AdvantageNAS: Efficient Neural Architecture Search with Credit Assignment

Rei Sato, Jun Sakuma, Youhei Akimoto

University of Tsukuba
RIKEN AIP
{reisato@bbo., jun@, akimoto@}cs.tsukuba.ac.jp

Abstract

Neural architecture search (NAS) is an approach for automatically designing a neural network architecture without human effort or expert knowledge. However, the high computational cost of NAS limits its use in commercial applications. Two recent NAS paradigms, namely one-shot and sparse propagation, which reduce the time and space complexities, respectively, provide clues for solving this problem. In this paper, we propose a novel search strategy for one-shot and sparse propagation NAS, namely AdvantageNAS, which further reduces the time complexity of NAS by reducing the number of search iterations. AdvantageNAS is a gradient-based approach that improves the search efficiency by introducing credit assignment in gradient estimation for architecture updates. Experiments on the NAS-Bench-201 and PTB dataset show that AdvantageNAS discovers an architecture with higher performance under a limited time budget compared to existing sparse propagation NAS. To further reveal the reliabilities of AdvantageNAS, we investigate it theoretically and find that it monotonically improves the expected loss and thus converges.

1 Introduction

Deep neural networks (DNNs) have achieved remarkable performance improvements in various domains. Finding a suitable DNN architecture is critical for applying a DNN to a new task and achieving higher performance. However, as the architecture is defined by a combination of tens to hundreds of hyperparameters (filter sizes of convolutional layers, existence of connections between some layers, etc.), improving such a combination requires considerable human effort and expert knowledge. Neural architecture search (NAS) has been studied as a promising approach for automatically identifying sophisticated DNN architectures and overcoming the problems in applying DNNs to previously unseen tasks.

However, the high computational cost (time and space complexity) of NAS is a bottleneck in its applications, especially commercial applications that do not involve multiple high-spec GPUs. Hence, reducing the computational cost of NAS is an important research direction.

One-shot NAS reduces the time complexity of NAS by simultaneously optimizing the architecture and the parameters

(weights) in a single training process or a few training processes. It maintains a redundant deep learning model comprising numerous layers and connections, called the parent network, which can represent all possible architectures in the search space as its sub-network. All the parameters required by candidate architectures are possessed by the parent network and shared among different architectures. The goodness of an architecture is evaluated using the shared parameters, which drastically reduces the computational time for the architecture search compared to conventional approaches that require the parameters to be trained from scratch for each candidate architecture evaluation.

From the viewpoint of GPU memory consumption, one-shot NAS can be categorized into dense propagation NAS (DP-NAS) and sparse propagation NAS (SP-NAS). DP-NAS requires the entire parent network to be loaded in the GPU memory to perform forward and/or backward propagation, whereas SP-NAS requires only a sub-network to be loaded. DP-NAS is more popular owing to the success of DARTS and its successors (Liu, Simonyan, and Yang 2019; Xie et al. 2019; Yao et al. 2020). However, because of its high memory consumption, its search space is limited. SP-NAS overcomes this limitation and provides more flexible search spaces on a standard GPU with limited memory. Hence, we focus on SP-NAS in this study.

To further reduce the computational cost of one-shot SP-NAS, we propose a novel one-shot SP-NAS, namely *AdvantageNAS*. Although various factors affect the performance of NAS (performance estimation, search space, etc.) and state-of-the-art (SOTA) performance may require a search for the best combination of these factors, this study focuses on the search strategy. We emphasize that the objective of this study is not to obtain the SOTA architecture but to investigate the efficiency of the search strategy.

AdvantageNAS is a gradient-based SP-NAS.[1] This cate-

---

[1] The advantages of gradient-based approaches are as follows. Approaches using evolutionary algorithms (e.g., (Real et al. 2017)) or progressive search methods (e.g., (Liu et al. 2018)) must define a mutation or expansion strategy depending on the search space. Such manual preparations depending on the search space are not required in gradient-based methods. Another advantage of gradient-based methods is that they can be easily implemented in most modern deep learning frameworks, as they support automatic differentiation of deep neural networks.

gory of NAS includes ENAS (Pham et al. 2018), Proxyless-NAS (Cai, Zhu, and Han 2019), PARSEC (Casale, Gordon, and Fusi 2019), and GDAS (Dong and Yang 2019). These methods introduce a family of parametric probability distributions from which an architecture is sampled, and the distribution parameter is updated with a gradient of the objective function estimated using the Monte Carlo method, which indirectly optimizes the architecture itself. As the gradient is estimated, reducing the variance of the gradient estimate intuitively improves the search efficacy. Inspired by the actor-critic approach (Konda and Tsitsiklis 2000), we adopt the notion of *advantage*, which has been introduced to solve the well-known *credit assignment* problem of policy gradients (Minsky 1961).

We investigate AdvantageNAS both theoretically and empirically. Empirical studies show that it performs well on different tasks with different search spaces, whereas theoretical studies guarantee its performance on empirically unforeseen tasks. Theoretical studies are relatively scarce in NAS research; however, they are important for investigating search strategies for NAS.

Our contributions are summarized as follows:

- We propose a novel SP-NAS, namely AdvantageNAS. Our analysis of a test scenario demonstrates that the estimation variance of the gradient is reduced compared to REINFORCE (Williams 1992), which is a baseline search strategy for ENAS.

- The theoretical investigation reveals that the estimated gradient in AdvantageNAS is an unbiased estimator of the gradient of the expected loss. From the unbiasedness, we find that AdvantageNAS monotonically improves the expected loss and thus converges. Further, we confirm that reducing the gradient variance contributes to an improved lower bound of the loss improvement in each iteration.

- We compared AdvantageNAS with various gradient-based NAS methods in three different settings to demonstrate its efficacy on different search spaces. Our experiments were performed on the same code base to compare the search strategies.[2] The results showed that AdvantageNAS converges faster than existing SP-NAS approaches.

2 One-shot Neural Architecture Search

DAG Representation of DNNs We formulate a deep neural network as a directed acyclic graph (DAG) following previous studies (Pham et al. 2018; Liu, Simonyan, and Yang 2019). In the DAG representation, each node corresponds to a feature tensor x and each edge corresponds to an operation $\mathcal{O}$ applied to the incoming feature tensor, e.g., convolution or pooling. The adjacency matrix of the DAG is pre-defined. Let $\mathcal{I}$ be a set of edges. The input and output nodes are two special nodes; the feature tensor at the input node is the input to the DNN and the input node has no incoming edge,

while the feature tensor at the output node is the output of the DNN and the output node has no outgoing edge.

The feature tensor x_ℓ at node ℓ, except for the input node, is computed as follows. Let $\mathcal{E}_\ell \subseteq \mathcal{I}$ be a subset of the edges that point to node ℓ. Let $Q(i)$ be the input node of edge $i \in \mathcal{I}$. For each edge $i \in \mathcal{E}_\ell$, the operation $\mathcal{O}_i$ is applied to $x_{Q(i)}$ and output $y_i = \mathcal{O}_i(x_{Q(i)})$. The feature tensor x_ℓ is computed by aggregating the outputs y_i of the incoming edges $i \in \mathcal{E}_\ell$ with some pre-defined aggregation function $F_\ell : \{y_i : i \in \mathcal{E}_\ell\} \mapsto x_\ell$ such as summation or concatenation.

Search Space for Neural Architecture For each edge $i \in \mathcal{I}$ of the DAG, a set of candidate operations is prepared. Let $\mathcal{J}_i$ be the indices of the candidate operations at edge i, and each candidate operation is denoted as $\mathcal{O}_i^j$ for $j \in \mathcal{J}_i$. Each candidate operation may have its own real-valued parameter vector w_i^j, often referred to as a *weights*, and $\mathcal{O}_i^j$ is assumed to be differentiable w.r.t. the weights. The operation $\mathcal{O}_i$ of edge i is then defined as a weighted average of the candidate operations with weight $\boldsymbol{a}_i = (a_i^1, \ldots, a_i^{|\mathcal{J}_i|})^{\mathrm{T}}$:

$$\mathcal{O}_i(x_{Q(i)}) = \sum_{j \in \mathcal{J}_i} a_i^j \mathcal{O}_i^j(x_{Q(i)}; w_i^j) \ , \qquad (1)$$

where $\sum_{j \in \mathcal{J}_i} a_i^j = 1$. The vector $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{|\mathcal{I}|})$ is referred to as the *architecture*.

The objective of NAS is to locate the best architecture under some performance criterion. The aim of our study, as well as many related studies (Liu, Simonyan, and Yang 2019; Xie et al. 2019; Cai, Zhu, and Han 2019; Dong and Yang 2019; Yao et al. 2020; Casale, Gordon, and Fusi 2019), is to find the best architecture such that $\boldsymbol{a}_i$ for each $i \in \mathcal{I}$ is a one-hot vector. In other words, we choose one operation at each edge.

Weight Sharing and One-shot NAS One-shot architecture search is a method for jointly optimizing $\boldsymbol{a}$ and $\boldsymbol{w}$ during a single training process.

Let R be a performance metric (referred to as *reward* in this study) that takes an architecture $\boldsymbol{a}$ and weights $\boldsymbol{w_a}$ required to compute the DAG defined by $\boldsymbol{a}$. The reward is typically the negative expected loss $-\mathbb{E}_{d \sim \mathcal{P}}[\mathcal{L}(\boldsymbol{a}, \boldsymbol{w_a}; d)]$, where $\mathcal{L}$ is a data-wise loss, $\mathcal{P}$ is a data distribution, and d represents data. For example, in the image classification task, d is a set of an image and its label, and $\mathcal{L}$ is the cross-entropy loss using d. The objective of NAS is written as $\max_{\boldsymbol{a}} R(\boldsymbol{a}, \boldsymbol{w_a^*})$, where $\boldsymbol{w_a^*} = \mathrm{argmax}_{\boldsymbol{w_a}} R(\boldsymbol{a}, \boldsymbol{w_a})$ is the optimal weights for architecture $\boldsymbol{a}$. The expected reward R is often approximated by the average over the validation dataset $\mathcal{V}$ for optimization of $\boldsymbol{a}$ and the training dataset $\mathcal{D}$ for optimization of $\boldsymbol{w_a}$. The difficulty in simultaneously optimizing $\boldsymbol{a}$ and $\boldsymbol{w_a}$ is that the domain of $\boldsymbol{w_a}$ changes with $\boldsymbol{a}$ during the training process. The time complexity of conventional NAS approaches (e.g., (Zoph and Le 2017)) is extremely high because they require $\boldsymbol{w_a}$ to be trained for each $\boldsymbol{a}$ from scratch.

Weight sharing (Pham et al. 2018) is a technique for simultaneous optimization. All the weights appearing in the candidate operations are stored in a single weight set, $\boldsymbol{w} = \{w_i^j : i \in \mathcal{I}, \ j \in \mathcal{J}_i\}$. The reward function is extended to

---

[2]NAS approaches using different components such as search spaces and training protocols have been compared in the literature. However, it has been reported, e.g., in (Yang, Esperança, and Carlucci 2020), that such differences in the components make it difficult to fairly compare the contributions of each component. Our code: https://github.com/madoibito80/advantage-nas.

Method	Propagation	a_i	Stochastic	TG
DARTS	dense	n-hot	no	
SNAS	dense	n-hot	yes	
NASP	dense[3]	1-hot	no	✓
ProxylessNAS	sparse	2-hot	yes	
PARSEC	sparse	1-hot	yes	
GDAS	sparse	1-hot	yes	
ENAS	sparse	1-hot	yes	
AdvantageNAS	sparse	1-hot	yes	✓

Table 1: Properties of various gradient-based NAS methods. The "Propagation" column indicates whether the method belongs to DP-NAS or SP-NAS discussed in Section 3. TG: Theoretical Guarantee.

accept w by defining $R(a, w) = R(a, w_a)$. Then, the objective of NAS is to find (a, w) that maximizes $R(a, w)$. A disadvantage of the weight sharing approach is that a large parameter set w is required during the search. However, this set can be stored in the CPU memory and only a subset $w_a \subseteq w$ is required to be loaded in the GPU, which typically involves much smaller memory. Moreover, if a is sparse during the training, the GPU memory consumption is limited.

3 Gradient-based One-shot NAS

The major stream of search strategies for one-shot NAS is gradient-based search (Pham et al. 2018; Liu, Simonyan, and Yang 2019; Xie et al. 2019; Cai, Zhu, and Han 2019; Dong and Yang 2019; Yao et al. 2020; Casale, Gordon, and Fusi 2019). The architecture a is either re-parameterized by a real-valued vector θ through some function $a = \mu(\theta)$ or sampled from a probability distribution $a \sim p_\theta$ parameterized by θ. Gradient steps on θ as well as on w are taken alternatively or simultaneously to optimize a and w.

A comparison of related studies is shown in Table 1.

Dense Propagation NAS (DP-NAS) DP-NAS has been popularized by DARTS (Liu, Simonyan, and Yang 2019), where the architecture is re-parameterized by the real-valued vector $\theta \in \mathbb{R}^{\sum_{i \in \mathcal{I}} |\mathcal{J}_i|}$ through the softmax function $\mu(\theta) = (\mu_1(\theta_1), \ldots, \mu_{|\mathcal{I}|}(\theta_{|\mathcal{I}|}))$, where $\mu_i(\theta_i) = (\mu_i^1(\theta_1), \ldots, \mu_i^{|\mathcal{J}_i|}(\theta_i))$ and

$$\mu_i^j(\theta_i) = \exp(\theta_i^j) / \sum_{k \in \mathcal{J}_i} \exp(\theta_i^k) \ . \tag{2}$$

In other words, the output of the jth edge in (1) is a weighted sum of the outputs of all candidate operations. The architecture $a = \mu(\theta)$ is optimized with the gradient $\nabla_\theta R(\mu(\theta), w)$.

Several follow-up works have been conducted. SNAS (Xie et al. 2019) uses a concrete distribution (Maddison, Mnih, and Teh 2017) to represent a. NASP (Yao et al. 2020) introduces a proximal iteration to update θ.

Sparse Propagation NAS (SP-NAS) SP-NAS introduces a parametric probability distribution p_θ over the architecture space and optimizes a indirectly through the optimiza-

tion of θ. Thus, all of the SP-NAS is the stochastic algorithm shown in Table 1. The objective is to maximize the reward function expected over the architecture distribution $J(\theta, w) = \mathbb{E}_{a \sim p_\theta}[R(a, w)]$. Different distributions and different update rules have been proposed. ENAS (Pham et al. 2018) uses LSTM (Hochreiter and Schmidhuber 1997) to represent p_θ and REINFORCE (Williams 1992) to estimate $\nabla_\theta J(\theta, w)$:

$$\begin{aligned} \nabla_\theta J(\theta, w) &= \nabla_\theta \mathbb{E}_{a \sim p_\theta(a)}[R(a, w)] \\ &= \mathbb{E}_{a \sim p_\theta(a)}[R(a, w) \nabla_\theta \log(p_\theta(a))] \ . \end{aligned} \tag{3}$$

Note that REINFORCE does not need to execute a backward process to update θ and (3) can be approximated using the Monte Carlo method with $a \sim p_\theta$.

Different follow-up studies have been conducted to accelerate the optimization of θ. PARSEC (Casale, Gordon, and Fusi 2019) replaces LSTM with a categorical distribution, where $\mu(\theta)$ is a probability vector. ProxylessNAS (Cai, Zhu, and Han 2019) and GDAS (Dong and Yang 2019) aim to incorporate the gradient information $\nabla_{a_i} R(a, w)$ rather than a single reward value $R(a, w)$ to update θ, as in the case of DP-NAS. To avoid dense architecture evaluation, they require different approximation techniques ((Courbariaux, Bengio, and David 2015) and (Maddison, Mnih, and Teh 2017), respectively) and the resulting parameter update procedures require only sparse architecture evaluation.

Characteristics of DP-NAS and SP-NAS The dense forward or backward propagation is a major drawback of DP-NAS. The forward propagation results of all the candidate operations at all the edges need to be obtained to compute $\nabla_\theta R(\mu(\theta), w)$. Thus, the GPU memory consumption in each iteration is approximately proportional to $\sum_{i \in \mathcal{I}} |\mathcal{J}_i|$, which prevents the construction of a large parent network.

A popular approach for overcoming this limitation is to optimize the architecture of a smaller network on a proxy task (Zoph et al. 2018) consisting of a similar to the original dataset but smaller one. For example, CIFAR (Krizhevsky 2009) is used as a proxy task for ImageNet (Deng et al. 2009). The obtained architecture is expanded by repeating it or increasing the number of nodes to construct a larger network. Approaches based on proxy tasks rely on transferability. However, the architectures obtained on the proxy task do not necessarily perform well on other tasks (Dong and Yang 2020). In addition, constructing a proxy task for DP-NAS requires human effort, and the incurred cost prevents practitioners from using NAS in commercial applications.

The advantage of SP-NAS over DP-NAS is that it needs to perform forward (and backward) propagation only in one operation at each edge. Therefore, its GPU memory consumption in each iteration is independent of the number of operations $|\mathcal{J}_i|$ at each edge. As its GPU memory consumption is comparable to that of a single architecture, it does not need any proxy task even when a search space designed for a task requires a large parent network. The advantage of such a search space with a large parent network is demonstrated by, for example, ProxylessNAS (Cai, Zhu, and Han 2019), which achieves SOTA accuracy on ImageNet under mobile latency constraints.

---

[3]NASP performs sparse propagation in the weight update step and dense propagation in the architecture update step.

4 AdvantageNAS

We propose a novel search strategy for one-shot SP-NAS. Inspired by the credit assignment problem of policy gradients, we improve the convergence speed of the REINFORCE algorithm, i.e., the baseline search strategy for ENAS, by introducing the so-called advantage function. As with other SP-NAS, our proposed approach, Advantage-NAS, has low GPU memory consumption.

4.1 Framework

The probability distribution for $\boldsymbol{a}$ is modeled by an independent categorical distribution. Each $\boldsymbol{a}_i$ for $i \in \mathcal{I}$ independently follows a categorical distribution p_{θ_i} parameterized by θ_i. The probability mass function for $\boldsymbol{a}$ is written as the product $p_\theta(\boldsymbol{a}) = \prod_{i \in \mathcal{I}} p_{\theta_i}(\boldsymbol{a}_i)$, where $\theta = (\theta_1, \ldots, \theta_{|\mathcal{I}|})$. The domain of θ_i is $\mathbb{R}^{|\mathcal{J}_i|}$ and the probability of $\boldsymbol{a}_i$ being 1 in the jth category is $\mu_i^j(\theta_i)$ defined by (2).

The objective function of AdvantageNAS is the reward function expected over the architecture distribution, i.e., $J(\theta, \boldsymbol{w}) = \mathbb{E}_{\boldsymbol{a} \sim p_\theta}[R(\boldsymbol{a}, \boldsymbol{w})]$, similar to REINFORCE. It is easy to see that the maximization of $J(\theta, \boldsymbol{w})$ leads to the maximization of $R(\boldsymbol{a}, \boldsymbol{w})$ since $J(\theta, \boldsymbol{w}) \leqslant \max_{\boldsymbol{a}} R(\boldsymbol{a}, \boldsymbol{w})$ and $\sup_\theta J(\theta, \boldsymbol{w}) = \max_{\boldsymbol{a}} R(\boldsymbol{a}, \boldsymbol{w})$, where p_θ represents the Dirac delta function concentrated at $\operatorname{argmax}_{\boldsymbol{a}} R(\boldsymbol{a}, \boldsymbol{w})$.

AdvantageNAS update $\boldsymbol{w}$ and θ alternately. The weights $\boldsymbol{w}$ is updated by estimating the gradient $\nabla_{\boldsymbol{w}} J(\theta, \boldsymbol{w})$ using the Monte Carlo method with an architecture sample $\boldsymbol{a} \sim p_\theta$ and a minibatch reward $r(\boldsymbol{a}, \boldsymbol{w}; D) = -\frac{1}{|D|} \sum_{d \in D} \mathcal{L}(\boldsymbol{a}, \boldsymbol{w}; d)$, where $D \subseteq \mathcal{D}$. Then, θ is updated by estimating the policy gradient $\nabla_\theta J(\theta, \boldsymbol{w})$ using the Monte Carlo method with another architecture sample $\boldsymbol{a} \sim p_\theta$ and a minibatch reward $r(\boldsymbol{a}, \boldsymbol{w}; V)$, where $V \subseteq \mathcal{V}$. Next, we discuss how to estimate the policy gradient to improve the convergence speed while maintaining the time and space complexities at levels similar to those in previous studies. The algorithm of AdvantageNAS is described in Algorithm 1 using pseudocode.

4.2 Estimation Variance of Policy Gradient

The REINFORCE algorithm estimates the gradient using a simple Monte Carlo method,

$$\begin{aligned} \nabla_\theta J(\theta, \boldsymbol{w}) &= \mathbb{E}_{\boldsymbol{a} \sim p_\theta}[R(\boldsymbol{a}, \boldsymbol{w}) \nabla_\theta \log(p_\theta(\boldsymbol{a}))] \\ &\approx r(\boldsymbol{a}, \boldsymbol{w}; V) \nabla_\theta \log(p_\theta(\boldsymbol{a})) \ , \end{aligned} \quad (4)$$

where $\boldsymbol{a}$ is an architecture sampled from p_θ. In general, this gradient is called a *policy gradient*, where p_θ is regarded as a policy that takes no input (i.e., no state) and outputs the probability of selecting each architecture (as an action). For our choice of p_θ, the gradient of the log-likelihood is $\nabla_\theta \log(p_\theta(\boldsymbol{a})) = \boldsymbol{a} - \mu(\theta)$. In other words, the probability vector $\mu(\theta)$ is updated to approach $\boldsymbol{a}$ if the minibatch reward $r(\boldsymbol{a}, \boldsymbol{w}; V)$ is positive. A higher reward results in a greater shift of $\mu(\theta)$ towards current sample $\boldsymbol{a}$.

Intuitively, a high estimation variance will lead to slow convergence. This is indeed theoretically supported by the subsequent result, whose proof follows (Akimoto et al. 2012); the details are presented in Appendix A.1. The lower

---

Algorithm 1 AdvantageNAS

1: **while** Until Search Process End **do**
2:   Sample a minibatch D from $\mathcal{D}$ ▷ $\boldsymbol{w}$ update step
3:   Sample an architecture $\boldsymbol{a} \sim p_\theta$
4:   Compute a minibatch gradient:
       $\delta_{\boldsymbol{w}} \leftarrow \nabla_{\boldsymbol{w}} r(\boldsymbol{a}, \boldsymbol{w}; D)$
5:   Update $\boldsymbol{w}$ with some SGD variant using $\delta_{\boldsymbol{w}}$
6:   Sample a minibatch V from $\mathcal{V}$ ▷ θ update step
7:   Sample an architecture $\boldsymbol{a} \sim p_\theta$
8:   Compute minibatch advantages using Equation (7):
       $\bar{A}_i \leftarrow A_i(\boldsymbol{a}, \boldsymbol{w}; V)$ for all $i \in \mathcal{I}$
9:   Compute minibatch gradients:
       $\delta_{\theta_i} \leftarrow \bar{A}_i \nabla_{\theta_i} \log(p_\theta(\boldsymbol{a}))$ for all $i \in \mathcal{I}$
10:   Update θ_i with some SGD variant using δ_{θ_i} $(\forall i \in \mathcal{I})$
11: **end while**

---

the estimation variance $\operatorname{Var}_\delta[\delta]$ of policy gradient δ, the greater is the lower bound of the expected improvement in one step. Therefore, reducing the estimation variance is key to speed up the architecture update.

Theorem 4.1. *Let* $J(\theta, \boldsymbol{w}) = \mathbb{E}[R(\boldsymbol{a}, \boldsymbol{w})]$ *and* δ *be an unbiased estimator of* $\nabla_\theta J(\theta, \boldsymbol{w})$. *Suppose that* $R(\boldsymbol{a}, \boldsymbol{w}) > 0$ *for any* $\boldsymbol{a}$. *Then, for any* $\epsilon > 0$,

$$\mathbb{E}_\delta[\log J(\theta + \epsilon \cdot \delta, \boldsymbol{w})] - \log J(\theta, \boldsymbol{w}) \geqslant$$
$$\left(\frac{\epsilon}{J(\theta, \boldsymbol{w})} - \frac{\epsilon^2}{2} \right) \|\nabla_\theta J(\theta, \boldsymbol{w})\|^2 - \frac{\epsilon^2}{2} \operatorname{Var}_\delta[\delta] \ . \quad (5)$$

Credit assignment in reinforcement learning tasks is a well-known cause of slow convergence of policy gradient methods (Minsky 1961). The reward is the result of a set of consecutive actions (in our case, the choice $\boldsymbol{a}_i$ at each edge $i \in \mathcal{I}$). A positive reward for $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{|I|})$ does not necessarily mean that all the actions $\boldsymbol{a}_i$ are promising. However, since (4) updates θ based on a single reward, it cannot recognize the contribution of each edge. To understand this problem, consider a situation in which we have two edges with two candidate operations for each edge. The reward is assumed to be the sum of the contributions of the choices on the two edges, r_i^j for $\boldsymbol{a}_i = \boldsymbol{e}_j$ for $i = 1$ and 2 and $j = 1$ and 2. If $r_1^1 \gg r_2^1 > 0$ and $r_1^2 \ll r_2^2 < 0$, i.e., the contribution of the first edge dominates, the sign of the reward is determined by $\boldsymbol{a}_1$. The update direction of $\mu_2(\theta_2)$ is determined by $\boldsymbol{a}_1$; it is not based on whether $\boldsymbol{a}_2$ is a better choice. This results in a high variance for the $\mu_2(\theta_2)$ update, leading to slow convergence.

4.3 Advantage

We introduce the notion of *advantage*, used in policy gradient methods, to address the issue of credit assignment (Minsky 1961). We regard $\boldsymbol{s}_i = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}, \boldsymbol{a}_{i+1}, \ldots, \boldsymbol{a}_{|I|})$ as a state and $\boldsymbol{a}_i$ as an action. By introducing the advantage, we evaluate the goodness of the choice $\boldsymbol{a}_i$ at each edge i rather than evaluate the goodness of successive actions $\boldsymbol{a}$ by a scalar reward r. We expect to reduce the estimation variance of the policy gradient by the advantage.

We define the advantage of $\boldsymbol{a}_i$ as (by letting $\boldsymbol{a} = (\boldsymbol{s}_i, \boldsymbol{a}_i)$)

$$A_i(\boldsymbol{a}, \boldsymbol{w}; V) = r(\boldsymbol{a}, \boldsymbol{w}; V) - r((\boldsymbol{s}_i, \boldsymbol{0}), \boldsymbol{w}; V) \ . \quad (6)$$

It is understood as follows. Given the state $\boldsymbol{s}_i$, the advantage of the action $\boldsymbol{a}_i$ is the rewards improvement over $\boldsymbol{a} = (\boldsymbol{s}_i, \boldsymbol{0})$, where the output of the ith edge is zero.[4] We replace the reward with the advantage in the policy gradient estimation in (4). If we replace the second term on the right-hand side of (6) with a single scalar independent of $\boldsymbol{s}_i$, the resulting policy gradient is known to be an unbiased estimator (Evans and Swartz 2000). However, a state-dependent advantage such as (6) is not unbiased in general. The following result states that the policy gradient estimate using this advantage is an unbiased estimate of (4). Its proof is presented in Appendix A.2.

Theorem 4.2. *Let $\boldsymbol{a} \sim p_\theta(\boldsymbol{a})$ and $d \in V$ be i.i.d. from $\mathcal{P}$. Then, with A_i defined in (6), $A_i(\boldsymbol{a}, \boldsymbol{w}; V)\nabla_{\theta_i} \ln p_\theta(\boldsymbol{a})$ is an unbiased estimate of $\nabla_{\theta_i} J(\theta, \boldsymbol{w})$, i.e., $\nabla_{\theta_i} J(\theta, \boldsymbol{w}) = \mathbb{E}_{\boldsymbol{a}, V}[A_i(\boldsymbol{a}, \boldsymbol{w}; V)\nabla_{\theta_i} \ln p_\theta(\boldsymbol{a})]$.*

Thus, we can use $|\mathcal{I}|$ different advantage values rather than a single reward value. A shortcoming of the above approach is that it requires the computation of an additional reward to compute $r((\boldsymbol{s}_i, \boldsymbol{0}), \boldsymbol{w}; V)$ for each $i \in \mathcal{I}$; hence, it requires $|\mathcal{I}|$ additional forward propagation to estimate (4). As our objective is to reduce the time complexity by improving the convergence speed of SP-NAS, we aim to compute an advantage with ideally no additional computational cost.

To reduce the computational cost, we approximate the advantage function (6) as (dropping $\boldsymbol{w}$ and V for short)

$$A_i(\boldsymbol{a}) \approx \nabla_{\boldsymbol{a}_i} r(\boldsymbol{a})|_{\boldsymbol{a}=(\boldsymbol{s}_i, \boldsymbol{a}_i)}^{\mathrm{T}} \boldsymbol{a}_i = \frac{\partial r(\boldsymbol{a})}{\partial \mathcal{O}_i^{\mathrm{T}}} \mathcal{O}_i^j(x_{Q(i)}) \ , \quad (7)$$

where $\mathcal{O}_i$ and $\mathcal{O}_i^j$ are as in (1) and $j \in \mathcal{J}_i$ is the index of the selected operation at edge i, i.e., $\boldsymbol{a}_i$ is the one-hot vector with 1 at jth. Also note that here we consider the situation where $\mathcal{O}_i$ and $\mathcal{O}_i^j$ are column vectors and the batch size equals 1 for simplicity. The above approximation follows the first-order Taylor approximation of the reward around $\boldsymbol{a}$. Compared to a naive policy gradient computation that uses only a single reward $r(\boldsymbol{a}, \boldsymbol{w}; V)$, the policy gradient estimate with the above approximate advantage additionally requires only one reward gradient computation, $\nabla_{\mathcal{O}_i} r(\boldsymbol{a}, \boldsymbol{w}; V)^{\mathrm{T}} \mathcal{O}_i^j(x_{Q(i)}; w_i^j)$. The output $\mathcal{O}_i^j(x_{Q(i)}; w_i^j)$ of the selected operation is computed in the forward process and $\nabla_{\mathcal{O}_i} r(\boldsymbol{a}, \boldsymbol{w}; V)$ is computed in the backward process. Therefore, our advantage involves no additional computational cost.

---

[4]When a zero operation (returns zero consistently) is included in the search space, we replace the advantage of the zero operation with the worst expected advantage of the other operations estimated by a moving average. The rationale for this modification is as follows. First, (6) for the zero operation is always zero, resulting in no architecture update, which is not preferred. Second, the zero operation can mostly be represented by other operations with specific weights. Hence, there is no reason to prefer it unless we consider a latency constraint. See Appendix B for further details.
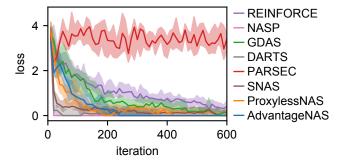
AdvantageNAS uses the gradient information of the reward rather than a scalar reward. Existing works on SP-NAS also use the gradient (Cai, Zhu, and Han 2019; Dong and Yang 2019). AdvantageNAS aims to reduce the estimation variance while keeping the policy gradient unbiased, whereas the previous methods simply try to incorporate the gradient information of the reward and they do not consider the estimation bias.

4.4 Variance Reduction Effect : An Example

We know from Theorem 4.2 that the advantage (6) provides an unbiased estimator of the gradient $\nabla_\theta J(\theta, \boldsymbol{w})$ and from Theorem 4.1 that reducing the variance in an unbiased estimator of the gradient contributes toward improving the lower bound of the expected improvement. Since (7) is an approximation of (6), we expect that our advantage contributes toward reducing the estimation variance, resulting in faster convergence.

To understand the variance reduction effect of the proposed advantage, we consider a simple situation in which the reward is given by $\boldsymbol{r}^{\mathrm{T}} \boldsymbol{a} = \sum_{i=1}^{|\mathcal{I}|} \boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{a}_i$ for some vector $\boldsymbol{r} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{\mathcal{I}})$. In this example, neither $\mathcal{V}$ nor $\boldsymbol{w}$ is introduced. In this linear objective situation, the advantages (6) and (7) are equivalent and $A_i(\boldsymbol{a}) = \boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{a}_i$. The square norm of the gradient component $\nabla_{\theta_i} J(\theta)$ is $\sum_{j \in \mathcal{J}_i} (r_i^j)^2 (\mu_i^j)^2 (1 - \mu_i^j)^2$. The estimation variance of the policy gradient δ_{adv} using the advantage (6) or (7) is $\mathrm{Var}[\delta_{\mathrm{adv}}] = \sum_j \mu_i^j (r_i^j)^2 (1 - 2\mu_i^j + \|\mu_i\|^2)$. Meanwhile, the estimation variance of the policy gradient δ_{base} using a reward value (i.e., (4)) is $\mathrm{Var}[\delta_{\mathrm{base}}] = \mathrm{Var}[\delta_{\mathrm{adv}}] + (1 - \|\mu_i\|^2) \sum_{k \neq i} \sum j (r_k^j)^2 \mu_k^j$. In other words, the variance reduction effect for each gradient component typically scales linearly with $|\mathcal{I}|$ (because $\sum_{k \neq i}$), whereas $\|\nabla_{\theta_i} J(\theta)\|^2$ and $\mathrm{Var}[\delta_{\mathrm{adv}}]$ do not. In light of Theorem 4.1, the proposed advantage potentially contributes toward improving the search performance, especially when the number of edges is $|\mathcal{I}| \gg 1$.

5 Experiments

We performed three experiments to compare the search efficiency of AdvantageNAS with that of existing gradient-based search strategies for NAS. Note that our objective is not to obtain the SOTA architecture; hence, the search spaces used in the experiments were relatively small for convenience. To compare the search strategies for NAS fairly, we implemented all the baseline NAS methods on the same code base and imposed the same setting except for the search strategy.

5.1 Toy Regression Task

Motivation Whereas one-shot NAS methods optimize the weights and architecture simultaneously, to solely assess the performance improvement due to the differences in the architecture updates, we evaluate the performance of the architecture optimization while the weights are fixed. In particular, we confirm that the variance reduction by our advantage contributes toward accelerating the architecture search

Figure 1: Results of the toy regression task. Median (solid line) and 25th—75th percentile area (colored area) over 10 trials.
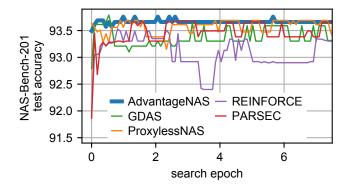

Figure 2: Improvement progress of search process. All the lines represent the median for three search trials. The values are obtained by querying NAS-Bench-201.

by comparing the performance of REINFORCE and AdvantageNAS, which are different only in terms of the reward computation for each edge as described in Section 4.3.

Task Description We prepare two networks: teacher-net $\phi_{\boldsymbol{a}^*}$ and student-net $\phi_{\boldsymbol{a}}$. Both networks are represented by the same DAG. The two networks have $|\mathcal{I}| = 10$ intermidiate nodes, and there are edges from one input node to all the intermidiate nodes. For each edge $i \in \mathcal{I}$, $|\mathcal{J}_i| = 10$ different convolutional filters are prepared as the candidate operations. These filters have a filter size of 7×7, stride of 2, and output channels of 1 followed by $\tanh$ activation. The architecture variable $\boldsymbol{a}^*$ of teacher-net is such that $\boldsymbol{a}_i^*$ are one-hot vectors, i.e., one convolutional filter per edge which connects the input node and the intermediate node. For each edge from an intermediate node to the output node, a convolution with filter size of 4×4, stride of 1, and output channels of 1 is applied. The average of the outputs is taken as the aggregation function for the output node. A minibatch V with batch size of 100 is sampled randomly for each iteration, where the input data $\boldsymbol{x}$ is generated by the uniform distribution over $[-1, +1]^{13 \times 13 \times 1}$ and its ground truth is given by $\phi_{\boldsymbol{a}^*}(\boldsymbol{x})$. The reward function is defined as the negative loss; the squared error between the ground truth $\phi_{\boldsymbol{a}^*}(\boldsymbol{x})$ and the student prediction $\phi_{\boldsymbol{a}}(\boldsymbol{x})$. The weights for the convolutions are fixed to be the same ($\boldsymbol{w}^*$) for both networks during the architecture optimization process. The $\boldsymbol{w}^*$ and $\boldsymbol{a}^*$ are defined randomly before the optimization process. The loss is consistently zero if and only if student-net learns the optimal architecture $\boldsymbol{a}^*$. We optimize θ using the Adam optimizer (Kingma and Ba 2015) with a learning rate of 0.001 and $\beta = (0.9, 0.999)$. The implementation details of gradient-based methods are presented in Appendix C.

Results Figure 1 shows the results. The test loss is computed for the architecture obtained by selecting the operation at each edge such that $\mu_i(\theta_i)$ is maximum. We observe that AdvantageNAS converges faster than existing SP-NAS such as REINFORCE, PARSEC and GDAS. Moreover, it is more stable than ProxylessNAS when the number of iterations is large. Existing DP-NAS approaches such as DARTS, SNAS and NASP converge faster than AdvantageNAS. However, as mentioned above, these methods require nearly $|\mathcal{J}|$ times the memory space to perform the architecture search, which

prevents their application to a large-scale task requiring a large architecture.

5.2 Convolutional Cell on CIFAR-10

Motivation We compare AdvantageNAS with existing SP-NAS by searching for a convolutional cell on the CIFAR-10 (Krizhevsky 2009) dataset and evaluating them on the NAS-Bench-201 (Dong and Yang 2020).

Search Settings The CIFAR-10 dataset contains 50000 training data and 10000 test data. We split the original training data into 25000 training data $\mathcal{D}$ and 25000 validation data $\mathcal{V}$, as in the case of NAS-Bench-201. We use $\mathcal{D}$ to optimize $\boldsymbol{w}$ and $\mathcal{V}$ to optimize θ. Our architecture search space is the same as NAS-Bench-201 except that the zero operations are removed.[5] We search for the convolutional cell architecture and stack it 15 times to construct a macro skeleton. All the stacked cells have same architecture. The cell structure has a total of $|\mathcal{I}| = 6$ edges. In each cell, there are $|\mathcal{J}| = 5$ candidate operations such as 1×1 convolution, 3×3 convolution, 3×3 average pooling and identity map. The total procedure was conducted in 100 epochs; however, θ was fixed in the first 50 epochs for pre-training $\boldsymbol{w}$. The other settings are presented in Appendix D.

Evaluation Settings To evaluate the performance of one-shot NAS, the weights for the architectures obtained by the search process are often retrained for longer epochs; then, the test performances are compared. To minimize the influence of the quality of retraining on the performance metric, we use NAS-Bench-201 (Dong and Yang 2020) for evaluation. It is a database containing average accuracy for all architectures in the search space. The accuracy was computed after 200 epochs of training on CIFAR-10 with each architecture. Each architecture was trained three times independ-

---

[5]Our preliminary experiments showed that zero operations disrupt the architecture search of existing SP-NAS (e.g., ProxylessNAS ended up with a final accuracy of 10%). We removed zero operations to avoid the effects of search space definition. See Appendix F for the results on the search space including the zero operations.

Method	argmax		process best	
	accuracy	params(MB)	accuracy	params(MB)
REINFORCE(Pham et al. 2018; Williams 1992)	93.01 ± 0.47	0.67 ± 0.18	92.69 ± 0.52	0.68 ± 0.20
PARSEC(Casale, Gordon, and Fusi 2019)	93.68 ± 0.12	1.39 ± 0.20	92.51 ± 0.65	0.47 ± 0.12
ProxylessNAS(Cai, Zhu, and Han 2019)	93.22 ± 0.08	0.74 ± 0.10	-	-
GDAS(Dong and Yang 2019)	93.46 ± 0.13	1.24 ± 0.10	92.53 ± 0.98	0.62 ± 0.20
AdvantageNAS	$\mathbf{93.76 \pm 0.00}$	1.53 ± 0.00	$\mathbf{93.29 \pm 0.21}$	0.94 ± 0.11

Table 2: Results on CIFAR-10. Average±std accuracy over three search trials as well as the parameter size associated with the final architecture. Note that `process best` for ProxylessNAS is missing as it samples two operations at each edge and the one-hot architecture is never evaluated during the architecture search process, as described in Table 1.

Method	validation perplexity	test perplexity	params(MB)
REINFORCE(Pham et al. 2018; Williams 1992)	99.48 ± 1.51	95.50 ± 1.73	7.40 ± 0.07
PARSEC(Casale, Gordon, and Fusi 2019)	99.68 ± 1.49	95.51 ± 1.01	7.33 ± 0.09
ProxylessNAS(Cai, Zhu, and Han 2019)	95.12 ± 0.22	$\mathbf{90.76 \pm 0.23}$	7.33 ± 0.09
GDAS(Dong and Yang 2019)	99.28 ± 2.00	95.40 ± 2.01	7.33 ± 0.09
AdvantageNAS	$\mathbf{94.90 \pm 0.18}$	91.18 ± 0.13	7.44 ± 0.00

Table 3: Results on PTB. Average±std accuracy over five search trials as well as the parameter size associated with the final architecture.

dently and its average accuracy is recorded in the database. We use this average accuracy for the performance metric.

There are different ways for determining the final architecture after architecture search. One way is to choose the best architecture based on its validation loss observed in the search process. This way is adopted by, e.g., ENAS (REINFORCE) and GDAS, and is denoted as `process best`. Another way (adopted by ProxylessNAS and PARSEC) is to take operation j for which $\mu_i^j(\theta_i)$ is maximum at each edge i, as in Section 5.1. It is denoted as `argmax`. In our experiment, we show the performance of these two ways and confirmed which one is better suited for AdvantageNAS.

Results Table 2 presents the results. AdvantageNAS achieves an improvement over REINFORCE solely owing to the introduction of the advantage described in Section 4.3. Moreover, AdvantageNAS outperforms all the existing SP-NAS in terms of both `argmax` and `process best`. Although `process best` was adopted in ENAS and GDAS, `argmax` delivered better performances in all the algorithms including AdvantageNAS. A comparison with DP-NAS is shown in Appendix G for reference.

The search progress in the initial steps is shown in Figure 2. The median of the average accuracy on NAS-Bench-201 is shown for each method with `argmax` over three independent search trials. At the early stage in the search process, AdvantageNAS achieved a higher accuracy than that of the existing SP-NAS and was more stable at the high accuracy. It can be inferred that AdvantageNAS outperforms the existing SP-NAS in terms of convergence speed.

To further assess the convergence speed, we analyzed the entropy of categorical distribution $\mu(\theta)$ in the search process. The details are presented in Appendix H.

5.3 Recurrent Cell on Penn Treebank

Motivation We confirm that the usefulness of AdvantageNAS for recurrent cell architecture search as well as for con-

volutional cells.

Settings We use Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993) (PTB) dataset to search for the recurrent cell architecture. Most of the settings is the same as those of DARTS (Liu, Simonyan, and Yang 2019). See Appendix E for the details. The recurrent cell structure has $|\mathcal{I}| = 6$ edges. They have $|\mathcal{J}| = 5$ operation candidates, namely ReLU, tanh, sigmoid, identity, and zero operation, after linear transformation is applied. The parameters of the linear transformation are trainable weights. In the search process, we use the training data of PTB to optimize both the weights and θ for 30 epochs. Finally, the obtained architectures are retrained in 100 epochs. According to previous studies, we chose the final architecture by `argmax` in ProxylessNAS and PARSEC and by `process best` for REINFORCE(ENAS) and GDAS. The final architecture of AdvantageNAS is determined by `argmax`. The validation and test data of PTB are used to measure the final performance.

Results The performances of the retrained architectures are shown in Table 3. AdvantageNAS achieved highest performance in validation data. In addition, AdvantageNAS achieved the competitive result obtained by ProxylessNAS in test data. This result implies that AdvantageNAS can be applied to various domains. The architectures obtained by AdvantageNAS and ProxylessNAS were similar, as shown in Appendix I.

6 Conclusion

We proposed AdvantageNAS, a novel gradient-based search strategy for one-shot NAS. AdvantageNAS aims to accelerate the architecture search process while maintaining the GPU memory consumption as in the case of sparse propagation NAS. For this purpose, we introduced the notion of *credit assignment* into the architecture search to reduce the variance of gradient estimation. A unique aspect of our

study is the theoretical guarantee of the monotonic improvement of the expected loss by AdvantageNAS. Empirical evaluations were performed on different datasets and search spaces, confirming the promising performance of AdvantageNAS. To compare only the search strategies, we implemented the existing approaches on the same publicly available code base. To make an important contribution to the NAS community, we focused on investigating only the efficacy of the search strategy. Finding SOTA architectures, which requires searching for best combinations of different NAS components, is beyond the scope of this study. Further investigation of larger search spaces is an important direction for future research.

Acknowledgements

References

Akimoto, Y.; Nagata, Y.; Ono, I.; and Kobayashi, S. 2012. Theoretical Foundation for CMA-ES from Information Geometry Perspective. *Algorithmica* 64(4): 698–716. ISSN 0178-4617.

Cai, H.; Zhu, L.; and Han, S. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*.

Casale, F. P.; Gordon, J.; and Fusi, N. 2019. Probabilistic Neural Architecture Search. *CoRR* arXiv:1902.05116.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In *NIPS*, 3123–3131.

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

Dong, X.; and Yang, Y. 2019. Searching for A Robust Neural Architecture in Four GPU Hours. In *CVPR*, 1761–1770.

Dong, X.; and Yang, Y. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *ICLR*.

Evans, M.; and Swartz, T. 2000. *Approximating integrals via Monte Carlo and deterministic methods*. Oxford University Press.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation* 9(8): 1735–1780.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Konda, V. R.; and Tsitsiklis, J. N. 2000. Actor-Critic Algorithms. In *NIPS*, 1008–1014.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.; Fei-Fei, L.; Yuille, A. L.; Huang, J.; and Murphy, K. 2018.

Progressive Neural Architecture Search. In *ECCV*, volume 11205 of *Lecture Notes in Computer Science*, 19–35.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *ICLR*.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *ICLR*.

Marcus, M. P.; Santorini, B.; and Marcinkiewicz, M. A. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19(2): 313–330.

Minsky, M. 1961. Steps toward Artificial Intelligence. *Proceedings of the IRE* 49(1): 8–30.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*, volume 80 of *PMLR*, 4092–4101.

Real, E.; Moore, S.; Selle, A.; Saxena, S.; Suematsu, Y. L.; Tan, J.; Le, Q. V.; and Kurakin, A. 2017. Large-Scale Evolution of Image Classifiers. In *ICML*, volume 70 of *PMLR*, 2902–2911.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8(3): 229–256.

Xie, S.; Zheng, H.; Liu, C.; and Lin, L. 2019. SNAS: stochastic neural architecture search. In *ICLR*.

Yang, A.; Esperança, P. M.; and Carlucci, F. M. 2020. NAS evaluation is frustratingly hard. In *ICLR*.

Yao, Q.; Xu, J.; Tu, W.; and Zhu, Z. 2020. Efficient Neural Architecture Search via Proximal Iterations. In *AAAI*, 6664–6671.

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *CVPR*, 8697–8710.