Rethinking Bi-Level Optimization in Neural Architecture Search: A Gibbs Sampling Perspective

Chao Xue[1], **Xiaoxing Wang**[2], **Junchi Yan**[2]*, **Yonggang Hu**[3], **Xiaokang Yang**[2], **Kewei Sun**[1]
[1] IBM Research – China [2] Shanghai Jiao Tong University [3] IBM System
{xuechao,sunkewei}@cn.ibm.com, {figure1_wxx,yanjunchi,xkyang}@sjtu.edu.cn, yhu@ca.ibm.com

Abstract

One-Shot architecture search, aiming to explore all possible operations jointly based on a single model, has been an active direction of Neural Architecture Search (NAS). As a well-known one-shot solution, Differentiable Architecture Search (DARTS) performs continuous relaxation on the architecture's importance and results in a bi-level optimization problem. As many recent studies have shown, DARTS cannot always work robustly for new tasks, which is mainly due to the approximate solution of the bi-level optimization. In this paper, one-shot neural architecture search is addressed by adopting a directed probabilistic graphical model to represent the joint probability distribution over data and model. Then, neural architectures are searched for and optimized by Gibbs sampling. We rethink the bi-level optimization problem as the task of Gibbs sampling from the posterior distribution, which expresses the preferences for different models given the observed dataset. We evaluate our proposed NAS method – GibbsNAS on the search space used in DARTS/ENAS as well as the search space of NAS-Bench-201. Experimental results on multiple search space show the efficacy and stability of our approach.

Introduction

Automated design of a deep neural network for specific tasks (often referred as NAS – network architecture search) has attracted wide attention for its state-of-the-art performance in different learning tasks e.g. computer vision and natural language. A series of mechanisms are designed for NAS, such as evolutionary based approaches (Real et al. 2017; Xie and Yuille 2017), random search based algorithms (Bergstra and Bengio 2012; Li, Jamieson, and DeSalvo 2017), Bayesian optimization (Domhan, Springenberg, and Hutter 2015; Zhou et al. 2019), reinforcement learning based methods (Zoph and Le 2017; Baker et al. 2017) and differentiable solutions (Liu, Simonyan, and Yang 2019; Zela et al. 2020).

To reduce the cost of the searching phase in NAS, many works have been devised, including evaluation prediction (Liu et al. 2018a; Baker et al. 2018), collaborative searching (Lindauer and Hutter 2018; Xue et al. 2019), weight sharing (Pham et al. 2018), and one-shot model (Liu, Simonyan, and Yang 2019; Bender et al. 2018). In (Baker et al. 2018), candidate network architectures and early observed test/validation accuracy are taken as inputs of an LSTM model, which is used to predict the final accuracy of those architectures. When multiple tasks are involved, collaborative searching by transferring knowledge across different tasks can be adopted to improve search efficiency, such as multi-task Bayesian optimization (Swersky, Snoek, and Adams 2013; Bardenet et al. 2013), warm-start schemes (Lindauer and Hutter 2018; Feurer et al. 2015), and hard decision solutions (Xue et al. 2019). On the other hand, the widely-used weight sharing technologies like ENAS (Pham et al. 2018), RSPS (Li and Talwalkar 2019) share weights among all child architectures to reduce the computing from thousands of GPU days to a single one. Similarly, one-shot model trains a single model that contains all possible operations. While differentiable architecture search (DARTS) (Liu, Simonyan, and Yang 2019) makes a continuous relaxation on the architecture's importance with an approximate solution of bi-level optimization.

Recently, evaluation of different search strategies is built on some benchmarks (Ying et al. 2019a,b; Dong and Yang 2020) with identical search space and same settings such as learning rate scheme, data augmentation and auxiliary network, etc. In (Yu et al. 2020), the authors evaluate different search strategies on NAS-Bench-101 (Ying et al. 2019b), and find that the weight sharing strategy degrades the final test performance. Furthermore, NAS-Bench-201 (Dong and Yang 2020) extends NAS-Bench-101 (Ying et al. 2019b), and provides a unified benchmark for almost any up-to-date NAS algorithms. Unfortunately, the promising differential method, DARTS (Liu, Simonyan, and Yang 2019), fails in NAS-Bench-201 (Dong and Yang 2020). Some works (Zela et al. 2020) argue that the approximation of the bi-level optimization degrades the final performance.

This paper focuses on one-shot neural architecture search, and proposes a novel method called GibbsNAS. The contributions of the paper can be summarized as follows:

1) Uncertainty-Aware NAS by Gibbs Sampling. Unlike previous differentiable NAS methods solving bi-level optimization problem by gradient descent, we provide a method to calculate the posterior distribution that expresses the preferences for different models given observed dataset, and propose an uncertainty-aware approach by using Gibbs sampling from the posterior distribution, which yields competitive performance compared to existing techniques. To our best knowledge, this is the first work to develop Gibbs sam-

*The correspondence author is Junchi Yan.

pling approach to NAS, and it also provides a principled view to the bi-level optimization problem in one-shot NAS.

2) Practical Second Order Optimization Implementation. To derive the posterior distribution that expresses the preference for different models given the observed dataset, we refer to the Quasi-Newton's Method. To reduce memory and computation costs, we implement a Hessian inverse approximation method, which includes second-order derivatives and statistics. The proposed practical solution makes second-order optimization feasible to implement and deploy.

3) Empirical Study. On one hand, it is highly recommended to evaluate NAS strategy on different search space and across multiple datasets. On the other hand, comparing different search strategies within the same search space and setting (learning rate policy, cutout, auxiliary network, drop path probability, etc.) is also essential. Based on these considerations, GibbsNAS is evaluated on the search space used in DARTS/ENAS as well as NAS-Bench-201 (Dong and Yang 2020) which is built on multiple datasets. Experimental results show the efficacy and stability of our approach.

Preliminaries and Related Work

Our work is closely related to one-shot architecture search, differentiable architecture search with bi-level optimization and Bayesian approach for architecture search.

One-Shot NAS and Differential Method with Bi-Level Optimization. The work (Bender et al. 2018) pioneers the direction of training a single model containing every possible operation in the search space which is referred as one-shot NAS. In this original paper, the feature maps are added together without considering different weights on the operations. DARTS (Liu, Simonyan, and Yang 2019) makes a continuous relaxation on the architecture's importance and model the search process as a bi-level optimization problem shown in outer objective Eq. 1 and inner objective Eq. 2, both of which are optimized iteratively by a single gradient descent step shown in Fig. 1a. Depending on the approximation method for outer variables' gradients, there are two versions for DARTS: first order and second order. DARTS attracts a lot of attentions and has become a promising way to solve neural architecture search due to its good performance on the similar search space of ENAS (Pham et al. 2018) and its small search cost. Recent works (Chen et al. 2019; Xu et al. 2020; Wang et al. 2020) are further dedicated to improving the performance of DARTS and reducing the search cost. However, most recent research works (Zela et al. 2020; Dong and Yang 2020; Yu et al. 2020) show that DARTS does not work robustly for new tasks and new search space. Recent work (Zela et al. 2020) points that the reason that DARTS (Liu, Simonyan, and Yang 2019) does not work robustly for new tasks is mainly due to its approximation of the bi-level optimization. Consequently, the work (Zela et al. 2020) proposes an early stop mechanism as well as a regularization scheme to improve the generalization properties of DARTS (Liu, Simonyan, and Yang 2019). In our paper, we provide a new perspective on bi-level optimization for one-shot NAS: using Gibbs Sampling to update architecture importance and network weights iteratively, which is shown in Fig. 1b.

$$\min_{a} \quad \mathcal{L}_{val}(\boldsymbol{\omega}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}) \tag{1}$$

$$\text{s.t.} \quad \boldsymbol{\omega}^*(\boldsymbol{\alpha}) = \operatorname*{argmin}_{\boldsymbol{\omega}} \mathcal{L}_{train}(\boldsymbol{\omega}, \boldsymbol{\alpha}) \tag{2}$$

Bayesian Learning for NAS. Bayesian approach has been utilized to find proper hyper-parameters for a few decades. Many works follow the Bayesian optimization framework to design hyper-parameters search algorithms (Bergstra et al. 2011; Domhan, Springenberg, and Hutter 2015). There is also a line of work falling into the combination of Hyperband (Li, Jamieson, and DeSalvo 2017) and Bayesian optimization. The work (Falkner, Klein, and Hutter 2018b) replaces the random selection of configurations at the beginning of each Hyperband iteration with a Tree Parzen Estimator (TPE) (Bergstra et al. 2011). Although neural architecture search can be viewed as hyper-parameters search with structure encoding, the unique characteristics of neural architecture search should also be taken into account. For instance, to adjust discrete search space, the method in (Kandasamy et al. 2018) derives the so-called OTMANN distance to measure the difference between the structures of two networks, and then the Gaussian process based methods (Snoek, Larochelle, and Adams 2012) can be applied to architecture search. BayesNAS (Zhou et al. 2019) proposes a Bayesian approach for one-shot NAS. It uses priors to build dependencies among the nodes and then forms an efficient graph after pruning. Therefore, it can also be applied to compress convolutional neural networks. In our work, we derive a data-model joint distribution, and then use Bayes' rule to calculate the conditional distribution. We also show the relationship between our Gibbs Sampling method and a non-weight-sharing Bayesian solution in terms of model evidence in Section (GibbsNAS vs. Bayesian Model Evidence).

GibbsNAS: Gibbs Sampling Based Neural Architecture Search

One-shot NAS can be considered as a generative probability graphical model. Based on Bayesian networks (Friedman, Geiger, and Goldszmidt 1997), one can represent the joint probability distribution over data and model[1]:

$$p(\mathcal{D}, \boldsymbol{\omega}, \boldsymbol{\alpha}|\sigma_\alpha^2, \sigma_\omega^2) = p(\boldsymbol{\alpha}|\sigma_\alpha^2) \cdot p(\boldsymbol{\omega}|\sigma_\omega^2) \cdot p(\mathcal{D}|\boldsymbol{\alpha}, \boldsymbol{\omega}) \tag{3}$$

The joint probability can be viewed as the product of the likelihood for the architecture importance $\boldsymbol{\alpha}$ and the network weights $\boldsymbol{\omega}$ with data $\mathcal{D}$ and their priors. Similar to the work (Zhou et al. 2019), we suppose that the prior distributions of $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ are independent. It should be noted that the independence of posterior distributions does not hold because $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ are not d-separated when the data is observed. Also, we follow the previous works (Zhou et al. 2019; Kandasamy et al. 2018) to choose the prior distributions of the architecture importance and the network weights to be Gaussian $p(\boldsymbol{\omega}|\sigma_\omega^2) = \mathcal{N}(\boldsymbol{\omega}|\mathbf{0}, \sigma_\omega^2\mathbf{I})$, $p(\boldsymbol{\alpha}|\sigma_\alpha^2) = \mathcal{N}(\boldsymbol{\alpha}|\mathbf{0}, \sigma_\alpha^2\mathbf{I})$. By using Bayesian rules, the posterior distribution $p(\boldsymbol{\omega}, \boldsymbol{\alpha}|\mathcal{D}, \sigma_\alpha^2, \sigma_\omega^2)$, which expresses the preference for different models given the observed

[1]In one-shot NAS, a model contains two parts: architecture importance $\boldsymbol{\alpha}$, and network weights $\boldsymbol{\omega}$.

dataset, can be derived. Then we apply Gibbs sampling from the distribution to update $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ iteratively. Therefore, neural architectures are searched for and optimized in a Gibbs sampling manner – see Fig. 1b. In the t^{th} step of the Gibbs sampling procedure, $\boldsymbol{\omega}_t$ is replaced by a value drawn from the posterior distribution of $\boldsymbol{\omega}$ conditioned on $\boldsymbol{\alpha}_t$; in the $(t+1)^{\text{th}}$ step, $\boldsymbol{\alpha}_{t+1}$ is replaced by a value drawn from the posterior distribution of $\boldsymbol{\alpha}$ conditioned on $\boldsymbol{\omega}_t$, and the procedure is repeated until the search budget is reached. Our work focuses on the sampling procedure of $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$, and thus the noise precisions σ_α^{-2} and σ_ω^{-2} are viewed as deterministic values (importance decays and weight decays). But it is easy to extend them as stochastic variables (with conjugate priors (Zhou et al. 2019)) and then they can also be sampled and replaced iteratively in the Gibbs sampling procedure.

Now, the posterior distribution of $\boldsymbol{\omega}$ conditioned on $\boldsymbol{\alpha}_t$ is calculated with $\sigma^2 = \{\sigma_\alpha^2, \sigma_\omega^2\}$.

$$p(\boldsymbol{\omega}|\mathcal{D}, \boldsymbol{\alpha}_t, \sigma^2) = \frac{p(\mathcal{D}, \boldsymbol{\omega}|\boldsymbol{\alpha}_t, \sigma_\omega^2) \cdot p(\boldsymbol{\alpha}_t|\sigma_\alpha^2)}{p(\mathcal{D}, \boldsymbol{\alpha}_t|\sigma_\alpha^2)} \tag{4}$$

$$\ln p(\boldsymbol{\omega}|\mathcal{D}, \boldsymbol{\alpha}_t, \sigma^2) = \ln p(\mathcal{D}, \boldsymbol{\omega}|\boldsymbol{\alpha}_t, \sigma_\omega^2) + \text{const} \tag{5}$$

Any term on the R.H.S not dependent on $\boldsymbol{\omega}$ can be absorbed into the additional constant. Considering one-shot NAS (Liu, Simonyan, and Yang 2019) and weight sharing NAS (Pham et al. 2018), where the network weights $\boldsymbol{\omega}$ at the t^{th} step are generated from those at the $(t-1)^{\text{th}}$ step as shown in the Fig. 1a and Fig. 1b, we therefore apply a Taylor expansion of the first term on the right-hand side in Eq. 5 centered on its preceding weights $\boldsymbol{\omega}_{t-1}$ with Peano remainder, so that

$$\begin{aligned}\ln p(\mathcal{D}, \boldsymbol{\omega}|\boldsymbol{\alpha}_t, \sigma_\omega^2) &= \ln p(\mathcal{D}, \boldsymbol{\omega}_{t-1}|\boldsymbol{\alpha}_t, \sigma_\omega^2) + \mathbf{b}_\omega^\top(\boldsymbol{\omega} - \boldsymbol{\omega}_{t-1}) \\ &- \frac{1}{2}(\boldsymbol{\omega} - \boldsymbol{\omega}_{t-1})^\top \mathbf{A}_\omega (\boldsymbol{\omega} - \boldsymbol{\omega}_{t-1}) + o(||\boldsymbol{\omega} - \boldsymbol{\omega}_{t-1}||^2)\end{aligned} \tag{6}$$

where $\mathbf{b}_\omega = \nabla_{\boldsymbol{\omega}} \ln p(\mathcal{D}, \boldsymbol{\omega}|\boldsymbol{\alpha}_t, \sigma_\omega^2)|_{\boldsymbol{\omega}=\boldsymbol{\omega}_{t-1}}$, $\mathbf{A}_\omega = -\nabla_{\boldsymbol{\omega}}^2 \ln p(\mathcal{D}, \boldsymbol{\omega}|\boldsymbol{\alpha}_t, \sigma_\omega^2)|_{\boldsymbol{\omega}=\boldsymbol{\omega}_{t-1}}$.

Additionally, $\mathbf{b}_\omega$ can be represented by the summation of the gradient of the cross-entropy loss plus the weight decay term of network weights as shown Eq. 7. Here, softmax activation is used for the last layer output, and thus the negative logarithm of the likelihood function is known as the cross-entropy loss for the multi-class classification problem.

$$\begin{aligned}\mathbf{b}_\omega &= \nabla_{\boldsymbol{\omega}} \ln p(\mathcal{D}, \boldsymbol{\omega}|\boldsymbol{\alpha}_t, \sigma_\omega^2)|_{\boldsymbol{\omega}=\boldsymbol{\omega}_{t-1}} \\ &= \nabla_{\boldsymbol{\omega}} \ln p(\mathcal{D}|\boldsymbol{\alpha}_t, \boldsymbol{\omega})|_{\boldsymbol{\omega}=\boldsymbol{\omega}_{t-1}} + \nabla_{\boldsymbol{\omega}} \ln p(\boldsymbol{\omega}|\sigma_\omega^2)|_{\boldsymbol{\omega}=\boldsymbol{\omega}_{t-1}} \\ &= -\nabla_{\boldsymbol{\omega}} \mathcal{L}_{CE}^{train}(\mathcal{D}, \boldsymbol{\alpha}_t, \boldsymbol{\omega})|_{\boldsymbol{\omega}=\boldsymbol{\omega}_{t-1}} - \sigma_\omega^{-2}\boldsymbol{\omega}_{t-1}\end{aligned} \tag{7}$$

Then $\mathbf{A}_\omega$ can be derived and rewritten as:

$$\mathbf{A}_\omega = \nabla_{\boldsymbol{\omega}}^2 \mathcal{L}_{CE}^{train}(\mathcal{D}, \boldsymbol{\alpha}_t, \boldsymbol{\omega})|_{\boldsymbol{\omega}=\boldsymbol{\omega}_{t-1}} + \sigma_\omega^{-2}\mathbf{I} \tag{8}$$

The $\nabla_{\boldsymbol{\omega}}^2 \mathcal{L}_{CE}$ is the Hessian matrix $\mathbf{H}$ with H_{ij} as the second derivatives of the cross-entropy loss. Then the posterior of $\boldsymbol{\omega}$ conditioned on $\boldsymbol{\alpha}_t$ in Eq. 4 can be calculated:

$$\begin{aligned}&p(\boldsymbol{\omega}|\mathcal{D}, \boldsymbol{\alpha}_t, \sigma^2) \propto \exp\left\{o(||\boldsymbol{\omega} - \boldsymbol{\omega}_{t-1}||^2)\right\} \cdot \\ &\exp\left\{\mathbf{b}_\omega^\top(\boldsymbol{\omega} - \boldsymbol{\omega}_{t-1}) - \frac{1}{2}(\boldsymbol{\omega} - \boldsymbol{\omega}_{t-1})^\top \mathbf{A}_\omega (\boldsymbol{\omega} - \boldsymbol{\omega}_{t-1})\right\}\end{aligned} \tag{9}$$

where $\mathbf{b}_\omega$ and $\mathbf{A}_\omega$ are obtained by Eq. 7 and Eq. 8. If Eq. 9 can be expressed properly, $\boldsymbol{\omega}_t$ can be drawn by:

$$\boldsymbol{\omega}_t \sim p(\boldsymbol{\omega}|\mathcal{D}, \boldsymbol{\alpha}_t, \sigma_\alpha^2, \sigma_\omega^2) \tag{10}$$

Similarly, we can obtain the posterior distribution over $\boldsymbol{\alpha}$ conditioned on $\boldsymbol{\omega}_t$, where $\sigma^2 = \{\sigma_\alpha^2, \sigma_\omega^2\}$.

$$\mathbf{b}_\alpha = -\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{CE}^{val}(\mathcal{D}, \boldsymbol{\omega}_t, \boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t} - \sigma_\alpha^{-2}\boldsymbol{\alpha}_t; \tag{11}$$

$$\mathbf{A}_\alpha = \nabla_{\boldsymbol{\alpha}}^2 \mathcal{L}_{CE}^{val}(\mathcal{D}, \boldsymbol{\omega}_t, \boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t} + \sigma_\alpha^{-2}\mathbf{I} \tag{12}$$

$$\begin{aligned}&p(\boldsymbol{\alpha}|\mathcal{D}, \boldsymbol{\omega}_t, \sigma^2) \propto \exp\Big(o(||\boldsymbol{\alpha} - \boldsymbol{\alpha}_t||^2)\Big) \cdot \\ &\exp\left(\mathbf{b}_\alpha^\top(\boldsymbol{\alpha} - \boldsymbol{\alpha}_t) - \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}_t)^\top \mathbf{A}_\alpha (\boldsymbol{\alpha} - \boldsymbol{\alpha}_t)\right)\end{aligned} \tag{13}$$

We follow the setting of DARTS to update $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ by train set and validation set, respectively. Therefore, $\boldsymbol{\alpha}_{t+1}$ can be drawn from this distribution:

$$\boldsymbol{\alpha}_{t+1} \sim p(\boldsymbol{\alpha}|\mathcal{D}, \boldsymbol{\omega}_t, \sigma_\omega^2, \sigma_\alpha^2) \tag{14}$$

Then we can replace $\boldsymbol{\alpha}_t$ with $\boldsymbol{\alpha}_{t+1}$ to calculate the posterior distribution over $\boldsymbol{\omega}$ in Eq. 9. This procedure is repeated until the maximal search time is reached.

However, there is a gap between the theoretical and practical optimization as shown in Eq. 9 and Eq. 13. In the following section, a practical second-order optimization approximation method is introduced for narrowing the gap and making GibbsNAS feasible to implement.

Hessian Inverse Approximation

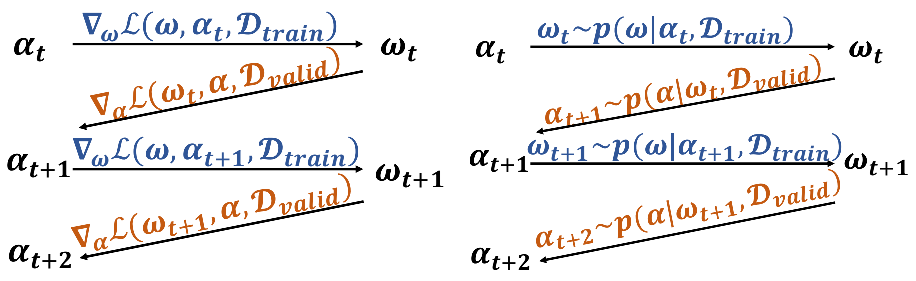
Finding the exact posterior distributions from Eq. 9 and Eq. 13 is a non-trivial task. Here, we omit the cubic and higher terms, and focus on the first order and second order. Instead of using numerical differentiation which is computationally intractable, we estimate the inverse of Hessian matrix for network weight $\tilde{\mathbf{A}}_\omega$ and architecture importance $\tilde{\mathbf{A}}_\alpha$ by Quasi-Newton methods. On one hand, $\tilde{\mathbf{A}}_\alpha^{-1}$ can be obtained by BFGS algorithm (Nocedal and Wright 2006), which is guaranteed to be symmetric, invertible and positive-definite in most cases[2]. On the other hand, as the space complexity of BFGS is $O(M^2)$, where M denotes the parameters number, it is intractable to approximate Hessian inverse for $\boldsymbol{\omega}$. Hence, we adopt the diagonal approximation (Becker, Lecun et al. 1988) for the Hessian of network weight $\mathbf{A}_\omega$ and use Limited-memory BFGS algorithm (LBFGS) (Liu and Nocedal 1989) to estimate the Hessian inverse $\tilde{\mathbf{A}}_\omega^{-1}$. Given the property of the symmetric and positive-definitive of $\tilde{\mathbf{A}}_\alpha^{-1}$ (non-diagonal estimation) and $\tilde{\mathbf{A}}_\omega^{-1}$ (diagonal estimation), the posterior distributions in Eq. 9 and Eq. 13 can be approximated as multivariate Gaussian distributions:

$$\boldsymbol{\omega}_t \sim p(\boldsymbol{\omega}|\mathcal{D}, \boldsymbol{\alpha}_t, \sigma_\alpha^2, \sigma_\omega^2) \approx \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\omega}_{t-1} - \tilde{\mathbf{A}}_\omega^{-1}\mathbf{b}_\omega, \tilde{\mathbf{A}}_\omega^{-1}) \tag{15}$$

$$\boldsymbol{\alpha}_{t+1} \sim p(\boldsymbol{\alpha}|\mathcal{D}, \boldsymbol{\omega}_t, \sigma_\omega^2, \sigma_\alpha^2) \approx \mathcal{N}(\boldsymbol{\alpha}|\boldsymbol{\alpha}_t - \tilde{\mathbf{A}}_\alpha^{-1}\mathbf{b}_\alpha, \tilde{\mathbf{A}}_\alpha^{-1}) \tag{16}$$

where $\mathbf{b}_\omega$ and $\mathbf{b}_\alpha$ denote the first derivative to $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ in Eq. 7 and Eq. 11, respectively. The update for network weights and architecture importance are conducted by sampling Eq. 15 and Eq. 16, respectively.

[2]Except all the elements of the gradient are zero.


α_t
$\nabla_\omega \mathcal{L}(\omega, \alpha_t, \mathcal{D}_{train})$
ω_t
$\nabla_\alpha \mathcal{L}(\omega_t, \alpha, \mathcal{D}_{valid})$
α_{t+1}
$\nabla_\omega \mathcal{L}(\omega, \alpha_{t+1}, \mathcal{D}_{train})$
ω_{t+1}
$\nabla_\alpha \mathcal{L}(\omega_{t+1}, \alpha, \mathcal{D}_{valid})$
α_{t+2}
α_t
$\omega_t \sim p(\omega|\alpha_t, \mathcal{D}_{train})$
ω_t
$\alpha_{t+1} \sim p(\alpha|\omega_t, \mathcal{D}_{valid})$
α_{t+1}
$\omega_{t+1} \sim p(\omega|\alpha_{t+1}, \mathcal{D}_{train})$
ω_{t+1}
$\alpha_{t+2} \sim p(\alpha|\omega_{t+1}, \mathcal{D}_{valid})$
α_{t+2}

(a) Differential method: DARTS (b) Gibbs Sampling method: GibbsNAS

Figure 1: Comparison of two solutions for bi-level optimization in one-shot NAS. Here we follow the notations of DARTS: $\boldsymbol{\alpha}$: architecture importance, $\boldsymbol{\omega}$: network weight, D_{train}: train set, D_{valid}: validation set.

GibbsNAS vs. DARTS

In the following, we will show the connection between GibbsNAS and DARTS, and also the relation with Bayesian model evidence learning and BayesNAS (Zhou et al. 2019).

First, consider DARTS in Fig. 1a. In every step, $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ are updated iteratively by gradient descent. The updated value of $\boldsymbol{\omega}$ in DARTS is exact the mean value of Eq. 15 in GibbsNAS, if the matrix $\mathbf{A}_\omega$ is estimated by $\lambda_w^{-1} * \mathbf{I}$, where λ_w can be viewed as the learning rate of $\boldsymbol{\omega}$. Depending on the approximation method for the gradient w.r.t $\boldsymbol{\alpha}$, DARTS has two versions. In the first order version, $\boldsymbol{\alpha}$ is updated like $\boldsymbol{\omega}$. Hence, if we use the mean values in Eq. 15 and Eq. 16 to replace the sampling values (omit the uncertainty), and estimate $\mathbf{A}_\omega$ as $\lambda_w^{-1} * \mathbf{I}$, GibbsNAS degrades to the first order of DARTS. In the second order of DARTS, the calculation of $\boldsymbol{\alpha}$ is differentiated w.r.t $\boldsymbol{\alpha}$ from the gradient of loss w.r.t $\boldsymbol{\omega}$, while in GibbsNAS, second order gradient of loss is w.r.t the same variable. Furthermore, if all the sampling procedures are omitted in GibbsNAS (just maintains the mean values and does not consider the uncertainty), $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ will be updated iteratively by Newton method, and thus it seems to be a "Newton method" version of DARTS.

GibbsNAS vs. SNAS/GDAS

SNAS (Xie et al. 2019), GDAS (Dong and Yang 2019b), and GibbsNAS involve the sampling concept in differentiable neural architecture search. However, the sampling methods are quite different. SNAS and GDAS sample a single-path (subgraph) architecture from the one-shot model. By contrast, GibbsNAS handles the network weight and the architecture importance directly on the one-shot model like DARTS and uses Gibbs sampling to approximate their joint distribution. GibbsNAS updates the architecture importance by sampling it from the multivariate Gaussian distributions in Eq. 16.

GibbsNAS vs. Bayesian Model Evidence

When it comes to Bayesian evidence, it is well known that the basic idea for NAS is to identify the architecture importance $\boldsymbol{\alpha}$ from given data $\mathcal{D}$, which can be written as:

$$p(\boldsymbol{\alpha}|\mathcal{D}) = \frac{p(\boldsymbol{\alpha}) \int p(\mathcal{D}, \boldsymbol{\omega}|\boldsymbol{\alpha}) p(\boldsymbol{\omega}) d\boldsymbol{\omega}}{p(\mathcal{D})} \tag{17}$$

The integral term in the numerator is also called model evidence, which expresses the preference shown by the data for different architectures. The discretization approximation to model evidence by using Monte Carlo sampling can be viewed as the probability of generating the data $\mathcal{D}$ from an architecture $\boldsymbol{\alpha}$ whose network weights $\boldsymbol{\omega}$ are sampled at random from the prior. However, this model evidence can not be estimated precisely without sampling a huge amount of network weights and then calculating their loss. An alternative approximation solution is to use Laplace approximation (MacKay 1992). Then the model evidence can be shown as

$$p(\mathcal{D}|\boldsymbol{\alpha}) \approx p(\mathcal{D}|\boldsymbol{\alpha}, \boldsymbol{\omega}^*) p(\boldsymbol{\omega}^*) \cdot \frac{(2\pi)^{M/2}}{|\mathbf{A}_\omega|^{1/2}} \tag{18}$$

where $\boldsymbol{\omega}^*$ is the local optimal value, and the matrix $\mathbf{A}_\omega$ is calculated at the point of $\boldsymbol{\omega}^*$. This solution leads to a non-weight-sharing scheme and its search costs are higher than weight-sharing ones because the network weights of each architecture need to be trained until convergence to get the $\boldsymbol{\omega}^*$. On the contrary, GibbsNAS updates the posterior distribution approximation at every step and search the neural architecture in a Gibbs sampling manner.

GibbsNAS vs. BayesNAS

BayesNAS (Zhou et al. 2019) adopts Bayesian learning to address the dependencies and pruning issues by modeling architecture parameters via hierarchical automatic relevance determination priors. Both BayesNAS and GibbsNAS formulate network search as a generative graphical model, and suppose the prior independence of architecture importance $\boldsymbol{\alpha}$ and network weights $\boldsymbol{\omega}$. While GibbsNAS builds a posterior that expresses the preferences for different models given the dataset, and samples from this distribution in a Gibbs manner. BayesNAS maximizes the marginal likelihood as:

$$\int \int p(\mathcal{D}|\boldsymbol{\alpha}, \boldsymbol{\omega}, \sigma^2) p(\boldsymbol{\omega}|\lambda) p(\boldsymbol{\alpha}|\boldsymbol{s}) d\boldsymbol{\omega} d\boldsymbol{\alpha} \tag{19}$$

where σ, λ and s are hyper-parameters. By maximizing Eq. 19, BayesNAS can get the update of dependency factors of information flow, and then uses them to prune the architecture. In a nutshell, BayesNAS updates the architecture importance $\boldsymbol{\alpha}$ and network weights $\boldsymbol{\omega}$ in a traditional deterministic BP manner, and involves Bayesian learning to shrink the search

Algorithm 1 GibbsNAS: Uncertainty-Aware One-Shot Neural Architecture Search by Gibbs Sampling

```
Input: Input data $\mathcal{D}$, search space of operations $\mathcal{O}$, search space
    of network connections $\mathcal{C}$ and hyper-parameters e.g. $\sigma_\alpha$, $\sigma_\omega$,
    learning rate $lr_\alpha$ and $lr_\omega$ with decay policies, budget $T$, etc.
Output: Optimal searched architecture $Arc^*$.
1: Initialize architecture importance $\boldsymbol{\alpha}_1$ and network weights $\boldsymbol{\omega}_0$.
   The elements of architecture importance $\boldsymbol{\alpha}(i, j)$ is formed by
   the importance of $i^{\text{th}}$ connection and $j^{\text{th}}$ operation;
2: Create a mixed operation $\hat{o}(c)$ parametrized by the row vector
   $\boldsymbol{\alpha}_1(c)$ for each connection $c$ in $\mathcal{C}$;
3: for t = 1,2,...,T do
4:     Sample $\boldsymbol{\omega}_t \sim p(\boldsymbol{\omega}|\mathcal{D}, \boldsymbol{\alpha}_t, \sigma_\alpha^2, \sigma_\omega^2)$ by Eq. 15, Eq. 7 and Eq.
       8 to update network weights;
5:     Sample $\boldsymbol{\alpha}_{t+1} \sim p(\boldsymbol{\alpha}|\mathcal{D}, \boldsymbol{\omega}_t, \sigma_\omega^2, \sigma_\alpha^2)$ by Eq. 16, Eq. 11 to
       update architecture importance;
6:     Update the posterior distribution over network weights and
       architecture importance by Eq. 15 and Eq. 16.
7: end for
8: Replace $\hat{o}(c)$ with $o^*(c) = \text{argmax}_{o \in \mathcal{O}} \boldsymbol{\alpha}(c, o)$ for each con-
   nection $c$ in $\mathcal{C}$. Set $Arc^* = \cup_{c \in \mathcal{C}} \{o^*(c)\}$.
```

space (prune the architecture). On the contrary, GibbsNAS updates the architecture importance and network weights in an uncertainty-aware manner.

Conclusive Remarks

Our method focuses on one-shot NAS. It calculates the posterior distribution over architecture importance conditioned on network weights and the posterior distribution over network weights conditioned on architecture importance, respectively. Newer value of architecture importance or network weights is drawn from its corresponding posterior distribution, and then it is used to replace the older one. As such, neural architectures are searched for and optimized in a Gibbs sampling manner rather than solving bi-level optimization Eq. 1 and Eq. 2 directly. We term our approach *uncertainty-aware one-shot neural architecture search by Gibbs sampling – addr. GibbsNAS* whose details are depicted in Algorithm 1.

GibbsNAS benefits neural architecture search in two aspects: GibbsNAS models one-shot NAS by a Bayesian learning framework and uses Gibbs sampling to update architecture importance and network weights iteratively, which gives a solid theoretical interpretation along with bi-level optimization in one-shot NAS. Another advantage of GibbsNAS is that it involves sampling operations of the posterior distributions in the update procedure, which makes GibbsNAS to be uncertainty-aware. Therefore, it can be more stable and thus has a better generalization capability comparing with the uncertainty-unaware scheme: DARTS (Liu, Simonyan, and Yang 2019), which is demonstrated in our experiments.

Experiments

GibbsNAS is evaluated in three settings: 1) the micro cell based search space used in ENAS (Pham et al. 2018) and DARTS (Liu, Simonyan, and Yang 2019); 2) the search space derived from NAS-Bench-201 (Dong and Yang 2020); 3) transferable performance of ImageNet (Russakovsky et al. 2015) classification from the basic cell searched on CIFAR-10 (Krizhevsky, Hinton et al. 2009a). In GibbsNAS, we need to update network weights $\boldsymbol{\omega}$ and architecture importance $\boldsymbol{\alpha}$ iteratively. As mentioned above, LBFGS and BFGS are used to calculate the second order terms for $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$, respectively. For the network weights $\boldsymbol{\omega}$ updates (Eq. 15), we set initial learning rate to 0.1 with batch size 64. For the architecture importance $\boldsymbol{\alpha}$ updates (Eq. 16), we set initial learning rate to 0.1 with batch size 2048. To avoid excessive uncertainty of weights $\boldsymbol{\omega}$ caused by sampling, a sampling clipping strategy and a warm-up strategy are applied to make the training process smooth, which is listed in supplementary material along with the learning rate decay policy and the positive-definite policy. Considering the variances of prior Gaussian distributions, we set both weight decay (σ_ω^{-2}) and importance decay (σ_α^{-2}) to 1e-4. In the first setting and the third setting, experiments are run on one Tesla V100, while in the second setting experiments are performed on one Tesla K80.

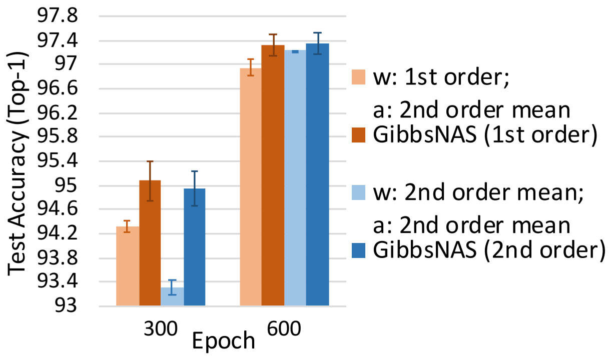


Figure 2: Ablation study on uncertainty-aware GibbsNAS with different training time. GibbsNAS is an uncertainty-aware NAS, whose 1st order version and 2nd order version are defined above. Their counterparts are updating network weights and architecture importance by their mean values.

Search on Micro Cell Based Search Space Used in ENAS and DARTS. We first evaluate GibbsNAS on micro cell based search space used in DARTS (Liu, Simonyan, and Yang 2019), which extends the cell search space of ENAS (Pham et al. 2018) by adding dilated convolutions. During the search phase, aligned with other baseline search strategies like DARTS, ENAS and PC-DARTS, GibbsNAS targets at maximizing the validation accuracy, regardless of other objectives (model size, inference latency, etc.). The test error is reported in Table 1, indicating that the architecture searched by GibbsNAS gets 2.53% error rate with 4.12M parameters on CIFAR-10 (Krizhevsky, Hinton et al. 2009a), which significantly outperforms DARTS and achieves a very competitive result with other cutting-edge NAS algorithms.

Note that to compare different single-objective (accuracy-oriented) NAS algorithms, if the search space and other settings (initial number of channels, number of cells, etc.) are the same among them, the accuracy with its bias and the searching cost are the most important metrics. On the contrary, the FLOPS and model size are meaningful for comparing dif-

Methods	**Test Error (%)**	**Params (M)**	**FLOPs (M)**	**#ops**	**Search GPU-day**	**Search Method**
DenseNet-BC (Huang et al. 2017)	3.46	25.6	–	–	–	manual
NASNet-A + cutout (Zoph et al. 2018)	2.65	3.3	605	13	1800	RL
AmoebaNet-B + cutout (Real et al. 2019)	2.55±0.05	2.8	490	19	3150	evolution
Hierarchical Evo (Liu et al. 2018c)	3.75±0.12	15.7	–	6	300	evolution
PNAS (Liu et al. 2018b)	3.41±0.09	3.2	–	8	225	SMBO
ENAS + cutout (Pham et al. 2018)	2.89	4.6	626	6	0.5	RL
DARTS-V1 + cutout (Liu, Simonyan, and Yang 2019)	3.00±0.14	3.3	519	7	0.4	gradient
DARTS-V2 + cutout (Liu, Simonyan, and Yang 2019)	2.76±0.09	3.4	547	7	1.0	gradient
SNAS (moderate) + cutout (Xie et al. 2019)	2.85±0.02	2.8	441	7	1.5	gradient
P-DARTS + cutout (Chen et al. 2019)	2.50	3.4	532	7	0.3	gradient
PC-DARTS (1st order) + cutout (Xu et al. 2020)	2.57±0.07	3.6	557	7	0.1	gradient
BayseNAS + cutout (Zhou et al. 2019)	2.81±0.04	3.4	–	7	0.2	gradient
GibbsNAS + cutout	2.53±0.02	4.1	571	7	0.5	gradient & sampling

Table 1: Comparison on CIFAR-10. In line with the existing protocol, GibbsNAS's search cost excludes the final evaluation cost. Numbers of compared methods are from original papers. We follow the DARTS search space as well as its other settings like initial number of channels, number of cells, making NAS as a single objective optimization (accuracy-oriented) problem.

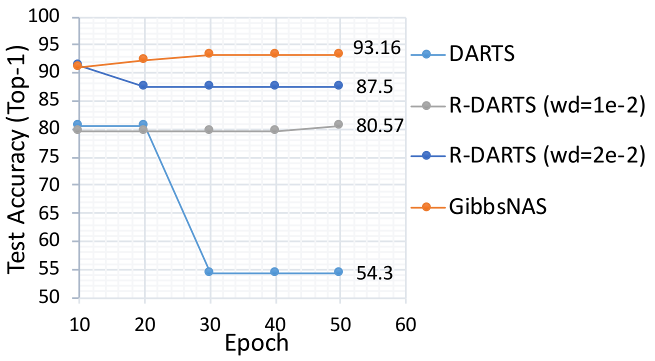


Figure 3: Convergence stability on CIFAR-10 with DARTS.

ω -1st -order -mean	ω -2nd -order -mean	ω -2nd -order -sample	α -2nd -order -mean	α -2nd -order -sample	Top-1 Test Accuracy %
√				√	97.43
√			√		97.33
		√		√	97.47
		√	√		97.29
	√		√		97.20

Table 2: Ablation study on uncertainty-aware GibbsNAS with different optimizers. ω-1st-order-mean means updating weights by SGD optimizer; 2nd-order-mean means updating weights or importance by the mean of Eq. 15 and Eq. 16; 2nd-order-sample means updating weights or importance by sampling from Eq. 15 and Eq. 16. Denote the first and third row as GibbsNAS (1st order) and (2nd order), respectively.

ferent NAS search strategies in the following three cases: 1) multiple objective NAS; 2) single objective NAS with different search space 3) single objective NAS with different initial settings like initial number of channels, number of cells, etc.

We conduct ablation studies to demonstrate the effectiveness of the proposed uncertainty-aware solution in Tab. 2 and Fig. 2. According to different optimizers, we denote the search strategies as GibbsNAS (1st order) and GibbsNAS (2nd order): GibbsNAS (1st order) approximates the Hessian inverse of architecture importance and updates it in a sampling manner, while updating network weights by SGD optimizer. GibbsNAS (2nd order) approximates the Hessian inverse for both network weights and architecture importance, and updates them in a sampling manner. [3] We compare them with their non-uncertainty-aware versions (update merely by mean values). For each searching method, three parallel tests are conducted, and the average and standard deviation are reported in Fig. 2. It shows that the uncertainty-aware versions (update by sampling) outperform the non-uncertainty-aware ones (update by mean values) both in an early inference stage (300 epochs) and a final inference stage (600 epochs). The ablation studies in Tab. 2 also empirically demonstrate that NAS can benefit from the uncertainty-aware sampling, and thus the efficacy of GibbsNAS is verified.

Search on NAS-Bench-201. We use the same search space as NAS-Bench-201 (Dong and Yang 2020), with five operations included, and the architectures are constructed by stacking 17 cells. Three datasets are used for evaluation: CIFAR-10 (Krizhevsky, Hinton et al. 2009b), CIFAR-100 (Krizhevsky, Hinton et al. 2009b), and ImageNet-16-120 (Dong and Yang 2020). For fairness, we follow the training settings and split protocol as the original paper (Dong and Yang 2020). We search for the architecture based on

[3] If no explicit labeled, GibbNAS denotes GibbsNAS (2nd order).

Methods	CIFAR-10		CIFAR-100		ImageNet-16-120	
	valid	test	valid	test	valid	test
RSPS (Li and Talwalkar 2019)	80.42±3.58	84.07±3.61	52.31±5.55	52.31±5.77	27.22±3.24	26.28±3.09
DARTS-V1 (Liu, Simonyan, and Yang 2019)	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS-V2 (Liu, Simonyan, and Yang 2019)	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS (Dong and Yang 2019b)	89.89±0.08	93.61±0.09	71.34±0.04	70.70±0.30	41.59±1.33	41.71±0.98
SETN (Dong and Yang 2019a)	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
ENAS (Pham et al. 2018)	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
GibbsNAS	90.02±0.60	92.72±0.60	68.88±1.43	69.20±1.40	42.31±1.69	42.08±1.95
REA (Real et al. 2019)	91.19±0.31	93.92±0.30	71.81±1.12	71.84±0.99	45.15±0.89	45.54±1.03
RS (Bergstra and Bengio 2012)	90.93±0.36	93.70±0.36	70.93±1.09	71.04±1.07	44.45±1.10	44.57±1.25
REINFORCE (Williams 1992)	91.09±0.37	93.85±0.37	71.61±1.12	71.71±1.09	45.05±1.02	45.24±1.18
BOHB (Falkner, Klein, and Hutter 2018a)	90.82±0.53	93.61±0.52	70.74±1.29	70.85±1.28	44.26±1.36	44.42±1.49
ResNet (He et al. 2016)	90.83	93.97	70.42	70.86	44.53	43.63
Optimal	91.61	94.37	73.49	73.51	46.77	47.31

Table 3: Top-1 test accuracy (%) for classification on NAS-Bench-201. The results of other architectures are obtained from the paper of NAS-Bench-201. The first block shows the results by parameter sharing based NAS methods, while the second block shows the non-parameter sharing algorithms. As GibbsNAS is a parameter-sharing one, it should be compared within the first block. "optimal" indicates the highest mean accuracy for each set. The mean and std of 3 trials for RSPS, DARTS, GDAS, SETN, ENAS, and GibbsNAS are reported with their ultimate performance (retrain from scratch).

Methods	Params (M)	Cost (GPU-day)	Top-1 Acc. (%)
NASNet-A	5.3	1800	0.740
AmoebaNet-A	5.1	3150	0.745
DARTS†	4.9	1	0.731
GibbsNAS (1st order)†	4.9	0.4	0.743
GibbsNAS (2nd order)†	5.1	0.5	0.741
P-DARTS‡	4.9	0.3	0.756
PC-DARTS‡	5.3	0.1	0.749
GibbsNAS‡	5.1	0.5	0.754

Table 4: Transferability study from CIFAR-10 to ImageNet. Peer methods' results are from original papers. † (‡) denotes using the same lr decay policy as in DARTS (P-DARTS).

GibbsNAS for three times with different random seeds, as reported in Table 3. We observe that GibbsNAS yields superior performance compared to DARTS, ENAS, SETN and RSPS, while it achieves competitive results with GDAS (slightly better than GDAS on ImageNet-16-120, while slightly worse than GDAS on CIFAR-100 and CIFAR-10).

Besides test accuracy, convergence stability of a NAS approach is also evaluated: the search algorithm should converge to a robust architecture. In other words, the performance (retrain from scratch) of the architectures during the search phase should be stabilizing rather than decreasing. Fig. 3 compares the stability of DARTS (Liu, Simonyan, and Yang 2019) and its amendment version (Zela et al. 2020) (by adding weight decay) with GibbsNAS. We observe that the ultimate performance of DARTS usually decreases over search epochs. In contrast, thanks to the uncertainty-aware solution, GibbsNAS can steadily achieve improved performance.

Transfer to ImageNet Classification. We further study the transferability from easy tasks to hard tasks. Following (Pham et al. 2018; Liu, Simonyan, and Yang 2019), we search for the cells on CIFAR-10, and then without extra searching efforts, we transfer it to ImageNet by expanding the depth and width of the searched model as DARTS (Liu, Simonyan, and Yang 2019) does. The search space is different between the first block and the last two blocks of Tab. 4, and the learning rate decay policy is different between the second block and the third block of Tab. 4. GibbsNAS follows DARTS's search space and compares different learning rate policy, the results in Table 4 indicate that the models searched by GibbsNAS outperform that searched by DARTS and are competitive to those searched by P-DARTS and PC-DARTS.

The empirical results also show that the performance rank on the surrogate dataset (GibbsNAS-2nd-order is better than GibbsNAS-1st-order on CIFAR-10) can not always be maintained when it is evaluated on the target dataset (GibbsNAS-2nd-order is sightly worst than GibbsNAS-1st-order on ImageNet). It remains an open issue for NAS to select the proper surrogate datasets for large scale target datasets.

Conclusion and Future Work

We have proposed GibbsNAS to provide an elegant solution for NAS by a Gibbs sampling procedure from the posterior distributions which express the preference for different architectures. Our approach is uncertainty-aware, and works competitively on different search space and datasets for image classification. It is still highly desired to design a more precise approximation of Hessian matrix for network weights, to calculate Eq. 15, which we leave to future work.

Acknowledgments

Junchi Yan, Xiaoxing Wang and Xiaokang Yang were partially supported by China Major State Research Development Program (2020AAA0107600), NSFC (61972250, U19B2035, 72061127003), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102).

References

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2017. Designing Neural Network Architectures using Reinforcement Learning. In *ICLR*.

Baker, B.; Gupta, O.; Raskar, R.; and Naik, N. 2018. Accelerating Neural Architecture Search using Performance Prediction. In *ICLR*.

Bardenet, R.; Brendel, M.; Kégl, B.; and Sebag, M. 2013. Collaborative hyperparameter tuning. In *ICML*, 199–207.

Becker, S.; Lecun, Y.; et al. 1988. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*.

Bender, G.; Kindermans, P.; Zoph, B.; Vasudevan, V.; and Le, Q. V. 2018. Understanding and Simplifying One-Shot Architecture Search. In *ICML*, 549–558.

Bergstra, J.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for Hyper-Parameter Optimization. In *NeurIPS*, 2546–2554.

Bergstra, J.; and Bengio, Y. 2012. Random search for hyperparameter optimization. *Journal of Machine Learning Research* .

Chen, X.; Xie, L.; Wu, J.; and Tian, Q. 2019. Progressive Differentiable Architecture Search: Bridging the Depth Gap Between Search and Evaluation. In *ICCV*, 1294–1303.

Domhan, T.; Springenberg, J. T.; and Hutter, F. 2015. Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves. In *IJCAI*, 3460–3468.

Dong, X.; and Yang, Y. 2019a. One-Shot Neural Architecture Search via Self-Evaluated Template Network. In *ICCV*, 3680–3689.

Dong, X.; and Yang, Y. 2019b. 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*.

Falkner, S.; Klein, A.; and Hutter, F. 2018a. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *ICML*, 1436–1445.

Falkner, S.; Klein, A.; and Hutter, F. 2018b. Practical Hyperparameter Optimization for Deep Learning. In *ICLR*.

Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J. T.; Blum, M.; and Hutter, F. 2015. Efficient and Robust Automated Machine Learning. In *NeurIPS*, 2962–2970.

Friedman, N.; Geiger, D.; and Goldszmidt, M. 1997. Bayesian Network Classifiers. *Mach. Learn.* 29(2-3): 131–163.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*, 770–778.

Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2017. Densely Connected Convolutional Networks. In *CVPR*.

Kandasamy, K.; Neiswanger, W.; Schneider, J.; Póczos, B.; and Xing, E. P. 2018. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. In *NeurIPS*, 2020–2029.

Krizhevsky, A.; Hinton, G.; et al. 2009a. Learning Multiple Layers of Features from Tiny Images. Technical report, Citeseer.

Krizhevsky, A.; Hinton, G.; et al. 2009b. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Li, L.; Jamieson, K.; and DeSalvo, G. 2017. Hyperband: bandit-based configuration evaluation for hyper-parameter optimization. In *ICLR*.

Li, L.; and Talwalkar, A. 2019. Random Search and Reproducibility for Neural Architecture Search. In *UAI*, 367–377.

Lindauer, M.; and Hutter, F. 2018. Warmstarting of Model-Based Algorithm Configuration. In *AAAI*, 1355–1362.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.; Fei-Fei, L.; Yuille, A. L.; Huang, J.; and Murphy, K. 2018a. Progressive Neural Architecture Search. In *ECCV*, 19–35.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.; Fei-Fei, L.; Yuille, A. L.; Huang, J.; and Murphy, K. 2018b. Progressive Neural Architecture Search. In *ECCV*, 19–35.

Liu, D. C.; and Nocedal, J. 1989. On the limited memory BFGS method for large scale optimization. *Math. Program.* 45(1-3): 503–528.

Liu, H.; Simonyan, K.; Vinyals, O.; Fernando, C.; and Kavukcuoglu, K. 2018c. Hierarchical Representations for Efficient Architecture Search. In *ICLR*.

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

MacKay, D. J. C. 1992. A Practical Bayesian Framework for Backpropagation Networks. *Neural Comput.* 4(3): 448–472.

Nocedal, J.; and Wright, S. 2006. *Numerical optimization*. Springer Science & Business Media.

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

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*, 4780–4789.

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*, 2902–2911.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M. S.; Berg, A. C.; and Li, F. 2015. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* 115(3): 211–252.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *NeurIPS*, 2960–2968.

Swersky, K.; Snoek, J.; and Adams, R. P. 2013. Multi-Task Bayesian Optimization. In *NeurIPS*, 2004–2012.

Wang, X.; Xue, C.; Yan, J.; Yang, X.; Hu, Y.; and Sun, K. 2020. MergeNAS: Merge Operations into One for Differentiable Architecture Search. In *IJCAI*, 3065–3072.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* 8: 229–256.

Xie, L.; and Yuille, A. L. 2017. Genetic CNN. In *ICCV*, 1388–1397.

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

Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Qi, G.; Tian, Q.; and Xiong, H. 2020. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *ICLR*.

Xue, C.; Yan, J.; Yan, R.; Chu, S. M.; Hu, Y.; and Lin, Y. 2019. Transferable AutoML by Model Sharing Over Grouped Datasets. In *CVPR*, 9002–9011.

Ying, C.; Klein, A.; Christiansen, E.; Real, E.; Murphy, K.; and Hutter, F. 2019a. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *ICML*, 7105–7114.

Ying, C.; Klein, A.; Christiansen, E.; Real, E.; Murphy, K.; and Hutter, F. 2019b. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *ICML*, 7105–7114.

Yu, K.; Sciuto, C.; Jaggi, M.; Musat, C.; and Salzmann, M. 2020. Evaluating The Search Phase of Neural Architecture Search. In *ICLR*.

Zela, A.; Elsken, T.; Saikia, T.; Marrakchi, Y.; Brox, T.; and Hutter, F. 2020. Understanding and Robustifying Differentiable Architecture Search. In *ICLR*.

Zhou, H.; Yang, M.; Wang, J.; and Pan, W. 2019. BayesNAS: A Bayesian Approach for Neural Architecture Search. In *ICML*, 7603–7613.

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.