) $\ell_{extra}$, the additional training loss.
3: $L'_x \leftarrow L_x$
4: $L_c \leftarrow$ empty list
5: $\ell_{extra} \leftarrow 0$
6: for $x_k$ in enumerate($L_x$) do
7:     if not is_out($x_k$) then
8:         continue
9:     end if
10:    Compute the eligible inputs In($x_k$), and index them as $z_1, ..., z_I$.
11:    $x_c \leftarrow \sum_{i=1}^{I} \sum_{j=1}^{J} \alpha_{i,j}^k op_j(\text{sg}(z_i))$.
12:    Insert the layer $x_c$ right before $x_k$ in $L'_x$.
13:    $\ell_{extra} \leftarrow \ell_{extra} + \lambda \sum_{i=1}^{I} \sum_{j=1}^{J} |\alpha_{i,j}^k|$.
14:    Append $x_c$ to $L_c$.
15:    Modify $x_k$ in $L'_x$ so that $x_k \leftarrow x_k + \text{sf}(x_c)$.
16: end for
```

we have six regular cells for each of the three scales of feature maps during training of the final found architectures, but have three regular cells per scale during search. Similarly, we have an initial channel size of $F = 32$ during final training and $F = 16$ during search. A transition cell is in between each neighboring resolutions, and it also starts as a modified residual unit. When we transfer the model to larger data-sets that require more than three resolutions, we use transition cells to first down-sample the image height and width to be no greater than 32 and then apply the found model. In macro-search, where no transition cells are specifically learned, we again use the the modified ResNet cells for initial transition in the transferred model.

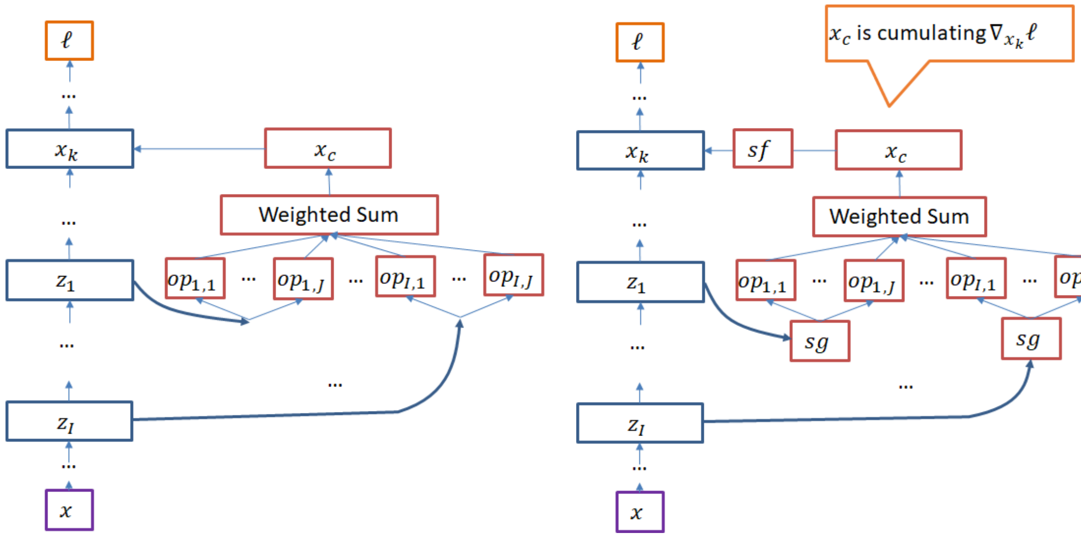
6.4.4 Joint Weak Learning

Given an intermediate layer x_k to expand at, we have $I = |\texttt{In}(x_k)|$ possible input layers and $J = |\texttt{Op}|$ possible operations. Hence, there are $\binom{IJ}{I_{max}}$ possible weak learners in the space $\mathcal{H}_k$, and it is computationally expensive to train each weak learner individually. Inspired by the parameter sharing works in NAS (Liu et al., 2019; Pham et al., 2018) and model compression in neural networks (Huang et al., 2017a), we propose to jointly train the weak learners in the union of them, and at the same time learn to select the shortcut connections. This process effectively amortizes the search through all weak learners against other weak learners so the computational cost is only a constant factor worse than for the chosen weak learner.

Algorithm 9 describes the proposed approach to train the weak learners. For now, let us assume the boolean variable This means that weak learning does not affect the parameters of the current model. Fig. 6.3b illustrates the weak learning modification to the current network.

Algorithm 10 Petridish.finalize_candidates

```
1: Inputs: (1) $L'_x$, the list of layers of the model in topological order; (2) $L_c$, list of selection
   modules in $L'_x$; (3) $\alpha^k_{i,j}$, the learned weights of the each $x_c$.
2: Output: A modified $L'_x$ with selected operations.
3: for $x_c$ in $L_c$ do
4:     Let $A = \{\alpha^k_{i,j} : i = 1, ..., I, j = 1, ..., J\}$ be the weights of operations in $x_c$.
5:     Sort $\{|a| : a \in A\}$, and let the operations associated with the largest $I_{max}$ value be
       $op_1, ..., op_{I_{max}}$.
6:     Replace $x_c$ with proj(concat($op_1, ..., op_{I_{max}}$)) in $L'_x$.
7: end for
8: Replace all sf($\cdot$) and sg($\cdot$) with identity in $L'_x$.
```



(a) (b)

Figure 6.3: Training of a weak learner x_c, so that it can (a) and cannot (b) affect the current model.

Combining Weak Learners. During joint weak learning, we combine all shortcut connections to x_k in a weighted sum as follows.

$$x_c = \sum_{i=1}^{I}\sum_{j=1}^{J} \alpha_{i,j} op_j(z_i), \tag{6.9}$$

where $op_j \in \texttt{Op}$ and $z_i \in \texttt{In}(x_k)$ enumerate all possible operations and inputs, and $\alpha_{i,j} \in \mathbb{R}$ is the weight of the shortcut $op_j(z_i)$. The next paragraphs explain how we simultaneously train and select the shortcuts to form a weak learner for x_k.

$L1$**-regularization.** Each $op_j(z_i)$ is normalized with batch-normalization to have zero mean and unit variance in expectation, so $\alpha_{i,j}$ reflects the importance of the operation. To learn the most important operations, we apply $L1$-regularization (Tibshirani, 1994) on the weight vector $\vec{\alpha}$ to encourage sparsity, i.e., we add the following loss during the fitting of x_c,

$$\lambda \|\vec{\alpha}\|_1 = \lambda \sum_{i=1}^{I} \sum_{j=1}^{J} |\alpha_{i,j}|, \tag{6.10}$$

where λ is a hyperparameter. $L1$-regularization, known as Lasso, induces sparsity in the parameter and is widely used for feature selection. It has also been successfully applied to model compression of neural networks such as in (Huang et al., 2017a).

Weak learning. The goal of weak learning is to match x_c with the negative gradient of the loss with respect to the layer x_k, i.e., we minimize

$$\langle \nabla_{x_k} \mathcal{L}, x_c \rangle = \langle \nabla_{x_k} \mathcal{L}, \sum_{i=1}^{I} \sum_{j=1}^{J} \alpha_{i,j} op_j(\text{sg}(z_i)) \rangle, \tag{6.11}$$

where sg is short for stop-gradient, an operation which treats each z_i as a constant, so that the optimization of weak learners does not affect the current network. Mathematically, $\text{sg}(x) = x$ during forward, and has zero gradient with respect to x during backward.

We add the loss 6.11 implicitly to the overall objective on line 15. A naive implementation adds the loss in Eq. 6.11 to the additional ℓ_{extra}, and backpropagates the network while only updating parameters in the weak learners x_c. However, this requires recording the intermediate gradients $\nabla_{x_k} \mathcal{L}$ during training. Interestingly, this can be avoided as described in Algorithm 9. Specifically, on line 15, we replace the layer x_k with $x_k + \text{sf}(x_c)$, where $\text{sf}(x_c) = x_c - \text{sg}(x_c)$, so that $\text{sf}(x_c) = 0$ during forward, and has gradient of identity with respect to x_c. As a result, for any parameter θ in weak learner x_c for intermediate layer x_k, its gradient during the backpropagation is

$$\nabla_\theta \mathcal{L} = \nabla_{x_k + \text{sf}(x_c)} \mathcal{L} \nabla_{x_c} \text{sf}(x_c) \nabla_\theta x_c = \nabla_{x_k} \mathcal{L} \nabla_\theta x_c = \nabla_\theta \langle \nabla_{x_k} \mathcal{L}, x_c \rangle. \tag{6.12}$$

This is the same as the gradient of the loss in Eq. 6.11 with respect to θ. Hence, exploiting sf and sg operations on line 15 and line 11, we can optimize both the current network and the weak learners at the same time without the weak learners affecting the network achieving amortization between network learning and weak learner learning. Furthermore, we do not force the training procedure to record $\nabla_{x_k} \mathcal{L}$ explicitly.

Warm-start. After appending the weak learners to an existing trained model, we warm-start the training with the parameters of the existing model, and initialize the weak-learner parameters randomly. Leveraging these existing model parameters, we can potentially spend fewer epochs per model, because we only need to fit the weak learners, which are shallow networks.

6.4.5 Weak Learner Finalization

In Algorithm 10, we finalize the weak learners. Since the weights $\alpha_{i,j}$ convey the importance of the associated shortcuts, we select for each x_c of Eq. 6.9 the top I_{max} shortcuts according to the absolute value of $\alpha_{i,j}$, and merge them to form the selected weak learner. The other operations are removed. We train the finalized model for a few epochs, warm-starting with the parameters from

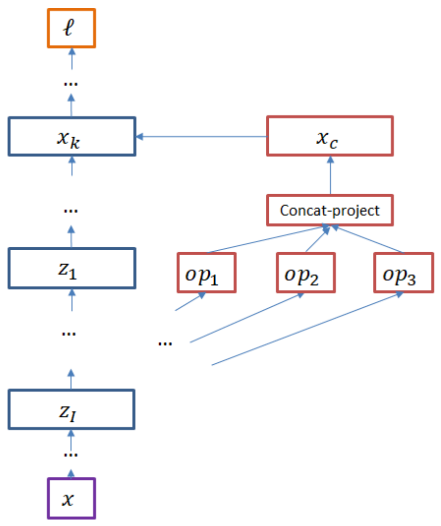


Figure 6.4: Weighted sum is replaced with concat-projection, when the top operations are chosen. Any sf or sg are also removed.

the weak learning phase. Although we train and select the shortcuts in a weighted sum, we found through ablation study in Sec. 6.5.4 that for vision tasks, the found models are more cost-effective if we merge the I_{max} selected shortcuts with concatenation-projection, as illustrated in Fig. 6.4. Existing NAS works (Liu et al., 2019; Pham et al., 2018; Real et al., 2018; Zoph et al., 2018) have a similar set-up, where intermediate layers within cells are concatenated, and the concatenation is immediately projected when it is an input to other cells.

6.4.6 Utilizing Parallel Workers

The proposed iterative architecture growth may be noisy due to the randomness during training of weak learners and the expanded models. By leveraging parallel workers, we can explore multiple growths to find more cost-effective models. The parallel workers can share knowledge and expand from any searched models, with this section describing their sampling procedure.

We maintain the lower convex hull of the performance of the searched models on the validation error versus test-time computation graph. The models on the hull are the most cost-efficient, because no mixture of other searched models is both more accurate and less expensive than any of them. To choose one model on the hull, we enter a while-loop iterating from the most accurate model that is within the computational budget K to the least accurate on the hull, and exit the loop with a model m with probability $1/(n(m)+1)$, where $n(m)$ is the number of times that model m has already been selected. This is because the next child model expanded from m is the best among the children with probability $1/(n(m)+1)$, assuming the children are uniformly drawn. We also favor the more accurate models as it is often more difficult to improve an already accurate model. In practice, we explore few models in total (< 50), so that the effect of different sampling on the hull is not clear given the limited search samples.

6.5 Selected Empirical Highlights

Following (Zoph et al., 2018), we first report the search results on CIFAR-10 (Krizhevsky et al., 2009) and the model transfer result to ImageNet (Russakovsky et al., 2015). Then we report ablation studies on hyper parameters of Petridish.

6.5.1 Search Results on CIFAR10

Set-up. We first apply the proposed algorithm to search for architectures on CIFAR-10 (Krizhevsky et al., 2009). During search, we use a fixed set of 45000 training images for training, and 5000 for validation. Both weak learner initialization and finalization are trained for 80 epochs, with a batch size 32 and a learning rate that decays from 0.025 to 0 in cosine decay (Loshchilov and Hutter, 2017). We apply drop-out (Larsson et al., 2017b) and cut-out (DeVries and Taylor, 2017) during search. The final found model is trained from scratch using the same parameters, except that it trains on all 50000 training images, and spends 600 epochs. Following (Liu et al., 2019; Zoph et al., 2018), we search on a shallower and slimmer version of the network, which has $N = 3$ normal cells per feature map resolution and $F = 16$ initial filter size. The final training is instead on a network with $N = 6$ and $F = 32$. Since Petridish macro-search is simply cell-search binding the cells to be the same, we transform macro-search results on $N = 3$ to models with $N = 6$ by repeating each normal cell twice. The initial seed model is trained for 200 epochs, and all subsequent children models with or without weak learners are trained for 80 epochs each, warm starting from their parent models' parameters.

Search Results. Table 6.1 depicts the test-errors, model parameters, and search computation of the proposed methods along with many state-of-the-art methods. Petridish cell search finds a model with 2.61% error rate with 2.5M parameters, in 5 GPU-days, which is at state-of-the-art level. Petridish macro search finds a model that achieves 2.83% error rate using 2.2M parameters in the same search computation. This is significantly better than any previous macro search results, and showcases that macro search can find cost-effective architectures that are previously only found through cell search.

Importance of initial models. Table 6.1 also showcase the impact of initial models to the final results of architecture search. This is an important topic, because existing literature has been moving away from macro architecture search, as early works (Pham et al., 2018; Real et al., 2018; Zoph et al., 2018) have shown that cell search results tend to be superior to those from macro search. However, this result may be explained by the superior initial models of cell search: the initial model of Petridish is one of the simplest models that any of the listed cell search methods would propose and evaluate, and it already achieves 4.6% error rate using only 0.4M parameters, a result is on-par or better than any other macro search results.

6.5.2 Transfer to ImageNet

We focus on the mobile setting for the model transfer results on ILSVRC (Russakovsky et al., 2015). Following (Zoph et al., 2018), we use 224x224 cropped input images, and apply to them a 3x3 conv with $F/4$ filters and stride of 2. Then we apply two transition cells to convert the feature map to 28x28 and F filters. For macro-search results, we apply the transition cell in the

Table 6.1: Comparison against state-of-the-art recognition results on CIFAR-10. Results marked with † are not trained with cutout. The first block represents approaches for macro-search. The second block represents approaches for cell-search.

Method	**# params** (mil.)	**Search** (GPU-Days)	**Test Error** (%)
Zoph and Le (2017)†	7.1	1680+	4.47
Zoph and Le (2017) + more filters†	37.4	1680+	3.65
Real et al. (2017)†	5.4	2500	5.4
ENAS macro (Pham et al., 2018)†	21.3	0.32	4.23
ENAS macro + more filters†	38	0.32	3.87
Lemonade I (Elsken et al., 2018a)	8.9	56	3.37
Petridish initial model ($N = 6$, $F = 32$)	0.4	–	4.6
Petridish macro	**2.2**	5	**2.83**
NasNet-A (Zoph et al., 2018)	3.3	1800	2.65
AmoebaNet-A (Real et al., 2018)	3.2	3150	3.3
AmoebaNet-B (Real et al., 2018)	2.8	3150	2.55
PNAS (Liu et al., 2017a)†	3.2	225	3.41
Heirarchical NAS (Liu et al., 2018)†	15.7	300	3.75
ENAS cell (Pham et al., 2018)	4.6	0.45	2.89
ENAS cell (Pham et al., 2018)†	4.6	0.45	3.54
Lemonade II (Elsken et al., 2018a)	3.98	56	3.50
Darts (Liu et al., 2019)	3.4	4	2.83
Darts random (Liu et al., 2019)	3.1	–	3.49
Cai et al. (2018)	5.7	8	2.49
Luo et al. (2018)†	3.3	0.4	3.53
PARSEC (Casale et al., 2019)	3.7	1	2.81
Petridish cell	**2.5**	5	**2.61**

seed model, i.e., residual units from (He et al., 2016) where conv is replaced with separated conv. We then treat the resulting tensor as the input image for the found architectures. We follow (Liu et al., 2019) to choose the training hyper parameters: we train for 250 epochs with batch size 128, weight decay $3 * 10^{-5}$, and initital SGD learning rate of 0.1 (decayed by a factor of 0.97 per epoch).

The top-1 error rate, the number of model parameters and the test-time computational cost in terms of mult-adds are shown in Table 6.2. The Petridish cell-search model achieves 26.0% error rate using 4.8M parameters and 598M multiply-adds, which is on par with state-of-the-art results listed in the second block of Table 6.2. By utilizing feature selection techniques to evaluate multiple model expansions at the same time, Petridish is able to find models faster by one or two orders of magnitude than early methods that train models independently, such as NASNet (Zoph et al., 2018), AmoebaNet (Real et al., 2018), and PNAS (Liu et al., 2017a). In comparison to super-graph methods such as DARTS (Liu et al., 2019), Petridish cell-search sacrifices about a factor of four search speed for the flexibility to grow from existing models.

Table 6.2: ILSVRC2012 transfer results. Petridish uses Isolated and the concat-projection (CP) modification by default.

Method	**# params** (mil.)	**# multi-add** (mil.)	**Search** (GPU-Days)	**top-1 Test Error** (%)
Inception-v1 (Szegedy et al., 2015)	6.6	1448	–	30.2
MobileNetV2 (Sandler et al., 2018)	6.9	585	–	28.0
NASNet-A (Zoph et al., 2017)	5.3	564	1800	26.0
NASNet-B (Zoph et al., 2017)	5.3	488	1800	27.2
AmoebaNet-A (Real et al., 2018)	5.1	555	3150	25.5
Path-level (Cai et al., 2018)	–	588	8.3	25.5
PNAS (Liu et al., 2017a)	5.1	588	225	25.8
DARTS (Liu et al., 2019)	4.9	595	4	26.9
SNAS (Xie et al., 2019)	4.3	522	1.6	27.3
Proxyless (Han Cai, 2019)	7.1	465	8.3	24.9
PARSEC (Casale et al., 2019)	5.6	–	1	26.0
Petridish macro (F=44)	4.3	511	5	28.5
Petridish cell (F=40)	3.2	500	5	27.0
Petridish cell (F=44)	4.8	598	5	26.0

The Petridish macro-search model achieves 28.5% error rate using 4.3M parameters and 511M multiply-adds, a comparable result to the human-designed models in the first block of Table 6.2. To the best of our knowledge, this is one of the first successful result to transfer macro-search results on CIFAR to ImageNet, showing that macro-search results can be transferred. However, we do observe a gap in error rates between Petridish macro and cell search both during search and the model transfer. This suggests that the larger macro search space is more difficult.

As Petridish gradually expand existing models, we naturally receive a gallery of models of various computational costs and accuracy. Figure 6.5 showcases the found models by Petridish with $F = 44$. We removed the seed model and points that are no longer on the lower convex hull.

6.5.3 Search Space: Direct versus Proxy

This section provides an ablation study on a common theme of recent neural architecture search works, where the search is conducted on a proxy space of small and shallow models, with results transferred to larger models later. In particular, since Petridish uses iterative growth, it need not consider the complexity of a super graph containing all possible models. Thus, Petridish can be applied directly to the final model setting on CIFAR-10, where $N = 6$ and $F = 32$. However, this implies each model takes about eight times the computation, and may introduce extra difficulty in convergence. Table 6.3 shows the transfer results of the two approaches to ILSVRC. We see that using a proxy not only results in a model with about 1% less errors, but also takes about one third of the search time, confirming that on image tasks the proxy approach is effective.

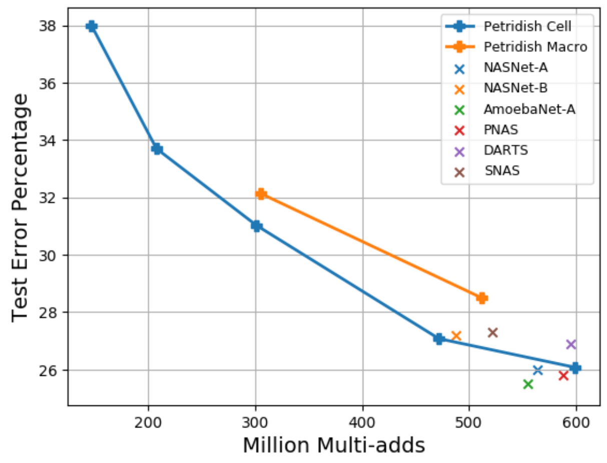


Figure 6.5: The performance convex hull of the found models by Petridish on ILSVRC. Petridish models are of parameter $N = 6$ and $F = 44$.

Method	**# params** (mil.)	**# multi-add** (mil.)	**Search** (GPU-Days)	**top-1 Test Error** (%)
Petridish cell proxy (F=44)	4.8	598	5	26.0
Petridish cell direct (F=40)	4.4	583	15.3	26.9

Table 6.3: Search space comparison between the direct space of $N = 6$ and $F = 32$ and the proxy space of $N = 3$ and $F = 16$ by evaluating their best mobile setting models on ILSVRC.

6.5.4 Weak Learner Space: Weighted Sum versus Concatenation-Projection

After selecting the shortcuts in Sec. 6.4.5, we concatenate them and project the result with 1x1 conv so that the result can be added to the output layer x_{out}. Here we empirically justify this design choice through consideration of two alternatives. We first consider applying the switch only to the final reported model. In other words, instead of using concat-project as the merge operation during search we switch all weak learner weighted-sums to concat-projections in the final model, which are trained from scratch to report results. We call this variant CP-end. Another variant where we never switch to concat-projection is called WS. Since concat-projection incurs additional computation to the model, we increase the channel size of WS variants so that the two variants have similar test-time multiply-adds for fair comparisons. The default Petridish option is switching the weak learner weighted-sums to concat-projections each time weak learners are finalized. We compare WS, CP-end and Petridish on the transfer results on ImageNet in Table 6.4, and observe that Petridish achieves similar or better prediction error using less test-time computation and training-time search.

6.5.5 Weak Learner Space: Number of Merged Operations

As we initialize all possible shortcuts during weak learning, we need decide I, the number of them to select for forming the weak learner. On one hand, adding complex weak learners can

Table 6.4: ILSVRC2012 transfer results. Ablation study on the choice of weighted-sum (WS), concat-projection at the end (CP-end), or the Petridish default merge operation in finalized weak learners. The searches were done with parameter initial channel $F = 32$ and s number of regular cells per resolution of $N = 6$.

Method	**# params** (mil.)	**# multi-add** (mil.)	**Search** (GPU-Days)	**top-1 Test Error** (%)
WS macro(F=48)	5.9	756	29.5	32.5
CP-end macro (F=36)	5.4	680	29.5	29.1
Petridish macro (F=32)	4.9	593	27.2	29.4
WS cell (F=48)	3.3	477	22.8	32.7
CP-end cell (F=44)	4.7	630	22.8	27.2
Petridish cell (F=40)	4.4	583	15.3	26.9

Table 6.5: Test error rates on CIFAR-10 by models found with different weak learner complexities.

Number of Shortcuts	Average Lowest Error Rate
$I = 2$	3.08
$I = 3$	2.68
$I = 4$	2.93

boost performance rapidly. On the other, this may add sub-optimal weak learners that hinder future growth. We test the choice of $I = 2, 3, 4$ during search. We run with each choice five times, and take the average of their most accurate models that take under 60 million multi-adds on the CIFAR model with $N = 3$ and $F = 16$. Models in this range are chosen, because their transferred models to ILSVRC can have 600 million multi-adds with standard setups of (Zoph et al., 2018), and hence, they are natural candidate models for ILSVRC mobile setting. Table 6.5 reports the test error rates on CIFAR10, and we see that $I = 3$ yields the best results.

6.5.6 Weaker Learner Training: Joint versus Isolated training with Parent Model

An interesting consideration is whether to stop the influence of the weak learners to the models during the weak learning. On one hand, we eventually want to add the weak learners into the model and allow them to be backpropagated together to improve the model accuracy. On the other hand, the introduction of untrained weak learners to trained models may negatively affect the training. Furthermore, the models may develop dependency on weak-learner shortcuts that are not selected, which can also negatively affect the future models. To study the effects through an ablation study, we replace the occurrence of sf and sg with identity in Algorithm 9, so that the weak learners are directly added to the models, as illustrated in Fig. 6.3a. We call this variant Joint, and compare it against the default Petridish. Table 6.6 showcases the transfer results of Isolated and Joint to ImageNet. We compare Petridish cell (F=40) with Joint cell (F=32), two models that have similar computational cost but very different accuracy, and we observe that Isolated leads to much better model than Joint for cell-search.

Table 6.6: ILSVRC2012 transfer results. Ablation study on the choice of Joint and Isolated for training the weak learners. The search were with parameter initial channel $F = 32$ and number of regular cell per resolution $N = 6$.

Method	**# params** (mil.)	**# multi-add** (mil.)	**Search** (GPU-Days)	**top-1 Test Error** (%)
Petridish Joint cell (F=32)	4.0	546	20.6	32.8
Petridish cell (F=40)	4.4	583	15.3	26.9

6.6 Discussion

Since the NAS problem is a combinatorial optimization, we have to approach it with either better approximation algorithms, or utilize the special conditions of the search space itself. In particular, a search space on CIFAR10 is studied by (Ying et al., 2019), which shows that architectures that are similar also have similar statistical performances. This suggests that a local search where models are changed iteratively and gradually can be very efficient if the starting model is already near the optimal model. Luckily this can often be the case. The benchmark results concludes that the best human designed models such as Resnets, DenseNet, and Inception, are all close to the pareto frontier of the computation versus error plot, so that these models are naturally good starting points, as evidenced by this work.

6.7 Conclusion

In this work, we formulate the neural architecture search problem (NAS) as a bi-level optimization problem, which also generalizes the anytime linear prediction problem. Insetad of exhaustive search, backward elimination, or sparse optimization approaches, we create an efficient forward search procedure inspired by gradient boosting and least-angle regression for feature selection. We also speed up the training of the weak learners by jointly training the union of all possible weak learners, and at the same time learn to select the most influential subset to form the final weak learner. We demonstrate the search on CIFAR10 and transfer the result to ILSVRC2012, with this iterative approach demonstrating state-of-the-art models with a small number of GPU-days for training.

Chapter 7

Discussion and Conclusion

7.1 Discussion and Future Works

7.1.1 Dynamic Models with Data-Dependent Computational Graphs

In this work, we only consider anytime predictors to have a sequential computational graph, where a fixed sequence of computation is used for generating anytime results for all data samples. However, it is also possible to form anytime predictions via computational graphs that depend on the input data samples, so that intermediate computation not only provides valid early predictions, but also determines the computational graph of the subsequent procedure. For instance, decision trees are natural anytime predictors with branching structures: we can stop the tree early, and the predict using the deepest tree node visited.

A number of existing works already considered dynamic models for balancing test-time computation and accuracy. (Karayev et al., 2012) approach the feature sequencing problem in anytime linear prediction by formulating it as a Markov decision process, and the partial results using computed features also determine which next features are computed. (Xu et al., 2013b) train a tree of classifiers to determine the order to compute features. (Wang et al., 2017) train neural networks to dynamically skip a number of layers based on early features. (Shazeer et al., 2017) train a large number of networks and use a controller network to determine for each data sample which networks are activated.

However, most of these existing works apply the dynamic models to the budgeted prediction problem, i.e., they minimize the average test computation, subject to not degrading prediction quality much. As a result, each data sample has a fixed early-exit, after which no improvement to the prediction on this sample is made. Particularly, it remains to be considered how to design dynamic neural networks for anytime predictions, i.e., each sample is predicted with an anytime neural network that is dynamically selected based on the sample itself.

7.1.2 Game Theoretical Approach to Training Anytime Predictors

In Chapter 4, we formulated training anytime neural networks as a multi-objective problem, and we approach it by optimizing the anytime losses in an adaptively weighted sum. An interesting alternative approach is to consider the problem as a game, where an adversary chooses the computational budget where the interruption occurs, and the learner is to ensure that at all budgets it is nearly at the best it can do. For instance, we may measure the performance at each budget with the relative increase in error rate in comparison to a model that specifically trained for that budget. Then the adversary maximizes over the budgets to increase this relative error rate, and the learner is to minimize the maximum relative increment.

One challenge to this approach, however, is to determine the objective function. The relative, instead of absolute, performance against an expert at each budget is necessary, because if otherwise, the problem may be overwhelmed by the different scales of the objectives at different budgets. However, computing this relative performance gap may require one to train many experts at various budgets. It will be interesting to consider how many experts we really need to compute, or whether there are formulations to avoid them.

7.1.3 Determine When to Grow Models in Anytime Learning

When a prediction model seems to not perform well, it can be the result of ill-optimized parameters, or it can be because the model architecture is not suitable for the problem. To address the former issue, one needs to optimize the model further, whereas against the latter issue, one needs to modify the architecture itself. It will be interesting to have a principled way to determine which action is the right one.

In Chapter 6, we studied anytime learning via neural architecture search, where we addressed the above problem in an ad-hoc manner. We trained each model with a small number of fixed epochs and then attempt to grow its architecture. In the visual recognition problem that we considered, these small number of epochs are often enough to tell apart performances of the different models. However, in general problem, we do not have a principled way to determine whether we have enough optimization on the existing models.

7.2 Conclusion

In this thesis we consider the trade-off between computation and accuracy for predictors at both testing and training time. We approach the balance between these two opposing factors with anytime algorithms, which always prepare valid partial results in case of budget depletion and produce better results if extra computation is given. Such a approach is taken because it can automatically adjust to and utilize any agnostic budget limit.

We start off with anytime linear predictors, for which we show that cost-aware greedy methods can achieve near-optimal predictions uniformly. However, we also discovered that by combining multiple weak predictors, such as features in linear prediction, ensemble-based anytime predictors have a a limitation in how well they can do in comparison to the optimal. Specifically, we establish a bi-criteria lower and upper bound for anytime predictors, showing that they can and only can compete against the optimal combination that has a lower cost than them.

This discovery dictates that anytime predictors need to look beyond ensemble methods, and thus, we develop anytime neural networks, where anytime predictions are trained jointly as a multi-objective problem within a single predictor. We also show that by combining anytime predictors instead of regular predictors, one can improve the bi-criteria bound. This indicates that the future of anytime prediction may rely on a combination of the traditional ensemble approaches and creative anytime models designs.

We also address the concern with training efficiency in multiple ways. For large stochastic data streams, we developed streaming gradient boosting, so that this traditionally iteratively trained model can be trained on a data stream. For the large search space of neural architecture design, we draw a connection between feature selection and neural architecture search, and develop an iterative growth algorithm that is inspired by gradient boosting, which we previously leveraged for anytime prediction. This suggests that anytime prediction and anytime learning are inherently connected, and they may be studied together in the future.

Bibliography

Elizabeth J Atkinson, Terry M Therneau, L Joseph Melton, Jon J Camp, Sara J Achenbach, Shreyasee Amin, and Sundeep Khosla. Assessing fracture risk using gradient boosting machine (gbm) models. *Journal of Bone and Mineral Research*, 2012. 5.1

L. J. Ba and R. Caruana. Do deep nets really need to be deep? In *Proceedings of NIPS*, 2014. 2.2.2, 4.2

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *ICML*, 2009. 2.2.1, 4.2

Alina Beygelzimer, Elad Hazan, Satyen Kale, and Haipeng Luo. Online gradient boosting. In *NIPS*, pages 2449–2457, 2015a. 5.1, 5.2, 5.3.1, 5.5.1, 5.8.2.2

Alina Beygelzimer, Satyen Kale, and Haipeng Luo. Optimal and adaptive algorithms for online boosting. In *ICML*, pages 2323–2331, 2015b. 1.2, 5.1, 5.2, 5.3.1, 5.4, 5.5.1.0.1

Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *IJCAI*, 1989. 4.1

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for fast test-time prediction. In *ICML*, 2017. 1.1, 2.2.3, 4.1, 4.2, 4.6

S. Brubaker, J. Wu, J. Sun, M. Mullin, and J. Rehg. On the Design of Cascades of Boosted Ensembles for Face Detection. *International Journal of Computer Vision*, pages 65–86, 2008. 1.2, 2.2.3, 3.1

Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In *ICML*, 2018. 6.2, 6.1

Zhaowei Cai, Mohammad J. Saberian, and Nuno Vasconcelos. Learning Complexity-Aware Cascades for Deep Pedestrian Detection. In *International Conference on Computer Vision, ICCV*, 2015. 1.2, 2.2.3, 3.1, 4.1, 4.2

Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. Probabilistic neural architecture search. In *arxiv.org/abs/1902.05116*, 2019. 6.1, 6.2

Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004. 1.2, 5.1, 5.3

Olivier Chapelle and Yi Chang. Yahoo! Learning to Rank Challenge Overview. JMLR Workshop and Conference Proceedings, 2011. 3.5.1, 3.5.5

Olivier Chapelle, Yi Chang, and Tie-Yan Liu, editors. *Proceedings of the Yahoo! Learning to Rank Challenge, held at ICML 2010*, volume 14 of *JMLR Proceedings*, 2011. 5.1

Minmin Chen, Kilian Q. Weinberger, Olivier Chapelle, Dor Kedem, and Zhixiang Xu. Classifier Cascade for Minimizing Feature Evaluation Cost. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012a. 3.1, 3.5.2, 4.2

Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *ICCV*, 2017. 2.2.1, 4.2, 4.3

Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu. An online boosting algorithm with theoretical justifications. In *ICML*, 2012b. 5.1, 5.2

Benot Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. In *Annals of operations research*, 2007. 6.2, 6.3

Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. In *ICML*, 2017. 6.2

Abhimanyu Das and David Kempe. Submodular meets Spectral: Greedy Algorithms for Subset Selection, Sparse Approximation and Dictionary Selection . In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011. 1.2, 3.1, 3.3, 3.3

Terrance DeVries and Graham Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. 6.5.1

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004. 2.2.2, 6.3.1

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. 2018a. 6.1, 6.2, 6.1

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *CoRR*, abs/1808.05377, 2018b. 1.2, 6.1

Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *NIPS*, 1990. 6.1, 6.2

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956. 5.2

Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995. 5.1, 5.2, 5.5.1.0.1

Yoav Freund and Robert E Schapire. A short introduction to boosting. In *Journal of Japanese Society for Artificial Intelligence*, 1999. 5.2

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001. 5.1, 5.4, 5.5.1, 5.6.2, 5.8.6, 5.8.6.1

Wei Gao, Lu Wang, Rong Jin, Shenghuo Zhu, and Zhi-Hua Zhou. One-pass auc optimization. In *Artificial Intelligence Journal*, volume 236, pages 1–29, 2016. 5.4

H. Grabner and H. Bischof. On-line boosting and vision. In *CVPR*, volume 1, pages 260–267, 2006. 5.2

H. Grabner, C. Leistner, and H Bischof. Semisupervised on-line boosting for robust tracking. In *ECCV*, page 234 247, 2008. 5.2

Joshua Grass and Shlomo Zilberstein. Anytime Algorithm Development Tools. *SIGART Bulletin*, 1996. 4.1

Alexander Grubb and Drew Bagnell. Generalized boosting algorithms for convex optimization. In *ICML*, 2011. 1.2, 5.1, 5.3.1, 5.4, 5.5.1, 5.5.1, 5.5.1.0.1

Alexander Grubb and Drew Bagnell. Speedboost: Anytime prediction with uniform near-optimality. In *AISTATS*, pages 458–466, 2012a. 5.1

Alexander Grubb and J. Andrew Bagnell. SpeedBoost: Anytime Prediction with Uniform Near-Optimality. In *the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012b. 1.2, 2.2.1, 3.1, 4.1, 4.2

Jiaqi Guan, Yang Liu, Qiang Liu, and Jian Peng. Energy-efficient amortized inference with cascaded deep classifiers. In *arxiv preprint, arxiv.org/abs/1710.03368*, 2017. 1.1, 4.1, 4.2, 4.6

Song Han Han Cai, Ligeng Zhu. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 6.1, 6.2

Elad Hazan and Satyen Kale. Beyond the regret minimization barrier: Optimal algorithms for stochastic strongly-convex optimization. *Journal of Machine Learning Research*, 15:2489–2512, 2014. 5.1, 5.3

Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007. 1.2, 5.1, 5.2, 5.5.2

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 4.1, 1, 2, 4.5.1, 4.8, 6.4.3, 6.4.3, 6.5.2

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *Deep Learning and Representation Learning Workshop, NIPS*, 2014. 2.2.2, 4.2

Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *UAI*, 1987. 4.1

Hanzhang Hu, Alexander Grubb, Martial Hebert, and J. Andrew Bagnell. Efficient feature group sequencing for anytime linear prediction. In *UAI*, 2016. 4.2

Furong Huang, Jordan Ash, John Langford, and Robert Schapire. Learning deep resnet blocks sequentially using boosting theory. In *ICML*, 2018a. 6.2

G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. In *ICLR*, 2018b. (document), 2.2.1, 4.1, 4.2, 4.3, 4.2, 4.5.1, 4.5.2, 4.5.3, 4.8

Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. *arXiv preprint arXiv:1711.09224*, 2017a. 2.2.2, 6.4.4, 6.4.4

Gao Huang, Zhuang Liu, Kilian Q. Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *CVPR*, 2017b. 4.1, 4.5.1, 4.8

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *NIPS*, 2016. 2.2.2, 4.2

Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt

Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. In *arxiv preprint: 1602.07360*, 2016. 2.2.2, 4.2

Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, and Francis R. Bach. Proximal Methods for Sparse Hierarchical Dictionary Learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010. 3

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26*, 2013. 5.5.2

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. In *NIPS*, 2018. 6.1

Nikos Karampatziakis and Paul Mineiro. Discriminative Features via Generalized Eigenvectors. In *the 31th International Conference on Machine Learning, (ICML)*, 2014. 3.7.1

Sergey Karayev, Tobias Baumgartner, Mario Fritz, and Trevor Darrell. Timely Object Recognition. In *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2012. 3.1, 3.5.2, 2, 4.2, 7.1.1

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR, arXiv:1412.6980*, 2015. 5.6.2

Ronny Kohavi and Barry Becker. Adult data set. UCI Machine Learning Repository, 1996. 5.6

Andreas Krause and Daniel Golovin. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*, 2012. 3.3

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 4.3, 4.5.1, 4.8, 6.5, 6.5.1

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012. 4.1

G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017a. 2.2.1, 4.2

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017b. 6.5.1

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. In *Intelligent Signal Processing*, 2001. 3.7.1, 5.6

Chen-Yu Lee, Saining Xie, Patrick W. Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *AISTATS*, 2015. 2.2.1, 4.1, 4.2, 4.3, 1

Leonidas Lefakis and Francois Fleuret. Joint Cascade Optimization Using a Product of Boosted Classifiers. In *Advances in Neural Information Processing Systems (NIPS)*. 2010. 1.2, 2.2.3, 3.1, 4.2

C. Leistner, A. Saffari, P. M. Roth, and H Bischof. On robustness of on-line boosting - a competitive study. In *ICCV Workshop on On-line Learning for Computer Vision*, 2009. 5.2

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In

ICLR, 2017. 2.2.2, 4.2

M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`. 5.6

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *CoRR*, abs/1712.00559, 2017a. 6.1, 6.2, 6.1, 6.5.2

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018. 6.1, 6.1

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. 2019. 6.1, 6.2, 6.4.3, 6.4.3, 6.4.4, 6.4.5, 6.5.1, 6.1, 6.5.2

Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *arxiv preprint:1708.06519*, 2017b. 2.2.2, 4.2

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017. 6.5.1

Aurelie C. Lozano, Grzegorz Swirszcz, and Naoki Abe. Grouped Orthogonal Matching Pursuit for Variable Selection and Prediction. In *Neural Information Processing Systems (NIPS)*, 2009. 3.1, 3.5.3, 3.5.4

Aurelie C. Lozano, Grzegorz Swirszcz, and Naoki Abe. Group Orthogonal Matching Pursuit for Logistic Regression. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15, 2011. 3.1, 3.5.4

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NIPS*, 2018. 6.1

Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. In *NIPS*, 2000. 5.1

P. McCullagh and J. A. Nelder. *Generalized Linear Models (Second edition)*. London: Chapman & Hall, 1989. 3.7

Alan J. Miller. Subset Selection in Regression. In *Journal of the Royal Statistical Society. Series A (General), Vol. 147, No. 3, pp. 389-425*, 1984. 3.1

Woonhyun Nam, Piotr Dollár, and Joon Hee Han. Local decorrelation for improved pedestrian detection. In *NIPS*, pages 424–432, 2014. 5.1

Feng Nan and Venkatesh Saligrama. Dynamic model selection for prediction under a budget. In *NIPS*, 2017. 4.2

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. 4.5.1, 4.8

A. Odena, D. Lawson, and C. Olah. Changing model behavior at test-time using reinforcement. In *Arxive preprint: 1702.07780*, 2017. 4.2

Nikunj C. Oza and Stuart Russell. Online bagging and boosting. In *AISTATS*, pages 105–112, 2001. 5.1, 5.2

Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal Matching Pursuit : recursive function approximation with application to wavelet decomposition. In *Asilomar Conference on Signals, Systems and Computers*, 1993. 2.2.2, 3.1

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018. 6.1, 6.2, 6.4.3, 6.4.3, 6.4.4, 6.4.5, 6.5.1, 6.1

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016. 2.2.2, 4.2

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *CoRR*, abs/1703.01041, 2017. 6.1, 6.4.3, 6.1

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. 6.1, 6.4.3, 6.4.3, 6.4.5, 6.5.1, 6.1, 6.5.2

Lev Reyzin. Boosting on a budget: Sampling for feature-efficient prediction. In *the 28th International Conference on Machine Learning (ICML)*, 2011. 1.2, 2.2.1, 3.1, 4.2

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015. 4.5.1, 2, 6.5, 6.5.2

Robert E Schapire and Yoav Freund. *Boosting: Foundations and algorithms*. MIT press, 2012. 5.5.1

Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2011. 5.6.1

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 *ICLR*, 2017. 7.1.1

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 4.1

J. Sochman and J. Matas. WaldBoost: Learning for Time Constrained Sequential Detection. In *the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 1.2, 2.2.3, 3.1

M. Streeter and D. Golovin. An Online Algorithm for Maximizing Submodular Functions. In *Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS)*, 2008. 3.1, 3.3

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017. 2.2.1, 4.2

Robert Tibshirani. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994. 2.2.2, 3.1, 6.4.4

Andreas Veit and Serge Belongie. Convolutional networks with adaptive computation graphs. *arXiv preprint arXiv:1711.11503*, 2017. 2.2.3, 4.2

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, volume 1. IEEE, 2001a. 5.1

Paul A. Viola and Michael J. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001b. 2.2.3, 3.1, 4.1, 4.2

Xin Wang, Fisher Yu, Zi-Yi Dou, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. *arXiv preprint arXiv:1711.09485*, 2017. 2.2.3, 4.2, 7.1.1

K.Q. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature Hashing for Large Scale Multitask Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, 2009. 2.2.1, 3.1, 4.2

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, 1992. 6.2

Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *ICCV*, 2015. 2.2.1, 4.2, 4.3

Saining Xie, Ross Girshick, Piotr Dollr, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 4.1

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: Stochastic neural architecture search. In *ICLR*, 2019. 6.2

Z. Xu, K. Weinberger, and O. Chapelle. The Greedy Miser: Learning under Test-time Budgets. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2012. 2.2.1, 3.1, 4.2

Z. Xu, M. Kusner, G. Huang, and K. Q. Weinberger. Anytime Representation Learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013a. 2.2.1, 3.1, 4.2

Z. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle. Classifier cascades and trees for minimizing feature evaluation cost. *Journal of Machine Learning Research*, 15(1): 2113–2144, 2014. 1.2, 2.2.3, 3.1, 4.1, 4.2

Zhixiang Xu, Matt Kusner, Kilian Q. Weinberger, and Minmin Chen. Cost-sensitive tree of classifiers. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 131–141, 2013b. 7.1.1

Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Convolutional channel features. In *ICCV*, pages 82–90, 2015. 5.1

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *arxiv.org/abs/1902.09635*, 2019. 6.6

Ming Yuan and Yi Lin. Model Selection and Estimation in Regression with Grouped Variables. *Journal of the Royal Statistical Society*, 2006. 2.2.2, 3.1, 3.5.2

Amir R. Zamir, Te-Lin Wu, Lin Sun, William Shen, Jitendra Malik, and Silvio Savarese. Feedback networks. In *CVPR*, 2017. 2.2.1, 4.1, 4.2, 4.3, 4.5.3

Tong Zhang. On the Consistency of Feature Selection using Greedy Least Squares Regression.

Journal of Machine Learning Research, 10:555–568, 2009. 3.1

Tong Zhang and Bin Yu. Boosting with early stopping: Convergence and consistency. 33: 15381579, 2005. 5.2

Yanru Zhang and Ali Haghani. A gradient boosting method to improve travel time prediction. *Transportation Research Part C: Emerging Technologies*, 58:308–324, 2015. 5.1

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017. 2.2.1, 4.2

Chao Zhu and Yuxin Peng. Group cost-sensitive boosting for multi-resolution pedestrian detection. In *AAAI*, 2016. 5.1

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 1.1, 1.2, 6.1, 6.2, 6.4.3, 6.1

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 6.1, 6.2, 6.4.3, 6.4.3, 6.4.3, 6.4.5, 6.5, 6.5.1, 6.5.2, 6.1, 6.5.5