AdaNet: Adaptive Structural Learning of Artificial Neural Networks

Corinna Cortes [1] **Xavier Gonzalvo** [1] **Vitaly Kuznetsov** [1] **Mehryar Mohri** [2 1] **Scott Yang** [2]

Abstract

We present new algorithms for adaptively learning artificial neural networks. Our algorithms (ADANET) adaptively learn both the structure of the network and its weights. They are based on a solid theoretical analysis, including data-dependent generalization guarantees that we prove and discuss in detail. We report the results of large-scale experiments with one of our algorithms on several binary classification tasks extracted from the CIFAR-10 dataset and on the Criteo dataset. The results demonstrate that our algorithm can automatically learn network structures with very competitive performance accuracies when compared with those achieved by neural networks found by standard approaches.

1. Introduction

Multi-layer artificial neural networks form a powerful learning model which has helped achieve a remarkable performance in several applications in recent years. Representing the input through increasingly more abstract layers of feature representation has shown to be very effective in natural language processing, image captioning, speech recognition and several other areas (Krizhevsky et al., 2012; Sutskever et al., 2014). However, despite the compelling arguments for adopting multi-layer neural networks as a general template for tackling learning problems, training these models and designing the right network for a given task has raised several theoretical questions and faced numerous practical challenges.

A critical step in learning a large multi-layer neural network for a specific task is the choice of its architecture, which includes the number of layers and the number of units within each layer. Standard training methods for neural networks return a model admitting precisely the number of layers and units specified since there needs to be at least one path through the network for the hypothesis to be non-trivial. Single weights can be pruned (Han et al., 2015) via a technique originally termed *Optimal Brain Damage* (LeCun et al., 1990), but the global architecture remains unchanged. Thus, this imposes a stringent lower bound on the complexity of the model, which may not match that of the learning task considered: complex networks trained with insufficient data may be prone to overfitting and, in reverse, simpler architectures may not suffice to achieve an adequate performance.

This places a considerable burden on the user who is left with the requirement to specify an architecture with the right complexity, which is often a difficult task even with a significant level of experience and domain knowledge. As a result, the choice of the network is typically left to a hyperparameter search using a validation set. This search space can quickly become exorbitantly large (Szegedy et al., 2015; He et al., 2015) and large-scale hyperparameter tuning to find an effective network architecture often wasteful of data, time, and resources (e.g. grid search, random search (Bergstra et al., 2011)).

This paper seeks precisely to address some of these issues. We present a theoretical analysis of the problem of learning simultaneously both the network architecture and its parameters. To the best of our knowledge, our results are the first generalization bounds for the problem of *structural learning of neural networks*. These general guarantees can guide the design of a variety of different algorithms for learning in this setting. We describe in detail two such algorithms (ADANET algorithms) that directly benefit from our theory.

Rather than enforcing a pre-specified architecture and thus a fixed network complexity, our ADANET algorithms *adaptively* learn the appropriate network architecture for a learning task. Starting from a simple linear model, our algorithms incrementally augment the network with more units and additional layers, as needed. The choice of the additional subnetworks depends on their complexity and is directly guided by our learning guarantees. Remarkably, the optimization problems for both of our algorithms turn out to be strongly convex and thus guaranteed to admit a unique global solution.

[1]Google Research, New York, NY, USA; [2]Courant Institute of Mathematical Sciences, New York, NY, USA. Correspondence to: Vitaly Kuznetsov <vitalyk@google.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

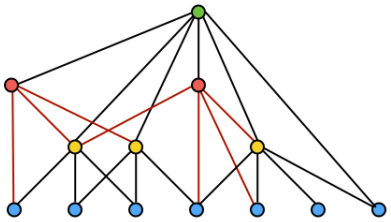

Figure 1. An example of a general network architecture: the output layer (in green) is connected to all of the hidden units as well as some input units. Some hidden units (in red and yellow) are connected not only to the units in the layer directly below, but also to units at other levels.

The paper is organized as follows. In Appendix A, we give a detailed discussion of previous work related to this topic. Section 2 describes the general network architecture and therefore the hypothesis set that we consider. Section 3 provides a formal description of our learning scenario. In Section 4, we prove strong generalization guarantees for learning in this setting, which help guide the design of the algorithm described in Section 5, as well as a variant described in Appendix C. We report the results of our experiments with AdaNet in Section 6.

2. Network architecture

In this section, we describe the general network architecture we consider for feedforward neural networks, thereby also defining our hypothesis set. To simplify the presentation, we restrict our attention to the case of binary classification. However, all our results can be straightforwardly extended to multi-class classification, including the network architecture, by augmenting the number of output units, and, our generalization guarantees, by using existing multi-class counterparts of the binary classification ensemble margin bounds we use.

A common model for feedforward neural networks is the multi-layer architecture where units in each layer are only connected to those in the layer below. We will consider more general architectures where a unit can be connected to units in any of the layers below, as illustrated by Figure 1. In particular, the output unit in our network architectures can be connected to any other unit. These more general architectures include as special cases standard multi-layer networks (by zeroing out appropriate connections) as well as somewhat more exotic ones (He et al., 2015; Huang et al., 2016). In fact, our definition covers any architecture that can be represented as a directed acyclic graph (DAG).

More formally, the artificial neural networks we consider are defined as follows. Let l denote the number of intermediate layers in the network and n_k the maximum number of units in layer $k \in [l]$. Each unit $j \in [n_k]$ in layer k represents a function denoted by $h_{k,j}$ (before composition with an activation function). Let $\mathcal{X}$ denote the input space and for any $x \in \mathcal{X}$, let $\mathbf{\Psi}(x) \in \mathbb{R}^{n_0}$ denote the corresponding feature vector. Then, the family of functions defined by the first layer functions $h_{1,j}$, $j \in [n_1]$, is the following:

$$\mathcal{H}_1 = \left\{ x \mapsto \mathbf{u} \cdot \mathbf{\Psi}(x) \colon \mathbf{u} \in \mathbb{R}^{n_0}, \|\mathbf{u}\|_p \leq \Lambda_{1,0} \right\}, \quad (1)$$

where $p \geq 1$ defines an l_p-norm and $\Lambda_{1,0} \geq 0$ is a hyperparameter on the weights connecting layer 0 and layer 1. The family of functions $h_{k,j}$, $j \in [n_k]$, in a higher layer $k > 1$ is then defined as follows:

$$\mathcal{H}_k = \left\{ x \mapsto \sum_{s=1}^{k-1} \mathbf{u}_s \cdot (\varphi_s \circ \mathbf{h}_s)(x) \colon \right. \\ \left. \mathbf{u}_s \in \mathbb{R}^{n_s}, \|\mathbf{u}_s\|_p \leq \Lambda_{k,s}, h_{k,s} \in \mathcal{H}_s \right\}, \quad (2)$$

where, for each unit function $h_{k,s}$, $\mathbf{u}_s$ in (2) denotes the vector of weights for connections from that unit to a lower layer $s < k$. The $\Lambda_{k,s}$s are non-negative hyperparameters and $\varphi_s \circ \mathbf{h}_s$ abusively denotes a coordinate-wise composition: $\varphi_s \circ \mathbf{h}_s = (\varphi_s \circ h_{s,1}, \ldots, \varphi_s \circ h_{s,n_s})$. The φ_ss are assumed to be 1-Lipschitz activation functions. In particular, they can be chosen to be the *Rectified Linear Unit function* (ReLU function) $x \mapsto \max\{0, x\}$, or the *sigmoid function* $x \mapsto \frac{1}{1+e^{-x}}$. The choice of the parameter $p \geq 1$ determines the sparsity of the network and the complexity of the hypothesis sets $\mathcal{H}_k$.

For the networks we consider, the output unit can be connected to all intermediate units, which therefore defines a function f as follows:

$$f = \sum_{k=1}^{l} \sum_{j=1}^{n_k} w_{k,j} h_{k,j} = \sum_{k=1}^{l} \mathbf{w}_k \cdot \mathbf{h}_k, \quad (3)$$

where $\mathbf{h}_k = [h_{k,1}, \ldots, h_{k,n_k}]^\top \in \mathcal{H}_k^{n_k}$ and $\mathbf{w}_k \in \mathbb{R}^{n_k}$ is the vector of connection weights to units of layer k. Observe that, for $\mathbf{u}_s = 0$ for $s < k-1$ and $\mathbf{w}_k = 0$ for $k < l$, our architectures coincides with standard multi-layer feedforward ones.

We will denote by $\mathcal{F}$ the family of functions f defined by (3) with the absolute value of the weights summing to one:

$$\mathcal{F} = \left\{ \sum_{k=1}^{l} \mathbf{w}_k \cdot \mathbf{h}_k \colon \mathbf{h}_k \in \mathcal{H}_k^{n_k}, \sum_{k=1}^{l} \|\mathbf{w}_k\|_1 = 1 \right\}.$$

Let $\widetilde{\mathcal{H}}_k$ denote the union of $\mathcal{H}_k$ and its reflection, $\widetilde{\mathcal{H}}_k = \mathcal{H}_k \cup (-\mathcal{H}_k)$, and let $\mathcal{H}$ denote the union of the families $\widetilde{\mathcal{H}}_k$: $\mathcal{H} = \bigcup_{k=1}^{l} \widetilde{\mathcal{H}}_k$. Then, $\mathcal{F}$ coincides with the convex hull of $\mathcal{H}$: $\mathcal{F} = \operatorname{conv}(\mathcal{H})$.

For any $k \in [l]$, we will also consider the family $\mathcal{H}_k^*$ derived from $\mathcal{H}_k$ by setting $\Lambda_{k,s} = 0$ for $s < k-1$, which

corresponds to units connected only to the layer below. We similarly define $\widetilde{\mathcal{H}}_k^* = \mathcal{H}_k^* \cup (-\mathcal{H}_k^*)$ and $\mathcal{H}^* = \cup_{k=1}^l \mathcal{H}_k^*$, and define $\mathcal{F}^*$ as the convex hull $\mathcal{F}^* = \mathrm{conv}(\mathcal{H}^*)$. Note that the architecture corresponding to the family of functions $\mathcal{F}^*$ is still more general than standard feedforward neural network architectures since the output unit can be connected to units in different layers.

3. Learning problem

We consider the standard supervised learning 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 $\mathcal{D}^m$.

For a function f taking values in $\mathbb{R}$, we denote by $R(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[1_{yf(x)\leq 0}]$ its generalization error and, for any $\rho > 0$, by $\widehat{R}_{S,\rho}(f)$ its empirical margin error on the sample S: $\widehat{R}_{S,\rho}(f) = \frac{1}{m}\sum_{i=1}^m 1_{y_i f(x_i)\leq\rho}$.

The learning problem consists of using the training sample S to determine a function f defined by (3) with small generalization error $R(f)$. For an accurate predictor f, we expect many of the weights to be zero and the corresponding architecture to be quite sparse, with fewer than n_k units at layer k and relatively few non-zero connections. In that sense, learning an accurate function f implies also learning the underlying architecture.

In the next section, we present data-dependent learning bounds for this problem that will help guide the design of our algorithms.

4. Generalization bounds

Our learning bounds are expressed in terms of the Rademacher complexities of the hypothesis sets $\mathcal{H}_k$. The empirical Rademacher complexity of a hypothesis set $\mathcal{G}$ for a sample S is denoted by $\widehat{\mathfrak{R}}_S(\mathcal{G})$ and defined as follows:

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) = \frac{1}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{G}}\sum_{i=1}^m \sigma_i h(x_i)\right],$$

where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)$, with σ_is independent uniformly distributed random variables taking values in $\{-1, +1\}$. Its Rademacher complexity is defined by $\mathfrak{R}_m(\mathcal{G}) = \mathbb{E}_{S\sim\mathcal{D}^m}[\widehat{\mathfrak{R}}_S(\mathcal{G})]$. These are data-dependent complexity measures that lead to finer learning guarantees (Koltchinskii & Panchenko, 2002; Bartlett & Mendelson, 2002).

As pointed out earlier, the family of functions $\mathcal{F}$ is the convex hull of $\mathcal{H}$. Thus, generalization bounds for ensemble methods can be used to analyze learning with $\mathcal{F}$. In particular, we can leverage the recent margin-based learning guarantees of Cortes et al. (2014), which are finer than those that can be derived via a standard Rademacher complexity analysis (Koltchinskii & Panchenko, 2002), and which admit an explicit dependency on the mixture weights $\mathbf{w}_k$ defining the ensemble function f. That leads to the following learning guarantee.

Theorem 1 (Learning bound)**.** *Fix $\rho > 0$. Then, for any $\delta > 0$, with probability at least $1 - \delta$ over the draw of a sample S of size m from $\mathcal{D}^m$, the following inequality holds for all $f = \sum_{k=1}^l \mathbf{w}_k \cdot \mathbf{h}_k \in \mathcal{F}$:*

$$R(f) \leq \widehat{R}_{S,\rho}(f) + \frac{4}{\rho}\sum_{k=1}^l \|\mathbf{w}_k\|_1 \mathfrak{R}_m(\widetilde{\mathcal{H}}_k) + \frac{2}{\rho}\sqrt{\frac{\log l}{m}} + C(\rho, l, m, \delta),$$

where $C(\rho, l, m, \delta) = \sqrt{\left\lceil \frac{4}{\rho^2}\log(\frac{\rho^2 m}{\log l})\right\rceil \frac{\log l}{m} + \frac{\log(2/\delta)}{2m}} = \widetilde{O}\left(\frac{1}{\rho}\sqrt{\frac{\log l}{m}}\right)$.

The proof of this result, as well as that of all other main theorems are given in Appendix B. The bound of the theorem can be generalized to hold uniformly for all $\rho \in (0, 1]$, at the price of an additional term of the form $\sqrt{\log\log_2(2/\rho)/m}$ using standard techniques (Koltchinskii & Panchenko, 2002).

Observe that the bound of the theorem depends only logarithmically on the depth of the network l. But, perhaps more remarkably, the complexity term of the bound is a $\|\mathbf{w}_k\|_1$-weighted average of the complexities of the layer hypothesis sets $\mathcal{H}_k$, where the weights are precisely those defining the network, or the function f. This suggests that a function f with a small empirical margin error and a deep architecture benefits nevertheless from a strong generalization guarantee, if it allocates more weights to lower layer units and less to higher ones. Of course, when the weights are sparse, that will imply an architecture with relatively fewer units or connections at higher layers than at lower ones. The bound of the theorem further gives a quantitative guide for apportioning the weights, depending on the Rademacher complexities of the layer hypothesis sets.

This data-dependent learning guarantee will serve as a foundation for the design of our structural learning algorithms in Section 5 and Appendix C. However, to fully exploit it, the Rademacher complexity measures need to be made explicit. One advantage of these data-dependent measures is that they can be estimated from data, which can lead to more informative bounds. Alternatively, we can derive useful upper bounds for these measures which can be more conveniently used in our algorithms. The next results in this section provide precisely such upper bounds, thereby leading to a more explicit generalization bound.

We will denote by q the conjugate of p, that is $\frac{1}{p} + \frac{1}{q} = 1$, and define $r_\infty = \max_{i\in[1,m]} \|\boldsymbol{\Psi}(x_i)\|_\infty$.

Our first result gives an upper bound on the Rademacher complexity of $\mathcal{H}_k$ in terms of the Rademacher complexity of other layer families.

Lemma 1. *For any $k > 1$, the empirical Rademacher complexity of $\mathcal{H}_k$ for a sample S of size m can be upper-bounded as follows in terms of those of $\mathcal{H}_s$s with $s < k$:*

$$\widehat{\mathfrak{R}}_S(\mathcal{H}_k) \leq 2 \sum_{s=1}^{k-1} \Lambda_{k,s} \, n_s^{\frac{1}{q}} \, \widehat{\mathfrak{R}}_S(\mathcal{H}_s).$$

For the family $\mathcal{H}_k^*$, which is directly relevant to many of our experiments, the following more explicit upper bound can be derived, using Lemma 1.

Lemma 2. *Let $\Lambda_k = \prod_{s=1}^k 2\Lambda_{s,s-1}$ and $N_k = \prod_{s=1}^k n_{s-1}$. Then, for any $k \geq 1$, the empirical Rademacher complexity of $\mathcal{H}_k^*$ for a sample S of size m can be upper bounded as follows:*

$$\widehat{\mathfrak{R}}_S(\mathcal{H}_k^*) \leq r_\infty \Lambda_k N_k^{\frac{1}{q}} \sqrt{\frac{\log(2n_0)}{2m}}.$$

Note that N_k, which is the product of the number of units in layers below k, can be large. This suggests that values of p closer to one, that is larger values of q, could be more helpful to control complexity in such cases. More generally, similar explicit upper bounds can be given for the Rademacher complexities of subfamilies of $\mathcal{H}_k$ with units connected only to layers $k, k-1, \ldots, k-d$, with d fixed, $d < k$. Combining Lemma 2 with Theorem 1 helps derive the following explicit learning guarantee for feedforward neural networks with an output unit connected to all the other units.

Corollary 1 (Explicit learning bound)**.** *Fix $\rho > 0$. Let $\Lambda_k = \prod_{s=1}^k 4\Lambda_{s,s-1}$ and $N_k = \prod_{s=1}^k n_{s-1}$. Then, for any $\delta > 0$, with probability at least $1 - \delta$ over the draw of a sample S of size m from $\mathcal{D}^m$, the following inequality holds for all $f = \sum_{k=1}^l \mathbf{w}_k \cdot \mathbf{h}_k \in \mathcal{F}^*$:*

$$R(f) \leq \widehat{R}_{S,\rho}(f) + \frac{2}{\rho} \sum_{k=1}^{l} \|\mathbf{w}_k\|_1 \left[\overline{r}_\infty \Lambda_k N_k^{\frac{1}{q}} \sqrt{\frac{2\log(2n_0)}{m}} \right] + \frac{2}{\rho} \sqrt{\frac{\log l}{m}} + C(\rho, l, m, \delta),$$

where $C(\rho, l, m, \delta) = \sqrt{\left\lceil \frac{4}{\rho^2} \log\left(\frac{\rho^2 m}{\log l}\right) \right\rceil \frac{\log l}{m} + \frac{\log\left(\frac{2}{\delta}\right)}{2m}} = \widetilde{O}\left(\frac{1}{\rho}\sqrt{\frac{\log l}{m}}\right)$, and where $\overline{r}_\infty = \mathbb{E}_{S \sim \mathcal{D}^m}[r_\infty]$.

The learning bound of Corollary 1 is a finer guarantee than previous ones by Bartlett (1998), Neyshabur et al. (2015), or Sun et al. (2016). This is because it explicitly differentiates between the weights of different layers while previous bounds treat all weights indiscriminately. This is crucial to the design of algorithmic design since the network complexity no longer needs to grow exponentially as a function of depth. Our bounds are also more general and apply to more other network architectures, such as those introduced in (He et al., 2015; Huang et al., 2016).

5. Algorithm

This section describes our algorithm, ADANET, for *adaptive* learning of neural networks. ADANET adaptively grows the structure of a neural network, balancing model complexity with empirical risk minimization. We also describe in detail in Appendix C another variant of ADANET which admits some favorable properties.

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

5.1. Objective function

Let $\{h_1, \ldots, h_N\}$ be a subset of $\mathcal{H}^*$. In the most general case, N is infinite. However, as discussed later, in practice, the search is limited to a finite set. For any $j \in [N]$, we will denote by r_j the Rademacher complexity of the family $\mathcal{H}_{k_j}$ that contains h_j: $r_j = \mathfrak{R}_m(\mathcal{H}_{k_j})$.

ADANET seeks to find a function $f = \sum_{j=1}^N w_j h_j \in \mathcal{F}^*$ (or neural network) that directly minimizes the data-dependent generalization bound of Corollary 1. This leads to the following objective function:

$$F(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m \Phi\Big(1 - y_i \sum_{j=1}^N w_j h_j\Big) + \sum_{j=1}^N \Gamma_j |w_j|, \quad (4)$$

where $\mathbf{w} \in \mathbb{R}^N$ and $\Gamma_j = \lambda r_j + \beta$, with $\lambda \geq 0$ and $\beta \geq 0$ hyperparameters. The objective function (4) is a convex function of $\mathbf{w}$. It is the sum of a convex surrogate of the empirical error and a regularization term, which is a weighted-l_1 penalty containing two sub-terms: a standard norm-1 regularization which admits β as a hyperparameter, and a term that discriminates the functions h_j based on their complexity.

The optimization problem consisting of minimizing the objective function F in (4) is defined over a very large space of base functions h_j. ADANET consists of applying coordinate descent to (4). In that sense, our algorithm is similar to the DeepBoost algorithm of Cortes et al. (2014). However, unlike DeepBoost, which combines decision trees, ADANET learns a deep neural network, which requires new methods for constructing and searching the space of func-

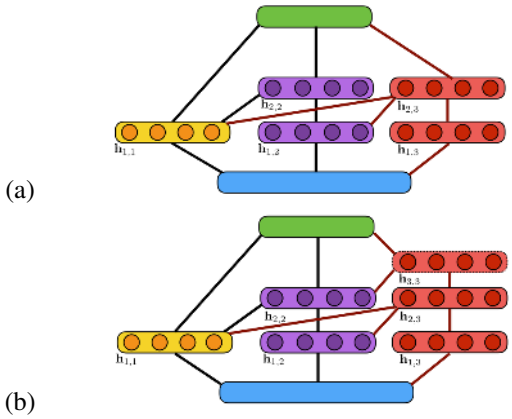

Figure 2. Illustration of the algorithm's incremental construction of a neural network. The input layer is indicated in blue, the output layer in green. Units in the yellow block are added at the first iteration while units in purple are added at the second iteration. Two candidate extensions of the architecture are considered at the the third iteration (shown in red): (a) a two-layer extension; (b) a three-layer extension. Here, a line between two blocks of units indicates that these blocks are fully-connected.

tions h_j. Both of these aspects differ significantly from the decision tree framework. In particular, the search is particularly challenging. In fact, the main difference between the algorithm presented in this section and the variant described in Appendix C is the way new candidates h_j are examined at each iteration.

5.2. Description

We start with an informal description of ADANET. Let $B \geq 1$ be a fixed parameter determining the number of units per layer of a candidate subnetwork. The algorithm proceeds in T iterations. Let l_{t-1} denote the depth of the neural network constructed before the start of the t-th iteration. At iteration t, the algorithm selects one of the following two options:

1. augmenting the current neural network with a subnetwork with the same depth as that of the current network $\mathbf{h} \in \mathcal{H}^{*B}_{l_{t-1}}$, with B units per layer. Each unit in layer k of this subnetwork may have connections to existing units in layer $k-1$ of ADANET in addition to connections to units in layer $k-1$ of the subnetwork.

2. augmenting the current neural network with a deeper subnetwork $\mathbf{h}' \in \mathcal{H}^{*B}_{l_{t-1}}$, with depth $l_{t-1}+1$. The set of connections allowed is defined in the same way as for $\mathbf{h}$.

The option selected is the one leading to the best reduction of the current value of the objective function, which depends both on the empirical error and the complexity of the subnetwork added, which is penalized differently in these two options.

Figure 2 illustrates this construction and the two options just described. An important aspect of our algorithm is that the units of a subnetwork learned at a previous iteration (say $\mathbf{h}_{1,1}$ in Figure 2) can serve as input to a deeper subnetwork added later (for example $\mathbf{h}_{2,2}$ or $\mathbf{h}_{2,3}$ in the Figure). Thus, the deeper subnetworks added later can take advantage of the embeddings that were learned at the previous iterations. The algorithm terminates after T rounds or if the ADANET architecture can no longer be extended to improve the objective (4).

More formally, ADANET is a boosting-style algorithm that applies (block) coordinate descent to (4). At each iteration of block coordinate descent, descent coordinates $\mathbf{h}$ (base learners in the boosting literature) are selected from the space of functions $\mathcal{H}^*$. These coordinates correspond to the direction of the largest decrease in (4). Once these coordinates are determined, an optimal step size in each of these directions is chosen, which is accomplished by solving an appropriate convex optimization problem.

Note that, in general, the search for the optimal descent coordinate in an infinite-dimensional space or even in finite but large sets such as that of all decision trees of some large depth may be intractable, and it is common to resort to a heuristic search (weak learning algorithm) that returns δ-optimal coordinates. For instance, in the case of boosting with trees one often grows trees according to some particular heuristic (Freund & Schapire, 1997).

We denote the ADANET model after $t-1$ rounds by f_{t-1}, which is parameterized by $\mathbf{w}_{t-1}$. Let $\mathbf{h}_{k,t-1}$ denote the vector of outputs of units in the k-th layer of the ADANET model, l_{t-1} be the depth of the ADANET architecture, $n_{k,t-1}$ be the number of units in k-th layer after $t-1$ rounds. At round t, we select descent coordinates by considering two candidate subnetworks $\mathbf{h} \in \widetilde{\mathcal{H}}^*_{l_{t-1}}$ and $\mathbf{h}' \in \widetilde{\mathcal{H}}^*_{l_{t-1}+1}$ that are generated by a weak learning algorithm WEAKLEARNER. Some choices for this algorithm in our setting are described below. Once we obtain $\mathbf{h}$ and $\mathbf{h}'$, we select one of these vectors of units, as well as a vector of weights $\mathbf{w} \in \mathbb{R}^B$, so that the result yields the best improvement in (4). This is equivalent to minimizing the following objective function over $\mathbf{w} \in \mathbb{R}^B$ and $\mathbf{u} \in \{\mathbf{h}, \mathbf{h}'\}$:

$$F_t(\mathbf{w}, \mathbf{u}) = \frac{1}{m} \sum_{i=1}^{m} \Phi\Big(1 - y_i f_{t-1}(x_i) - y_i \mathbf{w} \cdot \mathbf{u}(x_i)\Big) + \Gamma_{\mathbf{u}} \|\mathbf{w}\|_1, \quad (5)$$

where $\Gamma_{\mathbf{u}} = \lambda r_{\mathbf{u}} + \beta$ and $r_{\mathbf{u}}$ is $\mathfrak{R}_m\big(\mathcal{H}_{l_{t-1}}\big)$ if $\mathbf{u} = \mathbf{h}$ and $\mathfrak{R}_m\big(\mathcal{H}_{l_{t-1}+1}\big)$ otherwise. In other words, if $\min_{\mathbf{w}} F_t(\mathbf{w}, \mathbf{h}) \leq \min_{\mathbf{w}} F_t(\mathbf{w}, \mathbf{h}')$, then

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^B} F_t(\mathbf{w}, \mathbf{h}), \quad \mathbf{h}_t = \mathbf{h}$$

ADANET$(S = ((x_i, y_i)_{i=1}^m)$
1 $f_0 \leftarrow 0$
2 **for** $t \leftarrow 1$ **to** T **do**
3 $\quad \mathbf{h}, \mathbf{h}' \leftarrow$ WEAKLEARNER$\big(S, f_{t-1}\big)$
4 $\quad \mathbf{w} \leftarrow$ MINIMIZE$\big(F_t(\mathbf{w}, \mathbf{h})\big)$
5 $\quad \mathbf{w}' \leftarrow$ MINIMIZE$\big(F_t(\mathbf{w}, \mathbf{h}')\big)$
6 $\quad$ **if** $F_t(\mathbf{w}, \mathbf{h}') \leq F_t(\mathbf{w}', \mathbf{h}')$ **then**
7 $\quad\quad \mathbf{h}_t \leftarrow \mathbf{h}$
8 $\quad$ **else** $\mathbf{h}_t \leftarrow \mathbf{h}'$
9 $\quad$ **if** $F(\mathbf{w}_{t-1} + \mathbf{w}^*) < F(\mathbf{w}_{t-1})$ **then**
10 $\quad\quad f_t \leftarrow f_{t-1} + \mathbf{w}^* \cdot \mathbf{h}_t$
11 $\quad$ **else return** f_{t-1}
12 **return** f_T

Figure 3. Pseudocode of the ADANET algorithm. On line 3 two candidate subnetworks are generated (e.g. randomly or by solving (6)). On lines 3 and 4, (5) is solved for each of these candidates. On lines 5-7 the best subnetwork is selected and on lines 9-11 termination condition is checked.

and otherwise

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^B} F_t(\mathbf{w}, \mathbf{h}'), \quad \mathbf{h}_t = \mathbf{h}'$$

If $F(\mathbf{w}_{t-1} + \mathbf{w}^*) < F(\mathbf{w}_{t-1})$ then we set $f_t = f_{t-1} + \mathbf{w}^* \cdot \mathbf{h}_t$ and otherwise we terminate the algorithm.

There are many different choices for the WEAKLEARNER algorithm. For instance, one may generate a large number of random networks and select the one that optimizes (5). Another option is to directly minimize (5) or its regularized version:

$$\widetilde{F}_t(\mathbf{w}, \mathbf{h}) = \frac{1}{m} \sum_{i=1}^{m} \Phi\Big(1 - y_i f_{t-1}(x_i) - y_i \mathbf{w} \cdot \mathbf{h}(x_i)\Big) + \mathcal{R}(\mathbf{w}, \mathbf{h}), \quad (6)$$

over both $\mathbf{w}$ and $\mathbf{h}$. Here $\mathcal{R}(\mathbf{w}, \mathbf{h})$ is a regularization term that, for instance, can be used to enforce that $\|\mathbf{u}_s\|_p \leq \Lambda_{k,s}$ in (2). Note that, in general, (6) is a non-convex objective. However, we do not rely on finding a global solution to the corresponding optimization problem. In fact, standard guarantees for regularized boosting only require that each $\mathbf{h}$ that is added to the model decreases the objective by a constant amount (i.e. it satisfies δ-optimality condition) for a boosting algorithm to converge (Rätsch et al., 2001; Luo & Tseng, 1992).

Furthermore, the algorithm that we present in Appendix C uses a weak-learning algorithm that solves a convex subproblem at each step and that additionally has a closed-form solution. This comes at the cost of a more restricted search space for finding a descent coordinate at each step of the algorithm.

We conclude this section by observing that in our description of ADANET we have fixed B for all iterations and only two candidate subnetworks are considered at each step. Our approach easily extends to an arbitrary number of candidate subnetworks (for instance of different depth l) as well as varying number of units per layer B. Furthermore, selecting an optimal subnetwork among the candidates is easily parallelizable allowing for efficient and effective search for optimal descent directions. We also note that the choice of subnetworks need not be restricted to standard feedforward architectures and more exotic choices can be employed including the ones in (He et al., 2015; Huang et al., 2016). In our experiments we will restrict attention to simple feedforward subnetworks.

6. Experiments

In this section we present the results of our experiments with ADANET. Some additional experimental results are given in Appendix D and further implementation details presented in Appendix E.

6.1. CIFAR-10

In our first set of experiments, we used the `CIFAR-10` dataset (Krizhevsky, 2009). This dataset consists of 60,000 images evenly categorized in 10 different classes. To reduce the problem to binary classification, we considered five pairs of classes: `deer-truck`, `deer-horse`, `automobile-truck`, `cat-dog`, `dog-horse`. Raw images have been pre-processed to obtain color histograms and histogram of gradient features. The result is 154 real valued features with ranges in $[0, 1]$.

We compared ADANET to standard feedforward neural networks (NN) and logistic regression (LR) models. Note that convolutional neural networks are often a more natural choice for image classification problems such as `CIFAR-10`. However, the goal of our experiments was not to obtain state-of-the-art results for this particular task, but a proof-of-concept showing that our structural learning approach can be very competitive with traditional approaches for finding efficient architectures and training corresponding networks.

Our ADANET algorithm requires the knowledge of complexities r_j, which, in some cases, can be estimated from data. In our experiments, we used the upper bound of Lemma 2. Our algorithm admits a number of hyperparameters: regularization hyperparameters λ, β, number of units B in each layer of new subnetworks that are used to extend the model at each iteration, and a bound Λ_k on weights $\mathbf{u}$ in each unit. As discussed in Section 5, there are different approaches to finding candidate subnetworks in each iteration. In our experiments, we searched for candidate subnet-

Table 1. Experimental results for ADANET, NN, LR and NN-GP for different pairs of labels in `CIFAR-10`. Boldfaced results are statistically significant at a 5% confidence level.

Label pair	ADANET	LR	NN	NN-GP
`deer-truck`	**0.9372 ± 0.0082**	0.8997 ± 0.0066	0.9213 ± 0.0065	0.9220 ± 0.0069
`deer-horse`	**0.8430 ± 0.0076**	0.7685 ± 0.0119	0.8055 ± 0.0178	0.8060 ± 0.0181
`automobile-truck`	**0.8461 ± 0.0069**	0.7976 ± 0.0076	0.8063 ± 0.0064	0.8056 ± 0.0138
`cat-dog`	**0.6924 ± 0.0129**	0.6664 ± 0.0099	0.6595 ± 0.0141	0.6607 ± 0.0097
`dog-horse`	**0.8350 ± 0.0089**	0.7968 ± 0.0128	0.8066 ± 0.0087	0.8087 ± 0.0109

works by minimizing (6) with $\mathcal{R} = 0$. This also requires a learning rate hyperparameter η. These hyperparamers have been optimized over the following ranges: $\lambda \in \{0, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$, $B \in \{100, 150, 250\}$, $\eta \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. We have used a single Λ_k for all $k > 1$ optimized over $\{1.0, 1.005, 1.01, 1.1, 1.2\}$. For simplicity, we chose $\beta = 0$.

Neural network models also admit a learning rate η and a regularization coefficient λ as hyperparameters, as well as the number of hidden layers l and the number of units n in each hidden layer. The range of η was the same as for ADANET and we varied l in $\{1, 2, 3\}$, n in $\{100, 150, 512, 1024, 2048\}$ and $\lambda \in \{0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. Logistic regression only admits as hyperparameters η and λ which were optimized over the same ranges. Note that the total number of hyperparameter settings for ADANET and standard neural networks is exactly the same. Furthermore, the same holds for the number of hyperparameters that determine the resulting architecture of the model: Λ and B for ADANET and l and n for neural network models. Observe that, while a particular setting of l and n determines a fixed architecture, Λ and B parameterize a structural learning procedure that may result in a different architecture depending on the data.

In addition to the grid search procedure, we have conducted a hyperparameter optimization for neural networks using Gaussian process bandits (NN-GP), which is a sophisticated Bayesian non-parametric method for response-surface modeling in conjunction with a bandit algorithm (Snoek et al., 2012). Instead of operating on a pre-specified grid, this allows one to search for hyperparameters in a given range. We used the following ranges: $\lambda \in [10^{-5}, 1]$, $\eta \in [10^{-5}, 1]$, $l \in [1, 3]$ and $n \in [100, 2048]$. This algorithm was run for 500 trials, which is more than the number of hyperparameter settings considered by ADANET and NN. Observe that this search procedure can also be applied to our algorithm but we chose not to use it in this set of experiments to further demonstrate competitiveness of our structural learning approach.

In all our experiments, we use ReLu as the activation functions. NN, NN-GP and LR are trained using stochastic gradient method with batch size of 100 and maximum of 10,000 iterations. The same configuration is used for solving (6). We use $T = 30$ for ADANET in all our experiments although in most cases algorithm terminates after 10 rounds.

In each of the experiments, we used standard 10-fold cross-validation for performance evaluation and model selection. In particular, the 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 $i \in \{0, \ldots, 9\}$, fold i was used for testing, fold $i + 1 \pmod{10}$ 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 maximum average accuracy across validation folds. We report the average accuracy (and standard deviations) of the selected hyperparameter setting across test folds in Table 1.

Our results show that ADANET outperforms other methods on each of the datasets. The average architectures for all label pairs are provided in Table 2. Note that NN and NN-GP always select a one-layer architecture. The architectures selected by ADANET also typically admit a single layer, with fewer nodes than those selected by NN and NN-GP. However, for the more challenging problem `cat-dog`, ADANET opts for a more complex model with two layers, which results in a better performance. This further illustrates how our approach helps learn network architectures in an adaptive fashion, based on the complexity of the task.

As discussed in Section 5, different heuristics can be used to generate candidate subnetworks on each iteration of ADANET. In a second set of experiments, we varied the objective function (6), as well as the domain over which it is optimized. This allowed us to study the sensitivity of ADANET to the choice of a heuristic used to generate candidate subnetworks. In particular, we considered the following variants of ADANET. ADANET.R uses $\mathcal{R}(\mathbf{w}, \mathbf{h}) = \Gamma_{\mathbf{h}} \|w\|_1$ as a regularization term in (6). As

Table 2. Average number of units in each layer.

Label pair	ADANET		NN	NN-GP
	1st layer	2nd layer		
deer-truck	990	0	2048	1050
deer-horse	1475	0	2048	488
automobile-truck	2000	0	2048	1595
cat-dog	1800	25	512	155
dog-horse	1600	0	2048	1273

Table 3. Experimental results for different variants of ADANET, for the deer-truck label pair in CIFAR-10.

Algorithm	Accuracy ($\pm$ std. dev.)
ADANET.SD	0.9309 ± 0.0069
ADANET.R	0.9336 ± 0.0075
ADANET.P	0.9321 ± 0.0065
ADANET.D	0.9376 ± 0.0080

the ADANET architecture grows, each new subnetwork is connected to all the previous subnetworks, which significantly increases the number of connections in the network and the overall complexity of the model. ADANET.P and ADANET.D are restricting connections to existing subnetworks in different ways. ADANET.P connects each new subnetwork only to the subnetwork that was added on the previous iteration. ADANET.D uses dropout on the connections to previously added subnetworks. Finally, while ADANET is based on the upper bounds on the Rademacher complexities of Lemma 2, ADANET.SD uses instead standard deviations of the outputs of the last hidden layer on the training data as surrogates for Rademacher complexities. The advantage of using this data-dependent measure of complexity is that it eliminates the hyperparameter Λ, thereby reducing the hyperparameter search space. We report the average accuracies across test folds for the deer-truck pair in Table 3.

6.2. Criteo Click Rate Prediction

We also compared ADANET to NN on the Criteo Click Rate Prediction dataset (https://www.kaggle.com/c/criteo-display-ad-challenge). This dataset consists of 7 days of data where each instance is an impression and a binary label (clicked or not clicked). Each impression admits 13 count features and 26 categorical features. Count features have been transformed by taking the natural logarithm. The values of categorical features appearing less than 100 times are replaced by 0. The rest of the values are then converted to integers, which are then used as keys to look up embeddings (that are trained together with each model). If the number of possible values for a feature x is $d(x)$, then the embedding dimension is set to $6d(f)^{1/4}$ for $d(f) > 25$. Otherwise, the embedding dimension is $d(f)$. Missing feature values are set to 0. We split the labeled set provided in the link above into training, validation and test sets.[1] Our training set covered the first 5 days of data (32,743,299 instances) and the validation and test sets consisted of 1 day (6,548,659 instances). Gaussian processes bandits were used to find the best hyperparameter settings on validation set both for ADANET and NN. For ADANET we optimized over the following hyperparameter ranges: $B \in \{125, 256, 512\}$, $\Lambda \in [1, 1.5]$, $\eta \in [10^{-4}, 10^{-1}]$, $\lambda \in [10^{-12}, 10^{-4}]$. For NN the ranges were as follows: $l \in [1, 6]$, $n \in \{250, 512, 1024, 2048\}$, $\eta \in [10^{-5}, 10^{-1}]$, $\lambda \in [10^{-6}, 10^{-1}]$. We trained NNs for 100,000 iterations using mini-batch stochastic gradient method with batch size of 512. The same configuration was used at each iteration of ADANET to solve (6). The maximum number of hyperparameter trials was 2,000 for both methods. The results are presented in Table 4. In this experiment, NN chooses an architecture with four hidden layers and 512 units in each hidden layer. Remarkably, ADANET achieves a better accuracy with an architecture consisting of single layer with just 512 nodes. While the difference in performance appears to be small, it is in fact statistically significant in this challenging task.

Table 4. Experimental results for Criteo dataset.

Algorithm	Accuracy
ADANET	0.7846
NN	0.7811

7. Conclusion

We presented a new framework and algorithms for adaptively learning artificial neural networks. Our algorithm, ADANET, benefits from strong theoretical guarantees. It simultaneously learns a neural network architecture and its parameters, by balancing a trade-off between model complexity and empirical risk minimization. We reported favorable experimental results demonstrating that our algorithm is able to learn network architectures that perform better than those found via a grid search. Our techniques are general and can be applied to other neural network architectures such as CNNs and RNNs.

Acknowledgments

The work of M. Mohri and that of S. Yang were partly funded by NSF awards IIS-1117591 and CCF-1535987.

[1] The test set available from this link does not include ground truth labels and therefore could be used in our experiments.

References

Alvarez, Jose M and Salzmann, Mathieu. Learning the number of neurons in deep networks. In *NIPS*, 2016.

Arora, Sanjeev, Bhaskara, Aditya, Ge, Rong, and Ma, Tengyu. Provable bounds for learning some deep representations. In *ICML*, pp. 584–592, 2014.

Arora, Sanjeev, Liang, Yingyu, and Ma, Tengyu. Why are deep nets reversible: A simple theory, with implications for training. *arXiv:1511.05653*, 2015.

Baker, Bowen, Gupta, Otkrist, Naik, Nikhil, and Raskar, Ramesh. Designing neural network architectures using reinforcement learning. *CoRR*, 2016.

Bartlett, Peter L. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *Information Theory, IEEE Transactions on*, 44(2), 1998.

Bartlett, Peter L. and Mendelson, Shahar. Rademacher and Gaussian complexities: Risk bounds and structural results. *JMLR*, 3, 2002.

Bergstra, James S, Bardenet, Rémi, Bengio, Yoshua, and Kégl, Balázs. Algorithms for hyper-parameter optimization. In *NIPS*, pp. 2546–2554, 2011.

Chen, Tianqi, Goodfellow, Ian J., and Shlens, Jonathon. Net2net: Accelerating learning via knowledge transfer. *CoRR*, 2015.

Choromanska, Anna, Henaff, Mikael, Mathieu, Michael, Arous, Gérard Ben, and LeCun, Yann. The loss surfaces of multilayer networks. *arXiv:1412.0233*, 2014.

Cohen, Nadav, Sharir, Or, and Shashua, Amnon. On the expressive power of deep learning: a tensor analysis. *arXiv*, 2015.

Cortes, Corinna, Mohri, Mehryar, and Syed, Umar. Deep boosting. In *ICML*, pp. 1179 – 1187, 2014.

Daniely, Amit, Frostig, Roy, and Singer, Yoram. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *NIPS*, 2016.

Eldan, Ronen and Shamir, Ohad. The power of depth for feedforward neural networks. *arXiv:1512.03965*, 2015.

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

Ha, David, Dai, Andrew M., and Le, Quoc V. Hypernetworks. *CoRR*, 2016.

Han, Hong-Gui and Qiao, Jun-Fei. A structure optimisation algorithm for feedforward neural network construction. *Neurocomputing*, 99:347–357, 2013.

Han, Song, Pool, Jeff, Tran, John, and Dally, William J. Learning both weights and connections for efficient neural networks. In *NIPS*, 2015.

Hardt, Moritz, Recht, Benjamin, and Singer, Yoram. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv:1509.01240*, 2015.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

Huang, Gao, Liu, Zhuang, and Weinberger, Kilian Q. Densely connected convolutional networks. *CoRR*, 2016.

Islam, Md. Monirul, Yao, Xin, and Murase, Kazuyuki. A constructive algorithm for training cooperative neural network ensembles. *IEEE Trans. Neural Networks*, 14 (4):820–834, 2003.

Islam, Md. Monirul, Sattar, Md. Abdus, Amin, Md. Faijul, Yao, Xin, and Murase, Kazuyuki. A new adaptive merging and growing algorithm for designing artificial neural networks. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 39(3):705–722, 2009.

Janzamin, Majid, Sedghi, Hanie, and Anandkumar, Anima. Generalization bounds for neural networks through tensor factorization. *arXiv:1506.08473*, 2015.

Kawaguchi, Kenji. Deep learning without poor local minima. In *NIPS*, 2016.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Koltchinskii, Vladmir and Panchenko, Dmitry. Empirical margin distributions and bounding the generalization error of combined classifiers. *Annals of Statistics*, 30, 2002.

Kotani, Manabu, Kajiki, Akihiro, and Akazawa, Kenzo. A structural learning algorithm for multi-layered neural networks. In *International Conference on Neural Networks*, volume 2, pp. 1105–1110. IEEE, 1997.

Krizhevsky, Alex. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *NIPS*, pp. 1097–1105, 2012.

Kuznetsov, Vitaly, Mohri, Mehryar, and Syed, Umar. Multi-class deep boosting. In *NIPS*, 2014.

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

LeCun, Yann, Denker, John S., and Solla, Sara A. Optimal brain damage. In *NIPS*, 1990.

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

Leung, Frank HF, Lam, Hak-Keung, Ling, Sai-Ho, and Tam, Peter KS. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural Networks*, 14(1): 79–88, 2003.

Lian, Xiangru, Huang, Yijun, Li, Yuncheng, and Liu, Ji. Asynchronous parallel stochastic gradient for nonconvex optimization. In *NIPS*, pp. 2719–2727, 2015.

Livni, Roi, Shalev-Shwartz, Shai, and Shamir, Ohad. On the computational efficiency of training neural networks. In *NIPS*, pp. 855–863, 2014.

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

Ma, Liying and Khorasani, Khashayar. A new strategy for adaptively constructing multilayer feedforward neural networks. *Neurocomputing*, 51:361–385, 2003.

Narasimha, Pramod L, Delashmit, Walter H, Manry, Michael T, Li, Jiang, and Maldonado, Francisco. An integrated growing-pruning method for feedforward network training. *Neurocomputing*, 71(13):2831–2847, 2008.

Neyshabur, Behnam, Tomioka, Ryota, and Srebro, Nathan. Norm-based capacity control in neural networks. In *COLT*, 2015.

Rätsch, Gunnar, Mika, Sebastian, and Warmuth, Manfred K. On the convergence of leveraging. In *NIPS*, pp. 487–494, 2001.

Sagun, Levent, Guney, V Ugur, Arous, Gerard Ben, and LeCun, Yann. Explorations on high dimensional landscapes. *arXiv:1412.6615*, 2014.

Saxena, Shreyas and Verbeek, Jakob. Convolutional neural fabrics. *CoRR*, abs/1606.02492, 2016.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical Bayesian Optimization of Machine Learning Algorithms. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *NIPS*, pp. 2951–2959. Curran Associates, Inc., 2012.

Sun, Shizhao, Chen, Wei, Wang, Liwei, Liu, Xiaoguang, and Liu, Tie-Yan. On the depth of deep neural networks: A theoretical view. In *AAAI*, 2016.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott E., Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *CVPR*, 2015.

Telgarsky, Matus. Benefits of depth in neural networks. In *COLT*, 2016.

Zhang, Saizheng, Wu, Yuhuai, Che, Tong, Lin, Zhouhan, Memisevic, Roland, Salakhutdinov, Ruslan, and Bengio, Yoshua. Architectural complexity measures of recurrent neural networks. *CoRR*, 2016.

Zhang, Yuchen, Lee, Jason D, and Jordan, Michael I. ℓ_1-regularized neural networks are improperly learnable in polynomial time. *arXiv:1510.03528*, 2015.

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. *CoRR*, 2016.