Self-born Wiring for Neural Trees

Ying Chen[1], Feng Mao[2], Jie Song[1], Xinchao Wang[3], Huiqiong Wang[4*], and Mingli Song[1]
[1]Zhejiang University, [2]Alibaba Group, [3]National University of Singapore
[4]Ningbo Research Institute, Zhejiang University
{lynesychen,sjie,huiqiong_wang,brooksong}@zju.edu.cn,
maofeng.mf@alibaba-inc.com, xinchao@nus.edu.sg

Abstract

Neural trees aim at integrating deep neural networks and decision trees so as to bring the best of the two worlds, including representation learning from the former and faster inference from the latter. In this paper, we introduce a novel approach, termed as Self-born Wiring (SeBoW), to learn neural trees from a mother deep neural network. In contrast to prior neural-tree approaches that either adopt a pre-defined structure or grow hierarchical layers in a progressive manner, task-adaptive neural trees in SeBoW evolve from a deep neural network through a construction-by-destruction process, enabling a global-level parameter optimization that further yields favorable results. Specifically, given a designated network configuration like VGG, SeBoW disconnects all the layers and derives isolated filter groups, based on which a global-level wiring process is conducted to attach a subset of filter groups, eventually bearing a lightweight neural tree. Extensive experiments demonstrate that, with a lower computational cost, SeBoW outperforms all prior neural trees by a significant margin and even achieves results on par with predominant non-tree networks like ResNets. Moreover, SeBoW proves its scalability to large-scale datasets like ImageNet, which has been barely explored by prior tree networks.

1. Introduction

Deep Neural Networks (DNNs) [33, 11] have been arguably the most successful machine learning models in the last decade, dominating a large spectrum of applications such as computer vision and natural language processing. The unprecedented success is largely attributed to its hierarchical representation learning through the composition of nonlinear transformations, which alleviates the need for feature engineering. Nevertheless, DNNs are not flawless: they typically suffer from the expensive computation cost, the daunting architecture design process, as well the lack of interpretation, which hinders their more widespread applications.

Decision Trees (DTs), as an alternative category of machine learning models, have also demonstrated their power in real-world applications [9, 31]. Unlike DNNs where hierarchical representations with varying abstraction are learned, DTs are characterized by learning hierarchical clusters of data, so that in each cluster a linear model suffices to explain the data. As DTs conduct classification through a relatively short root-to-leaf sequence, they usually yield faster inference, in which the decision process is also task-adaptive. Moreover, the decisions in DTs are often made directly in the original feature space, resulting in better model interpretability. Albeit these appealing properties, DTs often require hand-engineering features; the capacity of a single DT is also limited due to the simple routing functions along the root-to-leaf path. Such attributes, unfortunately, substantially precludes DTs' applications to more complex real-world scenarios.

In light of the mutual exclusive benefits and limitations of DNNs and DTs, it is therefore desirable to integrate DNNs and DTs into a single model, termed as *neural trees*, in the hope that the complementary merits of both worlds are preserved. Several pilot endeavours have been made towards this ambitious goal. For example, Frosst and Hinton adopted a soft decision tree to imitate the output of a neural network for better interpretability [8]. However, as every decision is made in the original input space and no representation learning is performed, better interpretability is achieved by sacrificing the performance. Kontschieder *et al*. proposed *Deep Neural Decision Forest* (DNDF) [19], an ensemble of DTs, in which each DT is built upon Inception V1 [35]. Despite the promising accuracy, the cumbersome backbone and the separate optimization pipeline for network parameters and prediction distribution render both the training and the inference highly costly. The pre-defined model structure also makes it not adaptive to tasks. More recently, Tanno *et al*. proposed *Adaptive Neu-*

*Corresponding author

ral Trees (ANT) [36] that inherit representation learning from DNNs and lightweight inference from DTs. Unfortunately, ANT relies on a suboptimal progressive scheme to grow trees layer by layer, in which each operation is selected among several pre-defined ones in a greedy manner, making it prone to local optima and hardly scalable to large datasets like ImageNet [5].

In this paper, we propose a novel approach to learning neural trees from DNNs, termed as *Self-born Wiring* (SeBoW), through which the merits of both categories are naturally united. In contrast to existing methods that either relies on growing trees progressively or restricted to a specific tree structure, our task-customized neural trees evolve themselves from a user-designated mother DNN architecture like VGG, via a construction-by-destruction manner. Such a self-born learning procedure, in turn, allows for a global-level parameter optimization over the neural tree search. Neural trees derived in this way not only enjoy learnable hierarchical representations and efficient inference, but also yield results significantly better than those from prior neural-tree methods and even on par with those of DNN-based ones, thanks to the global tree-architecture optimization.

Specifically, SeBoW starts the learning process with destructing a user-designated network configuration like VGG [33], by removing all the connections across layers and turning the network into a collection of isolated layers. The neurons or filters in every layer, except the first layer that acts as the root, are further decomposed into several equal-sized filter groups, each of which has only a subset of truncated filters and acts as a feature learner. We then adopt routers to conduct wiring over these scattered filter groups so as to navigate the adaptive decision paths across different layers. In the last layer, we install a solver at the end of each filter group to make the final predictions.

The learners, the routers, and the solvers constitute the complete search space of SeBoW. Unlike prior neural trees where intermediate nodes are optimized greedily and separately, SeBoW runs over all the possible decision paths in the search space and adopts the widely used *Negative Log Likelihood* (NLL) loss to train the whole model, making the training as simple as training popular DNNs. Interestingly, our router design effectively imposes strong sparsity constraints on the wiring between groups across layers, resulting in only a few active modules with the rest silent. Finally, a lightweight decision tree can be extracted by safely removing those inactive modules.

Extensive experiments over several datasets including ImageNet have been conducted, in which SeBoW attains higher accuracy with lower computation cost compared to existing tree-based models. More encouragingly, SeBoW even achieves on par or sometimes superior performance when compared to non-tree networks, let alone its faster inference and better interpretability due to its endowed decision-tree nature.

In sum, our contribution is a novel neural-tree learning scheme, termed as SeBoW, that bears a task-customized neural tree from a mother DNN, enabling the merits from both DNNs and neural trees to be by nature preserved. The construction-by-destruction learning process allows for a global-level tree architecture search, which consequently gives rise to a competent neural tree with encouraging performance. Experiments demonstrate that the derived trees yield performances significantly superior to prior tree methods and even on par with results obtained from popular DNNs on large-scale datasets such as ImageNet.

2. Self-born Wiring

In this section, we depict the proposed method, SeBoW, to craft task-adaptive neural trees. The training data is denoted by $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\}$, $\mathbf{x} \in \mathcal{X}$, $y \in \mathcal{Y}$. The goal is to learn the conditional distribution $p(y|\mathbf{x})$. We first introduce the model topology and operations used for neural trees. Then we describe how to design the search space for neural trees based on prevailing DNN architectures. Finally, we describe how the tree architecture and the parameters are simultaneously optimized.

2.1. Model Topology and Operations

We define the model topology for neural trees as a pair $(\mathbb{T}, \mathbb{O})$, where $\mathbb{T}$ defines the model topology, and $\mathbb{O}$ denotes the set of operations on it. In this paper, we adopt a *Directed Acyclic Graph* (DAG) to represent the model topology, *i.e.*, $\mathbb{T} := \{\mathcal{N}, \mathcal{E}\}$, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges between them, $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$. The internal nodes are denoted by $\mathcal{N}_{int}$, and leaf nodes by $\mathcal{N}_{leaf}$. $\mathbb{T}$ consists of three primitive operations, *learning*, *routing*, and *solving*. Each node is assigned with operations in $\mathbb{O}$, which turns the DAG into a decision tree. These operations are implemented by the following modules:

- **Learners**: every internal node $i \in \mathcal{N}_{int}$ of the tree is assigned with a learner $l_i^{\boldsymbol{\psi}}$, parameterized by $\boldsymbol{\psi}$, that transforms data from the parent to its children. The learners are the key modules for representation learning, and they are implemented by stacking some widely used layers (detailed in Section 2.2), including convolution, ReLU, batch normalization, and pooling layers.

- **Routers**: following each learner in internal node $i \in \mathcal{N}_{int}$, a router module, $r_i^{\boldsymbol{\theta}} := \mathcal{X}_i \rightarrow [0, 1]^{N_i}$, parameterized by $\boldsymbol{\theta}$, is appended to send incoming data to its children nodes. Here $\mathcal{X}_i$ denotes the input data space for router $r_i^{\boldsymbol{\theta}}$. We make no restriction on the number of children here, which means the decision tree is not necessarily a binary tree, *i.e.*, $N_i \geq 2$. This relaxes the assumptions made in previous tree-based models.

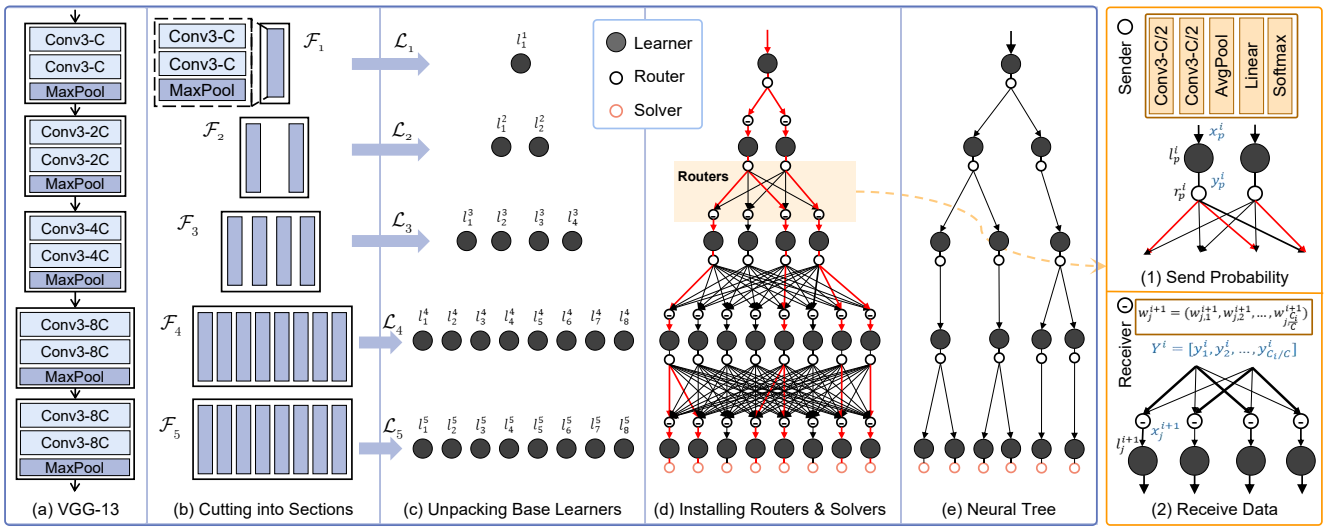


Figure 1. An illustrative example of the proposed self-born wiring pipeline. *Left*: (a) VGG-13 is utilized as mother DNN to design search space. (b) Cutting the DNN into 5 sections. (c) Unpacking base learners from sections. (d) Installing a router for each internal learner, a solver for each leaf learner, then the complete search space is obtained. The red path is selected according to Sect. 2.3.2. (e) One neural tree chosen from (d). *Right*: Two essential components for routers. (1) Senders explore probability distributions. (2) Receivers fuse various features. "Conv3-C" denotes 3×3 convolution with C filters. The formulas in blue represent data, while others donate network modules.

- **Solvers**: each leaf $i \in \mathcal{N}_{leaf}$ is assigned a solver module, $s_i^{\phi} : \mathcal{X}_i \to \mathcal{Y}$, parameterized by ϕ, which operates on the transformed data and outputs the estimate for the conditional distribution $p(y|\mathbf{x})$. In this paper, we focus on classification tasks, and thus implement the solver by a fully connected layer and a softmax layer.

Note that unlike ANT [36], we do not assign any operation to edges. They are only used for data flow. The model topology and the defined operations sketch a broad view of a neural tree. In the following sections, we introduce how a task-adaptive neural tree is generated by SeBoW.

2.2. Search Space for Neural Trees

To design the search space for neural trees, we use the network configuration from a mother DNN like VGG [33] as the starting point, as illustrated in Figure 1 (a). This moderately relieves us of the burden of designing the search space. For example, we need not consider how many times the data should be down-pooled along the decision route in the neural tree. The pipeline of designing space is depicted in Figure 1. Then for simplicity, we describe the work of designing the tree space as a three-step process: *unpacking base learners*, *installing routers* and *mounting solvers*.

2.2.1 Unpacking Base Learners

We unpack base learners, which will be used as learners in neural trees, from a widely used DNN. To this end, we first cut the DNN into several sections, each with a pooling layer inside. Formally, let $\mathcal{F}$ denotes the function underlying a DNN, then assume the DNN is cut into S sections. The function underlying the i-th fragment is symbolized by $\mathcal{F}_i$. Then $\mathcal{F}$ can be written as $\mathcal{F} = \mathcal{F}_S \circ ... \circ \mathcal{F}_2 \circ \mathcal{F}_1$, where the symbol $\circ$ denotes function composition. Each section embodies more than one convolution layer, and each convolution layer in the same section contains the same number of filters[1]. Let C_i denote the number of filters in the i-th section, then the filters can be denoted by a $W_i \times H_i \times C_{i-1} \times C_i$ tensor. Following the idea of group convolution [20], for each $i \in \{1, 2, ..., S\}$, we split the i-th section into C_i/C groups, in which the convolution layers contains the fixed-sized $W_i \times H_i \times C \times C$ filters. C is a pre-defined hyper-parameter. We view these split groups as the base learners and symbolize them by $\mathcal{L} = \{\mathcal{L}_1, ..., \mathcal{L}_S\}$, where $\mathcal{L}_i = \{l_1^i, ..., l_{C_i/C}^i\}$ is the set of base learners unpacked from the i-th section, as seen in Figure 1 (b) and (c). These base learners play important roles in representation learning in the neural tree. For better illustration, we denote the input data of learner l_j^i as $\mathbf{x}_j^i$ and the output as $\mathbf{y}_j^i$, *i.e.*, $\mathbf{y}_j^i = l_j^i(\mathbf{x}_j^i)$.

2.2.2 Installing Routers

The number of learners determines the number of nodes in the search space. We view $\mathcal{L}_S$ as the set of *leaf learners* and $l \in \mathcal{L} \setminus \mathcal{L}_S$ as *internal learners*. In this step, we install routers to turn the isolated learners into a connected DAG. For any two learners l_p^i and l_q^j, the data flow from l_p^i to l_q^j is allowed if and only if $j = i + 1$. In this work, we adopt two types of routers, named *senders* and *receivers*, to direct

[1] In prevailing design philosophy, the number of filters only changes after the pooling layers.

the data to flow through the network. The senders are installed after each internal learner to route the path for the output data. Ideally, for the output from a specific learner, the sender chooses only a subset of learners in the next section to pass the data, so that we devise the sender as a softmax classifier. Specifically, the sender is implemented by two convolution layers, an adaptive average pooling layer for extracting feature vectors, and a fully connected layer followed by a softmax layer. For learner l_p^i, the installed sender is denoted by r_p^i. It is actually a $\frac{C_{i+1}}{C}$-way classifier that makes choices among the following learners to pass the data, as seen in Figure 1 (1).

It seems sufficient to route the data to flow through the network with the aid of senders. However, the learners in different sections are interleaved in such a way, which leaves the model still a neural network rather than a neural tree and difficult to interpret. One important characteristic of DTs, which brings better interpretability, is that each node in DTs receives data only from one parent node. To enforce the current learner receives data from only one learner in the previous section, we place a receiver before each learner to decide on which input data the learner receives. The receiver samples a categorical value from the continuous sampling distribution. To enable the differentiability for the sampling operation, we utilize Gumbel-Softmax [15] to implement the receiver. Formally, for the j-th learner l_j^i in section i, we construct a vector $\mathbf{w}_j^i = \{w_{j,1}^i, w_{j,2}^i, ..., w_{j,C_{i-1}/C}^i\}$ to represent the connectivity from learners in the previous section to the learner l_j^i. Each element $w_{j,k}^i$ stores the probability value that denotes how likely the output of learner l_k^{i-1} would be sampled by the receiver to pass to learner l_j^i. During the forward propagation, the receiver makes a discrete decision drawn from the categorical distribution based on the distribution:

$$\mathbf{h}_j^i = \text{one\_hot}\{\arg\max_k (\log w_{j,k}^i) + \epsilon_k\}. \tag{1}$$

Here $\mathbf{h}_j^i$ is a one-hot vector with the dimension the same as the number of learners in the previous section. $\epsilon \in \mathbb{R}^{C_i/C}$ is a vector in which the elements are i.i.d samples drawn from the Gumbel distribution $(0, 1)$ to add a small amount of noise to avoid the argmax operation always selecting the element with the highest probability value.

To enable differentiability of the discreet sampling function, we use the Gumbel-Softmax trick to relax $\mathbf{h}_j^i$ during backward propagation as

$$\mathbf{h}_j^i = \frac{\exp((\log \mathbf{w}_j^i + \epsilon)/\tau)}{\sum_k \exp((\log w_{j,k}^i + \epsilon_k)/\tau)}, \tag{2}$$

where τ is the temperature that controls how sharp the distribution is after the approximation. Ultimately, we formula the sampling operation as: $\mathbf{x}_j^i = \mathbf{h}_j^i \cdot \mathbf{Y}^{i-1}$, where $\mathbf{Y}^{i-1} = [\mathbf{y}_1^{i-1}, \mathbf{y}_2^{i-1}, \ldots, \mathbf{y}_{C_{i-1}/C}^{i-1}]$ and x_j^i donates the output of the j-th receiver in section i.

2.2.3 Mounting Solvers

After leaf learners, the data features become low-dimensional. Finally, we append a solver s_p^s to each leaf learner l_p^s to make the final task predictions. The solvers are implemented by a fully connected layer and a softmax layer.

2.3. Optimization

As shown in Figure 1 (d), the entire search space can be viewed as a deep forest or a *Hierarchical Mixture of Experts* (HMEs) [17], each of which is implemented by a multi-layered network and a root-to-leaf decision path. Each expert learns some specialized features, which may be useful for different sub-tasks. The goal of optimization is training the entire model by the final objective with implicit sparsity constraints from routers, which makes the model adapt to task and produce a lightweight neural tree.

2.3.1 Probabilistic Model and Inference

The input $\mathbf{x}$ stochastically traverses the tree based on the decisions of the routers and undergoes a sequence of transformations until it reaches a leaf node where the corresponding solver predicts the label y. Recalling that $\boldsymbol{\psi}$, $\boldsymbol{\theta}$, and $\boldsymbol{\phi}$ denote the parameters of the learners, the routers and the solvers, respectively. We use Θ to indicate involved parameters, *i.e.*, $\Theta = \{\boldsymbol{\psi}, \boldsymbol{\phi}, \boldsymbol{\theta}\}$. The predictive distribution is

$$p(y|\mathbf{x}, \Theta) = \sum_j^{C_S/C} \underbrace{p(z_j = 1|\mathbf{x}, \boldsymbol{\psi}, \boldsymbol{\theta})}_{\text{Leaf-reaching probability}} \underbrace{p(y|\mathbf{x}, z_j = 1, \boldsymbol{\phi}, \boldsymbol{\psi})}_{\text{solver prediction}}, \tag{3}$$

where the first term $p(z_j|\mathbf{x}, \boldsymbol{\psi}, \boldsymbol{\theta})$ denotes the probability of reaching the j-th leaf. The second term $p(y|\mathbf{x}, z_j = 1, \boldsymbol{\phi}, \boldsymbol{\psi})$ represents the prediction distribution produced by the j-th solver. The leaf reaching probability is calculated by propagating the routing probability from the root to the leaf nodes with the following propagation rule:

$$p(z_j^i = 1|\mathbf{x}, \Theta_j^i) = \sum_{k=1}^{\frac{C_i}{C}} p(z_k^{i-1}|\mathbf{x}, \Theta_j^{i-1}) \cdot r_k^{i-1}(z_j^i = 1|\mathbf{x}, \boldsymbol{\theta}), \tag{4}$$

where $p(z_j^i = 1|\mathbf{x}, \Theta)$ denotes the probability of reaching the j-th node in the i-th section. r_k^{i-1} denotes the routing probability produced by the sender after k-th learner in the $(i-1)$-th section. Θ_j^i denotes all the involved parameters for calculating the path probability of reaching learner l_j^i.

2.3.2 Loss Functions and Tree Selection

Training of the proposed model proceeds in two stages: *search phase* and *retraining phase*. During the searching phase the whole model is optimized to search for the lightweight neural tree in the large search space, as shown in Figure 1 (e). We adopt the *Negative Log-Likelihood* (NLL) loss as the objective function to optimize the model:

$$\mathcal{J} = -\log p(\mathbf{Y}|\mathbf{X}, \Theta) = -\sum_{n=1}^{N} \log(\sum_{i=1}^{C_s/C}) p(y_{y^{(n)}}|\mathbf{x}^{(n)}, \Theta), \tag{5}$$

where $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$ and $\mathbf{Y} = \{y^{(1)}, y^{(2)}, ..., y^{(N)}\}$ denote the training inputs and targets, respectively. With the Gumbel-Softmax trick in the routers, all modules are differentiable with respect to their parameters, so that we adopt stochastic gradient descent to optimize the model in an end-to-end way. After the search phase, we retain the node in i-th section if its conditional probability obtained from previous senders is greater than the threshold $C/(2 \times C_i)$. After that, the retraining phase retrains the derived neural tree from scratch [23].

3. Experiments

3.1. Experimental Settings

Datasets. Four classification benchmarking datasets, including CIFAR10 [2], CIFAR100 [2], tiny-ImageNet [21] and ImageNet [5], with varying complexities, are adopted to comprehensively evaluate the generalisation and effectiveness of the proposed method SeBoW. These datasets span over a range of sizes and input resolutions. CIFAR10 [5] and CIFAR100 [5] each comprises a collection of 60k 32×32 pixel images. Tiny-ImageNet [21] consists of 110k images in resolution 64×64, and the ImageNet [5] dataset contains 1.33 million images from 1000 different classes with 224×224 resolution.

Network architecture. For ImageNet, we construct a search space based on the network configuration of truncated VGG-13 [33], in which all the fully connected layers are removed. The input data are down-pooled by 5 times through the VGG-13, so we cut the model into 5 sections, as described in Section 2.2. The number of learners in each section is $\{1, 2, 4, 8, 8\}$, respectively. For CIFAR-10, CIFAR-100 and tiny-ImageNet, as the image resolution is much smaller than ImageNet, four times of down-pooling are sufficient to extract low-dimensional features. We construct the search space with only the former four sections, with number of learners $\{1, 2, 4, 8\}$. To comprehensively evaluate the proposed method, we devise three variants of SeBoW, symbolized by SeBoW-A, SeBoW-B and SeBoW-C, with varying capacities. Details of these models are summarized in Table 1.

Model	Sender	Learner	Solver
SeBoW-A	2× Conv3-48 + GAP + LC	2× Conv3-96 + BN + ReLU + MaxPool	GAP + LC
SeBoW-B	2× Conv3-72 + GAP + LC	2× Conv3-144 + BN + ReLU + MaxPool	GAP + LC
SeBoW-C	2× Conv3-128 + GAP + LC	2× Conv3-256 + BN + ReLU + MaxPool	GAP + LC

Table 1. Details of the primitive modules. "Conv3-48" represents the 3×3 convolution with 48 filters. "GAP", "LC", "BN" and "MaxPool" denote global-average-pooling, linear classifier, batch-normalization and max-pooling operations, respectively.

Training details. We use SGD with the initial learning rate of 0.1. After 30 epochs, the learning rate is decayed by half for every 20 epochs until reaching 100 epochs where the training ceases. We set the batch size to 128, the weight decay to 10^{-4}, and the Nesterov momentum to 0.9. During the network search phase, we initialize the connectivity vectors in all receivers with uniform distributions to encourage free exploration in the early stage. The temperature τ is set to 10 and decayed by the number of epochs to exploit the converged topology distribution in the later stage. We retrain the selected final architecture on the training sets from scratch using the same training set in the search phase but set the weight decay to be 5×10^{-4}. Please refer to the supplementary materials for more details.

Inference schema. Thanks to the decision-tree nature of SeBoW, the inference can be executed in two manners: *multi-path inference* and *single-path inference*. Multi-path inference computes the weighted predictive distribution by running over all possible paths in the derived neural tree, such that all solvers in the tree will contribute to the final prediction. However, in the single-path inference, only the most probable paths are executed based on the routing probability from routers, which enjoys less inference cost with some accuracy drop.

3.2. Benchmark Comparisons

We compare the performance of SeBow against three groups of existing models: (1) typical human-engineered DNNs, including VGG [33], ResNet [11], GoogleNet [35], and MobileNet [12]; (2) neural decision trees, including Adaptive Neural Trees (ANT) [36], Neual Decision Tree Towards Neural Graph(NDT) [37], Deep Neural Decision Forests (DNDF) [19], Conditional CNN [13] and Explainable Observer-Classifier (XOC) [1]; (3) as the proposed neural tree can be viewed as a multi-branch network, we also compare it with some multi-branch models widely used in multi-task learning. These models include Routing network [29], Learning to Branch [10] and Cross-stitch [24] networks. We implement some of these competitors with the same training settings as the proposed method.

Experimental results in both accuracy and model com-

Method	Params.	Accuracy (%)
MobileNet [12]	2.2M	85.90 (±0.23)
VGG-13 [33]	28.3M	92.51 (±0.15)
ResNet-18 [11]	11.2M	92.98 (±0.17)
Max-Cut DT [4]	N/A	*34.90*
Compact BT [25]	N/A	*48.56*
gcForest [42]	N/A	*61.78* / *61.78*
Conditional CNN [13]	*> 0.5M*	*< 90.00*
ANT-CIFAR10-C [36]	*0.7M* / *0.5M*	*90.69* / *90.66*
ANT-CIFAR10-B [36]	*0.9M* / *0.6M*	*90.85* / *90.82*
ANT-CIFAR10-A [36]	*1.4M* / *1.0M*	*91.69* / *91.68*
ANT-CIFAR10-A (ensemble) [36]	*8.7M* / *7.4M*	*92.29* / *92.21*
XOC [1] + ResNet-18	> 11.2M	93.12 (±0.32)
LearnToBranch-Deep-Wide [10]	3.5M	91.98 (±0.57)
SeBoW-A	1.0M / 0.7M	93.45 (±0.12) / 93.41
SeBoW-B	2.7M / 1.6M	94.00 (±0.18) / 93.93
SeBoW-C	5.8M / 4.6M	**94.33** (±0.14) / **94.24**

Table 2. Performance comparison on CIFAR-10. Underlined numbers denote the results of single-path inference. Italic fonts mean that the results are copied from the original paper. "N/A" means not applicable.

Method	Params.	Accuracy (%)
MobileNet [12]	2.4M	53.91 (±0.32)
VGG-13 [33]	28.7M	72.70 (±0.42)
ResNet-18 [11]	11.2M	72.28 (±0.28)
Max-Cut DT [4]	N/A	*12.40*
NDT [37]	*14.1M*	*15.48*
DNDF [19] + ResNet-18	> 11.2M	67.18
ANT-Extend [36]	4.2M / 4.2M	65.81 (±0.12) / 65.71
Cross Stitch [24, 29]	-	*53.0*
Routing network [29]	-	*60.50 (±0.75)*
LearnToBranch-Deep-Wide [10]	6.7M	72.04 (±0.23)
SeBoW-B	1.9M / 1.5M	71.79 (±0.23) / 71.59
SeBoW-C	4.2M / 4.2M	74.59 (±0.33) / **74.59**

Table 3. Performance comparisons on the CIFAR-100 dataset.

Method	Params.	Accuracy (%)
MobileNet [12]	2.5M	46.12 (±0.73)
GoogleNet [35]	6.8M	48.85 (±0.52)
VGG-13 [33]	28.7M	56.10 (±0.57)
ResNet-18 [11]	11.2M	55.32 (±0.75)
DNDF [19] + ResNet-18	> 11.2M	44.56
SeBoW-C	8.4M / 4.8M	**58.77** (±0.39) / **58.43** (±0.45)

Table 4. Performance comparison on the tiny-ImageNet dataset.

plexity (the number of parameters) on the four benchmark datasets are provided in Table 2, 3, 4 and 5, respectively. All these results are computed by averaging over three individual runs. Note that we provide the results of both the multi-path and the single-path (shown in underlined font) inference for some tree-based models to give a more comprehensive view of the proposed method. From these results, we can make the following conclusions: (1) On CIFAR-10, CIFAR-100 and tiny-ImageNet, SeBoW yields consistently superior performance to almost all types of competitors with smaller model size. For example, on CIFAR-10, SeBoW achieves 94.24% accuracy with only 4.6M parameters. However, ResNet-18 reaches only 92.98% with 11.2M parameters. (2) On ImageNet, which is seldom explored by prior tree models, SeBoW still produces competitive performance with fewer parameters compared to its search space VGG-13. (3) It can be seen that the accuracy comparison result is SeBoW-A < SeBoW-B < SeBoW-C. With more filters in the base learners, the model capacity becomes larger. The larger capacity leads to a significant performance boost. (4) For all experiments, the single path inference produces only a slight accuracy drop compared to multi-path inference. It demonstrates the effectiveness of SeBoW in searching the sparsely connected tree in the densely wired search space.

Method	Params.	Top-1 Acc.	Top-5 Acc.
VGG-13 [33]	*133.0M*	*69.93*	*89.25*
ResNet-18 [11]	*11.7M*	*69.76*	*89.08*
Conditional CNN [13]	*≈ 30M*	-	*86.20*
XOC [1] + ResNet-152	*> 60.2M*	*60.77*	-
SeBow-C (Multi-Path)	16.90M	**70.13**	**89.98**
SeBow-C (Single-Path)	14.78M	69.86	89.17

Table 5. Performance comparison on ImageNet. Both top-1 and top-5 accuracy are provided.

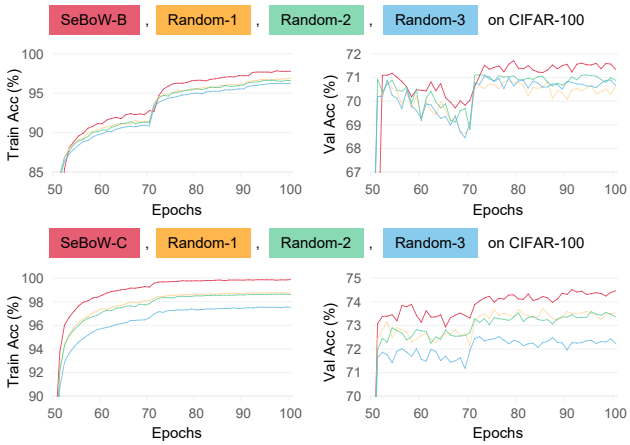


Figure 2. The accuracy curves of SeBoW and randomly wiring models. Best viewed in color.

3.3. Ablation study

Self-born wiring *versus* random wiring. To validate the effectiveness of the proposed self-born wiring (SeBoW), we compare it with *random wiring*, where the connections between different base learners are randomly determined. Experimental results are provided in Table 6, and some accuracy curves are depicted in Figure 2. We can see that SeBoW produces consistently higher final accuracy compared to random wiring under different experimental set-

Dataset	Method	Architecture	Params.	Accuracy (%)
CIFAR10	SeBoW-A	{1, 1, 1, 3}	1.0M / 0.7M	**93.45 / 93.41**
	Random	Randomly generated	2.9M / 0.9M	91.71 / 91.23
			0.6M / 0.6M	92.55 / 92.55
			1.4M / 0.7M	92.18 / 92.07
			1.0M / 0.7M	83.32 / 83.21
	SeBoW-B	{1, 1, 2, 3}	2.7M / 1.6M	**94.00 / 93.93**
	Random	Randomly generated	6.4M / 1.9M	93.41 / 93.09
			1.3M / 1.3M	93.63 / 93.63
			3.2M / 1.7M	93.57 / 93.49
			2.7M / 1.6M	84.04 / 84.01
CIFAR100	SeBoW-B	{1, 1, 1, 2}	1.9M / 1.5M	**71.79 / 71.59**
	Random	Randomly generated	6.5M / 1.9M	70.40 / 70.08
			1.3M / 1.3M	71.12 / 71.12
			3.3M / 1.6M	70.63 / 70.40
			1.8M / 1.5M	63.42 / 63.38
	SeBoW-C	{1, 1, 1, 1}	4.2M / 4.2M	**74.59 / 74.59**
	Random	Randomly generated	20.5M / 6.2M	72.52 / 72.22
			7.0M / 4.6M	73.71 / 73.59
			10.3M / 5.4M	73.58 / 73.33
			6.9M / 4.6M	67.04 / 67.02

Table 6. Ablation study on network architecture. "Architecture" represents the number of learners in each section.

tings. These results validate the effectiveness of the proposed wiring method for neural trees.

Senders and Receivers. We propose another two variants of SeBoW to validate the necessity of routers, one without senders and the other without receivers, in Table 7. The model without receivers sends data weighted with path probability rather than sampling vector, but the resulting architecture always degenerates into a single branch with sub-optimal results, *i.e.* $\{1, 1, 1, 1\}$-architecture. The model without senders selects the final network architecture using the distribution given by Equation 1 but without the noise ϵ. Notably, this model cannot calculate path probability, so single-path inference is not applicable. We find that the model using both senders and receivers always produces the best top-1 accuracy. We speculate that the sampling distribution fuses various features for more robustness architecture selection, and the probability distribution under multiple paths helps the network obtain the optimal result.

Dataset	Method	S.	R.	Params.	Accuracy (%)
CIFAR10	SeBoW-A	✓		0.6M / 0.6M	92.55 / 92.55
			✓	1.1M	91.33
		✓	✓	1.0M / 0.7M	**93.45 / 93.41**
	SeBoW-B	✓		1.3M / 1.3M	93.63 / 93.63
			✓	2.5M	92.72
		✓	✓	2.7M / 1.6M	**94.00 / 93.93**
CIFAR100	SeBoW-B	✓		1.3M / 1.3M	71.12 / 71.12
			✓	3.9M	52.89
		✓	✓	1.9M / 1.5M	**71.79 / 71.59**
	SeBoW-C	✓		4.2M / 4.2M	**74.59 / 74.59**
			✓	10.3M	58.12
		✓	✓	4.2M / 4.2M	**74.59 / 74.59**
Tiny-ImageNet	SeBoW-C	✓		4.2M / 4.2M	54.42 / 54.42
			✓	18.0M	42.01
		✓	✓	8.4M / 4.8M	**58.77 / 58.43**
ImageNet	SeBoW-C	✓		5.6M / 5.6M	68.12 / 68.12
			✓	19.80M	62.29
		✓	✓	16.90M / 14.78M	**70.13 / 69.86**

Table 7. Ablation study on architecture modules. The columns "S." and "R." denote the senders and receivers.

Method	Flops	Speed(Batches/ms.)	Accuracy(%)
VGG-13	248.3M	64.85	92.51
ANT-B	163M / 149M	87.51 / 90.43	90.85 / 90.82
ANT-A	254M / 243M	40.00 / 41.50	91.69 / 91.68
SeBoW-A	**151M / 146M**	**89.23 / 90.91**	**93.45 / 93.41**

Table 8. Flops, inference speed on CIFAR-10. Underlined numbers denote the results of single-path inference.

Dataset	Model	Stage 1		Stage2	
		Time	Epochs	Time	Epochs
CIFAR10	ANT-A [36]	1.7 (hr)	265	1.5 (hr)	200
	SeBoW-B	**1.3 (hr)**	100	**0.5 (hr)**	100
CIFAR100	ANT-Extend [36]	1.8 (hr)	255	2.0 (hr)	200
	SeBoW-C	**1.5 (hr)**	100	**0.8 (hr)**	100

Table 9. Training time of ANT and SeBoW. The two stages are growth and refinement phases in ANT [36], architecture search and retraining phases in SeBoW.

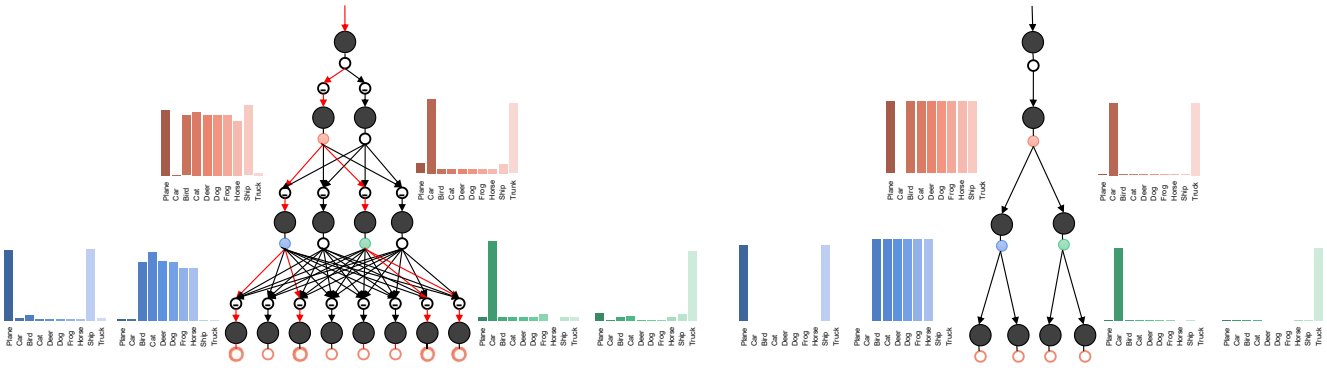
Inference Speed. To validate the lightweight of SeBoW, we investigate the Flops and inference speed of various models. We select CIFAR-10 as experimental dataset and show the results in Table 8. Experiments are run on a single GeForce GTX 1080 Ti with batch size of 256. It can be seen that our SeBoW achieves higher test accuracy and inference speed than neural trees and mother DNN, which indicates the value of the self-born wiring.

Training time. To demonstrate the superiority of SeBoW in training efficiency, we compare ANT [36] and SeBoW under a similar amount of parameters. Table 9 summarises the average time taken with three individual runs on a single Quadro P5000 GPU with 16GB memory. It indicates that SeBoW is much time-efficient than ANT thanks to its global optimization in differentiable architecture spaces instead of greedy evolution for architecture growth.

3.4. Interpretability

Here we demonstrate that SeBoW exhibits better interpretability to DNNs thanks to its endowed decision-tree nature. The training of the SeBoW is divided into two stages.

We visualize the class distribution on the decision paths of SeBoW-C on CIFAR10 in Figure 3. The results show that SeBoW is able to partition categories into several groups with similar semantics or visual cues. The model first divides all categories into two groups as $\{car, truck\}$ and $\{animal, ship, plane\}$. Cars and trucks are vehicles with wheels, which are similar in appearance. In the other branch, since aircraft design principles are derived from bionics related to bird characteristics, such as aircraft wings



(a) Search Space (b) Searched Structure

Figure 3. Visualization of class distributions on decision paths. (a) shows that the model captures a hierarchy from complete search space. (b) exhibits the searched architecture further polarises path probabilities of samples in different classes.

and bird wings, there are certain morphological similarities between these artifacts and animals. This category group is further divided into $\{ship, plane\}$ and $\{animal\}$ by the model, which may be due to the difference in characteristics between metal materials and animal fur. However, it should be pointed out that the human intuition of the category relationship is not necessarily equal to the optimal network architecture. SeBoW can explore the relationship among categories from the perspective of model itself, thus supporting the interpretability of our model. Visualizations of all learned network architectures are provided in the supplementary materials.

4. Related Works

Neural Decision Trees, combines the characteristics of decision trees and neural networks to explore the network structures with the complementary benefits of both approaches, yielding a lightweight model for disentangled inference [39, 38]. Earlier, tree-based neural networks [17, 8] just perform a top-to-bottom path selection for a given sample without any representation learning, thus limiting their performance. Modern tree-based neural networks [19, 16] strengthen the performances by integrating nonlinear transformations of features into trees. Xiao [37] proposes an approach with the root transformer MLP and optimises the network to minimise the information gain loss. Kontschieder *et al*. [19] and Ji *et al*. [16] employ the traditional hand-crafted network [33, 11, 35] as the root transformer, resulting in surprising performances. All the models described above are pre-specified and fixed, exhibiting finite flexibility with tasks. Soft decision trees [34, 14] greedily grow new nodes with termination criteria based on validation set error. Adaptive neural trees [36] optimise architecture through progressive growth with level limitation to avoid over-fitting. Decision jungles [32] consider the input space of multiple subtrees to be mergeable, thus to avoid the local optimum that "split" algorithms may fall into.

Neural Architecture Search (NAS), emerges for minimizing human intervention by allowing automatically designing the network architectures. Existing NAS research works can be categorized into modular search strategy and continuous search space [28]. The cell-based search space [22, 7, 40] is widely used in various NAS tasks as it can be migrated to different tasks by stacking cells, which effectively reduces the cost of NAS compared with global search. Earlier works regard network architecture search as a black-box optimization problem in discrete search space based on Bayesian optimization [6, 18, 41], evolutionary algorithms [27, 26] and reinforcement learning [3, 43]. For this, DAS [30] is committed to transforming the discrete network architecture space into a continuously differentiable form, and uses gradient optimization techniques to search for the best network architecture.

5. Conclusion

In this paper, we introduced a novel approach, termed as Self-born Wiring, to automatically construct neural trees from a pre-defined deep neural network. Unlike prior neural-tree methods those either limit themselves to predefined structures or rely on greedy algorithms to grow hierarchical layers, SeBoW carries out a construction-by-destruction process that enables a global-level wiring optimization to learn tree architectures. Experimental results showcase that the derived neural trees yield results even on par with those of DNNs on large-scale datasets. In our future work, we will explore SeBoW on other vision tasks beyond classification problems.

Acknowledgments. This work is supported by Key Research and Development Program of Zhejiang Province (2020C01024)), the Fundamental Research Funds for the Central Universities (2021FZZX001-23), Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies, and the Startup Fund of National University of Singapore.

References

[1] Stephan Alaniz and Zeynep Akata. XOC: explainable observer-classifier for explainable binary decisions. *CoRR*, abs/1902.01780, 2019.

[2] Geoffrey Hinton Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

[4] Jonathan Bodine and Dorit S. Hochbaum. The max-cut decision tree: Improving on the accuracy and running time of decision trees. *CoRR*, abs/2006.14118, 2020.

[5] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[6] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, pages 3460–3468, 2015.

[7] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *ECCV*, pages 540–555, 2018.

[8] Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. In *(AI*IA)*, 2017.

[9] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *CVPR*, pages 1022–1029, 2009.

[10] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. In *ICML*, 2020.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[12] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *CVPR*, 2017.

[13] Yani Ioannou, Duncan P. Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *CoRR*, abs/1603.01250, 2016.

[14] Ozan Irsoy, Olcay Taner Yildiz, and Ethem Alpaydin. Soft decision trees. In *ICPR*, pages 1819–1822, 2012.

[15] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

[16] Ruyi Ji, Longyin Wen, Libo Zhang, Dawei Du, Yanjun Wu, Chen Zhao, Xianglong Liu, and Feiyue Huang. Attention convolutional binary neural tree for fine-grained visual categorization. In *CVPR*, pages 10465–10474, 2020.

[17] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, (2):181–214, 1994.

[18] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *NeurIPS*, pages 2020–2029, 2018.

[19] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulò. Deep neural decision forests. In *IJCAI*, pages 4190–4194, 2016.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1106–1114, 2012.

[21] Y. Le and X. Yang. Tiny imagenet visual recognition challenge, 2015.

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

[23] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

[24] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *CVPR*, pages 3994–4003, 2016.

[25] Natalia Ponomareva, Thomas Colthurst, Gilbert Hendry, Salem Haykal, and Soroush Radpour. Compact multi-class boosted trees. In *BigData*, pages 47–56, 2017.

[26] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, pages 4780–4789, 2019.

[27] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *ICML*, pages 2902–2911, 2017.

[28] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *CoRR*, abs/2006.02903, 2020.

[29] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *ICLR*, 2018.

[30] Richard Shin, Charles Packer, and Dawn Song. Differentiable neural network architecture search. In *ICLR*, 2018.

[31] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *CVPR*, pages 1–8, 2008.

[32] Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John M. Winn, and Antonio Criminisi. Decision jungles: Compact and rich models for classification. In *NeurIPS*, pages 234–242, 2013.

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

[34] Alberto Suárez and James F. Lutsko. Globally optimal fuzzy decision trees for classification and regression. *PAMI*, (12):1297–1311, 1999.

[35] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015.

[36] Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, Antonio Criminisi, and Aditya V. Nori. Adaptive neural trees. In *ICML*, pages 6166–6175, 2019.

[37] Han Xiao. NDT: neual decision tree towards fully functioned neural graph. *CoRR*, abs/1712.05934, 2017.

[38] Yiding Yang, Zunlei Feng, Mingli Song, and Xinchao Wang. Factorizable graph convolutional networks. In *NeurIPS*, 2020.

[39] Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. Distilling knowledge from graph convolutional networks. In *CVPR*, pages 7074–7083, 2020.

[40] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *CVPR*, pages 2423–2432, 2018.

[41] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, pages 7603–7613, 2019.

[42] Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. In *IJCAI*, pages 3553–3559, 2017.

[43] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.