AdaNet: Adaptive Structural Learning of Artificial Neural Networks

Conference Paper · July 2016

CITATIONS
13

READS
189

5 authors, including:

Corinna Cortes
Google Inc.
89 PUBLICATIONS **34,247** CITATIONS
SEE PROFILE

Xavi Gonzalvo
Google Inc.
22 PUBLICATIONS **143** CITATIONS
SEE PROFILE

Mehryar Mohri
Courant Institute of Mathematical Sciences
200 PUBLICATIONS **8,083** CITATIONS
SEE PROFILE

AdaNet: Adaptive Structural Learning of Artificial Neural Networks

Corinna Cortes
Google Research
New York, NY 10011
corinna@google.com

Xavi Gonzalvo
Google Research
New York, NY 10011
xavigonzalvo@google.com

Vitaly Kuznetsov
Google Research
New York, NY 10011
vitalyk@google.com

Mehryar Mohri
Courant Institute and Google
New York, NY 10011
mohri@cims.nyu.edu

Scott Yang
Courant Institute
New York, NY 10011
yangs@cims.nyu.edu

Abstract

We present a new theoretical framework for analyzing and learning artificial neural networks. Our approach simultaneously and adaptively learns both the structure of the network as well as its weights. The methodology is based upon and accompanied by strong data-dependent theoretical learning guarantees. We present some preliminary results to show that the final network architecture adapts to the complexity of a given problem.

1 Introduction

Deep learning has become a very powerful framework for machine learning and has enjoyed strong success in many areas in recent years. However, despite the compelling arguments for using neural networks as a general template for solving machine learning problems, the training and design of the right network has been filled with many theoretical gaps and practical concerns.

For training a network, one needs to specify an often large network architecture with several layers, and then solve a difficult non-convex optimization problem. The pre-specified architecture is often treated as a hyperparameter which is tuned using a validation set. From an optimization perspective, there is no guarantee of stability of an output model or near optimality of the learning objective, and often, one needs to implement ad hoc methods (e.g. gradient clipping [Pascanu et al., 2013]) to produce coherent models. From the statistical standpoint, large-scale hyperparameter tuning is extremely wasteful of data (due to cross validation), and can also exhaust a lot of time and resources (e.g. random search [Bergstra et al., 2011]). Moreover, if a network architecture is specified a priori this imposes a stringent lower bound on the complexity of the model making it prone to overfitting when there is insufficient data.

In this paper, we attempt to remedy some of these issues. Accepting the general structure of a neural network as an effective parametrized model for supervised learning we introduce a framework for training neural networks that adapts the structure and complexity of the network to the difficulty of the particular problem at hand, with no pre-defined architecture. Starting from a simple single layer neural network, we will add more neurons and additional layers as needed. The additional neurons that we add will be carefully selected and penalized according to rigorous estimates from the theory of statistical learning. This will serve as a catalyst for the sparsity of our model as well as the strong generalization bonds that we will be able to derive. Incredibly, our method will also turn out to be convex and hence more stable than the current methodologies employed.

There has been extensive work involving structure learning for neural networks (e.g. [Kwok and Yeung, 1997, Leung et al., 2003, Islam et al., 2003, Lehtokangas, 1999, Islam et al., 2009]). All these publications seek to grow and prune the neural network architecture using some heuristic (e.g. genetic, information theoretic, or correlation). The structure learning algorithm introduced in this paper is based directly on optimizing generalization performance, which is precisely the learning goal in the batch setting.

2 Theory

Let $\mathcal{X}$ denote the input space. We consider the standard supervised binary classification scenario and assume that training and test points are drawn i.i.d. according to some distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1, +1\}$ and denote by $S = ((x_1, y_1), \ldots, (x_m, y_m))$ a training sample of size m drawn according to $\mathcal{D}^m$. Given any $x \in \mathcal{X}$, we denote by $\mathbf{\Phi}(x) \in \mathbb{R}^{n_0}$ the feature representation of x.

We consider a set of neural network models that are parametrized as follows $f = \sum_{k=1}^{l} \sum_{j=1}^{n_k} w_{k,j} h_{k,j}$, where $h_{k,j}$ denotes j-th node in k-the layer and f can be thought of an output unit. Note that in our model the output unit is connected to every node in the network with weights $w_{k,j}$ on each edge. Standard architectures are a special case of this with $w_{k,j} = 0$ for all layers k except the penultimate one. Each $h_{k,j}$ is chosen from a family of functions $\mathcal{H}_k^{(p)}$ defined in the following way:

$$\mathcal{H}_1^{(p)} = \left\{ x \mapsto \mathbf{u} \cdot \mathbf{\Phi}(x) \colon \mathbf{u} \in \mathbb{R}^{n_0}, \|\mathbf{u}\|_p \leq \Lambda_1 \right\} \tag{1}$$

$$\mathcal{H}_k^{(p)} = \left\{ x \mapsto \left(\sum_{j=1}^{n_{k-1}} u_j (\varphi_{k-1} \circ h_j)(x) \right) \colon \mathbf{u} \in \mathbb{R}^{n_{k-1}}, \|\mathbf{u}\|_p \leq \Lambda_k, h_j \in \mathcal{H}_{k-1}^{(p)} \right\}, \ (\forall k > 1) \tag{2}$$

where $\Lambda_k > 0$ is a hyperparameter and where φ_k is an activation function. The choice of norm p here is left to the learner and will determine both the sparsity of the network and the accompanying learning guarantee of the resulting model. The following data-dependent learning guarantee holds.

Theorem 1 (AdaNet Generalization Bound)**.** *Let $\hat{l} \in [1, l]$ and for each $k \in [1, \hat{l}]$, let $\hat{n}_k \in [1, n_k]$. For $k \in [1, \hat{l}]$ and $j \in [1, \hat{n}_k]$, let $w_{k,j} \in \mathbb{R}$ and $h_{k,j} \in \mathcal{H}_k^{(p)}$. Finally, let $r_\infty = \max_{j \in [1, n_1], i \in [1, m]} |[\mathbf{\Phi}(x_i)]_j|$, and $\frac{1}{p} + \frac{1}{q} = 1$. Then for any $\delta > 0$, with probability at least $1 - \delta$ over the sample S of size m drawn i.i.d. according to distribution D^m, the following inequality holds for any $f = \sum_{k=1}^{\hat{l}} \sum_{j=1}^{\hat{n}_k} w_{k,j} h_{k,j}$ with $\sum_{k=1}^{\hat{l}} \sum_{j=1}^{\hat{n}_k} w_{k,j} = 1$:*

$$R(f) \leq \hat{R}_{S,\rho} + \frac{4}{\rho} \sum_{k=1}^{\hat{l}} \sum_{j=1}^{\hat{n}_k} w_{k,j} 4^{k-1} r_\infty \left(\prod_{i=1}^{k} \Lambda_i n_{i-1}^{\frac{1}{q}} \right) \sqrt{\frac{2 \log(2 n_0)}{m}} + C(\rho, l, m, \delta),$$

where $C(\rho, l, m, \delta) = \frac{2}{\rho} \sqrt{\frac{\log(l)}{m}} + \sqrt{\lceil \frac{4}{\rho^2} \log(\frac{\rho^2 m}{\log(l)}) \rceil \frac{\log(l)}{m} + \frac{\log(\frac{2}{\delta})}{2m}}$.

All proofs, derivations and pseudocode of the results in this section as well as the ones in the sequel are deferred to the appendix. The generalization bound above shows that the complexity of the neural network returned is a weighted combination of the complexities of each node in the neural network, where the weights are precisely the ones that define our network. This agrees with the intuition that deeper networks are more complex and suggests that if we can find a model that has both small empirical error and most of its weight on the shallower nodes, then such a model will generalize well.

In the next section, we present an algorithm that directly seeks to minimize upper bounds of this generalization bound. In the process, our algorithm will train neural networks that discriminate deeper networks from shallower ones efficiently learning the architecture of the network. This is a novel property that existing regularization techniques in the deep learning toolbox do not enforce. Techniques such as l_2 and l_1 regularization and dropout (see e.g. Goodfellow et al. [2016]) are generally applied uniformly across all nodes in the network.

3 Algorithm

This section describes our algorithm, ADANET, for *adaptive* deep learning. ADANET adaptively grows the structure of a neural network, balancing model complexity with margin maximization.

Let $x \mapsto \Phi(-x)$ be a non-increasing convex function upper bounding the $0/1$ loss, $x \mapsto 1_{x \leq 0}$, with Φ differentiable over $\mathbb{R}$ and $\Phi'(x) \neq 0$ for all x. Φ may, for instance, be the exponential function, $\Phi(x) = e^x$ as in the AdaBoost of Freund and Schapire [1997] or the logistic function, $\Phi(x) = \log(1 + e^x)$ as in logistic regression. Our algorithm will apply coordinate descent to the following objective function over $\mathbb{R}^N$:

$$F(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} \Phi\Big(1 - y_i \sum_{j=1}^{N} w_j h_j(x_i)\Big) + \sum_{j=1}^{N} \Gamma_j |w_j|, \tag{3}$$

where $\Gamma_j = \lambda r_j + \beta$ with $\lambda \geq 0$ and $\beta \geq 0$ hyperparameters. The objective function is the sum of the empirical error based on a convex surrogate loss function $x \mapsto \Phi(-x)$ of the binary loss and a regularization term. The regularization term is a weighted-l_1 penalty that contains two sub-terms: a standard norm-1 regularization which admits β as a parameter, and a term that discriminates functions h_j based on their complexity (i.e. r_j) and which admits λ as a parameter. Note that by solving this optimization problem we are directly optimizing the learning guarantee in the previous section.

Our algorithm starts with the network reduced to the input layer, corresponding to the input feature vector, and an output unit, fully connected, and then augments or modifies the network over T rounds. At each round, it either augments the network with a new node or updates the weights defining the function $h \in \mathcal{H}_k^{(p)}$ of an existing node of the network at layer k. A new node may be selected at any layer $k \in [1, l]$ already populated or start on a new layer but, in all cases, it is chosen with links only to existing nodes in the network in the layer below plus a connection to the output unit. Existing nodes are either those of the input layer or nodes previously added to the network by the algorithm.

The choice of the node to construct or update at each round is a key aspect of our algorithm. This is done by iteratively optimizing an objective function that we describe in detail later. At each round, the choice of the best node minimizing the current objective is subject to the following trade-off: the best node selected from a lower layer may not help reduce the objective as much as one selected from a higher layer; on the other hand, nodes selected from higher layers augment the network with substantially more complex functions, thereby increasing the risk of overfitting. To resolve this tension quantitatively, our algorithm selects the best node at each round based on a combination of the amount by which it helps reduce the objective and the complexity of the family of hypotheses defined nodes at that layer. There are many potential methods to construct new candidate nodes, and at first glance, scoring every possible new node with connections to existing nodes may seem a computational impediment. However, by using Banach space duality, we can compute directly and efficiently in closed form the new node that best optimizes the objective at each layer.

At each round t, our algorithm maintains a distribution over examples $\mathcal{D}_t$, that can often be efficiently computed from $\mathcal{D}_{t-1}$. For instance, in case of exponential loss $\mathcal{D}_t(i) = \mathcal{D}(i)_{t-1} \exp(-y_i h_t(x_i)$, where h_t is the node updated or added on the previous iteration. Given a distribution $\mathcal{D}$ over the sample S and a tuple of hypotheses $\mathbf{h}_k = (h_{k,1}, \ldots, h_{k,n_k}) \subset \mathcal{H}_k^{(p)}$, we denote by $\mathrm{Margin}(\mathcal{D}, h_{k,j})$ the weighted margin of hypothesis $h_{k,j}$ composed with its activation on distribution $\mathcal{D}$:

$$\mathrm{Margin}(\mathcal{D}, h_{k,j}) = \mathbb{E}_{i \sim \mathcal{D}}[y_i(\varphi_k \circ h_{k,j})(x_i)],$$

and we denote by $\mathrm{Margin}(\mathcal{D}, \mathbf{h}_k)$ the vector of weighted margins of all nodes in layer k:

$$\mathrm{Margin}(\mathcal{D}, \mathbf{h}_k) = \big(\mathbb{E}_{i \sim D}[y_i(\varphi_k \circ h_{k,1})(x_i)], \ldots, \mathbb{E}_{i \sim \mathcal{D}}[y_i(\varphi_k \circ h_{k,n_k})(x_i)]\big).$$

For any layer k in an existing neural network, a vector $u \in \mathbb{R}^{n_k}$ uniquely specifies a node that connects from nodes in the previous layer $k-1$. Let $\tilde{u}_k$ denote such a new node in layer k, and let $\hat{l}$ be the number of layers with non-zero nodes. Then for layers $2 \leq k \leq \hat{l}$, if the number of nodes is less than the maximum allowed size n_k, we will consider as candidates the nodes with the largest weighted margin. Remarkably, these nodes can be computed efficiently and in closed form:

Table 1: Accuracy results for different datasets.

ionosphere	Baseline	AdaNet	diabetes	Baseline	AdaNet	phishing	Baseline	AdaNet
Error	0.877	0.900	Error	0.766	0.769	Error	0.928	0.957
(std dev)	0.062	0.054	(std dev)	0.037	0.047	(std dev)	0.021	0.014
retinopathy	**Baseline**	**AdaNet**	**german**	**Baseline**	**AdaNet**	**cifar**	**Baseline**	**AdaNet**
Error	0.720	0.760	Error	0.761	0.770	Error	0.891	0.916
(std dev)	0.042	0.055	(std dev)	0.040	0.036	(std dev)	0.007	0.007

Lemma 2 (Construction of new candidate nodes)**.** *Fix* $(h_{k-1,j})_{j=1}^{n_{k-1}} \subset \mathcal{H}_{k-1}^{(p)}$. *Then the solution* $\tilde{u}_k$ *to the optimization problem*

$$\max_{\|u\|_p \leq \Lambda_k} \mathbb{E}_{i \sim \mathcal{D}} \left[y_i \sum_{j=1}^{n_{k-1}} u_j (\varphi_{k-1} \circ h_{k-1,j})(x_i) \right],$$

can be computed coordinate-wise as:

$$(\tilde{u}_k)_j = \frac{\Lambda_k}{\|\mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q^{\frac{q}{p}}} |\mathrm{Margin}(\mathcal{D}, h_{k-1,j})|^{q-1} \operatorname{sgn}\left(\mathrm{Margin}(\mathcal{D}, h_{k-1,j})\right),$$

and the solution has value: $\Lambda_k \|\mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q$.

Thus, at each iteration of our algorithm either one of those candidates is introduced to the network or the weight $w_{k,j}$ of an existing node $h_{k,j}$ is updated. That option depends on which one provides the maximal reduction in the value of the objective function. It can be shown that this corresponds to a coordinate descent algorithm, which combined with the fact that our objective is convex leads to convergence guarantees for our algorithm. This should be contrasted with standard methods for training neural networks that typically solve non-convex problems and hence do not enjoy the same favorable convergence guarantees.

Our algorithm can be viewed as an instance of the DeepBoost algorithm Cortes et al. [2014]. However, unlike DeepBoost, which combines decision trees, ADANET algorithm learns a deep neural network, which requires both deep learning-specific theoretical analysis as well as an online method for constructing and searching new nodes. Both of these aspects differ significantly from the decision tree framework in DeepBoost, and the latter is particularly challenging due to the fact that our hypothesis space $\mathcal{H}$ is infinite.

4 Experimental Results

In this section, we present some preliminary experimental results comparing Adanet with grid search over the neural network architectures trained using standard back-propagation on a number of datasets: `ionosphere`, `diabetes`, `phishing`, `german`, `retinopathy`, `german` and `cifar-10` (2 classes sub-selected). The experiments with AdaNet described below use a single network limiting the maximum number of layers and nodes to 3 and 2000 respectively.

We used standard 10-fold cross-validation for performance evaluation and model selection. In particular, each dataset was randomly partitioned into 10 folds, and each algorithm was run 10 times, with a different assignment of folds to the training set, validation set and test set for each run. Specifically, for each $i \in \{0, \ldots, 10\}$, fold i was used for testing, fold $i + 1$ (mod 10) was used for validation, and the remaining folds were used for training. For each setting of the parameters, we computed the average validation error across the 10 folds, and selected the parameter setting with minimum average validation error. The average test error across the 10 folds was then computed for this particular parameter setting.

In the first set of experiments reported in Table 1, we compared AdaNet and a standard feedforward network. AdaNet was configured with the following hyperparameter ranges: a maximum of 10 layers, no more than 2000 nodes per layer, a constant $\Lambda = [1.0, 1.045]$ for all layers, $\lambda = [10^{-8}, 10^{-3}]$, $\beta = [10^{-8}, 10^{-3}]$, dual norm fixed to 2 and a ReLu activation.

5 Conclusion

We presented a new framework for analyzing and learning artificial neural networks. Our method optimizes for generalization performance, and it explicitly and automatically addresses the trade-off between model architecture and empirical risk minimization, ideas that have been under-explored in deep learning. We presented preliminary experimental results showing that Adanet can efficiently learn neural network architectures that achieve results that are comparable with baseline methods.

Our techniques are general and can be applied to other neural network architectures, including CNNs and LSTMs as well as to other learning settings such as multi-class classification and regression. Furthermore, our general framework allows to the use of other more general weak learners as well as different optimization techniques for learning network architectures. All of these serve as interesting avenues for future work.

References

J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, pages 2546–2554, 2011.

C. Cortes, M. Mohri, and U. Syed. Deep boosting. In *ICML*, pages 1179 – 1187, 2014.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer System Sciences*, 55(1):119–139, 1997.

I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016.

M. Islam, A. Sattar, F. Amin, X. Yao, and K. Murase. A new adaptive merging and growing algorithm for designing artificial neural networks. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(3):705–722, 2009.

M. M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *Neural Networks, IEEE Transactions on*, 14(4):820–834, 2003.

T.-Y. Kwok and D.-Y. Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *Neural Networks, IEEE Transactions on*, 8(3):630–645, 1997.

M. Lehtokangas. Modelling with constructive backpropagation. *Neural Networks*, 12(4):707–716, 1999.

F. H. Leung, H.-K. Lam, S.-H. Ling, and P. K. Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *Neural Networks, IEEE Transactions on*, 14(1):79–88, 2003.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

G. Rätsch, S. Mika, and M. K. Warmuth. On the convergence of leveraging. In *NIPS*, pages 487–494, 2001.

V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

A Preliminaries

Let $\mathcal{X}$ denote the input space. We consider the standard supervised binary classification scenario and assume that training and test points are drawn i.i.d. according to some distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1, +1\}$ and denote by $S = ((x_1, y_1), \ldots, (x_m, y_m))$ a training sample of size m drawn according to $\mathcal{D}^m$. Given any $x \in \mathcal{X}$, we denote by $\mathbf{\Phi}(x) \in \mathbb{R}^{n_0}$ the feature representation of x.

The standard description of a modern feedforward network is a network of layers of nodes, where each layer is mapped to the layer above it via a linear mapping composed with a component-wise nonlinear transformation. To make this precise, we define a neural network as follows.

Let l denote the number of layers in the network. The networks we learn can be potentially very deep, that is l can be very large. For each $k \in [l]$, denote by n_k the maximum number of nodes in layer k.

Let $1 \leq p \leq \infty$ and $k \geq 1$. Then define the set $\mathcal{H}_k^{(p)}$ to be the family of functions at layer k of the network in the following way:

$$\mathcal{H}_1^{(p)} = \left\{ x \mapsto \mathbf{u} \cdot \mathbf{\Phi}(x) \colon \mathbf{u} \in \mathbb{R}^{n_0}, \|\mathbf{u}\|_p \leq \Lambda_1 \right\} \tag{4}$$

$$\mathcal{H}_k^{(p)} = \left\{ x \mapsto \left(\sum_{j=1}^{n_{k-1}} u_j (\varphi_{k-1} \circ h_j)(x) \right) \colon \mathbf{u} \in \mathbb{R}^{n_{k-1}}, \|\mathbf{u}\|_p \leq \Lambda_k, h_j \in \mathcal{H}_{k-1}^{(p)} \right\}, \ (\forall k > 1) \tag{5}$$

where $\Lambda_k > 0$ is a hyperparameter and where φ_k is an activation function. Common activation functions include the Rectified Linear Unit (ReLU) $\varphi_k(x) = \max\{0, x\}$ and the sigmoid function $\varphi_k(x) = \frac{1}{1+e^{-x}}$ (see e.g. [Goodfellow et al., 2016]), although our work will allow for any 1-Lipschitz activation function. The choice of norm p here is left to the learner and will determine both the sparsity of the network and the accompanying learning guarantee of the resulting model.

Let $\mathcal{H}$ denote the union of these families of functions and their reflections: $\mathcal{H} = \bigcup_{k=1}^{l} (\mathcal{H}_k^{(p)} \cup (-\mathcal{H}_k^{(p)}))$. Any feedforward neural network, then, can be written as a composition of mappings $f = f_l \circ f_{l-1} \circ \ldots \circ f_1$, where $f_k \in \mathcal{H}_k^{(p)}$. Intuitively, each transformation represents the encoding of the original data into an abstract layer of feature representation from which learnability is presumed to be "easier."

Note that in our definition, the activation function is not directly included in the unit itself but instead only applied when feeding the neuron into the next layer. While this choice of notation ultimately represents the same family of models, it is a subtle but important distinction that will be crucial for our theory as well as algorithmic design, in particular for deriving Lemma 3 and Lemma 6.

B Theoretical properties of artificial neural networks

We measure the performance of a hypothesis $f \in \mathcal{H}$ by its expected loss over the data's distribution $\mathcal{D}$, also known as the generalization error: $R(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[1_{yf(x)\leq 0}]$.

We typically also want to measure the performance of the model on our training sample S. This will be done using the empirical margin loss: $\hat{R}_{S,\rho}(f) = \frac{1}{m} \sum_{i=1}^{m} 1_{y_i f(x_i) \leq \rho}$, where the margin refers to the ρ term. Hypothesis functions that allow for large margin with small empirical margin loss intuitively represent classifiers with high confidence of accuracy.

Given a hypothesis set $\mathcal{H}$ of functions mapping from $\mathcal{X}$ to $\mathbb{R}$, we denote by $\widehat{\mathfrak{R}}_S(\mathcal{H})$ the empirical Rademacher complexity of $\mathcal{H}$ for the sample S: $\widehat{\mathfrak{R}}_S(\mathcal{H}) = \frac{1}{m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{h \in H} \sum_{i=1}^{m} \sigma_i h(x_i) \right]$, and by $\mathfrak{R}_m(\mathcal{H})$ the Rademacher complexity of $\mathcal{H}$ defined by $\mathfrak{R}_m(\mathcal{H}) = \mathbb{E}_{S\sim\mathcal{D}^m}[\widehat{\mathfrak{R}}_S(\mathcal{H})]$.

The empirical Rademacher complexity measures the correlation of a hypothesis set with random noise over the sample and is a problem-dependent measure of model complexity. It can also be shown to relate to other classical notions of model complexity, such as the VC-dimension and covering number (see e.g. [Vapnik, 1998, Mohri et al., 2012]).

B.1 Learning bounds on the Rademacher complexity of network hypothesis sets

Since neural networks are built as compositions of layers, it is natural from the theoretical standpoint to first analyze the complexity of any layer in terms of the complexity of its previous layer. Our first result demonstrates that this can indeed be done, and that the empirical Rademacher complexity of any intermediate layer k in the network is bounded by the empirical Rademacher complexity of its input times a term that depends on a power of the size of the layer:

Lemma 3. *Let $\frac{1}{p} + \frac{1}{q} = 1$. Then for $k \geq 2$, the empirical Rademacher complexity of $\mathcal{H}_k^{(p)}$ for a sample S of size m can be upper bounded as follows in terms of that of $\mathcal{H}_{k-1}^{(p)}$:*

$$\widehat{\mathfrak{R}}_S(\mathcal{H}_k^{(p)}) \leq 2\Lambda_k n_{k-1}^{\frac{1}{q}} \widehat{\mathfrak{R}}_S(\mathcal{H}_{k-1}^{(p)}).$$

The proofs of this result, as well as all those that follow in this section, can be found in Appendix D.

By analyzing the complexity of the initial layer, we can derive a bound on the complexity of every layer in closed form without the need for any recurrence relation:

Lemma 4. *Let $r_\infty = \max_{j \in [1,n_1], i \in [1,m]} |[\mathbf{\Phi}(x_i)]_j|$, and $\frac{1}{p} + \frac{1}{q} = 1$. Then for any $k \geq 1$, the empirical Rademacher complexity of $\mathcal{H}_k^{(p)}$ for a sample S of size m can be upper bounded as follows:*

$$\widehat{\mathfrak{R}}_S(\mathcal{H}_k^{(p)}) \leq 2^{k-1} r_\infty \left(\prod_{j=1}^{k} \Lambda_j n_{j-1}^{\frac{1}{q}} \right) \sqrt{\frac{2\log(2n_0)}{m}}.$$

The above two lemmas are instructive and intuitive in the sense that they convey the message that additional layers in a neural network contribute to increased complexity of a model. Because of this, while large models are more powerful, they also become increasingly more prone to overfitting. Moreover, the Rademacher complexity bounds also suggest that model complexity can increase much more due to a single additional layer as opposed to an additional node.

Guided by this insight, we will seek to learn models that will be parsimonious in model complexity. Specifically, we will learn adaptive neural networks that consider *all* layers of feature representation simultaneously, emphasize shallower layers of representation more heavily, and only activate deeper ones when necessary. We will represent such models using the notation: $f = \sum_{k=1}^{l} \sum_{j=1}^{n_k} w_{k,j} h_{k,j}$, where $w_{k,j}$'s are weights and $h_{k,j} \in \mathcal{H}_k^{(p)}$. The motivation for this type of network is reinforced by the following learning guarantee:

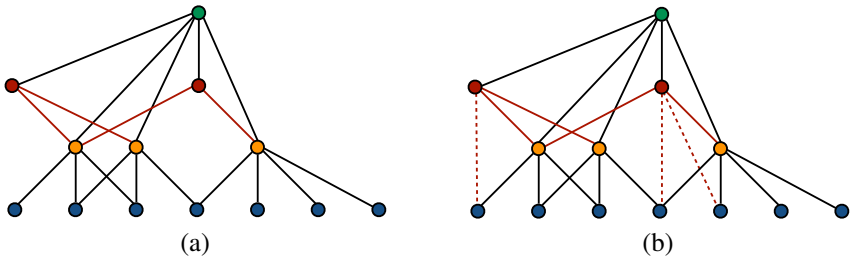
Theorem 5 (AdaNet Generalization Bound)**.** *Let $\hat{l} \in [1, l]$ and for each $k \in [1, \hat{l}]$, let $\hat{n}_k \in [1, n_k]$. For $k \in [1, \hat{l}]$ and $j \in [1, \hat{n}_k]$, let $w_{k,j} \in \mathbb{R}$ and $h_{k,j} \in \mathcal{H}_k^{(p)}$. Finally, let $r_\infty = \max_{j \in [1,n_1], i \in [1,m]} |[\mathbf{\Phi}(x_i)]_j|$, and $\frac{1}{p} + \frac{1}{q} = 1$. Then for any $\delta > 0$, with probability at least $1 - \delta$ over the sample S of size m drawn i.i.d. according to distribution D^m, the following inequality holds for any $f = \sum_{k=1}^{\hat{l}} \sum_{j=1}^{\hat{n}_k} w_{k,j} h_{k,j}$ with $\sum_{k=1}^{\hat{l}} \sum_{j=1}^{\hat{n}_k} w_{k,j} = 1$:*

$$R(f) \leq \hat{R}_{S,\rho} + \frac{4}{\rho} \sum_{k=1}^{\hat{l}} \sum_{j=1}^{\hat{n}_k} w_{k,j} 4^{k-1} r_\infty \left(\prod_{i=1}^{k} \Lambda_i n_{i-1}^{\frac{1}{q}} \right) \sqrt{\frac{2\log(2n_0)}{m}} + C(\rho, l, m, \delta),$$

where $C(\rho, l, m, \delta) = \frac{2}{\rho} \sqrt{\frac{\log(l)}{m}} + \sqrt{\lceil \frac{4}{\rho^2} \log(\frac{\rho^2 m}{\log(l)}) \rceil \frac{\log(l)}{m} + \frac{\log(\frac{2}{\delta})}{2m}}$.

The generalization bound above informs us that the complexity of the neural network returned is a weighted combination of the complexities of each node in the neural network, where the weights are precisely the ones that define our network. Specifically, this again agrees with our intuition that deeper networks are more complex and suggests that if we can find a model that has both small empirical error and most of its weight on the shallower nodes, then such a model will generalize well.

Toward this goal, we will design an algorithm that directly seeks to minimize upper bounds of this generalization bound. In the process, our algorithm will train neural networks that discriminate deeper networks from shallower ones. This is a novel property that existing regularization techniques in the deep learning toolbox do not enforce. Techniques such as l_2 and l_1 regularization and dropout (see e.g. Goodfellow et al. [2016]) are generally applied uniformly across all nodes in the network.



(a) (b)

Figure 1: Illustration of the algorithm's incremental construction of a neural network: (a) standard construction where nodes at each layer other than the output layer can only be connected to the layer below; (b) more complex construction where nodes can be connected to other layers as well.

C Algorithm

This section describes our algorithm, ADANET, for *adaptive* deep learning. ADANET adaptively grows the structure of a neural network, balancing model complexity with margin maximization. We start with a high-level description of the algorithm before proceeding to a more detailed presentation.

C.1 Overview

Our algorithm starts with the network reduced to the input layer, corresponding to the input feature vector, and an output unit, fully connected, and then augments or modifies the network over T rounds. At each round, it either augments the network with a new node or updates the weights defining the function $h \in \mathcal{H}_k^{(p)}$ of an existing node of the network at layer k. A new node may be selected at any layer $k \in [1, l]$ already populated or start on a new layer but, in all cases, it is chosen with links only to existing nodes in the network in the layer below plus a connection to the output unit. Existing nodes are either those of the input layer or nodes previously added to the network by the algorithm. Figure 1(a) illustrates this design.

The choice of the node to construct or update at each round is a key aspect of our algorithm. This is done by iteratively optimizing an objective function that we describe in detail later. At each round, the choice of the best node minimizing the current objective is subject to the following trade-off: the best node selected from a lower layer may not help reduce the objective as much as one selected from a higher layer; on the other hand, nodes selected from higher layers augment the network with substantially more complex functions, thereby increasing the risk of overfitting. To resolve this tension quantitatively, our algorithm selects the best node at each round based on a combination of the amount by which it helps reduce the objective and the complexity of the family of hypotheses defined nodes at that layer.

The output node of our network is connected to all the nodes created during these T rounds, so that the hypothesis will directly use all nodes in the network. As our theory demonstrated in Theorem 5, this can significantly reduce the complexity of our model by assigning more weight to the shallower nodes. At the same time, it also provides us the flexibility to learn larger and more complex models. In fact, the family of neural networks that we search over is actually larger than the family of feedforward neural networks typically considered using back-propagation due to these additional connections.

An additional more sophisticated variant of our algorithm is depicted in Figure 1(b). In this design, the nodes created at each round can be connected not just to the nodes of the previous layer, but to those of any layer below. This allows for greater model flexibility, and by modifying the definitions of the hypotheses sets 4 and 5, we can adopt a principled complexity-sensitive way for learning these types of structures as well.

In the next section, we give a more formal description of our algorithm, including the exact optimization problem as well as a specific search process for new nodes.

C.2 Objective function

Recall the definition of our hypothesis space $\mathrm{conv}(\mathcal{H}) = \bigcup_{k=1}^{l} (\mathcal{H}_k^{(p)} \cup (-\mathcal{H}_k^{(p)}))$, which is the convex hull of all neural networks up to depth l and naturally includes all neural networks of depth l

– the common hypothesis space in deep learning. Note that the set of all functions in $\mathcal{H}$ is infinite, since the weights corresponding to any function can be any real value inside their respective l_p balls.

Despite this challenge, we will efficiently discover a finite subset of $\mathcal{H}$, denoted by $\{h_1, \ldots, h_N\}$, that will serve as the basis for our convex combination. Here, N will also represent the maximum number of nodes in our network. Thus, we have that $N = \sum_{k=1}^{l} n_k$, and N will also generally be assumed as very large. Moreover, we will actually define and update our set $\{h_1, \ldots, h_N\}$ *online*, in a manner that will be made precise in Section C.4. We will also rely on the natural bijection between the two enumerations $\{h_1, \ldots, h_N\}$ and $\{h_{k,j}\}_{k \in [l], j \in [n_k]}$, depending on which is more convenient. The latter is useful for saying that $h_{k,j} \in \mathcal{H}_k^{(p)}$.

Moreover, for any $j \in [N]$, we will denote by $k_j \in [l]$, the layer in which hypothesis h_j lies. For simplicity, we will also write as r_j the Rademacher complexity of the family of functions $\mathcal{H}_{k_j}^{(p)}$ containing h_j: $r_j = \mathfrak{R}_m(\mathcal{H}_{k_j}^{(p)})$.

Let $x \mapsto \Phi(-x)$ be a non-increasing convex function upper bounding the $0/1$ loss, $x \mapsto 1_{x \leq 0}$, with Φ differentiable over $\mathbb{R}$ and $\Phi'(x) \neq 0$ for all x. Φ may, for instance, be the exponential function, $\Phi(x) = e^x$ as in the AdaBoost of Freund and Schapire [1997] or the logistic function, $\Phi(x) = \log(1 + e^x)$ as in logistic regression.

As in regularized boosting style methods (e.g. [Rätsch et al., 2001]), our algorithm will apply coordinate descent to the following objective function over $\mathbb{R}^N$:

$$F(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} \Phi\Big(1 - y_i \sum_{j=1}^{N} w_j h_j(x_i)\Big) + \sum_{j=1}^{N} \Gamma_j |w_j|, \tag{6}$$

where $\Gamma_j = \lambda r_j + \beta$ with $\lambda \geq 0$ and $\beta \geq 0$ hyperparameters. The objective function is the sum of the empirical error based on a convex surrogate loss function $x \mapsto \Phi(-x)$ of the binary loss and a regularization term. The regularization term is a weighted-l_1 penalty that contains two sub-terms: a standard norm-1 regularization which admits β as a parameter, and a term that discriminates functions h_j based on their complexity (i.e. r_j) and which admits λ as a parameter.

Our algorithm can be viewed as an instance of the DeepBoost algorithm of Cortes et al. [2014]. However, unlike DeepBoost, which combines decision trees, AdaNet algorithm learns a deep neural network, which requires both deep learning-specific theoretical analysis as well as an online method for constructing and searching new nodes. Both of these aspects differ significantly from the decision tree framework in DeepBoost, and the latter is particularly challenging due to the fact that our hypothesis space $\mathcal{H}$ is infinite.

C.3 Coordinate descent

Let $\mathbf{w}_t = (w_{t,1}, \ldots, w_{t,N})^\top$ denote the vector obtained after $t \geq 1$ iterations and let $\mathbf{w}_0 = \mathbf{0}$. Let e_k denote the kth unit vector in $\mathbb{R}^N$, $k \in [1, N]$. The direction e_k and the step η selected at the tth round are those minimizing $F(\mathbf{w}_{t-1} + \eta e_k)$. Let $f_{t-1} = \sum_{j=1}^{N} w_{t-1,j} h_j$. Then we can write

$$F(\mathbf{w}_{t-1} + \eta e_k) = \frac{1}{m} \sum_{i=1}^{m} \Phi\Big(1 - y_i f_{t-1}(x_i) - \eta y_i h_k(x_i)\Big) + \sum_{j \neq k} \Gamma_j |w_{t-1,j}| + \Gamma_k |w_{t-1,k} + \eta|,$$

For any $t \in [1, T]$, we will maintain the following distribution $\mathcal{D}_t$ over our sample: $\mathcal{D}_t(i) = \frac{\Phi'\left(1 - y_i f_{t-1}(x_i)\right)}{S_t}$, where S_t is a normalization factor, $S_t = \sum_{i=1}^{m} \Phi'(1 - y_i f_{t-1}(x_i))$. Moreover, for any $s \in [1, T]$ and $j \in [1, N]$ and a given hypothesis h_j bounded by $C > 0$, we will consider $\epsilon_{s,j}$, the weighted error of hypothesis h_j over the distribution $\mathcal{D}_s$: $\epsilon_{s,j} = \frac{C}{2}\left[1 - \mathbb{E}_{i \sim \mathcal{D}_s}\left[\frac{y_i h_j(x_i)}{C}\right]\right]$. These weighted errors will be crucial for "scoring" the direction that the algorithm takes at each round.

C.4 Search and active coordinates

As already mentioned, a key aspect of our AdaNet algorithm is the construction of new hypotheses at each round. We do not enumerate all N hypotheses at the beginning of the algorithm, because it would

ADANET$(S = ((x_i, y_i)_{i=1}^m, (n_k, \Lambda_k, \Gamma_k, C_k)_{k=1}^l)$

$(w_{0,k,j})_{k\in[1,l],j\in[1,n_k]}, \mathcal{D}_1, \hat{l}, (\hat{n}_k)_{k\in[1,l],j\in[1,\hat{n}_k]}) \leftarrow$ INIT$\big(m, l, (n_k)_{k=1}^l\big)$
for $t \leftarrow 1$ **to** T **do**
  $(d_{k,j})_{k\in[1,\hat{l}],j\in[1,\hat{n}_k]} \leftarrow$ EXISTINGNODES$\big(\mathcal{D}_t, (\mathbf{h}_k)_{k=1}^{\hat{l}}, (\hat{n}_k, C_k, \Gamma_k)_{k=1}^l\big)$
  $(\tilde{d}_k, \tilde{u}_k)_{k=2}^{(\hat{l}+1)\wedge l} \leftarrow$ NEWNODES$\big(\mathcal{D}_t, (\mathbf{h}_k)_{k=1}^{\hat{l}}, (n_k, \hat{n}_k, C_k, \Lambda_k, \Gamma_k)_{k=1}^l\big)$
  $((k,j), \epsilon_t, \hat{l}, (\hat{n}_k)_{k=1}^l) \leftarrow$ BESTNODE$\big((d_{k,j})_{k\in[1,\hat{l}],j\in[1,\hat{n}_k]}, (\tilde{d}_k)_{k\in[2,\hat{l}+1\wedge l]}\big)$
  $\mathbf{w}_t \leftarrow$ APPLYSTEP$\big((k,j), w_{t-1}\big)$
  $(\mathcal{D}_{t+1}, S_{t+1}) \leftarrow$ UPDATEDISTRIBUTION$\big(\mathbf{w}_t, S, (h_{k,j})_{k\in[1,\hat{l}],j\in[1,\hat{n}_k]}\big)$
$f \leftarrow \sum_{k=1}^{\hat{l}} \sum_{j=1}^{\hat{n}_k} w_{T,k,j} h_{k,j}$
return f

Figure 2: Pseudocode of the AdaNet algorithm.

EXISTINGNODES$(\mathcal{D}_t, (h_{k,j})_{k\in[1,\hat{l}],j\in[1,\hat{n}_k]}, (\hat{n}_k, C_k)_{k=1}^l)$

for $k \leftarrow 1$ **to** $\hat{l}$ **do**
  for $j \leftarrow 1$ **to** $\hat{n}_k$ **do**
    $\epsilon_{t,k,j} \leftarrow \frac{C_k}{2}\Big[1 - \mathbb{E}_{i\sim\mathcal{D}_t}\big[\frac{y_i h_{k,j}(x_i)}{C_k}\big]\Big]$
    if $(w_{t-1,k,j} \neq 0)$ **then**
      $d_{k,j} \leftarrow \big(\epsilon_{t,k,j} - \frac{1}{2}C_k\big) + \operatorname{sgn}(w_{t-1,k,j})\frac{\Gamma_k m}{2S_t}$
    elseif $\big(\big|\epsilon_{t,k,j} - \frac{1}{2}C_k\big| \leq \frac{\Gamma_k m}{2S_t}\big)$ **then**
      $d_{k,j} \leftarrow 0$
    else $d_{k,j} \leftarrow \big(\epsilon_{t,k,j} - \frac{1}{2}C_k\big) - \operatorname{sgn}(\epsilon_{t,k,j} - \frac{1}{2}C_k)\frac{\Gamma_k m}{2S_t}$
return $(d_{k,j})_{k\in[1,\hat{l}],j\in[1,\hat{n}_k]}$

Figure 3: Pseudocode for computing the potential contribution of each existing node.

be extremely difficult to select good candidates before seeing the data. At the same time, searching through all node possible combinations using the data would be a computationally infeasible task.

Instead, our search procedure will be online, building upon the nodes that we already have in our network and selecting at most a single at a time. Specifically, at each round, the algorithm selects a node out of the following set of "active" candidates: existing nodes in our network or new nodes with connections to existing nodes in some layer k.

There are many potential methods to construct new candidate nodes, and at first glance, scoring every possible new node with connections to existing nodes may seem a computational impediment. However, by using Banach space duality, we can compute directly and efficiently in closed form the new node that best optimizes the objective at each layer.

C.4.1 New candidate nodes

Given a distribution $\mathcal{D}$ over the sample S and a tuple of hypotheses $\mathbf{h}_k = (h_{k,1}, \ldots, h_{k,n_k}) \subset \mathcal{H}_k^{(p)}$, we denote by $\operatorname{Margin}(\mathcal{D}, h_{k,j})$ the weighted margin of hypothesis $h_{k,j}$ composed with its activation on distribution $\mathcal{D}$:

$$\operatorname{Margin}(\mathcal{D}, h_{k,j}) = \mathbb{E}_{i\sim\mathcal{D}}[y_i(\varphi_k \circ h_{k,j})(x_i)],$$

and we denote by $\operatorname{Margin}(\mathcal{D}, \mathbf{h}_k)$ the vector of weighted margins of all nodes in layer k:

$$\operatorname{Margin}(\mathcal{D}, \mathbf{h}_k) = \big(\mathbb{E}_{i\sim D}[y_i(\varphi_k \circ h_{k,1})(x_i)], \ldots, \mathbb{E}_{i\sim\mathcal{D}}[y_i(\varphi_k \circ h_{k,n_k})(x_i)]\big).$$

For any layer k in an existing neural network, a vector $u \in \mathbb{R}^{n_k}$ uniquely specifies a node that connects from nodes in the previous layer $k-1$. Let $\tilde{u}_k$ denote such a new node in layer k, and let $\hat{l}$ be the number of layers with non-zero nodes. Then for layers $2 \leq k \leq \hat{l}$, if the number of nodes is

NEWNODES$(\mathcal{D}_t, (\mathbf{h}_k)_{k=1}^{\hat{l}}, (n_k, \hat{n}_k, C_k, \Lambda_k, \Gamma_k))_{k=1}^{l}$

for $k \leftarrow 2$ **to** $(\hat{l}+1) \wedge l$ **do**
  if $\hat{n}_k < n_k$ and $\hat{n}_{k-1} > 0$ **then**
    $\tilde{\epsilon}_k \leftarrow \frac{C_k}{2}\left(1 - \frac{\Lambda_k}{C_k}\left\|\mathcal{E}(\mathcal{D}_t, \mathbf{h}_{k-1})\right\|_q\right)$
    $\tilde{u}_k \leftarrow \frac{\Lambda_k |\mathcal{E}(\mathcal{D}_t, h_{k-1,j})|^{q-1} \operatorname{sgn}(\mathcal{E}(\mathcal{D}_t, h_{k-1,j}))}{\|\mathcal{E}(\mathcal{D}_t, \mathbf{h}_{k-1})\|_q^{\frac{q}{p}}}$
  else $\tilde{\epsilon}_k \leftarrow \frac{1}{2}C_k$
  if $\left(\left|\tilde{\epsilon}_k - \frac{1}{2}C_k\right| \leq \frac{\Gamma_k m}{2S_t}\right)$ **then**
    $\tilde{d}_k \leftarrow 0$
  else $\tilde{d}_k \leftarrow \left(\tilde{\epsilon}_k - \frac{1}{2}C_k\right) - \operatorname{sgn}(\tilde{\epsilon}_k - \frac{1}{2}C_k)\frac{\Gamma_k m}{2S_t}$
return $(\tilde{d}_k, \tilde{u}_k)_{k=2}^{(\hat{l}+1)\wedge l}$

Figure 4: Pseudocode for the construction of new candidate nodes. Remarkably, we can use l_p duality to derive a closed-form expression of the node with largest weighted error at each layer.

less than the maximum allowed size n_k, we will consider as candidates the nodes with the largest weighted margin. Remarkably, these nodes can be computed efficiently and in closed form:

Lemma 6 (Construction of new candidate nodes, see proof in Appendix E)**.** *Fix* $(h_{k-1,j})_{j=1}^{n_{k-1}} \subset \mathcal{H}_{k-1}^{(p)}$. *Then the solution* $\tilde{u}_k$ *to the optimization problem*

$$\max_{\|u\|_p \leq \Lambda_k} \mathbb{E}_{i \sim \mathcal{D}}\left[y_i \sum_{j=1}^{n_{k-1}} u_j(\varphi_{k-1} \circ h_{k-1,j})(x_i)\right],$$

can be computed coordinate-wise as:

$$(\tilde{u}_k)_j = \frac{\Lambda_k}{\|\operatorname{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q^{\frac{q}{p}}} |\operatorname{Margin}(\mathcal{D}, h_{k-1,j})|^{q-1} \operatorname{sgn}\left(\operatorname{Margin}(\mathcal{D}, h_{k-1,j})\right),$$

and the solution has value: $\Lambda_k \left\|\operatorname{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\right\|_q$.

C.5 Pseudocode

In this section, we present the pseudocode for our algorithm, ADANET, which applies the greedy coordinate-wise optimization procedure described in Section C.3 on the objective presented in Section C.2 with the search procedure described in Section C.4.

The algorithm takes as input the sample S, the maximum number of nodes per layer $(n_k)_{k=1}^{l}$, the complexity penalties $(\Gamma_k)_{k=1}^{l}$, the l_p norms of the weights defining new nodes $(\Lambda_k)_{k=1}^{l}$, and upper bounds on the nodes in each layer $(C_k)_{k=1}^{l}$. ADANET then initializes all weights to zero, sets the distribution to be uniform, and considers the active set of coordinates to be the initial layer in the network. Then the algorithm repeats the following sequence of steps: it computes the scores of the existing nodes in the method EXISTINGNODES, of the new candidate nodes in NEWNODES, and finds the node with the best score ($|d_{k,j}|$ or $|\tilde{d}_k|$) in BESTNODE. After finding this "coordinate", it updates the step size and distribution before proceeding to the next iteration (as described in Section C.3). The precise pseudocode is provided in Figure 2, and details of its derivation are given in Section E.

C.6 Convergence of ADANET

Remarkably, the neural network that ADANET outputs is competitive against the *optimal* weights for any sub-network that it sees during training. Moreover, it achieves this guarantee in linear time. The precise statement and proof are provided in Appendix G.

C.7 Large-scale implementation of AdaNet

We describe a large-scale implementation of the ADANET optimization problem using state-of-the-art techniques from stochastic optimization in Appendix H.

D Proofs of theoretical guarantees

Lemma 3. *Let $\frac{1}{p}+\frac{1}{q}=1$. Then for $k\geq 2$, the empirical Rademacher complexity of $\mathcal{H}_k^{(p)}$ for a sample S of size m can be upper bounded as follows in terms of that of $\mathcal{H}_{k-1}^{(p)}$:*

$$\widehat{\mathfrak{R}}_S(\mathcal{H}_k^{(p)}) \leq 2\Lambda_k n_{k-1}^{\frac{1}{q}} \widehat{\mathfrak{R}}_S(\mathcal{H}_{k-1}^{(p)}).$$

Proof.

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(\mathcal{H}_k^{(p)}) &= \frac{1}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\substack{h_j\in\mathcal{H}_{k-1}^{(p)}\\ \|\mathbf{u}\|_p\leq\Lambda_k}} \sum_{i=1}^m \sigma_i \sum_{j=1}^{n_{k-1}} u_j(\varphi_{k-1}\circ h_j)(x_i)\right] \\
&= \frac{1}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\substack{h_j\in\mathcal{H}_{k-1}^{(p)}\\ \|\mathbf{u}\|_p\leq\Lambda_k}} \sum_{j=1}^{n_{k-1}} u_j \sum_{i=1}^m \sigma_i(\varphi_{k-1}\circ h_j)(x_i)\right] \\
&= \frac{\Lambda_k}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h_j\in\mathcal{H}_{k-1}^{(p)}} \left\|\sum_{i=1}^m \sigma_i(\varphi_{k-1}\circ h_j)(x_i)\right\|_q\right] && \text{(def. of dual norm)} \\
&= \frac{\Lambda_k n_{k-1}^{\frac{1}{q}}}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h_j\in\mathcal{H}_{k-1}^{(p)}} \left\|\sum_{i=1}^m \sigma_i(\varphi_{k-1}\circ h_j)(x_i)\right\|_\infty\right] && \text{(equiv. of } l_p \text{ norms and sup)} \\
&= \frac{\Lambda_k n_{k-1}^{\frac{1}{q}}}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}_{k-1}^{(p)}} \left|\sum_{i=1}^m \sigma_i(\varphi_{k-1}\circ h)(x_i)\right|\right] \\
&= \frac{\Lambda_k n_{k-1}^{\frac{1}{q}}}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\substack{h\in\mathcal{H}_{k-1}^{(p)}\\ s\in\{-1,+1\}}} s\sum_{i=1}^m \sigma_i(\varphi_{k-1}\circ h)(x_i)\right] && \text{(def. of absolute value)} \\
&\leq \frac{\Lambda_k n_{k-1}^{\frac{1}{q}}}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}_{k-1}^{(p)}} \sum_{i=1}^m \sigma_i(\varphi_{k-1}\circ h)(x_i)\right] \\
&\quad + \frac{\Lambda_k}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}_{k-1}^{(p)}} \sum_{i=1}^m -\sigma_i(\varphi_{k-1}\circ h)(x_i)\right] \\
&= \frac{2\Lambda_k n_{k-1}^{\frac{1}{q}}}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}_{k-1}^{(p)}} \sum_{i=1}^m \sigma_i(\varphi_{k-1}\circ h)(x_i)\right] \\
&\leq \frac{2\Lambda_k n_{k-1}^{\frac{1}{q}}}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}_{k-1}^{(p)}} \sum_{i=1}^m \sigma_i h(x_i)\right] && \text{(Talagrand's inequality)} \\
&\leq 2\Lambda_k n_{k-1}^{\frac{1}{q}} \widehat{\mathfrak{R}}_S(\mathcal{H}_{k-1}^{(p)})
\end{aligned}$$

□

Lemma 4. *Let $r_\infty = \max_{j\in[1,n_1],i\in[1,m]} |[\boldsymbol{\Phi}(x_i)]_j|$, and $\frac{1}{p}+\frac{1}{q}=1$. Then for any $k\geq 1$, the empirical Rademacher complexity of $\mathcal{H}_k^{(p)}$ for a sample S of size m can be upper bounded as follows:*

$$\widehat{\mathfrak{R}}_S(\mathcal{H}_k^{(p)}) \leq 2^{k-1} r_\infty \left(\prod_{j=1}^k \Lambda_j n_{j-1}^{\frac{1}{q}}\right)\sqrt{\frac{2\log(2n_0)}{m}}.$$

Proof.

$$
\begin{aligned}
\widehat{\mathfrak{R}}_S(\mathcal{H}_1^{(p)}) &= \frac{1}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\|\mathbf{u}\|_p \leq \Lambda_1} \sum_{i=1}^m \sigma_i \mathbf{u}\cdot\boldsymbol{\Phi}(x_i)\right] \\
&= \frac{1}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\|\mathbf{u}\|_p \leq \Lambda_1} \mathbf{u}\cdot\sum_{i=1}^m \sigma_i \boldsymbol{\Phi}(x_i)\right] \\
&= \frac{\Lambda_1}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\left\|\sum_{i=1}^m \sigma_i[\boldsymbol{\Phi}(x_i)]\right\|_q\right] && \text{(def. of dual norm)} \\
&\leq \frac{\Lambda_1 n_0^{\frac{1}{q}}}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\left\|\sum_{i=1}^m \sigma_i[\boldsymbol{\Phi}(x_i)]\right\|_\infty\right] && \text{(equivalence of } l_p \text{ norms)} \\
&= \frac{\Lambda_1 n_0^{\frac{1}{q}}}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\max_{j\in[1,n_1]}\left|\sum_{i=1}^m \sigma_i[\boldsymbol{\Phi}(x_i)]_j\right|\right] && \text{(def. of } l_\infty \text{ norm)} \\
&= \frac{\Lambda_1 n_0^{\frac{1}{q}}}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\max_{\substack{j\in[1,n_1]\\ s\in\{-1,+1\}}}\sum_{i=1}^m \sigma_i s[\boldsymbol{\Phi}(x_i)]_j\right] && \text{(def. of absolute value)} \\
&\leq \Lambda_1 n_0^{\frac{1}{q}} r_\infty \sqrt{m}\frac{\sqrt{2\log(2n_0)}}{m} = r_\infty \Lambda_1 n_0^{\frac{1}{q}}\sqrt{\frac{2\log(2n_0)}{m}}. && \text{(Massart's lemma)}
\end{aligned}
$$

The result then follows by application of Lemma 3. □

Theorem 5 (AdaNet Generalization Bound)**.** *Let $\hat{l} \in [1,k]$ and for each $k \in [1,\hat{l}]$, let $\hat{n}_k \in [1,n_k]$. For $k \in [1,\hat{l}]$ and $j \in [1,\hat{n}_k]$, let $w_{k,j} \in \mathbb{R}$ and $h_{k,j} \in \mathcal{H}_k^{(p)}$. Then for any $\delta > 0$, with probability at least $1-\delta$ over the sample S of size m drawn i.i.d. according to distribution D^m, the following inequality holds for any $f = \sum_{k=1}^{\hat{l}}\sum_{j=1}^{\hat{n}_k} w_{k,j}h_{k,j}$ with $\sum_{k=1}^{\hat{l}}\sum_{j=1}^{\hat{n}_k} w_{k,j} = 1$:*

$$
\begin{aligned}
R(f) \leq \hat{R}_{S,\rho} &+ \frac{4}{\rho}\sum_{k=1}^{\hat{l}}\sum_{j=1}^{\hat{n}_k} w_{k,j} 4^{k-1} r_\infty \left(\prod_{i=1}^k \Lambda_i n_{i-1}^{\frac{1}{q}}\right)\sqrt{\frac{2\log(2n_0)}{m}} + \frac{2}{\rho}\sqrt{\frac{\log(l)}{m}} \\
&+ \sqrt{\left\lceil \frac{4}{\rho^2}\log\left(\frac{\rho^2 m}{\log(l)}\right)\right\rceil \frac{\log(p)}{m} + \frac{\log(\frac{2}{\delta})}{2m}},
\end{aligned}
$$

Proof. By considering the symmetrized sets $\mathcal{H} = \bigcup_{k=1}^l (\mathcal{H}_k^{(p)} \cup (-\mathcal{H}_k^{(p)}))$, we can assume without loss of generality that the weights $w_{k,j}$ are non-negative.

We will now use the following structural learning guarantee for ensembles of hypotheses:

Lemma 7 (DeepBoost Generalization Bound, Theorem 1 Cortes et al. [2014])**.** *Let $l > 1$. Assume there exists some decomposition of the hypothesis space $\mathcal{H} = \cup_{i=1}^l \mathcal{H}_i$. Fix $\rho > 0$. Given any $h \in \mathcal{H}$ and a sample S, denote the empirical margin loss as $\hat{R}_{S,\rho}$. Then for any $\delta > 0$, with probability at least $1-\delta$ over the sample S of size m drawn i.i.d. according to distribution D^m, for any $\alpha_t \in \mathbb{R}_+$ such that $\sum_{t=1}^T \alpha_t = 1$, the following inequality holds for $f = \sum_{t=1}^T \alpha_t h_t$:*

$$
\begin{aligned}
R(f) \leq \hat{R}_{S,\rho} &+ \frac{4}{\rho}\sum_{t=1}^T \alpha_t \mathfrak{R}_m(\mathcal{H}_{k_t}) + \frac{2}{\rho}\sqrt{\frac{\log(l)}{m}} \\
&+ \sqrt{\left\lceil \frac{4}{\rho^2}\log\left(\frac{\rho^2 m}{\log(l)}\right)\right\rceil \frac{\log(l)}{m} + \frac{\log(\frac{2}{\delta})}{2m}},
\end{aligned}
$$

where for each $h_t \in \mathcal{H}$, k_t denotes the smallest $k \in [1,l]$ such that $h_t \in \mathcal{H}_{k_t}$.

Lemma 7 implies that

$$R(f) \leq \hat{R}_{S,\rho} + \frac{4}{\rho}\sum_{k=1}^{\hat{l}}\sum_{j=1}^{\hat{n}_k} w_{k,j}\mathfrak{R}_m(\mathcal{H}_k^{(p)} \cup (-\mathcal{H}_k^{(p)})) + \frac{2}{\rho}\sqrt{\frac{\log(l)}{m}} + \sqrt{\lceil \frac{4}{\rho^2}\log\left(\frac{\rho^2 m}{\log(l)}\right)\rceil \frac{\log(p)}{m} + \frac{\log(\frac{2}{\delta})}{2m}}.$$

By using symmetrization in Lemma 4,

$$\sum_{k=1}^{\hat{l}}\sum_{j=1}^{\hat{n}_k} w_{k,j}\mathfrak{R}_m(\mathcal{H}_k^{(p)} \cup (-\mathcal{H}_k^{(p)})) \leq \sum_{k=1}^{\hat{l}} 4^{k-1} r_\infty \Big(\prod_{i=1}^{k}\Lambda_i n_{i-1}^{\frac{1}{q}}\Big)\sqrt{\frac{2\log(2n_0)}{m}}\sum_{j=1}^{\hat{n}_k} w_{k,j}.$$

□

E Proofs for algorithmic design

E.1 New candidate nodes

Lemma 6 (Construction of new candidate nodes)**.** *Fix* $(h_{k-1,j})_{j=1}^{n_{k-1}} \subset \mathcal{H}_{k-1}^{(p)}$. *Then the solution* $\tilde{u}_k$ *to the optimization problem*

$$\max_{\|u\|_p \leq \Lambda_k} \mathbb{E}_{i\sim\mathcal{D}}[y_i \sum_{j=1}^{n_{k-1}} u_j(\varphi_{k-1} \circ h_{k-1,j})(x_i)],$$

can be computed coordinate-wise as:

$$(\tilde{u}_k)_j = \frac{\Lambda_k}{\|\mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q^{\frac{q}{p}}} |\mathrm{Margin}(\mathcal{D}, h_{k-1,j})|^{q-1} \operatorname{sgn}\left(\mathrm{Margin}(\mathcal{D}, h_{k-1,j})\right),$$

and the solution has value: $\Lambda_k \|\mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q$

Proof. By linearity of expectation,

$$\tilde{u}_k = \operatorname*{argmax}_{\|u\|_p \leq \Lambda_k} \mathbb{E}_{i\sim\mathcal{D}}[y_i \sum_{j=1}^{n_{k-1}} u_j(\varphi_{k-1} \circ h_{k-1,j})(x_i)] = \operatorname*{argmax}_{\|u\|_p \leq \Lambda_k} \mathbf{u} \cdot \mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1}).$$

We claim that

$$(\tilde{u}_k)_j = \frac{\Lambda_k}{\|\mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q^{\frac{q}{p}}} |\mathrm{Margin}(\mathcal{D}, h_{k-1,j})|^{q-1} \operatorname{sgn}\left(\mathrm{Margin}(\mathcal{D}, h_{k-1,j})\right).$$

To see this, note first that by Holder's inequality,

$$\mathbf{u} \cdot \mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1}) \leq \|u\|_p \| \mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q \leq \Lambda_k \| \mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q,$$

and the expression on the right-hand side is our proposed value. At the same time, our choice of $\tilde{u}_k$ also satisfies this upper bound:

$$\begin{aligned}
\tilde{u}_k \cdot \mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1}) &= \sum_{j=1}^{n_{k-1}} (\tilde{u}_k)_j \mathrm{Margin}(\mathcal{D}, h_{k-1,j}) \\
&= \sum_{j=1}^{n_{k-1}} \frac{\Lambda_k}{\|\mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q^{\frac{q}{p}}} |\mathrm{Margin}(\mathcal{D}, h_{k-1,j})|^q \\
&= \frac{\Lambda_k}{\|\mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q^{\frac{q}{p}}} \|\mathrm{Margin}(\mathcal{D}, h_{k-1,j})\|_q^q \\
&= \Lambda_k \| \mathrm{Margin}(\mathcal{D}, \mathbf{h}_{k-1})\|_q.
\end{aligned}$$

Thus, $\tilde{u}_k$ is a solution to the optimization problem and achieves the claimed value. □

E.2 Derivation of coordinate descent update

Recall the form of our objective function:

$$F(w_t) = \frac{1}{m}\sum_{i=1}^{m}\Phi\big(1 - y_i\sum_{k=1}^{l}\sum_{j=1}^{n_k} w_{t,k,j}h_{k,j}(x_i)\big) + \sum_{k=1}^{l}\sum_{j=1}^{n_k}\Gamma_k|w_{k,j}|.$$

We want to find the directional derivative with largest magnitude as well as the optimal step-size in this coordinate direction.

Since F is non-differentiable at 0 for each coordinate (due to the weighted l_1 regularization), we must choose a representative of the subgradient. Since, F is convex, it admits both left and right directional derivatives, which we denote by

$$\nabla^+_{k,j}F(w) = \lim_{\eta\to 0^+}\frac{F(w+\eta e_{k,j}) - F(w)}{\eta}, \quad \nabla^-_{k,j}F(w) = \lim_{\eta\to 0^-}\frac{F(w+\eta e_{k,j}) - F(w)}{\eta}.$$

Moreover, convexity ensures that $\nabla^-_{k,j}F \le \nabla^+_{k,j}F$. Now, let $\delta_{k,j}F$ be the element of the subgradient that we will use to compare descent magnitudes, so that $(k_t, j_t) = \text{argmax}_{k\in[1,l],j\in[1,n_k]}\left|\delta_{k,j}F(w_t)\right|$. This subgradient will always be chosen as the one closest to 0:

$$\begin{aligned}
\delta_{k,j}F(w) &= 0 && \text{if } \nabla^-_{k,j}F(w) \le 0 \le \nabla^+_{k,j}F(w)\\
&= \nabla^+_{k,j}(w) && \text{if } \nabla^-_{k,j}F(w) \le \nabla^+_{k,j}F(w) \le 0\\
&= \nabla^-_{k,j}(w) && \text{if } 0 \le \nabla^-_{k,j}F(w) \le \nabla^+_{k,j}F(w).
\end{aligned}$$

Suppose that $w_{t,k,j} \neq 0$. Then by continuity, for η sufficiently small, $w_{t,k,j}$ and $w_{t,k,j} + \eta e_{k,j}$ have the same sign so that

$$\begin{aligned}
F(w_t + \eta e_{k,j}) = \frac{1}{m}\sum_{i=1}^{m}\Phi\bigg(1 - y_i\sum_{k=1}^{l}\sum_{j=1}^{n_k} w_{t,k,j}h_{k,j}(x_i) - \eta y_i h_{k,j}(x_i)\bigg)\\
+ \sum_{(\tilde{k},\tilde{j})\neq(k,j)}\Gamma_k|w_{t,\tilde{k},\tilde{j}}| + \Gamma_k\,\text{sgn}(w_{t,k,j})(w_{t,k,j}+\eta).
\end{aligned}$$

Furthermore, F is differentiable in the (k,j)-th at w_t, which implies that

$$\begin{aligned}
\nabla_{k,j}F(w_t) &= \frac{1}{m}\sum_{i=1}^{m} -y_i h_{k,j}(x_i)\Phi'\bigg(1 - y_i\sum_{k=1}^{l}\sum_{j=1}^{n_k} w_{t,k,j}h_{k,j}(x_i)\bigg) + \text{sgn}(w_{t,k,j})\Gamma_k\\
&= \frac{1}{m}\sum_{i=1}^{m} y_i h_{k,j}(x_i)\mathcal{D}_t(i)S_t + \text{sgn}(w_{t,k,j})\Gamma_k\\
&= (2\epsilon_{t,k,j} - C_k)\frac{S_t}{m} + \text{sgn}(w_{t,k,j})\Gamma_k
\end{aligned}$$

When $w_{t,k,j} = 0$, we can consider the left and right directional derivatives:

$$\begin{aligned}
\nabla^+_{k,j}F(w_t) &= (2\epsilon_{t,k,j} - C_k)\frac{S_t}{m} + \Gamma_k\\
\nabla^-_{k,j}F(w_t) &= (2\epsilon_{t,k,j} - C_k)\frac{S_t}{m} - \Gamma_k
\end{aligned}$$

Moreover,

$$\begin{aligned}
\left|\epsilon_{t,k,j} - \frac{C_k}{2}\right| \le \frac{\Gamma_k m}{2S_t} &\qquad \Leftrightarrow \nabla^-_{k,j}F(w) \le 0 \le \nabla^+_{k,j}F(w)\\
\epsilon_{t,k,j} - \frac{C_k}{2} \le \frac{-\Gamma_k m}{2S_t} &\qquad \Leftrightarrow \nabla^-_{k,j}F(w) \le \nabla^+_{k,j}F(w) \le 0\\
\frac{\Gamma_k m}{2S_t} \le \epsilon_{t,k,j} - \frac{C_k}{2} &\qquad \Leftrightarrow 0 \le \nabla^-_{k,j}F(w) \le \nabla^+_{k,j}F(w),
\end{aligned}$$

INIT$(m, l, n_1, \ldots, n_l)$
1 **for** $k \leftarrow 1$ **to** l **do**
2 **for** $j \leftarrow 1$ **to** n_k **do**
3 $w_{0,k,j} \leftarrow 0$
4 **for** $i \leftarrow 1$ **to** m **do**
5 $\mathcal{D}_1(i) \leftarrow \frac{1}{m}$
6 $(\hat{l}, \hat{n}_1) \leftarrow (1, n_1)$
7 **for** $k \leftarrow 2$ **to** l **do**
8 $\hat{n}_k \leftarrow 0$
9 **return** $(w_{0,k,j})_{k\in[1,l],j\in[1,n_k]}, \mathcal{D}_1, \hat{l}, (\hat{n}_k)_{k\in[1,l],j\in[1,\hat{n}_k]}$

Figure 5: Pseudocode for initializing parameters.

so that we have

$$\begin{aligned}
\delta_{k,j}F(w_t) &= (2\epsilon_{t,k,j} - C_k)\frac{S_t}{m} + \operatorname{sgn}(w_{t,k,j})\Gamma_k && \text{if } \left|\epsilon_{t,k,j} - \frac{C_k}{2}\right| \le \frac{\Gamma_k m}{2S_t} \\
&= 0 && \text{else if } \left|\epsilon_{t,k,j} - \frac{C_k}{2}\right| \le \frac{\Gamma_k m}{2S_t} \\
&= (2\epsilon_{t,k,j} - C_k)\frac{S_t}{m} - \operatorname{sgn}\left(\epsilon_{t,k,j} - \frac{C_k}{2}\right)\Gamma_k && \text{otherwise .}
\end{aligned}$$

F Other components of pseudocode

Figures 5, 6, 7, 8 present the remaining components of the pseudocode for ADANET with exponential loss.

The initial weight vector $\mathbf{w}_0$ is initialized to 0, and the initial weight distribution $\mathcal{D}_1$ is uniform over the coordinates.

The best node is simply the one with the highest score $|d_{k,j}|$ (or $|\tilde{d}_k|$) among all existing nodes and the new candidate nodes.

The step-size taken at each round is the optimal step in the direction computed. For exponential loss functions, this can be computed exactly, and in general, it can be approximated numerically via line search methods (since the objective is convex).

The updated distribution at time t will be proportional to $\Phi'(1 - y_i f_{t-1}(x_i)$, as explained in Section C.3.

G Convergence of ADANET

Theorem 8. *Let Φ be a twice-continuously differentiable function with $\Phi'' > 0$, and suppose we terminate* ADANET *after $\mathcal{O}\big(\log(1/\epsilon)\big)$ iterations if it does not add a new node. Let $I_s \subset [1, N]$ be the first j nodes added by* ADANET*, and let $w^*_{I_s} = \operatorname{argmin}_{w\in P_{I_s}(\mathbb{R}^N_+)} F(w)$, where P_{I_s} denotes projection onto $\mathbb{R}^{I_s}$. Let $\hat{N} = \sum_{k=1}^{\hat{l}} \hat{n}_k$ be the total number of nodes constructed by the* ADANET *algorithm at termination. Then* ADANET *will terminate after at most $\mathcal{O}(\hat{N}\log(1/\epsilon))$ iterations, producing a neural network and a set of weights such that*

$$F(w_{AdaNet}) - \min_{s\in[1,\hat{N}]} F(w^*_{I_s}) < \epsilon$$

Proof. Recall that

$$F(w) = \frac{1}{m}\sum_{i=1}^{m} \Phi\Big(1 - y_i \sum_{j=1}^{N} w_j h_j(x_i)\Big) + \sum_{j=1}^{N} \Gamma_j |w_j|.$$

BESTNODE$(\hat{l}, (d_{k,j})_{k\in[1,\hat{l}],j\in[1,\hat{n}_k]}, (\tilde{d}_k)_{k\in[2,\hat{l}+1\wedge l]}, (\hat{n}_k)_{k=1}^{l})$

if $\max_{k\in[1,\hat{l}],j\in[1,\hat{n}_k]} |d_{k,j}| > \max_{j\in[2,\hat{l}+1]} |\tilde{d}_j|$ **then**
  $(k, j) \leftarrow \operatorname*{argmax}_{k\in[1,\hat{l}],j\in[1,\hat{n}_k]} |d_{k,j}|$
else $k_{\text{new}} \leftarrow \operatorname*{argmax}_{j\in[2,\hat{l}+1]} |\tilde{d}_j|$
  if $k_{\text{new}} > \hat{l}$ **then**
    $\hat{l} \leftarrow \hat{l} + 1$
  $\hat{n}_{k_{\text{new}}} \leftarrow \hat{n}_{k_{\text{new}}} + 1$
  $(k, j) \leftarrow (\tilde{k}, \hat{n}_{k_{\text{new}}})$
  $(h_{k,j}, \epsilon_{t,k,j}) \leftarrow (\tilde{u}_k, \tilde{\epsilon}_{k_{\text{new}}})$
$\epsilon_t \leftarrow \epsilon_{t,k,j}$
return $((k, j), \epsilon_t, \hat{l}, (n_1, \ldots, \hat{n}_{\hat{l}}))$

Figure 6: Pseudocode for finding the node with the largest contribution toward decreasing the objective.

APPLYSTEP$((k, j), w_{t-1})$

$\eta_t \leftarrow \operatorname{argmin}_\eta F(w_t + \eta e_{k_t,j_t})$
$\mathbf{w}_t \leftarrow \mathbf{w}_{t-1}$
$w_{t,k,j} \leftarrow w_{t,k,j} + \eta_t$
return $\mathbf{w}_t$

Figure 7: Pseudocode for applying the optimal step size to the network. In general, the step size can be computed via line search.

Since ADANET initializes all weights to 0 and grows the coordinate space in an online fashion, we may consider the algorithm in epochs, so that if the support of w_t at any given t is I_s, then

$$F(w) = F_{I_s}(w) = \frac{1}{m}\sum_{i=1}^{m} \Phi\Big(1 - y_i \sum_{j\in I_s} w_j h_j(x_i)\Big) + \sum_{j\in I_s} \Gamma_j |w_j|.$$

Fix an epoch $s \in [1, \hat{N}]$. The optimal set of weights within the support I_s is given by $w^*_{I_s}$, and this solution exists because F is a coercive convex function (due to the weighted l_1 regularization). Let $M \in \mathbb{R}^{m\times N}$ be a matrix with elements given by $M_{i,j} = h_j(x_i)y_i$, and let $e_i \in \mathbb{R}^m$ be a i-th elementary basis vector of $\mathbb{R}^m$. Then for any vector $w \in \mathbb{R}^N$, $e_i^\top Mw = \sum_{j=1}^N w_j h_j(x_i) y_i$. Thus, if we define denote by $G_{I_s}(z) = \frac{1}{m}\sum_{i=1}^m \Phi(1 - e_i^\top z)$, then the first component of F_{I_s} can be written as $G_{I_s}(Mw)$.

Moreover, since Φ is twice continuously differentiable and $\Phi'' > 0$, it follows that G_{I_s} is twice continuously differentiable and strictly convex. We can also compute that for any $w \in \mathbb{R}^{\hat{N}}$,

$$\nabla^2 g(Mw) = \frac{1}{m}\sum_{i=1}^{m} \Phi''(1 - e_i^\top Mw) e_i e_i^\top,$$

UPDATEDISTRIBUTION$(\mathbf{w}_t, S, (h_{k,j})_{k\in[1,\hat{l}],j\in[1,\hat{n}_k]})$

$S_{t+1} \leftarrow \sum_{i=1}^m \Phi'\big(1 - y_i \sum_{k=1}^{\hat{l}} \sum_{j=1}^{\hat{n}_k} w_{t,k,j} h_{k,j}(x_i)\big)$
for $i \leftarrow 1$ **to** m **do**
  $\mathcal{D}_{t+1}(i) \leftarrow \frac{\Phi'\left(1 - y_i \sum_{k=1}^{\hat{l}} \sum_{j=1}^{\hat{n}_k} w_{t,k,j} h_{k,j}(x_i)\right)}{S_{t+1}}$
return $(\mathcal{D}_{t+1}, S_{t+1})$

Figure 8: Pseudocode for updating the distribution for the next iteration.

which is positive definite since $\Phi'' > 0$. Finally, since $\mathcal{H}$ is symmetric, we can, at the cost of flipping some $h_j \to -h_j$, equivalently write: $F_{I_s}(w) = g(Mw) + \sum_{j \in I_s} \Gamma_j w_j$, subject to $w_j \geq 0$.

Thus, the problem of minimizing F_{I_s} is equivalent to the optimization problem studied in Luo and Tseng [1992], and we have verified that the conditions are satisfied as well. If the algorithm doesn't add another coordinate, then it performs the Gauss-Southwell method on the coordinates I_s. By Theorem 2.1 in Luo and Tseng [1992], this method will converge to the optimal set of weights with support in I_s linearly. This implies that if the algorithm terminates, it will maintain the error guarantee: $F_{I_s}(w_{\text{AdaNet}}) - F_{I_s}(w^*_{I_s}) < \epsilon$.

If the algorithm does add a new coordinate before termination, then we can apply the same argument to $F_{I_{s+1}}$.

Thus, when the algorithm does finally terminate, we maintain the error guarantee for every subset $I_s \subset [1, N]$ of nodes that we create, and the total run time is at most $\mathcal{O}(\hat{N} log(1/\epsilon))$ iterations. □

H Large-scale implementation: AdaNet+

Our optimization problem is a regularized empirical risk minimization problem of the form: $F(x) = G(x) + \psi(x) = \frac{1}{m} \sum_{i=1}^{m} f_i(x) + \sum_{j=1}^{N} \psi_j(x_j)$, where each f_i is smooth and convex and ψ, also convex, decomposes across the coordinates of x.

For any subset $\mathcal{B} \subset [1, m]$, let $G_{\mathcal{B}} = \frac{1}{m} \sum_{i \in \mathcal{B}} f_i$ denote the components of the ERM objective that correspond to that subset. For any $j \in [1, N]$, let ∇_j denote the partial derivative in the coordinate j.

We can leverage the Mini-Batch Randomized Block Coordinate Descent (MBRCD) technique introduced by Zhao et al. [2014]. Their work can be viewed as the randomized block coordinate descent variant of the Stochastic Variance Reduced Gradient (SVRG) family of algorithms (Johnson and Zhang [2013], Xiao and Zhang [2014]).

Our algorithm divides the entire training period into K epochs. At every step, the algorithm samples two mini-batches from $[1, m]$ uniformly. The first is used to approximate the ERM objective for the descent step, and the second is used to apply the NewNodes subroutine in Figure 4 to generate new candidate coordinates. After generating these coordinates, the algorithm samples a coordinate from the set of new coordinates and the existing ones. Based on the coordinate chosen, it updates the active set. Then the algorithm computes a gradient descent step with the SVRG estimator in place of the gradient and with the sampled approximation of the true ERM objective. It then makes a proximal update with ψ as the proximal function. Finally, the "checkpoint parameter" of the SVRG estimator is updated at the end of every epoch.

While our optimization problem itself is not strongly convex, we can apply the MBRCD indirectly by adding a strongly convex regularizer, solving: $\tilde{F}(x) = F(x) + \gamma R(x)$, where R is a 1-strongly convex function, to still yield a competitive guarantee.

Moreover, since our non-smooth component is a weighted-l_1 term, the proximal update can be solved efficiently and in closed form in a manner that is similar to the Iterative Shrinkage-Thresholding Algorithm (ISTA) of Beck and Teboulle [2009].

Figure 9 presents the pseudocode of the algorithm, AdaNet+.

Appendix References

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

Z.-Q. Luo and P. Tseng. On the convergence of coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7 – 35, 1992.

L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

ADANET+$(\tilde{w}^0, \eta, (x_i, y_i)_{i=1}^m, (n_k, \Lambda_k, \Gamma_k, C_k)_{k=1}^l)$

$(\hat{N}, \hat{l}, (\hat{n}_k)_{k=1}^l) \leftarrow$ INIT+$((n_k)_{k=1}^l)$
for $s \leftarrow 1$ **to** S **do**
  $\tilde{w} \leftarrow \tilde{w}^{(s-1)}$
  $\tilde{\mu} \leftarrow \nabla\Phi(\tilde{w}^{(s-1)})$
  $w^{(0)} \leftarrow \tilde{w}^{(s-1)}$
  for $t \leftarrow 1$ **to** T **do**
    $\mathcal{B}_1 \leftarrow U([1, m])$
    $\mathcal{B}_2 \leftarrow U([1, m])$
    $(\tilde{d}_k, \tilde{u}_k)_{k=1}^l \leftarrow$ NEWNODES+$(\mathcal{B}_2, (\mathbf{h}_k, n_k, \hat{n}_k, \Lambda_k)_{k=1}^l)$
    $j \leftarrow U([1, \hat{N} + \hat{l}])$
    $(\hat{N}, \hat{l}, (\hat{n}_k, \mathbf{h}_k)_{k=1}^l) \leftarrow$ UPDATEACTIVESET+$(j, \hat{N}, \hat{l}, (\hat{n}_k, \mathbf{h}_k, \tilde{u}_k)_{k=1}^l)$
    $w_{t,j} \leftarrow \mathrm{prox}_{\Gamma_j|\cdot| + \frac{\gamma}{2}|\cdot|^2}(w_{t-1,j} - \eta[\nabla_j G_{\mathcal{B}_1}(w_{t-1}) - \nabla_j F_{\mathcal{B}_1}(\tilde{w}) + \tilde{\mu}_j])$
return $(d_{k,j})_{k \in [1,\hat{l}], j \in [1,\hat{n}_k]}$

Figure 9: Pseudocode for large-scale implementation of ADANET.

T. Zhao, M. Yu, Y. Wang, R. Arora, and H. Liu. Accelerated mini-batch randomized block coordinate descent method. In *Advances in neural information processing systems*, pages 3329–3337, 2014.