C02029: Doctor of Philosophy
CRICOS Code: 009469A
PhD Thesis: Information Technology
May 2021

Automated Deep Learning:
A Study on Neural Architecture Search

Miao Zhang

School of Biomedical Engineering
Faculty of Eng. & IT
University of Technology Sydney
NSW - 2007, Australia

Automated Deep Learning: A Study on Neural Architecture Search

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy
in
Information Technology

by

Miao Zhang

to

School of Biomedical Engineering
Faculty of Engineering and Information Technology
University of Technology Sydney
NSW - 2007, Australia

May 2021

© 2021 by Miao Zhang
All Rights Reserved

Abstract

Deep Learning (DL) has shown its superiority in various research areas in recent years, including computer vision, natural language processing, and autonomous driving. Through designing different deep neural networks (DNNs), deep learning techniques have achieved the state-of-the-art performance in numerous real-world applications. Deep neural network has become the first choice for most researchers when solving different machine learning problems. However, the performance of deep neural networks is very sensitive to the structures, and engineers need to choose or design appropriate network structures through tedious and repeated experiments so that deep neural networks can reach their potentials for different problems. Automated Deep Learning (AutoDL) aims to build a better deep learning model in a data-driven and automated manner, so that most practitioners in deep learning can also build a high-performance machine learning model, with being relieved from a labor-intensive and time-consuming neural network design process. AutoDL can bring new research ideas to deep neural networks, and lower the threshold of deep learning in various research areas through automated neural network design.

The process of automated neural network design is termed as Neural Architecture Search (NAS). As the name suggested, the goal of NAS is to automatically design deep neural networks without human intervention. Most recent works on NAS adopt a *weight-sharing* paradigm to find competitive architectures with greatly reducing the computational complexity. Instead of separating training architectures, weight sharing strategy encodes the whole search space as a supernet, and all neural networks directly inherit weights from the supernet for evaluation without needing to be trained from scratch. Pioneer studies on weight-sharing NAS follow two sequential steps. They first adopt an architecture sampling controller to sample architectures for the supernet training. Then, a heuristic search method is adopted to search promising architectures over a discrete search space based on the trained supernet. Since only the supernet is trained for once, this paradigm is also called as *one-shot NAS*. To further improve the efficiency, later researches further employ the continuous relaxation to make the neural architecture differentiable, so that gradient descent can be used to optimize the architecture with respect to validation accuracy, and this paradigm is also referred to as *differentiable NAS*. This thesis focuses on the two specific research directions: *one-shot NAS* and *differentiable NAS*.

Most state-of-the-art one-shot NAS methods use the validation accuracy based on inheriting weights from the supernet as the stepping stone to search for the best performing architecture, adopting a bilevel optimization pattern with assuming this validation accuracy approximates to the test accuracy after re-training. However, recent works have found that there is no positive correlation between the above validation accuracy and test accuracy for these weight-sharing methods, and this reward based sampling for supernet training also entails the rich-get-richer problem. To handle this deceptive problem, **Chapter 2** presents a new approach, **E**fficient **N**ovelty-driven **N**eural **A**rchitecture **S**earch (EN^{2}AS), to sample the most abnormal architecture to train the supernet. Specifically, a single-path supernet is adopted, and only the weights of a single architecture sampled by our novelty search are optimized in each step to reduce the memory demand greatly. Experiments demonstrate the effectiveness and efficiency of our novelty search based architecture sampling method.

Although one-shot NAS significantly improves the computational efficiency, it also introduces multi-model forgetting during the supernet training, where the performance of previous architectures degrades when sequentially training new architectures with partially-shared weights. To overcome such catastrophic forgetting, **Chapter 3** formulates the supernet training in the one-shot NAS as a constrained optimization problem of continual learning that the learning of current architecture should not degrade the performance of previous architectures during the supernet training. We propose a Novelty Search based Architecture Selection (**NSAS**) loss function and demonstrate that the posterior probability could be calculated without the strict assumption when maximizing the diversity of the selected constraints. Extensive experiments demonstrate that our method enhances the predictive ability of the supernet in one-shot NAS and achieves remarkable performance on CIFAR-10, CIFAR-100, and PTB with efficiency.

Existing works on differentiable NAS adopt a bilevel optimization to alternatively optimize the supernet weights and architecture parameters after relaxing the discrete search space into differentiable, to further improve the efficiency. However, there is non-negligible incongruence in this simple transformation, and it is hard to guarantee that the differentiable optimization in the continuous latent space is equivalent to the optimization in the discrete space. In **Chapter 4**, we utilize a variational graph autoencoder to injectively transform discrete architecture space into an equivalently continuous latent space, to resolve the incongruence. We further devise a probabilistic exploration enhancement method to encourage intelligent exploration during the architecture search in latent space. The catastrophic forgetting is an inevitable problem in weight-sharing NAS, which deteriorates supernet predictive ability and makes the bilevel optimization inefficient in differentiable NAS. This paper proposes an architecture complementation method to relieve this deficiency in differentiable NAS. We analyze the effectiveness of the proposed method in differentiable NAS, and a series of experiments have been conducted to compare the proposed method with state-of-the-art differentiable NAS methods.

Despite notable benefits on computational efficiency from differentiable NAS, more recent works find that existing differentiable NAS techniques struggle to outperform naive baselines, yielding deteriorative architectures as the search proceeds. Rather than directly optimizing the architecture parameters, **Chapter 5** formulates the neural architecture search as a distribution learning problem through relaxing the architecture weights into Gaussian distributions. By leveraging the recently-proposed natural-gradient variational inference (NGVI), the architecture distribution can be easily optimized based on existing codebases without incurring more memory and computational consumption. We demonstrate how the differentiable NAS benefits from Bayesian principles, enhancing exploration and improving stability. The experimental results on benchmark datasets confirm the significant improvements the proposed framework can make. Furthermore, to enhance the searched architectures' transferability in the complicated search space, we propose a simple yet effective depth-aware differentiable neural architecture search. Specifically, we achieve state-of-the-art results on the NAS-Bench-201 and NAS-Bench-1Shot1 benchmark datasets. Our best architecture in the DARTS search space also obtains competitive test errors with 2.37%, 15.72%, and 24.2% on CIFAR-10, CIFAR-100, and ImageNet datasets, respectively.

While much has been discussed about several potentially fatal factors in DARTS, the architecture gradient, a.k.a. hypergradient, has received less attention. In **Chapter 6**, we tackle the hypergradient computation in DARTS based on the implicit function theorem, making it only depends on the obtained solution to the inner-loop optimization and agnostic to the optimization path. To further reduce the computational requirements, we formulate a stochastic hypergradient approximation for differentiable NAS, and theoretically show that the architecture optimization with the proposed method, named iDARTS, is expected to converge to a stationary point. Comprehensive experiments on two NAS benchmark search spaces and the common NAS search space verify the effectiveness of our proposed method. It leads to architectures outperforming, with large margins, those learned by the baseline methods.

Author's declaration

I, *Miao Zhang* declare that this thesis, submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy, in the *School of Biomedical Engineering*, *Faculty of Engineering and Information Technology* at the University of Technology Sydney, Australia, is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis. This document has not been submitted for qualifications at any other academic institution. This research is supported by the Australian Government Research Training Program.

Production Note:
Signature removed prior to publication.

SIGNATURE: ______________________

[Miao Zhang]

DATE: 13th May, 2021

PLACE: Sydney, Australia

Dedication

To my dear parents, brother, and lovely Shufen ...

Acknowledgments

I would like to express my earnest thanks to my supervisors, A/Professor Steven Su, Dr. Shirui Pan, Professor Huiqi Li, and Dr. Steve Ling. Without the tremendous support and guidance from them, this thesis cannot be finished. First of all, I would like to sincerely thank my main supervisor, Professor Su, who led me into the realm of machine learning and deep learning. In the last three years of studying with Professor Su, I have learned a lot of research skills and theoretic foundation knowledge. Furthermore, Professor Su's broad research horizon and rigorous research attitude have benefited me a lot, and these will also have a positive impact in my future work and study. His optimistic and open-minded character are what I have been learning. Dr. Shirui Pan is my associate supervisor. From Dr. Shirui Pan, I learned the rigor and patience in research. His hard-working and self-motivation have set a good example for me.

I also sincerely thank the other mentors in Australia and China, including Dr. Steve Ling, Dr. Xiaojun Chang, and Professor Huiqi Li. I would also like to thank all the staffs in the School of Biomedical Engineering, University of Technology Sydney. Their help makes my research and life in UTS much easier. I will remember them all in my heart.

I would also like to thank all mates at UTS who all had a positive impact on my life and research, Taoping Liu, Li Wang, Yanhao Zhang, Huan Yu, Yongbo Chen, Jiaheng Zhao, Fang Bai, Kairui Guo, Wentian Zhang, Yao Huang, Juan Lyu, Hairong Yu, Ye Shi, Wei Huang. I will also express my gratitude to my friends, Xinyu Lin, Zuyi Chen, Fangzhou Liu, Hantang Liu, Lin Tian, Ming Wei, Qian Li, Huasu Jin, Jiawei Chen, Haocheng Liu, Kang Liu, Mingkun Pei. Their accompany and encouragement also give me a lot of support during my PhD research life.

Lastly, and above all, I would like to thank my parents, brother and people loved me for their selfless love and support. I wish them health and happiness.

List of Publications

Related to the Thesis :

- Chapter 2:

  1. Miao Zhang, Huiqi Li, Shirui Pan, Taoping Liu, Steven Su, One-Shot Neural Architecture Search via Novelty Driven Sampling, In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2020 [162]. ***CORE Rank A****

- Chapter 3:

  2. Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Steven Su, Overcoming Multi-Model Forgetting in One-Shot NAS with Diversity Maximization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020 [159]. ***CORE Rank A****
  3. Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Zongyuan Ge, and Steven Su, One-Shot Neural Architecture Search: Maximising Diversity to Overcome Catastrophic Forgetting. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020 [161]. ***CORE Rank A****

- Chapter 4:

  4. Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Zongyuan Ge, Steven Su, Differentiable Neural Architecture Search in Equivalent Space with Exploration Enhancement. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020 [158]. ***CORE Rank A****
  5. Miao Zhang, Steven Su, Shirui Pan, Xiaojun Chang, Huiqi Li, Gholamreza Haffari, Differentiable Neural Architecture Search in Equivalent Space with Enhancing Exploration and Relieving Forgetting. Submitted to *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021. ***CORE Rank A****

- Chapter 5:

  6. Miao Zhang, Steven Su, Shirui Pan, Xiaojun Chang, Li Wang, Gholamreza Haffari, Differentiable Neural Architecture Search via Bayesian Learning Rule. Submitted to *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021. ***CORE Rank A****

- Chapter 6:

  7. Miao Zhang, Steven Su, Shirui Pan, Xiaojun Chang, Huiqi Li, Ehsan Abbasnejad, Gholamreza Haffari, iDARTS: Differentiable Architecture Search with Stochastic Implicit Gradients. Accepted by *International Conference on Machine Learning (ICML)*, 2021 [165]. ***CORE Rank A****

OTHERS :

- **8. Miao Zhang**, Huiqi Li, Steven Su, High Dimensional Bayesian Optimization via Supervised Dimension Reduction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, ***CORE Rank A****

- **9. Miao Zhang**, Huiqi Li, Juan Lyu, Steve Ling, Steven Su, Hyperparameter Optimization with Non-stationary Kernel for CNN based Lung Nodule Classification. In *IEEE Transaction on Evolutionary Computing (TEvC)*, 2021, ***CORE Rank A****

- **10. Miao Zhang**, Steven Su, Shirui Pan, Xiaojun Chang, Gholamreza Haffari, Differentiable Architecture Search Without Training Nor Labels: A Pruning Perspective. In Submitting to *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021, ***CORE Rank A****

Table of Contents

LIST OF FIGURES

LIST OF TABLES

TABLE Page

Chapter 1

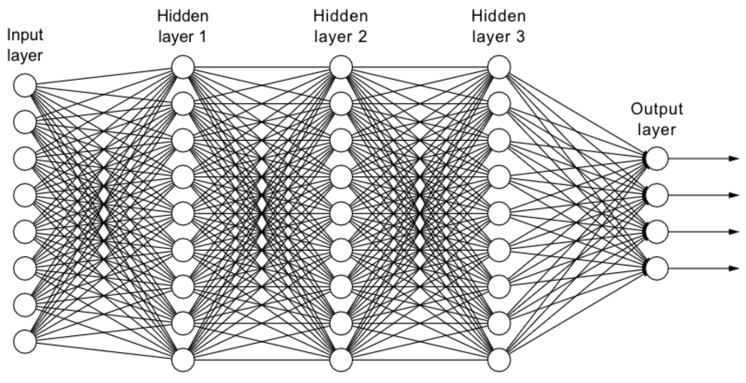
Introduction

1.1 Background: Deep Learning and Automated Deep Learning

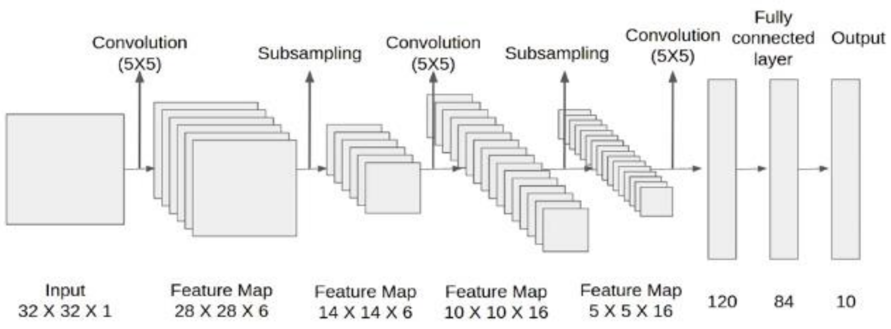
1.1.1 Deep Learning

In many industries such as finance and manufacturing, automation has completely revolutionized the production process with greatly improving the productivity efficiency. However, in the field of artificial intelligence (AI) and machine learning (ML), most applications are usually designed by experts through tedious and repeated experiments[63]. Deep learning (DL), as the most promising subfield of machine learning, has recently achieved remarkable progress in many fields, such as computer vision, natural language processing, autonomous driving, healthcare, bancassurance, fault diagnosis in industry, and so on [49, 78]. Deep learning system builds deep neural networks (DNNs) to learn the meaningful representations of high-dimensional data, which has become the first choice for most researchers when solving different machine learning tasks.

The first form of deep learning is the fully-connected neural network (FNN), where deep learning stacks several fully-connected layers to build the model [57]. Figure 1.1 (a) gives a simple example of FNN. There are two types of layers, one is liner layer to learn the linear transformation, and the other is the nonliner layer which is with non-linear activation function. Later, researchers found that, compared to fully-connection, the convolutional operation is a better way to extract spatial features for image processing. In 1994, the LeNet-5 is proposed for



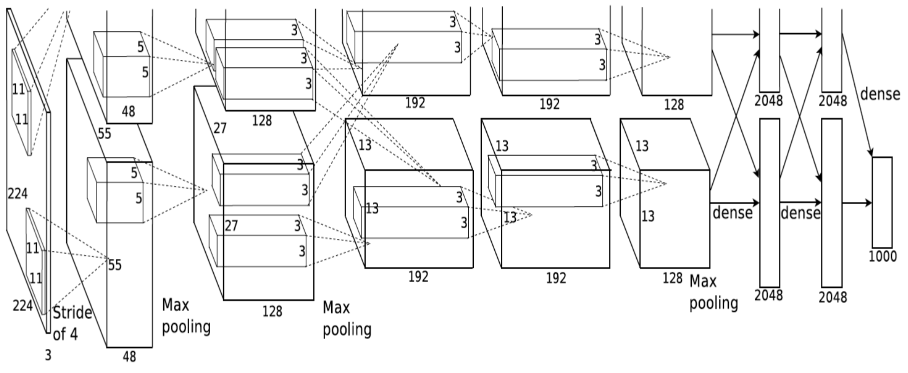
(a) Fully-connected neural network



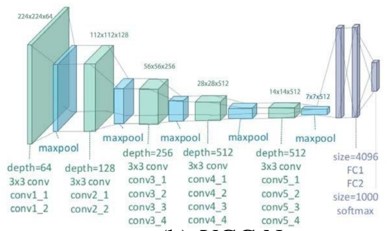
(b) LeNet-5

Figure 1.1: Illustrations of neural network structures in the early stage [57, 78].

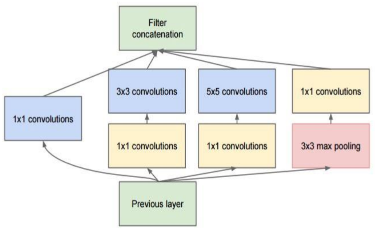
handwritten zip code recognition. The LeNet-5 architecture is fundamental to image processing, which bring deep learning into a new era [78]. As shown in Figure 1.1 (a), LeNet-5 contains convolutional layer, pooling layer, and fully-connected layer, which are still the most popular layers nowadays. However, due to the limit of computational resources, LeNet-5 could hardly be applied to more complicated images, and draw litter attention from the community. In 2012, Alex Krizhevsky designed the AlexNet [77], which was a deeper and wider version of the LeNet-5, for the ImageNet competition, and won other baselines with significant margins. With leveraging the GPUs computational ability, Alex Krizhevsky showed the promising future of deep learning. Hereafter, the community focuses designing different neural architectures to improve the performance and solve different tasks. For example, the VGG network [128] from Oxford utilized several smaller 3×3 convolutional filter to replace the large convolutional filter in the AlexNet. The Google team proposed the GoogLeNet and different variants of inception architectures [132] that further leveraged the 1×1 convolutional filter to reduce the number of features, which was also adopted by the most famous ResNet [59]. A typical revolution in deep learning community was the ResNet proposed in 2016, which utilized a simple idea to design the structure of deep neural networks: feeding the output of a convolutional layer and



(a) AlexNet


maxpool
maxpool
maxpool
maxpool
maxpool
depth=64 3x3 conv conv1_1 conv1_2
depth=128 3x3 conv conv2_1 conv2_2
depth=256 3x3 conv conv3_1 conv3_2 conv3_3 conv3_4
depth=512 3x3 conv conv4_1 conv4_2 conv4_3 conv4_4
depth=512 3x3 conv conv5_1 conv5_2 conv5_3 conv5_4
size=4096 FC1 FC2 size=1000 softmax

(b) VGG Net



(c) GoogLeNet

Figure 1.2: Typical structures of modern deep neural networks [77, 128, 132].

also bypassing the input to the next layers through residual connection. All above attempts show that the structure of deep neural network plays an important role in the deep learning.

1.1.2 Automated Deep Learning

Similar to artificial intelligence and machine learning systems, the deep learning systems are still designed by experts through tedious and repeated experiments for hyperparameter selection and network structure design. Since there are only a limited number of experts with the knowledge of deep learning systems, seeking to automate the design of deep learning systems has gradually become a popular research direction. On a high level, this automation can shorten the development time of deep learning systems, significantly relieve the burden for experts, and accelerate the development of deep learning technology, thereby make artificial general intelligence (AGI) possible to a certain extent[63].

Automated Deep Learning (AutoDL) aims to build a better deep learning system in a data-driven and automated manner, so that most practitioners in deep learning can build a high-performance machine learning model, without being an expert in the field of deep learning [43, 139, 148, 163, 164]. AutoDL is the process of automatically applying deep learning to real-world problems, which covers the entire process from data processing to the deployment of deep learning models. AutoDL can provide end-to-end deep learning solutions, and these solutions are usually better than hand-designed deep learning systems. AutoDL brings new research ideas to deep neural networks, and lowers the threshold of deep learning through automated neural network design. AutoDL has not only attracted the interest from the research community, technology companies such as Amazon, Facebook, and Google have also leverage the AutoDL to automatically build deep neural networks for solving various businesses tasks [4, 6, 48], including computer vision [84, 133], natural language processing [67], autonomous driving, and so on [28, 119]. AutoDL makes deep learning easier to use, thereby improving the business capabilities of enterprises to obtain more profits. There are two main research directions in AutoDL: Hyperparameter Optimization (HO) and Neural Architecture Search (NAS), which are very crucial to apply deep learning algorithms in practice.

1.1.2.1 Hyperparameter Optimization (HO)

Determining appropriate values of hyperparameters of DNN is a frustratingly difficult task where all feasible hyperparameter configurations form a huge space, from which we need to choose the optimal case. Setting correct hyperparameters is often critical for reaching the full potential of the deep neural network chosen or designed, otherwise it may severely hamper the performance of deep neural networks.

Hyperparameter optimization in DNN is a global optimization to find a D-dimensional hyperparameter setting x that minimize the validation error f of a DNN with learned parameters θ. The optimal x could be obtained through optimizing f as follows:

$$\begin{aligned} &\min_{x \subseteq \mathbb{R}^D} \quad f(x,\theta;Z_{val}) \\ &s.t. \quad \theta = \arg\min_{\theta} f(x,\theta;Z_{train}) \end{aligned} \tag{1.1}$$

where Z_{train} and Z_{val} are training and validation datasets respectively. Solving Eq.(1.1) is very challenging for the high complexity of the function f, and it is usually accomplished manually in the deep learning community, which largely depends on expert's experience or intuition. It is also hard to reproduce similar results when this configuration is applied on different datasets or problems. There are several systematic approach to tune hyperparameters in deep learning community, including Grid search [79], Random search [12], Tree-structured Parzen Estimator Approach (TPE)[14] and Bayesian optimization[130], which have shown their superiority than manual search method in hyperparameters optimization of deep neural network. Grid search is the most common strategy in hyper-parameter optimization [79], and it is simple to implement with parallelization, which makes it reliable in low dimensional spaces (e.g., 1-d, 2-d). However, Grid search suffer from the curse of dimensionality because the search space grows exponentially with the number of hyper-parameters. Random search [12] proposes to randomly sample points from the hyperparameter configuration space. Although this approach looks simple, but it could find comparable hyperparameter configuration to grid search with less computation time. Hyperparameter optimization in deep neural networks is a computational expensive problem where evaluating a hyperparameter choice may cost several hours or even days. This property also makes it unrealistic to sample many enough points to be evaluated in Grid and Random search. One popular approach is using efficient surrogates to approximate the computationally expensive fitness functions to guide the optimization process. Bayesian optimization [130] built a probabilistic Gaussian model surrogate to estimate the distribution of computationally expensive validation errors. Hyperparameter configuration space is usually modeled smoothly, which means that knowing the quality of certain points might help infer the quality of their nearby points, and Bayesian optimization [13, 14, 126] utilizes the above smoothness assumption to assist the search of hyperparameters. Gaussian Process is the most common method for modeling loss functions in Bayesian optimization for it is simple and flexible. There are several acquistion functions to determin the next promising points in Gaussian process, including Probability of Improvement (PI), Expected Improvement (EI), Upper Confidence Bound (UCB) and the Predictive Entropy Search (PES) [61, 130].

1.1.2.2 Neural Architecture Search (NAS)

Neural Architecture Search (NAS), another sub-field of AutoDL, is an efficient and effective method for automating the process of neural network design, which has attracted increasing attention recently as it relieves human experts from the labor-intensive and time-consuming neural network design process. NAS has achieved remarkable success on image recognition [84, 133], language modeling [67], and other deep learning applications [28, 119]. The search space of neural architecture $\mathscr{A}$ is generally represented as a directed acyclic graph (DAG), and the subgraph in the search space is denoted as $\alpha \in \mathscr{A}$ corresponding to a neural architecture $\mathcal{U}(\alpha, w)$ with weights w. NAS aims to find a subgraph α with the best validation loss after being trained on the training set, as:

$$\alpha^* = \underset{\alpha \in \mathscr{A}}{\mathtt{argmin}}\ \mathscr{L}_{\mathtt{val}}(\mathcal{U}(\alpha, w_\alpha)) \tag{1.2}$$

where $\mathscr{L}_{\mathtt{val}}$ is the loss function on the validation set, and w_α are the weights of the architecture after being trained on the training set to minimize the training loss $\mathscr{L}_{train}$:

$$w_\alpha = \underset{w}{\mathtt{argmin}}\ \mathscr{L}_{\mathtt{train}}(\mathcal{U}(\alpha, w)) \tag{1.3}$$

Early NAS works adopt a nested manner to optimize weights and architectures, which samples numerous architectures to be trained on the training set and utilize different search strategies, e.g., evolutionary algorithm (EA) or reinforcement learning (RL) [118], to find promising architectures based on those evaluated architectures. Despite its capacity to find competitive architectures, the computational complexity of NAS is highly expensive since exactly evaluating architectures has highly computational demand. Zoph *et al.* [173] spends more than 1800 GPU days for reinforcement learning (RL) based NAS and Real *et al.* [118] uses 450 GPUs for 7 days through evolutionary algorithm (EA) to train the model. As described, the downside is that NAS comes with an extremely high demand for computation power. To mitigate this problem, many recent studies have been devoted to reducing these search costs through different performance estimation strategies, including performance prediction [7], weight generation [156], and the well-known weight sharing method [114].

1.2 Literature Review

The primary drawback in NAS is the extremely high computational cost, and more and more recent works turn to relieve this issue. The most popular strategies include the performance prediction [7], weight generation [156], and the weight sharing [114], where the weight sharing takes the dominant position among the three. This subsection will review the related works based on these three strategies in NAS. After that, we will review several common NAS search spaces used in this thesis.

1.2.1 Performance Prediction

Since evaluating a deep neural network usually requires several GPU hours or even days, it is hard to find the optimal or suboptimal neural network based on heuristic algorithms through evaluating a large number of deep neural networks. Therefore, accelerating the neural network evaluation has become a promising research direction in the neural architecture search. First, the deep neural network can be evaluated based on the low-fidelity indicators of its real performance. Low-fidelity indicators based strategies include: shorter training time [153, 173], training on a small part of the full dataset [153], training on low-resolution images [32], reducing the number of channels or neurons in the deep neural networks [118, 173]. These low-fidelity indicators based strategies can greatly reduce the computational cost. Although the low-fidelity based methods will introduce bias in the performance prediction, the neural architecture search mainly depends on the ranking among the neural networks rather than their real performance.

Therefore, as long as the ranking of the neural networks is consistent, the architecture search method based on low fidelity can find a high-performance structure. However, recent studies have shown that when the difference between low-fidelity indicator and real performance is too large, the architecture ranking will be greatly affected, and the literature [93] proposes to search for the network structure by gradually increasing the fidelity.

Another performance prediction method is based on the learning curve of the neural network [7, 39, 131]. It evaluates the neural network according to the initial learning curve, and discards those neural networks with poor performance in the initial learning curve, speeding up the architecture search process. Similar to the surrogate model based hyperparameter optimization, the performance of architectures can be predicted by establishing a proxy model. Kandasamy *et al.* [69] propose a novel kernel function to define the relationship among different network structures, with using the optimal transport program to define the distance. Luo *et al.* [99] transform the discrete neural architectures into continuous variables with an autoencoder, where the continuous encoding of the neural architectures transformed by encoder could be used to establish a proxy model for the architecture search. After obtaining a continuous representation of a promising architecture, the decoder will generate a discrete architecture for evaluation.

1.2.2 Weights Generation

The weight generation method defines a hypernetwork to generate weights for different neural architectures. The paper [56] proposes a dynamic hypernetwork, which takes the network structures as input to generate the weights for different neural architectures. The hypernetwork can rank different neural architectures to obtain the most promising structure, which is then trained from scratch for evaluation. Therefore, the weight generation methods only need to train the hypernetwork during the neural architecture search phase, which thus greatly reduces the computational cost. Zhang *et al.* [156] propose a Graph-hypernetwork (GHN) for the weights generation based neural architecture search: given a neural architecture, GHN utilize the graph neural network to generate weights for this neural network directly. GHN can model the topology of the network, so it could more accurately predict the performance of the neural architecture compared with the normal hypernetwork.

1.2.3 Weights Sharing

Weight-sharing neural architecture search (weight-sharing NAS) [9, 114] defines a supernet subsuming all possible architectures in the search space, and all architectures share the weights from the supernet for the evaluation. Weight-sharing NAS is the most popular paradigm in current neural architecture search research, which attracts a lot of attention from the AutoDL community. This thesis mainly focuses on the neural architecture search based on the weight sharing strategy. There are two main paradigms in weight-sharing NAS: one-shot NAS and differentiable NAS.

1.2.3.1 One-Shot NAS

Pioneer studies on weights-sharing NAS contains two sequential steps [54, 86, 114]:

1. First utilize a sampling controller to sample architectures for the supernet training:

$$\mathcal{W}_{\mathcal{A}} = \underset{\mathcal{W}}{\texttt{argmin}}\ \mathcal{L}_{\texttt{train}}(\mathcal{U}(\mathcal{A},\mathcal{W})); \tag{1.4}$$

2. Then use a heuristic search algorithm for the promising architectures over a discrete search space based on the trained supernet:

$$\alpha^* = \underset{\alpha\in\mathcal{A}}{\texttt{argmin}}\ \mathcal{L}_{\texttt{val}}(\mathcal{U}(\alpha,\mathcal{W}_{\mathcal{A}}(\alpha))). \tag{1.5}$$

Since only the supernet is trained for once, these weight-sharing NAS methods are also called one-shot NAS. ENAS [114] utilizes the validation accuracy with shared weights as the reward to optimize the reinforcement learning (RL) based architecture sampling controller. Whereas, Guo *et al.* [54] and Li *et al.* [86] train the supernet based on a uniform sampling strategy, and the best-performing architecture from the trained supernet is found through a random search or evolutionary method.

1.2.3.2 Differentiable NAS

Recent state-of-the-art weigh-sharing methods use continuous relaxation to transform discrete architectures into a continuous space $\mathcal{A}_\theta$ to further improve efficiency [40, 95, 109, 140]. The supernet weights and architecture parameters can be jointly optimized through:

$$(\alpha_\theta^*,\mathcal{W}_{\mathcal{A}_\theta}(\alpha_\theta^*)) = \underset{\alpha_\theta,\mathcal{W}}{\texttt{argmin}}\ \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}_\theta}(\alpha_\theta^*)), \tag{1.6}$$

making it possible to optimize the architecture with gradient descent. The best architecture α^* is determined through argmax based on the continuous architecture representation α_θ^*.

Most differentiable NAS [20, 95, 140] methods apply a **softmax** function to calculate the magnitude of each operation and relax the discrete architectures into a continuous representation. A discrete architecture is obtained by applying an **argmax** function to the magnitude matrix after the supernet has been trained. Once the discrete architectures have been transformed into a continuous space, continuous optimization is performed to update the continuous architecture representation α_θ along the gradient of validation performance [20, 95, 99]:

$$\alpha_\theta^{i+1} \leftarrow \alpha_\theta^i - \gamma \nabla_{\alpha_\theta} \mathscr{L}_{\mathtt{val}}(\alpha_\theta, \mathscr{W}^*), \tag{1.7}$$

where γ is the learning rate, and $\mathscr{W}^*$ is approximated by adapting $\mathscr{W}$ using only a single training step with descending $\nabla_\omega \mathscr{L}_{\mathtt{train}}(\mathscr{W}_{\mathscr{A}}(\alpha_\theta^i))$ [40, 86].

Since Eq.(1.6) is supposed to train the entire supernet in each step, it has a much higher memory requirement than ENAS. Hence, ProxylessNAS [20] transforms the real-valued architecture parameters into binary representations through binary gates, and only a single path is activated during the supernet training. In this way, the memory requirement for ProxylessNAS is the same as training a single architecture. Yao *et al.* [149] developed a constrained optimization method to force each step of the architecture optimization process in the continuous space to arrive at a binary result, thus reducing the memory requirement of supernet training. Unlike continuous relaxation, NAO [99] uses an LSTM-based autoencoder to transform discrete neural architectures into continuous representations. A differentiable method is then used to search for architectures in the continuous space.

Unlike directly optimizing the architecture parameters, several recent works formulate the differentiable NAS as a distribution learning problem by relaxing architecture parameters into different distributions. SNAS [140] and GDAS [40] formulate the architecture as a discrete distribution with concrete relaxation and utilize the Gumbel-softmax trick to obtain the discrete architecture. DrNAS [24] treats the continuous architecture parameters as random variables being modeled by a learnable Dirichlet distribution. This distribution is parameterized by a concentration parameter β, which controls the sampling behavior and is optimized via pathwise derivative estimators [66]. Zheng *et al.* [169] consider the whole search space as a joint multinomial distribution and learn the probabilities of candidate operations among all nodes based on the multinomial distribution learning. A common point in these previous methods is that they formulate the architecture parameters as simple distributions in which only one parameter needs to be learned. In this way, these learning paradigms are easy to fit with existing DARTS codebases.

1.2.4 NAS Search Spaces

The extensive experiments in this thesis are conducted on several common neural architecture search spaces, including the DARTS convolutional search space, DARTS recurrent search space, NAS-Bench-101 search space, NAS-Bench-201 search space, and NAS-Bench-1shot1

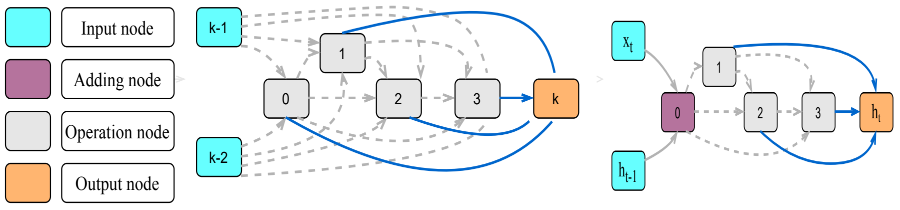


Figure 1.3: Description of DARTS convolutional (middle) and recurrent (right) search space.

Table 1.1: Summarize of common search spaces in NAS

Name	Size	Number of operations	Number of edges
DARTS convolutional search space	$\approx 10^{25}$	8	14
DARTS recurrent search space	$\approx 1.5 \times 10^{10}$	5	36
NAS-Bench-101 search space	423624	3	≤ 9
NAS-Bench-1shot1 search space	423624	3	≤ 9
NAS-Bench-201 search space	15625	5	6

search space, which are summarized in the Table 1.1, with describing the search space size, the number of candidate operations, and the number of the candidate edges. In the following, we detailed describe the above search spaces.

DARTS convolutional search space: DARTS searches for micro-cell structures on CIFAR-10 to stack more cells to form the final structure for architecture evaluation. The cell structure in this space contains eight different types of operations: 3×3 max pooling and average pooling operation, 3×3 and 5×5 separable convolution operation, 3×3 and 5×5 dilated separable convolutions operation, identity, and $zero$. There are seven nodes in each cell: two input nodes, four operation nodes, and one output node. The inputs to a cell are the outputs of two of its former cells, and the output of the cell is the sum of the outputs of all operation nodes. Fig. 1.3 describes a unified convolutional cell search space in DARTS. The best-found cells on CIFAR-10 are then transferred to CIFAR-100 and ImageNet datasets to evaluate its transferability. There are two types of cells with the unified search space: a normal cell α_{normal} and a reduction cell α_{reduce}. Cell structures are repeatedly stacked to form the final CNN structure. There are only two reduction cells in the final CNN structure, located in the 1/3 and 2/3 depths of the network. There are seven nodes in each cell: two input nodes, four operation nodes, and one output node. Each operation node selects two of the previous nodes' output as input nodes in this search space, resulting in 2+3+4+5=14 connections. Each input node will select one operation from $|\mathscr{O}| = 8$ candidate operations. In this way, a cell structure could be represented as a 14×8 matrix, and DARTS relax the categorical matrix to a softmax over all possible operations in every connection. The output of the connection is the mixing-weighted over all

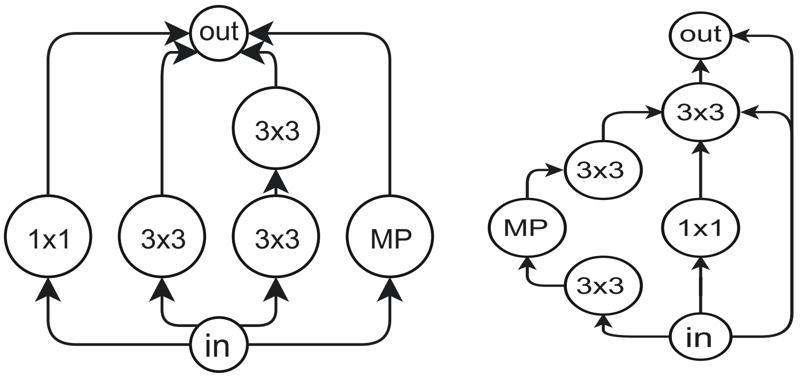


Figure 1.4: Example architectures in NAS-Bench-101 search space

possible operation:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{i,j})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{i,j})} o(x), \tag{1.8}$$

so each connection is parameterized by a continuous vector $\alpha_\theta^{(i,j)}$ with dimension $|\mathcal{O}|$ and learned by gradient descent. After the optimization in the continuous space, DARTS applies the *argmax* on α_θ^* to obtain the discrete architecture α^*. Specifically, DARTS first applies the *argmax* on the connection level, $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \alpha_\theta^{(i,j)}$, to specify the operation type for each connection. After that, DARTS will select two top connections based on $\alpha_\theta^{o^{(i,j)}}$.

DARTS recurrent search space: Beyond the convolutional cell structure search, we also conducted experiments on RNN cell structure search with the PTB dataset. For a fair comparison, the search space and hyperparameters were set following [86, 95]. The RNN search space only contains four different types of operations: $identity$, $relu$, $tanh$, $sigmoid$. The recurrent cell contains 12 nodes: 2 input nodes, 1 adding node, 8 operation nodes, and 1 output node. The adding node adds the two inputs and applies the $tanh$ activation function. The input of each node is the output of one of its previous nodes, and the output of the cell is the summation of outputs of all operation nodes. Unlike the CNN structure, our RNN architecture only contains a single cell. Fig.1.3 shows the common search spaces of the RNN cell structure.

NAS-Bench-101 search space: The high computational cost of evaluating architectures is a major obstacle when analyzing and reproducing one-shot NAS methods. For this reason, it can be hard to reproduce results with the current NAS methods under the same experimental settings for a fair comparison. To alleviate this problem, several benchmark datasets have been published in recent studies [41, 150, 154]. The NAS-Bench-101 dataset [41] is the first benchmark dataset for the neural architecture search research. This benchmark dataset provides the train time, validation and test result for all 423624 models in the search space, where each

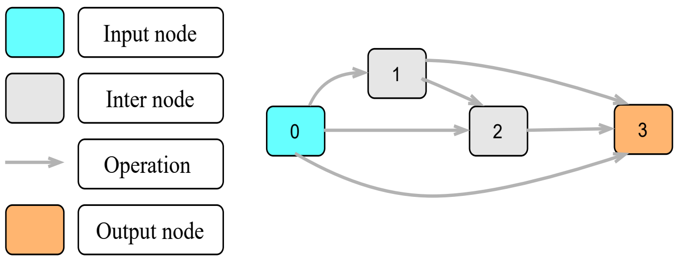


Figure 1.5: Search Space in NAS-Bench-201.

model is trained three times in a consistent manner on CIFAR-10. NAS-Bench-101 considers a cell-based search space, where each cell contains an input node, an output node, and up to 5 operation nodes, and there are three candidate operations for each operation nod: max pooling, 3×3 convolution, and 1×1 convolution. Fig. 1.4 describes two example architectures in NAS-Bench-101 search space, where MP is the max pooling operation.

NAS-Bench-1shot1 search space: NAS-Bench-1shot1 is built from the NAS-Bench-101 benchmark dataset [150], through dividing all architectures in NAS-Bench-101 into 3 different unified cell-based search spaces. The architectures in each search space have the same number of nodes and connections, making the one-shot NAS could be directly applied to each search space. The three search spaces contain 6240, 29160, and 363648 architectures with the CIFAR-10 performance, respectively.

NAS-Bench-201 search space: NAS-Bench-201 is similar to the recent cell-based NAS methods [86, 95, 150], which repeatedly stacks computational cells to form the final structure. The architectural skeleton of this search space contains three stages connected by a basic residual block [59]] with a stride of 2 between them. In each stage, the cell structure was stacked $N = 5$ times. In this search space, the cell structure is represented as a **densely** connected directed acyclic graph (DAG) with four nodes. There are six different edges between these nodes, and each edge is associated with five candidate operations, resulting in $5^6 = 15625$ candidate cell structures. The candidate operations included: 1×1 convolution, 3×3 convolution, 3×3 average pooling, skip connection, and $zero$. The $zero$ helps to drop the associated edge. NAS-Bench-201 reports the CIFAR-10, CIFAR-100, and Imagenet performance for all architecture in this search space. Fig. 1.5 outlines the search space of NAS-Bench-101 search space.

1.3 Motivations and Challenges

1.3.1 Motivations

With the development of deep learning, there are more and more candidate structures for deep learning on different applications, and the performance of deep learning is also sensitive to the selected neural architectures and hyperparameters. In addition, most deep neural networks only achieve excellent performance on specific datasets or problems, and their performance usually drops significantly when applied to different tasks, and it is currently impossible to design a generalized neural network for different tasks. Due to the lack of theoretical understanding of DNN, it is difficult to predict the performance of a DNN on different tasks based on its performance on the benchmark datasets. Designing a high-performance deep neural network is still a black-box optimization process. A deep neural network is usually designed by experts based on their past experience and tedious and repeated experiments to select appropriate hyperparameters and network structures. This neural network design process is highly iterative, so automated deep learning (AutoDL) is an ideal alternative to manually designing. AutoDL is an up-and-coming tool to take advantage of automatically building deep learning systems. By leveraging AutoDL, experts can get rid of tedious and repeated experiments, and thus focus on tasks that require more creativity. At the same time, AutoDL can also provide different insights for those real-world tasks from another perspective. Therefore, AutoDL can improve the efficiency for data scientists, and scientists in other fields without deep learning knowledge can also build high-performance deep learning models without relying on data scientists [43, 139, 148]. The main motivation of this thesis is to explore neural architecture search in the AutoDL for automatically building deep neural networks, from one-shot NAS to differentiable NAS.

1.3.2 Challenges

Neural architecture search (NAS) is a computationally expensive optimization problem, and most of the existing NAS methods are based on weight sharing paradigm to reduce computational costs. An essential assumption in the weight-sharing NAS is that the validation accuracy with inherited weights from the supernet approximates to the test accuracy after re-training, or at least be highly predictive. However, several recent works [9, 124, 129, 160] point out that there is no positive correlation between the above validation accuracy and test accuracy for most one-shot NAS methods. It indicates that we could not utilize the validation accuracy with inherited weights as a useful feedback for controller improvement. In other words, searching for the optimal architecture based on weight sharing is deceptive because architectures with optimal performance on proxy tasks are not guaranteed to perform the best in the target task [20]. Furthermore, since architectures with updated weights are supposed to have higher rewards, those performance reward based controllers have the potential to select those previously visited architectures with updated weights. Sampling architectures solely based on this deceptive reward without encouraging intelligent exploration entails the rich-get-richer problem [87] and leads to the local optima.

Catastrophic Forgetting [74, 89, 134] usually occurs when sequentially training a model for several tasks. Given a neural network with optimal parameters ω^* on task T_1, its performance on T_1 declines dramatically after the model has been trained on task T_2, because the network weights have been changed to optimize the objectives of T_2. Catastrophic forgetting with weight-sharing NAS has been observed in quite a few recent studies [11, 87, 124, 160], where performance degrades as the supernet learns new architectures to replace old ones. Sciuto *et al.* [124] and Singh *et al.* [129] observed this deceptiveness that weight sharing is supposed to disorder the architecture rank, with deteriorating the performance of other architectures when training a new generated architecture. Benyahia *et al.* [11] define it as *multi-model forgetting* that the architecture learning in each step will deteriorate the performance of other architectures with shared connections, and make the proxy reward based on the supernet unreliable.

The approach with the most recent works on NAS is to relax the discrete search space into a differentiable latent space and then alternatively optimize the supernet weights and architecture parameters via a bilevel optimization scheme, which is referred as *differentiable NAS*. However, despite its efficiency, the current differentiable NAS approaches are generally considered to be rather unreliable [23, 152]. For example, DARTS [95] does not consistently yield excellent solutions, and the architecture choices get worse and worse as the search proceeds. Performance can even be worse than a random search in some cases [124]. One potential reason for the **instability** of DARTS is that there is no theoretical foundation to show that, with continuous relaxation, an optimization in a continuous latent space is equivalent to an optimization over a discrete space. The lack of injective constraints in a simple continuous relaxation means there can be no guarantee that performing an optimization on a continuous latent space is the equivalent of doing so with a discrete space. With differentiable NAS, this *incongruence* might even increase during the architecture search [23, 152].

Despite notable benefits on computational efficiency from differentiable NAS, more recent works find it is still unreliable [23, 152] to directly optimize the architecture magnitudes. For example, DARTS [95] is unable to stably obtain excellent solutions and yields deteriorative architectures during the search proceeds, performing even worse than random search in some cases [124]. This critical weakness is termed as *instability* in differentiable NAS [152]. Zela *et al.* [152] empirically point out that the instability of DARTS is highly correlated with the dominant eigenvalue of the Hessian of the validation loss with respect to the architectural parameters, where this dominant eigenvalue increases during the architecture search. On the other hand, [24, 87, 127, 161] state that directly optimizing the architecture parameters without exploration easily entails the rich-gets-richer problem, leading to those architectures that converge faster at the beginning while achieving poor performance at the end of the training, e.g. architectures with intensive *skip-connections* [33, 91].

With a gradient-based bi-level optimization, differentiable NAS methods alternately optimize the inner model weights and the outer architecture parameter in a weight-sharing supernet. Differentiable NAS methods simply assume the model weights w in the inner-loop could reach the optimal points w^* with only one training step, leveraging the one-step unroll learning paradigm to calculate the hypergradient. A key challenge to the scalability and quality of the learned architectures is the need for differentiating through the inner-loop optimization, while which can impose considerable computational and memory burdens.

1.4 Thesis Contributions

This thesis studies two paradigms of neural architecture search in automated deep learning: (a) one-shot NAS; and (b) differentiable NAS. The main contributions of this thesis lie in the following five aspects:

- To relieve the deceptive reward problem in the supernet training, we innovatively introduce novelty search to one-shot NAS rather than devising a complicated reward-based controller, which samples architectures to train the supernet through novelty search. Compared with the existing reward-based sampler, novelty search has the potential to fairly training the supernet in One-Shot NAS, as it always samples those untrained or less trained parts of the supernet to be trained. Furthermore, since our method always samples architectures containing few shared connections with previously visited architectures, it could also effectively relieve the multi-model forgetting that occurs during the supernet training and make the supernet more predictive.

- Addressing multi-model forgetting during supernet training is an urgent issue if we are to better leverage one-shot NAS and improve the predictive ability of supernets. Hence, we have formulated supernet training as a constrained optimization problem for continual learning, where learning the current architecture should not degrade the performance of previous architectures with partially-shared weights. However, it is intractable to consider all previously visited architectures during each step of supernet training. Therefore, we only choose the most representative subset of previous architectures to regularize learning of the current architecture, where we have designed an efficient greedy novelty search method based on maximizing diversity to select a subset of the constraints that best approximate the feasible region formed by all previous architectures.

- The non-negligible incongruence in the relaxation methods adopted by existing differentiable NAS methods makes it hard to guarantee that a differentiable optimization in the continuous latent space will be equivalent to an optimization in the discrete space. To address the potential incongruence, we use a variational graph autoencoder with an asynchronous message passing scheme to injectively transform the discrete architectures into an equivalent continuous space. Using an injective approach lends a solid theoretical foundation to the equivalence between performing optimization in the continuous latent space versus the discrete space [141, 157].

- Unlike existing works that directly optimize the architecture parameters, we formulate the neural architecture search as a distribution learning problem and builds a generalized Bayesian framework for architecture optimization in differentiable NAS. We investigate differentiable NAS from a Bayesian learning perspective, and introduce the Bayesian Learning rule [72, 73, 106, 111] to the architecture optimization in differentiable NAS. We demonstrate that the proposed Bayesian framework is a practical solution to enhance exploration for differentiable NAS and improve stability as a by-product via implicitly regularizing the Hessian norm.

- While there are many variants on improving the DARTS from various aspects, limited research attention has been paid to the approximation of the architecture parameter gradient, which is also called the outer-loop gradient or hypergradient. To fill the gap, this thesis focuses on the hypergradient calculation in the differentiable NAS, with proposing the differentiable architecture search with stochastic implicit gradients (iDARTS). Specifically, we first revisit the DARTS from the bi-level optimization perspective and utilize the implicit function theorem (IFT) [10, 96], instead of the one-step unroll learning paradigm adopted by DARTS, to calculate the architecture parameter gradient.

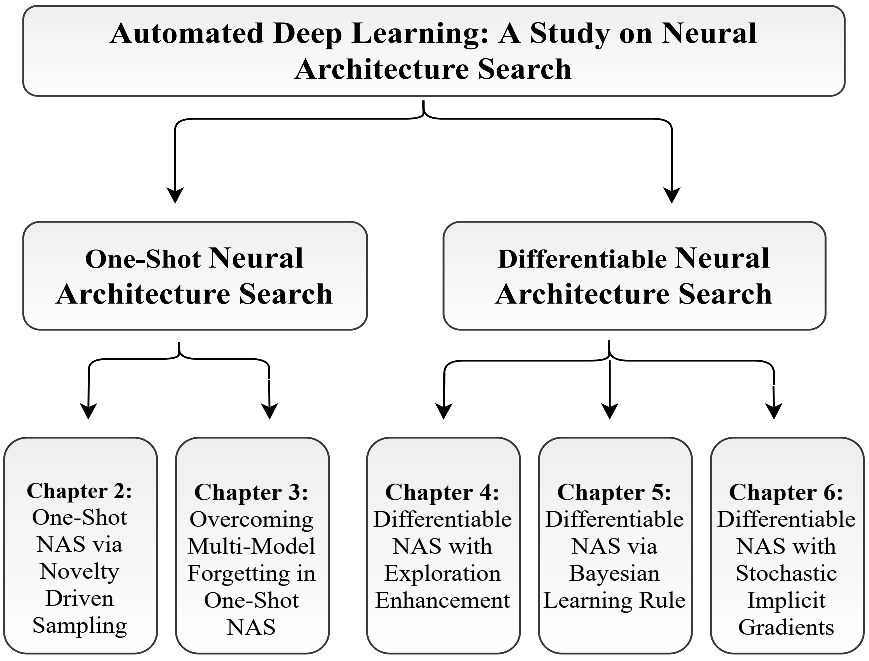

Automated Deep Learning: A Study on Neural Architecture Search
One-Shot Neural Architecture Search
Differentiable Neural Architecture Search
Chapter 2: One-Shot NAS via Novelty Driven Sampling
Chapter 3: Overcoming Multi-Model Forgetting in One-Shot NAS
Chapter 4: Differentiable NAS with Exploration Enhancement
Chapter 5: Differentiable NAS via Bayesian Learning Rule
Chapter 6: Differentiable NAS with Stochastic Implicit Gradients

Figure 1.6: Framework of the thesis.

1.5 Thesis Structure

The thesis is structured into two parts, where Part I focuses on the existing issues in the one-shot NAS and Part I deepens our understanding of exploring differentiable neural architecture searches, as demonstrated by Figure 1.6. The detailed roadmap of the thesis is summarized as follows:

- Chapter 2 introduces the **E**fficient **N**ovelty-driven **N**eural **A**rchitecture **S**earch (EN^{2}AS) to handle the deceptive reward problem and the rich-gets-richer problem in one-shot NAS.

- Chapter 3 introduces the continual learning into the one-shot NAS and has proposed the novelty search-based architecture selection (**NSAS**) loss function accordingly to overcoming the multi-model forgetting in the one-shot NAS.

- Chapter 4 introduces an approach that uses a variational graph autoencoder to injectively transform discrete architectures into an equivalent continuous latent space, to guarantee that a differentiable optimization in the continuous latent space will be equivalent to an optimization in the discrete space.

- Chapter 5 formulates the neural architecture search as a distribution learning problem, rather than directly optimizing the architecture parameters, through relaxing the architecture parameters into Gaussian distributions.
- Chapter 6 tackles the hypergradient computation in differentiable NAS based on the implicit function theorem, making it only depends on the obtained solution to the inner-loop optimization and agnostic to the optimization path.
- Chapter 7 summarizes this thesis and points out several future directions of this study.

Part I

One-Shot Neural Architecture Search

CHAPTER 2

ONE-SHOT NAS VIA NOVELTY DRIVEN SAMPLING

2.1 Introduction

One-shot neural architecture search (NAS) has received wide attention due to its computational efficiency. Most state-of-the-art one-shot NAS methods use the validation accuracy based on inheriting weights from the supernet as the stepping stone to search for the best performing architecture, adopting a bilevel optimization pattern with assuming this validation accuracy approximates to the test accuracy after re-training.

One-shot NAS, also called weight sharing NAS [9, 114], defines a supernet subsuming all possible architectures in the search space so that architectures can directly inherit weights from the supernet to avoid training from scratch. ENAS [114] utilizes the validation accuracy with shared weights as the reward to optimize the RL based architecture sampling controller. An important assumption in the weight-sharing NAS is that the validation accuracy with inherited weights from the supernet approximates to the test accuracy after re-training, or at least be highly predictive. However, several recent works [9, 124, 129, 160] point out that there is no positive correlation between the above validation accuracy and test accuracy for most one-shot NAS methods. It indicates that we could not utilize the validation accuracy with inherited weights as useful feedback for controller improvement. In other words, searching for the optimal architecture based on weight sharing is deceptive because architectures with optimal performance on proxy tasks are not guaranteed to perform the best in the target task [20].

Sciuto *et al.* [124] and Singh *et al.* [129] observed this deceptiveness that weight sharing is supposed to disorder the architecture rank, which usually deteriorates the performance of other architectures when training a new generated architecture. Benyahia *et al.* [11] defined it as *multi-model forgetting* that the architecture learning in each step will deteriorate the performance of other architectures with shared connections, and make the proxy reward based on the supernet unreliable. Furthermore, since architectures with updated weights are supposed to have higher rewards, those performance reward based controllers have the potential to select those previously visited architectures with updated weights. Sampling architectures solely based on this deceptive reward without encouraging intelligent exploration entails the rich-get-richer problem [87] and leads to the local optima. As suggested by curiosity-driven exploration in deep reinforcement learning [35], novelty-seeking could help the agent to learn new knowledge and avoid the local optima in RL domains with deceptive or sparse rewards. Different from the RL controller or gradient method, novelty search can alleviate this problem by encouraging the agent to visit unexplored areas rather than those areas with high performance. Compared with random sampling, novelty search further has the potential to fairly train the supernet in one-shot NAS, as it always samples those untrained or less trained parts of the supernet to be trained, which could improve the predictive ability of the supernet [33]. Instead of devising a complicated reward-based controller, we innovatively introduce novelty search to NAS, which samples architectures to train the supernet through novelty search. Since our method always samples architectures containing few shared connections with previously visited architectures, it could effectively relieve the multi-model forgetting that occurs during the supernet training and make the supernet more predictive. A weight-sharing based single-path model is adopted to reduce computational cost and memory storage, where all candidate architectures share weights and only the single-path weights are optimized in each step. Our contributions in this chapter can be summarized as follows.

- Firstly, a novelty search based mechanism is innovatively applied to architecture sampling in one-shot NAS for supernet training, where an efficient novelty-driven approach is devised to sample architectures without performance reward.
- Secondly, this chapter adopts a weight-sharing based single-path model for neural architecture search, which could reduce not only the computational cost but also the memory storage significantly.
- Thirdly, extensive experimental results illustrate the superiority of our method, which achieves remarkable performance on benchmark datasets with efficiency. Our approach obtains a competitive test error of 2.51% for CIFAR-10 with only 7.5 hours of search time in a single GPU, and a competitive validation perplexity of 57.83 and a test perplexity of 55.88 on PTB with 4 hours search time. After transferring to larger datasets, our best

models achieve a competitive test error of 16.56% on CIFAR-100 and a 26.66% on ImageNet, and a validation perplexity of 70.14 and a test perplexity of 69.31 on WT2. Our method also beats baselines on the NAS-Bench-201 benchmark dataset.

This chapter is based on the publication "*Miao Zhang, Huiqi Li, Shirui Pan, Taoping Liu, Steven Su, One-Shot Neural Architecture Search via Novelty Driven Sampling, In International Joint Conferences on Artificial Intelligence (IJCAI), 2020*" [162]. Miao Zhang conceived the original idea of focusing on the rich-get-richer problem in one-shot NAS. The EN^{2}AS algorithm was originally proposed by Miao Zhang. Steven Su helped Miao Zhang to verify all findings in this paper. This was then discussed with Huiqi Li, Shirui Pan, and Taoping Liu. Miao Zhang conducted all experiments, with the help from Shirui Pan. The first version of the paper was written by Miao Zhang with some help from Steven Su. The authors Huiqi Li, Shirui Pan, and Taoping Liu provided feedback during the writing of the paper.

2.2 Problem Definition and Preliminaries

2.2.1 Neural Architecture Search

The goal of NAS is to automatically design deep neural networks without human intervention. In general, the architecture of a deep neural network α is usually represented as a directed acyclic graph (DAG), which is also a subgraph of the whole search space $\alpha \in \mathscr{A}$. A deep neural network could be defined as $\mathscr{U}(\alpha, w_\alpha)$, where w_α are the weights associated with architecture α. With NAS, one tries to find the architecture with the best validation performance according to:

$$\alpha^* = \underset{\alpha \in \mathscr{A}}{\texttt{argmin}}\ \mathscr{L}_{\texttt{val}}(\mathscr{U}(\alpha, w_\alpha^*)), \tag{2.1}$$

where w_α^* is derived by training architecture α on the training set while minimizing the training loss function $\mathscr{L}_{train}$:

$$w_\alpha^* = \underset{w}{\texttt{argmin}}\ \mathscr{L}_{\texttt{train}}(\mathscr{U}(\alpha, w_\alpha)). \tag{2.2}$$

Early studies on NAS usually used a nested approach to finding promising architectures by training numerous architectures from scratch and leveraging evolutionary algorithm (EA) [118] or reinforcement learning (RL) [172] to reveal the promising ones. However, from a practical standpoint, it is computationally inefficient and often unaffordable to evaluate numerous architectures in this way. Therefore, more recently, researchers have shifted their attention to reducing computation costs with strategies such as performance prediction [7, 138], weights generation [19, 156], weight sharing [95, 114], and so on[173].

2.2.2 One-Shot Neural Architecture Search

One-shot NAS encodes a search space $\mathscr{A}$ as a supernet $\mathscr{W}_{\mathscr{A}}$ that consumes all possible candidates. Only the supernet is trained, while all candidate architectures α directly inherit weights from the supernet without needing to be trained from scratch. Search times are therefore greatly reduced because only one neural network needs to be trained during the architecture search phase. The most promising architecture α^* is based on validation performance with weights inherited from the supernet:

$$\begin{aligned} &\min_{\alpha\in\mathscr{A}} \quad \mathscr{L}_{\mathtt{val}}(\mathscr{W}^*_{\mathscr{A}}(\alpha)) \\ &\mathtt{s.t.} \quad \mathscr{W}^*_{\mathscr{A}}(\alpha) = \mathtt{argmin}\ \mathscr{L}_{\mathtt{train}}(\mathscr{W}_{\mathscr{A}}(\alpha)). \end{aligned} \tag{2.3}$$

Eq. (2.3) is more than a challenging bilevel optimization problem, and the discrete characteristics of the architecture space make it impossible to use a gradient-based method to solve the formula directly. For this reason, ENAS [114] uses an LSTM controller to sample the architectures. Whereas, [54] and [86] train the supernet based on a uniform sampling strategy and the best-performing architecture from the trained supernet is found through a random search or evolutionary method.

2.2.3 Novelty Search

Novelty search comes from the evolutionary community [81, 118], which encourages the population to search for notably different areas to enhance the exploration. This approach utilizes the novelty as the stepping stone instead of the reward function, making it easy to avoid local optima in return. Previous novelty search based evolutionary algorithms [81] have shown their superiority in searching for small neural networks. Recent works on deep reinforcement learning [35] also suggested that hybridized with novelty search evolutionary algorithm could effectively avoid local optima in RL domains with deceptive reward functions. We investigate the effects of novelty search on neural architecture search in this chapter and present how to use the novelty search mechanism as the controller to sample architectures for supernet training in the following section.

2.3 Efficient Novelty-driven Neural Architecture Search

In this section, we will describe our **E**fficient **N**ovelty-driven **N**eural **A**rchitecture **S**earch (EN^{2}AS). Algorithm 1 presents a simple implementation of EN^{2}AS, and we detailedly describe

Algorithm 1 EN^{2}AS

Input: Training, validation, test datase $\mathscr{D}_{train}$, $\mathscr{D}_{val}$, $\mathscr{D}_{test}$, initialized $\mathscr{W}$, architecture archive $A = \emptyset$, maximum number of stored architectures S, batch size b, training iteration T.

```
1: for $i = 1,2,...,(T * \texttt{size}(\mathscr{D}_{train})/b)$ do
2:     if $\texttt{size}(A) < S$ then
3:         randomly sample an architecture $\alpha$;
4:         update $\mathscr{W}_A(\alpha)$ by descending $\nabla_{\mathscr{W}_A(\alpha)}\mathscr{L}_{train}(\mathscr{W}_A(\alpha))$, and add architecture $\alpha$ into $A$;
5:     else
6:         randomly select $\alpha_\theta^m$ from $A$, update it with $\alpha_\theta^{m'}$ according Eq.(2.6);
7:         Apply argmax operation on the updated architecture to obtain $\alpha$;
8:         Update the shared weights $\mathscr{W}_A(\alpha)$ by descending $\nabla_{\mathscr{W}_A(\alpha)}\mathscr{L}_{train}(\mathscr{W}_A(\alpha))$;
9:     end if
10: end for
11: Perform random search or EA on the trained supernet with validation dataset $\mathscr{D}_{val}$ to get
    $\alpha^*$ based on Eq.(2.7).
12: Retrain $\alpha^*$ and get the best performance on the test dataset.
```

Return: architecture α^* with best performance.

the architecture sampling for supernet training based on novelty search and also discuss architecture selection from trained supernet in following subsections.

2.3.1 Single Path Supernet Training based on Novelty Search

As described in Eq.(2.3), the inherited weights $\mathscr{W}_{\mathscr{A}}(\alpha)$ of architecture α from the supernet $\mathscr{A}$ should approximate to the optimal weights w_α^* or be highly predictive. Therefore, the key to weight sharing based NAS is how to train the supernet. As discussed in [9, 124], a reward gradient-based architecture sampling controller is easy to be trapped in local optima, where there is no positive correlation between the validation accuracy with inherited weights and the test accuracy after re-training for such one-shot NAS methods. Recent work [35] on deep reinforcement learning demonstrates the effectiveness of novelty search as it could help the agent get out of local optimal when the reward function is very deceptive. In this chapter, we utilize the novelty search to sample architectures for supernet training in one-shot NAS.

The novelty search policy is defined as π and a behavior characterization $b(\pi)$ to describe its behavior. During the architecture search phase, every architecture α sampled from π is described as $b(\pi_\alpha)$ and added into archive A after calculating the novelty particular policy $N(b(\pi_\alpha), A)$. A simple and common novelty measurement is to calculate the mean distance of

α and its k-nearest neighbors from A:

$$\begin{aligned} N(\alpha,A) = N(b(\pi_\alpha),A) &= \frac{1}{|S|}\sum_{j\in S}\left\|b(\pi_\alpha)-b(\pi_j)\right\|_2 \\ S = kNN(b(\pi_\alpha),A) &= \{b(\pi_1),b(\pi_2),...,b(\pi_k)\} \end{aligned} \tag{2.4}$$

However, the distance calculation between neural architectures is not efficient because we need to compare all nodes and connections of two subgraphs, and calculating distances between the sampled architecture and all previously visited architectures in every search step. In this section, we introduce an archive based novelty search to relieve the high computational complexity for the novelty calculation. Given an archive A_θ containing a fixed number of continuous representation of sampled architectures as $\alpha_\theta^i = \alpha_\theta + \sigma\epsilon_i$, the gradient of expected novelty could be approximated as:

$$\nabla_{\alpha_\theta}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[N(\alpha_\theta+\sigma\epsilon,A)|A] \approx \frac{1}{n\sigma}\sum_{i=1}^{n}N(\alpha_\theta^i,A)\epsilon_i \tag{2.5}$$

where $\epsilon_i \sim \mathcal{N}(0,I)$, α_θ^i is the *i-th* architecture with continuous parameters representation in the archive, n is the number of sampled perturbations to α_θ^t, and the archive is fixed at the beginning of the iteration and updated at the end. Eq. (2.5) demonstrates how to change the current architectures to increase the novelty of the archive, and we could update *m-th* architecture in the archive according to:

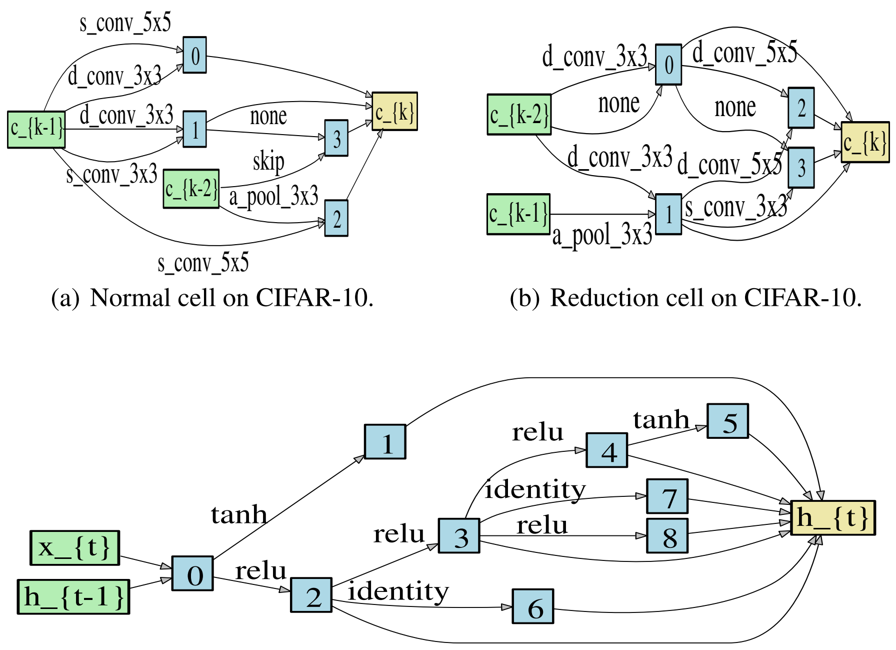
$$\alpha_\theta^{m'} \leftarrow \alpha_\theta^m + \gamma\frac{1}{n\sigma}\sum_{i=1}^{n}N(\alpha_\theta^{m,i},A)\epsilon_i \tag{2.6}$$

where γ is the stepsize. Based on Eq.(2.6), we only need to calculate the distance of the sampled architecture and an archive with a fixed number of architectures in every search step. It is straightforward to randomly select an architecture from the archive, and update it accordingly to optimize the novelty. In our practical implementation, only the architectures stored in the archive are continuous, and they are also applied with the **argmax** operation before calculating the distance to the sampled architectures.

2.3.2 Model Selection

Since evaluating an architecture is very efficient based on the trained supernet, it is possible to utilize a heuristic approach to find the most promising architecture, where random search and evolutionary algorithms are the two most common methods [54, 86]. In this chapter, we adopt the validation accuracy as the optimizing goal in model selection as:

$$\underset{\alpha}{\texttt{maximize}} \quad ACC(\mathcal{W}_{\mathcal{A}}(\alpha)) \tag{2.7}$$



(a) Normal cell on CIFAR-10.

(b) Reduction cell on CIFAR-10.

(c) Recurrent cell on PTB.

Figure 2.1: Best cell structures found by EN^{2}AS.

where $ACC(\mathscr{W}_{\mathscr{A}}(\alpha))$ is the validation accuracy of α with inhered weights from the supernet, and a baseline evolutionary algorithm is adopted to find the most promising architecture from the trained supernet.

2.4 Experimental Result

The experimental design is following [86, 95, 140] for a fair comparison, which contains three stages: architecture search, architecture evaluation, and transfer to larger datasets. We perform our EN^{2}AS on small datasets, CIFAR-10 and PTB, to search for cell architectures on a smaller supernet architecture with fewer cells in the architecture search phase, and stack more multiple cells to construct larger architecture for full training and evaluation. Finally, the best-learned cells are also transferred to CIFAR-100, ImageNet and WT2 to investigate the transferability. We also evaluate the supernet predictive ability of our novelty based sampling method compared with two baselines in the following subsections. Fig.2.1 show the best cell structures found by our EN^{2}AS. For CNN cells, each node needs to select two former nodes with applied operations as its input. As to RNN cells, each node only selects one former node with applied operation as its input. The outputs for the three types of cells are the summation of

Table 2.1: Comparison results with state-of-the-art weight sharing NAS methods on CIFAR-10, CIFAR-100 and ImageNet.

Method	Test Error (%)			Param.	Search	Memory	Supernet
	CIFAR10	CIFAR100	ImageNet	(M)	Cost	Consumption	Optimization
NAO-WS [99]	3.53	-	-	2.5	0.3	single path	gradient
ENAS [114]	2.89	18.91	-	4.6	-	single path	RL
SNAS [140]	2.85±0.02	20.09	27.3	2.8	1.5	whole supernet	gradient
BayesNAS [170]	2.81±0.04	-	26.5	3.40	**0.2**	whole supernet	gradient
MdeNAS [169]	2.51	-	-	4.06	0.16	single path	MDL
MdeNAS* [169]	2.87	17.61	26.8	3.78	0.16	single path	MDL
GDAS [40]	2.93	18.38	27.5	3.4	**0.21**	single path	gradient
DARTS (1st) [95]	2.94	-	-	2.9	1.5	whole supernet	gradient
DARTS (2nd) [95]	2.76±0.09	17.54	26.9	3.4	4	whole supernet	gradient
RandomNAS [86]	2.85±0.08	17.63	-	4.3	2.7	single path	random
EN^{2}AS	**2.61±0.06**	**16.45**	**26.66**	3.1	**0.3**	single path	novelty search
EN^{2}AS +	**2.51±0.05**	-	-	3.1	**0.3**	single path	novelty search

“MdeNAS*" indicates that we reproduce the results based on the best-reported model in MdeNAS. All models are trained with 600 (250 for ImageNet) epochs, where the batch size is 96, and the initial channel is 36, to obtain the test error. We also further train our best architecture with 1000 epochs on CIFAR-10 and 500 epochs on ImageNet to achieve state-of-the-art results.

Table 2.2: Comparison results with state-of-the-art NAS approaches on PTB and WT2.

Method	Perplexity(PTB)		Perplexity(WT2)		Param.	Search	Memory	Search
	Valid	Test	Valid	Test	(M)	Cost	Consumption	Method
ENAS [114]	60.8	58.6	72.4‡	70.4‡	24	0.5	single path	RL
DARTS (1st) [95]	60.2	57.6	-	-	23	0.5	whole supernet	gradient
DARTS (2nd) [95]	58.1	55.7	71.2	69.6	23	1	whole supernet	gradient
DARTS (2nd)* [95]	**59.21**	**56.70**	-	-	23	1	whole supernet	gradient
GDAS* [40]	60.23	57.69	-	-	23	0.4	single path	gradient
RandomNAS* [86]	60.34	57.8	73.35	70.86	23	0.25	single path	random
EN^{2}AS	**59.28**	**57.26**	-	-	23	0.67	single path	novelty&reward
EN^{2}AS +	**57.83**	**55.88**	**70.14**	**69.31**	23	0.67	single path	novelty&reward

Since the results on PTB reported in these peer methods are with different training epochs, we reproduce the results of the best models reported in these approaches with the same experimental setting as ours, which are indicated by “*", for a fair comparison. ‡ means that the results are reproduced by DARTS with the same search space as ours. All models are trained with 1600 epochs with 64 batch size to obtain the perplexity, and we also further train our best-found architecture with 3600 epochs on to achieve competitive results.

outputs for all nodes.

2.4.1 Architecture Search for Convolutional Cells

We first consider the most popular DARTS convolutional search space to compare our EN^{2}AS with NAS baselines. In this search space, we search for micro-cell structures on a small dataset CIFAR-10 [76] to stack more cells to form the final structure for architecture evaluation. The best-found cells on CIFAR-10 are then transferred to CIFAR-100 [76] and ImageNet datasets

[38] to evaluate its transferability. The CIFAR-10 dataset only has 10 classes, where each class contains 6000 images resulting in 60000 32x32 colour images in total. The CIFAR-100 dataset is similar to CIFAR-10 dataset, except that CIFAR-100 dataset contains 100 classes where each class contain 6000 32x32 colour images. The ImageNet dataset is much larger than CIFAR datasets, and we consider the ImageNet-1K, which contains 1000 classes, in this thesis. More details of this search space can be found in Sec.1.2.4.

2.4.1.1 Results on CIFAR-10

The comparison results on CIFAR-10 with the state-of-the-art NAS methods are demonstrated in Table 2.1. We report the results of the best found structure from 10 independent search experiments. It is impressive that the Random Search WS could obtain satisfactory results, which randomly sample architectures for supernet training. Random sampling strategy beats most reward-based sampling methods for one-shot NAS with the same search space, except for DARTS (2nd) and BayesNet, which are with an elaborate controller. The result is also in line with the observation from [9]. It is inspiring that the best architecture searched by our EN^{2}AS obtains the state-of-the-art test error on CIFAR-10 for weight sharing NAS. Our approach is also very efficient since the architecture search phase only costs about 7.5 hours (0.3 GPU day), and the memory consumption is the same as training a single architecture. The convolutional cell obtained by our EN^{2}AS is also very efficient, which has fewer parameters than most NAS methods.

2.4.1.2 Results on CIFAR-100 and ImageNet

The architecture evaluation setting on CIFAR-100 is the same as CIFAR-10, and the comparison results are also presented in Table 2.1. Our model could obtain a competitive result with 17.58% Top1 test errors with only 3.13M parameters. We further increase the number of initial filters from 36 to 50 (and the parameters increase to 5.88 M), and our network achieves state-of-the-art results with a test error of 16.45% among all compared methods. The mobile setting on ImageNet also follows [95] and we stacked the best found structure by 14 cells with batch size 128. Our model could obtain a competitive result with Top1/Top5 test errors as 26.66%/8.58% with only 4.5M parameters.

2.4.2 Architecture Search for Recurrent Cells

Beyond the convolutional cell structure search, we also conducted experiments on RNN cell structure search with the Penn Treebank (PTB) dataset [104], which a well-studied dataset for language model. Similar to DARTS convolutional search space, the best searched RNN structure on PTB dataset is also transferred to WikiText-2 (WT2) [107] dataset, which contains more realistic vocabularies and larger corpora, to verify the transferability. For a fair comparison, the search space and hyperparameters were set following [86, 95].

2.4.2.1 Results on PTB

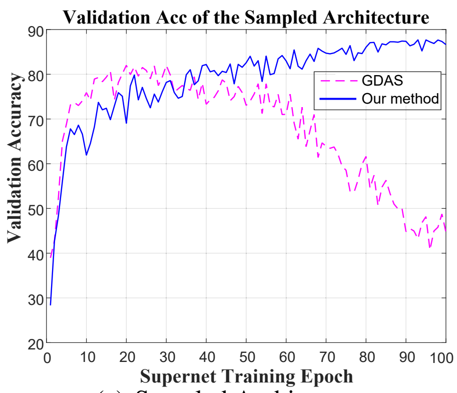
The comparison results on PTB with the state-of-the-art manually-designed architectures and NAS methods are demonstrated in Table 2.2. We can find that the DARTS (2nd) achieves state-of-the-art results on PTB among those NAS methods, which obtains a validation perplexity of 59.21 and a test perplexity of 56.71 and shows the efficiency of gradient method in the recurrent search space. Our EN^{2}AS obtains a competitive validation perplexity of 59.28 and a test perplexity of 57.26, which is much better than DARTS (1st) and on par with the state-of-the-art NAS methods on PTB. As discussed before, our EN^{2}AS is a first-order iterative optimization based on novelty. The results clearly show that enhancing the exploration instead of sampling architecture based on performance reward could improve the supernet predictive ability, as evidenced by the fact that our EN^{2}AS beats the DARTS with first-order approximation. We further train our best found recurrent cell structure with more training epochs and achieve a competitive validation perplexity of 57.83 and test perplexity of 55.88.

2.4.2.2 Results on WT2

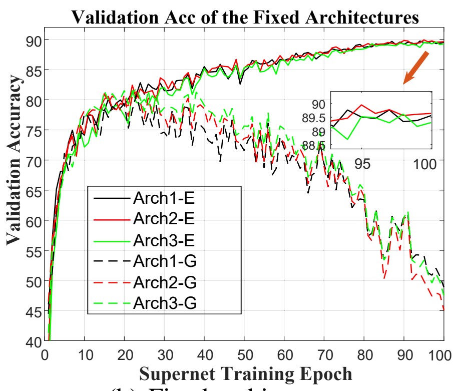
We also transfer those promising models obtained on PTB to WT2 following the experimental settings in [95]. The embedding and hidden sizes are changed to 700, weight decay to 5×10^{-7}, hidden-node variational dropout to 0.15, and other hyperparameter settings are the same as PTB. The results of different models on WT2 are presented in Table 2.2. We train our best model with 3600 epochs on WT2 and achieve a state-of-the-art validation perplexity of 70.14 and a test perplexity of 69.31.

2.4.3 Empirical Comparison with Baselines

Supernet Training Comparison with Reward based Sampling Strategy. As discussed previously, the reward based controller entails the rich-get-richer problem [87], and the multi-model forgetting that occurs during the supernet training will also deteriorate the supernet's validation performance. In this section, we investigate the validation performance of architectures during the supernet training for our proposed novelty search based sampling strategy compared with reward based sampling strategy. We adopt the GDAS [40] as the reward based



(a) Sampled Architecture

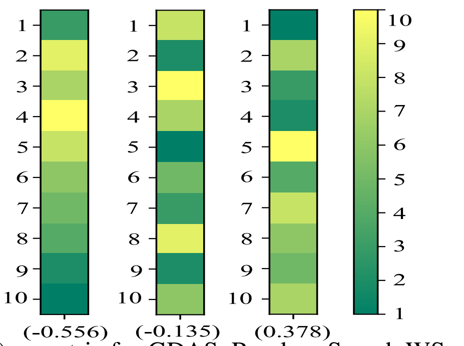


(b) Fixed architectures

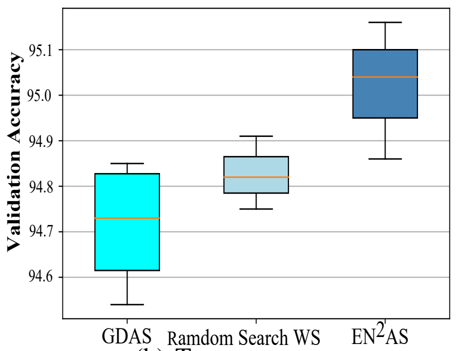
Figure 2.2: Validation accuracy of sampled architecture and fixed architectures during the supernet training for GDAS (dash lines) and EN^{2}AS (solid lines).

sampling baseline as it also only trains a single path in each step during the architecture search phase. We conduct this comparison experiment on CIFAR-10 and train the supernet with the two different sampling methods with 100 epochs, respectively. We tracked the validation accuracy of the sampled architecture in each step and also three fixed architectures through inheriting weights during the supernet training for the two sampling strategies. We present the validation accuracy of the sampled architectures during the supernet training in Fig.2.2 (a), and the validation accuracy of those fixed architectures in Fig.2.2 (b). It is straightforward that architectures are supposed to increase their validation accuracy with the supernet training. However, the performance of all those architectures shockingly gets worse during the supernet training for GDAS after several generations, as shown in Fig. 2.2, which demonstrates the existence of multi-model forgetting [11] induced by reward-based sampling methods in one-shot NAS that makes the supernet unreliable. Differently, our E^{2}NAS always samples abnormal architectures containing few shared connections with previously visited architectures to overcome this forgetting. Fig. 2.2 shows that the performance of architectures does not get worse during the supernet training based on novelty based sampling. It suggests that our proposed novelty search based sampling strategy can effectively relieve the multi-model forgetting and is more reliable than reward based sampling strategy.

Supernet Predictive Ability Evaluation. To demonstrate the effectiveness of our approach in relieving the rank disorder caused by weight sharing, we further conduct experiments to verify the supernet predictive ability of the proposed method compared with random sampling (Random Search WS, depicted as RS WS) and reward gradient based sampling (GDAS) in one-shot NAS. We also replace the baseline evolutionary algorithm in the model selection



(a) τ metric for GDAS, Random Search WS, and EN^{2}AS



(b) Test accuracy

Figure 2.3: The τ metric and mean test accuracy for architectures obtained through different architecture sampling methods.

Table 2.3: Comparison with two baselines on NAS-Bench-201 dataset.

Method	Test Acc(%)	τ metric	s-τ metric
EN^{2}AS	**93.36±0.3**	**0.228±0.066**	**0.333**
RS WS [86]	91.93±1.2	-0.016±0.100	0.111
GDAS [40]	92.05±0.2	-0.067±0.109	-0.092

of our EN^{2}AS with a random search for a fair comparison in this experiment. We conduct this comparison experiment on CIFAR-10 and also train the supernet with different sampling strategies for 100 epochs. Then we randomly sample 10 architectures and evaluate these architectures based on the three different trained supernets. We measure the correlation of architecture ranking based on weight sharing and retraining, and demonstrate the supernet predictive ability of three different sampling methods based on the Kendall Tau (τ) metric [70]. Kendall Tau (τ) is to demonstrate the difference of ranking based on weight sharing and retraining for the three sampling methods. As shown in Figure 2.3 (a), the random sampling and gradient based sampling both obtain negative values that show the rank disorder in the two baselines. Our approach obtains a much better τ=0.378 with a positive correlation between the architecture ranking based on weight sharing and retraining, which indicates that the supernet trained based on novelty search sampling archives a better predictive ability. Since the supernet with better predictive ability tends to obtain better architectures, we further compare the retraining validation accuracy of sampled architectures from the trained supernet based on the three different architecture sampling methods. Figure 2.3 (b) plots the mean retraining validation accuracy, and we could observe that our novelty search based architecture sampling achieves the best results.

2.4.4 Experiments on Benchmark Dataset

The high computational cost of evaluating architectures is the major obstacle of analyzing one-shot NAS methods, and several recent works try to build benchmark datasets [150] to relieve this difficulty. We adopt NAS-Bench-201 [41] as a benchmark dataset to analyze our approach in this experiment. The search space in NAS-Bench-201 contains 4 nodes with 5 associated operations, which results in 15625 cell candidates. Although the search space in NAS-Bench-201 is much simpler than the common search space, the ground-truth test accuracy of all candidates in the search space is reported, which could greatly reduce the computational requirements in the analysis of one-shot NAS methods. We run our EN^{2}AS on NAS-Bench-201 for three independent times with the same experimental settings in [41], and report the mean test accuracy of the best-found architectures in Table 2.3. To evaluate the supernet predictive ability, we further measure the Kendall Tau (τ) metric to demonstrate the difference of ranking based on supernet and ground truth for the three sampling methods. Apart from τ, we also calculate the **s-**τ to measure the stability of generated ranks from different runs, which is defined as $\frac{2}{N(N-1)}\sum_{1\le i<j\le N}\tau(R_i,R_j)$, where $N=3$ in this experiment. We ranked 15 randomly generated architectures based on supernet and the ground-truth to obtain the (τ) and **s-**τ for the three methods, and the results are presented in Table 2.3, where our method beats the two baselines.

2.5 Chapter Summary and Discussion

This chapter originally focuses on resolving the rich-get-richer problem in supernet training for weight-sharing neural architecture search, where a novelty search is proposed to enhance the exploration for architecture sampling during the supernet training. In particular, a novelty search mechanism is developed to efficiently find the most abnormal architecture, and the single-path model is adopted to greatly reduce computational and memory demand. Experimental results show the proposed approach could find the state-of-the-art or competitive CNN and RNN models, and also improve the predictive ability of the supernet in one-shot NAS.

Rather than considering the validation performance-based indicator for architecture sampling, this chapter simply devises a diversity-based indicator to sample diversified architectures for the supernet training in the one-shot NAS. As we can observe from the experimental results, the proposed architecture sampling method obtains to a better predictive ability for the supernet in one-shot NAS, showing in the Sec.2.4.3. The underline reason is that, the performance-based indicator usually leads to the “rich-get-richer" problem while the diversity-based indicator has the potential to fairly train the supernet. This phenomenon suggests that the validation performance by inheriting the weights from the supernet is deceptive, and devising a more effective indicator for the architecture sampling or selection is a promising research direction, which is also in line with several concurrent works [22, 105, 166].

CHAPTER 3

Overcoming Multi-Model Forgetting in One-Shot NAS

3.1 Introduction

Pioneer studies on one-shot NAS follow two sequential steps [9, 33, 86, 114]. They first adopt an architecture sampling controller to sample architectures for training the supernet. Then, a heuristic search method finds promising architectures over a discrete search space based on the trained supernet [54, 86, 114, 147]. Later studies [20, 40, 95, 99, 140, 149, 158] have further employed continuous relaxation to differentiate between architectures so that the gradient descent can be used to optimize the architecture with respect to validation accuracy. The architecture parameters and supernet weights are alternatively optimized through a bilevel optimization method, and the most promising architecture is obtained once the supernet is trained.

Since one-shot NAS evaluates candidate architectures based on the validation accuracy of the weights it inherits from the supernet as opposed to training them from scratch, the success of one-shot NAS relies on a critical assumption that the validation accuracy should approximate the test accuracy after training from scratch or be highly predictive. The authors of the first study on one-shot NAS [9] observed a strong positive correlation between the validation accuracy and the test accuracy when the supernet was trained through random path dropout. Subsequent studies all rightly considered this assumption to be true for all one-shot NAS methods. However, several recent studies have revealed that this assumption may not hold

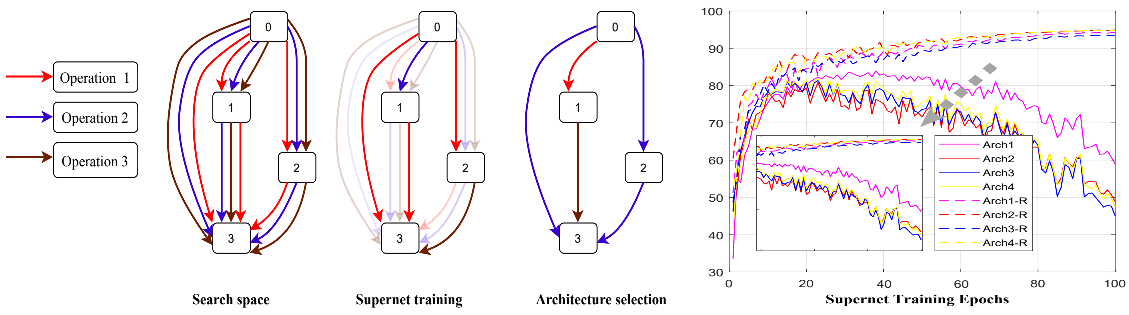


Figure 3.1: **Left**: The general process of one-shot NAS. First, the search space is defined as a supernet containing all candidate architectures. Then a single path of the supernet (an architecture) is trained in each step of the supernet training process. Promising architectures are selected based on the validation accuracy of weights inherited from the trained supernet without the need for training from scratch. **Right**: The validation accuracy for four different architectures during the supernet training. The solid lines (“Arch") are the accuracies returned using weights inherited from the supernet; the dashed lines (“Arch-R") are the accuracies after retraining.

in most popular one-shot NAS approaches. For instance, Sciuto *et al.* [124] show that there is no observable correlation between the validation and test accuracy of the weight-sharing paradigm with ENAS [114], and Adam *et al.* [2] show that the RNN controller in ENAS does not depend on past sampled architectures, which means its performance is the same as a random search. Similarly, Singh *et al.* [129] find that there is no visible progress in terms of the retrained performance for found architectures based on supernet during the architecture search phase, implying the supernet training is useless for improving the predictive ability of one-shot NAS. Further, Yang *et al.* [146] conducted extensive experiments that demonstrated that the current one-shot NAS techniques struggle to outperform naive baselines. Rather, the success of one-shot NAS is mostly due to the design of the search space.

Most one-shot NAS approaches [20, 33, 54, 86] adopt a single-path training method for their supernet training, where only a single path (one architecture) in the supernet is trained in each step. This is the scenario we consider. However, Benyahia et al. [11] observed that when training multiple models (architectures) with partially-shared weights for a single task, the training of each model may lower the performance of other models. Benyahia et al. [11] defined this phenomenon as multi-model forgetting, also known as catastrophic forgetting. They also observed this catastrophic forgetting in one-shot NAS. For example, consider a large supernet containing multiple models with shared weights across them. Sequentially training each model on a single task could mean that the accuracy of each model tends to drop when training another model containing partially-shared weights [11, 124]. This multi-model forgetting in one-shot NAS is illustrated in Fig.3.1 in terms of the validation accuracy of four different architectures during supernet training. What is clear from the figure is that inheriting weights makes performance deteriorate even further during supernet training.

So, although weight sharing can greatly reduce computation hours, it can also introduce catastrophic forgetting into the supernet training, which results in unreliable architecture rankings. Addressing multi-model forgetting during supernet training is an urgent issue if we are to better leverage one-shot NAS and improve the predictive ability of supernets. Hence, we have formulated supernet training as a constrained optimization problem for continual learning to avoid degrading the performance of previous architectures when training a new one.

That said, it is intractable to consider all previously visited architectures. Therefore, only the most representative subset of previous architectures is used to regularize learning of the current architecture. We have devised an efficient greedy novelty search method based on maximizing diversity to select the constraints. We have also implemented the approach in two one-shot baselines. The experimental results demonstrate that our strategy is able to relieve multi-model forgetting in one-shot NAS methods. A summary of our main contributions in this chapter follows.

- We first formulate supernet training with one-shot NAS as a constrained optimization problem of continual learning, where learning the current architecture should not degrade the performance of previous architectures with partially-shared weights.
- We have also designed an efficient greedy novelty search method based on maximizing diversity to select a subset of the constraints that best approximate the feasible region formed by all previous architectures.
- With these two techniques, we then incorporate this **NSAS** loss function (novelty search-based architecture selection) into the RandomNAS [86] and GDAS [40] one-shot NAS baselines to form RandomNAS-NSAS and GDAS-NSAS, with the goal of reducing the level of multi-model forgetting during supernet training. Our best-found models from the common search space [95] returned a competitive test error of 2.59% on CIFAR-10 and only took 0.7 GPU days of search time.
- To improve transferability, we further devised a variant of NSAS, called **NSAS-C**, which searches for "deeper" architectures in the convolutional cell search. Experiments on the common search space demonstrate increased transferability of the found models, and competitive test errors of 16.69% on CIFAR-100 and 25.5% on ImageNet.
- A series of experiments conducted with the NAS benchmark dataset NAS-Bench-201 [41] also verify that **NSAS** significantly reduce forgetting and improve the performance of one-shot NAS baselines.

This chapter is based on the publication "*Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Steven Su, Overcoming Multi-Model Forgetting in One-Shot NAS with Diversity Maximization. In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020*" [159] and "*Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Zongyuan Ge, and Steven Su, One-Shot Neural Architecture Search: Maximising Diversity to Overcome Catastrophic Forgetting. In IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2020*" [161], where the second one is the extended journal version of the previous one. Miao Zhang conceived the original idea of focusing on the catastrophic forgetting in one-shot NAS. The NSAS algorithm was originally devised by Miao Zhang. Steven Su helped Miao Zhang to verify all derivations of theoretical parts in this paper. This was then discussed with Shirui Pan and Huiqi Li. Miao Zhang conducted all experiments, with the help from Shirui Pan, and with feedback from Steven Su. The first version of the paper was written by Miao Zhang with some help from Steven Su. Huiqi Li and Shirui Pan revised the paper many times. The authors Xiaojun Chang, Chuan Zhou, and Zongyuan Ge provided feedback during the writing of the paper.

3.2 Preliminaries

3.2.1 Catastrophic Forgetting

Catastrophic Forgetting is a common problem in artificial general intelligence and multi-task learning. It describes the phenomenon of where a model forgets what it has learned about a previous task(s) after being trained on a new task [60, 74, 80, 112]. Formally, a model with optimal parameters θ_A^* for dataset $\mathscr{D}_A$ will perform substantially less well on $\mathscr{D}_A$ after it has trained on another dataset $\mathscr{D}_B$. Methods to resolve such issues are defined as *continual learning*. Some examples include learning without forgetting (LwF) [89], which adds a response from the old task as a regularization term to prevent catastrophic forgetting, and elastic weight consolidation (EWC) [74], which maximizes the likelihood of a conditional probability $p(\theta \mid \mathscr{D})$, where $\mathscr{D}$ containing two independent data sets $\mathscr{D}_A$ and $\mathscr{D}_B$, and $\mathscr{D}_A$ is not available when trained on $\mathscr{D}_B$.

3.2.2 Multi-model Forgetting

Multi-model Forgetting occurs when training multiple models with a single dataset. Unlike training a model on several tasks sequentially, one-shot NAS applies different models to a single dataset $\mathscr{D}$, e.g., $\theta_a = (\theta_a^p, \theta^s)$ and $\theta_b = (\theta_b^p, \theta^s)$, to a single dataset $\mathscr{D}$, where θ^s is the shared weight and θ_a^p and θ_b^p are private weights. Several recent studies [87, 124, 137] have shown that the interactions between networks can degrade the performance of a whole network, and that catastrophic forgetting with one-shot NAS can lower the performance of previous architectures after training a new architecture in the supernet. To alleviate this problem, Benyahia et al. [11] proposed a weight plasticity loss (WPL), which maximizes the posterior probability $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathscr{D})$ as:

$$\begin{aligned} p(\theta \mid \mathscr{D}) &= \frac{p(\theta_a^p, \theta_b^p, \theta^s, \mathscr{D})}{p(\mathscr{D})} = \frac{p(\theta_a^p \mid \theta_b^p, \theta^s, \mathscr{D}) p(\theta_b^p, \theta^s, \mathscr{D})}{p(\mathscr{D})} \\ &= \frac{p(\theta_a^p, \theta^s \mid \mathscr{D}) p(\mathscr{D} \mid \theta_b^p, \theta^s) p(\theta_b^p, \theta^s)}{p(\theta^s, \mathscr{D})} = \frac{p(\theta_a \mid \mathscr{D}) p(\mathscr{D} \mid \theta_b) p(\theta_b)}{p(\theta^s, \mathscr{D})}. \end{aligned} \tag{3.1}$$

However, it is intractable to calculate $(\theta^s, \mathscr{D})$ in Eq.(3.1), so Benyahia et al. [11] made several presuppositions to make the calculation feasible: a) that θ_a^p and θ_s are independent, and b) that the weights θ_a for previous model are in optimal points. This way, $p(\theta^s, \mathscr{D})$ can be estimated by the distance of θ^s to the optimal θ_s^* with the diagonal of the Fisher information defining the importance of each parameter. In WPL, the loss function to maximize the likelihood of $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathscr{D})$ is calculated as:

$$\mathscr{L}_{WPL}(\theta_b) = \mathscr{L}_c(\theta_b) + \frac{\eta}{2}(\|\theta_b^p\|^2 + \|\theta^s\|^2) + \sum_{\theta_{s_i} \in \theta_s} \frac{\varepsilon}{2} F_{\theta_{s_i}}(\theta_{s_i} - \theta_{s_i}^*), \tag{3.2}$$

where $\mathscr{L}_c$ is the cross-entropy loss function, and $F_{\theta_{s_i}}$ is the diagonal element of the Fisher information matrix corresponding to parameter θ_{s_i}. $F_{\theta_{s_i}}$ is estimated by presupposing parameters (θ_a^p, θ_b^p) are independent, and that θ_s^* are the shared parameters θ^s after the previous model has been trained, which are assumed to be in the optimal points. A detailed derivation of Eq.(3.2) can be found in [11].

Limitations: Weight plasticity loss (WPL) only considers one previous architecture in each step of supernet training. This method is also based on the assumption that the shared weights are optimal. However, these two assumptions are hard to hold when training a supernet in a one-shot NAS scheme given numerous architectures shared weights with the current architecture. Plus, the shared weights are usually far away from the optimal points. To address these concerns, we formulated supernet training with one-shot NAS as a constrained optimization problem, where learning the current architecture does not degrade the performance of previously-visited architectures. We consider a subset of previous architectures as constraints to regularize the learning of the current architecture. We also demonstrate that the loss function of the posterior probability $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathscr{D})$ can be calculated without assuming that the shared weights are optimal when maximizing the diversity of the selected architectures.

Algorithm 2 Greedy Novelty Search

```
Input: constraints archive $\mathscr{M}$, recent architectures archive $\mathscr{C}$, selected architecture $\alpha^m$, $n$.
    $N(\alpha^m, \mathscr{M}) \leftarrow$ calculate the novelty score of $\alpha^m$ in $\mathscr{M}$ based on Eq.(3.5);
 2: for $i = 1, 2, ..., n$ do
        randomly sample an architecture $\alpha^r$ from $\mathscr{C}$;
 4:     if $N(\alpha^r, \mathscr{M}) > N(\alpha^m, \mathscr{M})$ then
            replace $\alpha^m$ with $\alpha^r$;
 6:     end if
    end for
Return: architecture $\alpha^m$.
```

3.3 Novelty Search based Architecture Selection Loss Function

3.3.1 Problem Formulation

Unlike jointly optimizing the posterior probability under the assumption that θ_a is near-optimal as per WPL [11] or keeping the shared weights fixed as per Learn to Grow [88], we formulate supernet training as a constrained optimization problem. Specifically, we enforce the architectures with inherited weights in the current step so as to perform better than the last step. Without loss of generality, we consider a typical scenario where only one architecture in the supernet is trained in each step, and the constrained optimization problem is defined as:

$$\begin{aligned} \mathscr{W}_{\mathscr{A}}^{t} &= \underset{\theta \in \mathscr{W}_{\mathscr{A}}(\alpha^t)}{\texttt{argmin}} \quad \mathscr{L}_{\texttt{train}}(\mathscr{W}_{\mathscr{A}}(\alpha^t)), \\ \texttt{s.t.} \quad & \mathscr{L}_{\texttt{train}}(\mathscr{W}_{\mathscr{A}}^{t}(\alpha^i)) \leq \mathscr{L}_{\texttt{train}}(\mathscr{W}_{\mathscr{A}}^{t-1}(\alpha^i));\ \forall i \in \{0...t-1\}. \end{aligned} \tag{3.3}$$

Here, $\mathscr{L}_{\texttt{train}}(\mathscr{W}_{\mathscr{A}}(\alpha)) = \mathscr{L}_c(\mathscr{W}_{\mathscr{A}}(\alpha)) + \lambda\mathscr{R}(\mathscr{W}_{\mathscr{A}}(\alpha))$, and $\mathscr{W}_{\mathscr{A}}$ represents the total of all weights in the supernet. α^t is the current architecture in step t, and $\mathscr{W}_{\mathscr{A}}(\alpha^t)$ is the weights of α^t inherited from the supernet, and only $\mathscr{W}_{\mathscr{A}}(\alpha^t)$ is optimized in each step t.

3.3.2 Constraints Selection based on Novelty Search

The constraints in Eq.(3.3) prevent the learning the current architecture from degrading the performance of previous architectures as a strategy to overcome multi-model forgetting in one-shot NAS. However, the number of constraints in Eq.(3.3) increases linearly with the step, which makes it intractable to consider all constraints in the optimization. Therefore, it is more practical to try and select a subset of M constraints from the previous architectures that forms as close a feasible region to the original feasible region as possible. Intuitively, maximizing

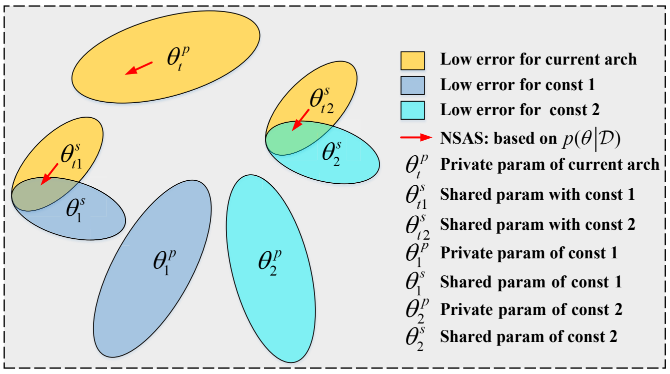

θ_t^p
θ_{t1}^s
θ_1^s
θ_{t2}^s
θ_2^s
θ_1^p
θ_2^p
Low error for current arch
Low error for const 1
Low error for const 2
NSAS: based on $p(\theta|\mathcal{D})$
θ_t^p Private param of current arch
θ_{t1}^s Shared param with const 1
θ_{t2}^s Shared param with const 2
θ_1^p Private param of const 1
θ_1^s Shared param of const 1
θ_2^p Private param of const 2
θ_2^s Shared param of const 2

Figure 3.2: **NSAS** loss function ensures that the learning of current architecture will not deteriorate the performance of previous architectures in the constraint subset.

the diversity of the subset is an efficient way to find the most representative samples from the previous architectures. Based on this observation and motivated by [5], we propose a surrogate for constraint selection:

$$\begin{aligned} &\texttt{maximize}_{\mathcal{M}} \quad \sum_{\alpha^i,\alpha^j\in\mathcal{M}} dis(\alpha^i,\alpha^j), \\ &\texttt{s.t.} \quad \mathcal{M}\subset\{\alpha^1...\alpha^{t-1}\}; |\mathcal{M}|=M, \end{aligned} \tag{3.4}$$

where $dis(\alpha^i,\alpha^j)$ is a function to calculate the distance between architectures. Further, to solve this equation, we proposed a greedy novelty search method to maximize the diversity of the subset. Before the archive is full, all the new coming architectures are added into the subset. Once full, the most similar one to the current architecture is chosen to replace the one that maximizes the novelty score of the archive. Algorithm 2 sets out a simple implementation of our greedy novelty search method. A simple and standard novelty measurement, defined as $N(\alpha,\mathcal{M})$, measures the mean distance of its k-nearest neighbors in $\mathcal{M}$:

$$\begin{aligned} &N(\alpha,\mathcal{M}) = \frac{1}{|S|}\sum_{\alpha^j\in S} dis(\alpha,\alpha^j) \\ &S = kNN(\alpha,\mathcal{M}) = \{\alpha^1,\alpha^2,...,\alpha^k\}. \end{aligned} \tag{3.5}$$

In this chapter, we measure the difference of the input edges for each node in an architecture. The input edge of the same node for two architectures is considered to be the same only when the two edges have the same input node and the same operations. M constraint architectures are then selected from $|\mathscr{C}|$ recent architectures rather than all previous architectures.

3.3.3 The NSAS Loss Function

After finding the M most representative architectures $\{\alpha_1, ..., \alpha_M\}$ by maximizing diversity, we need to forcibly optimize the learning of the current architecture in the feasible region formed by these constraints. A common approach is to convert the constraints into a soft regularization loss or apply a replay buffer [5]. The weights of these architectures in the subset are described as $\{\theta_1, ..., \theta_M\}$. When the selected constraints are converted to a soft regularization loss, the loss function for the constrained optimization problem in Eq. (3.3) could be described as Eq.(3.6):

$$\mathscr{L}_N(\mathscr{W}_{\mathscr{A}}(\alpha^t)) = (1-\beta)(\mathscr{L}_c(\mathscr{W}_{\mathscr{A}}(\alpha^t)) + \lambda\mathscr{R}(\mathscr{W}_{\mathscr{A}}(\alpha^t))) + \frac{\beta}{M}\sum_{i=1:M}(\mathscr{L}_c(\mathscr{W}_{\mathscr{A}}(\alpha^i)) + \lambda\mathscr{R}(\mathscr{W}_{\mathscr{A}}(\alpha^i))), \tag{3.6}$$

where $\mathscr{L}_c$ is the cross-entropy loss function, $\mathscr{R}$ is the ℓ_2 regularization term, and β are the trade-offs. The term $\mathscr{L}_N(\mathscr{W}_{\mathscr{A}}(\alpha^t))$ is the **NSAS** loss function. While learning the current architecture α_t, **NSAS** protects the performance of those constraint architectures by ensuring the shared parameters stay in a region of low error for these constraints, as shown schematically in Fig. 3.2.

3.3.4 From Weight Plasticity Loss (WPL) to NSAS

WPL [11] regularizes the learning of current architecture by maximizing the posterior probability $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathscr{D})$, where $\theta_a = \{\theta_a^p, \theta^s\}$ is the weights of the last architecture, $\theta_b = \{\theta_b^p, \theta^s\}$ is the weights of the current architecture, and θ^s is their shared weights. Unlike WPL, which only considers one previous architecture, we consider a subset of previously visited architectures - $\theta_a = \{\theta_1, ..., \theta_M\} = \{(\theta_1^p, \theta_1^s), ..., (\theta_M^p, \theta_M^s)\}$ - where θ_i^p is the private weights, and θ_i^s is the weights shared with the current architecture. When selected constraints make the following two assumptions hold true, then **Lemma** 1 describes the relationship between WPL and NSAS.

Assumption 1. *The architectures in the constraint subset cover all weights of the current architecture α_t that $\theta_b^{(e)} \subseteq \{\theta_1^{(e)} \cup ... \cup \theta_M^{(e)}\}$ for every edge e in α_t, where $\theta_m^{(e)}$ is the weight of the operations assigned to each edge of α_m.*

Assumption 2. *There are no shared weights among these constraint architectures, i.e, $\theta_1 \cap \theta_2 = ... = \theta_{M-1} \cap \theta_M = \emptyset$.*

Lemma 1. *When the selected constraint architectures satisfy the above two assumptions, the **NSAS** loss function with the WPL can be formulated as:*

$$\mathscr{L}_N(\mathscr{W}_{\mathscr{A}}(\alpha^t)) = \mathscr{L}_c(\mathscr{W}_{\mathscr{A}}(\alpha^t)) + \gamma\mathscr{L}_{WPL}(\mathscr{W}_{\mathscr{A}}(\alpha^t)). \tag{3.7}$$

Algorithm 3 One-Shot NAS-NSAS

Input: $\mathscr{D}_{train}$, $\mathscr{D}_{val}$, $\mathscr{W}$, constraints archive $\mathscr{M} = \emptyset$, M, neural architecture search iteration T, batch size b

```
1: for $t = 1, 2, ..., (T * \texttt{size}(\mathscr{D}_{train})/b)$ do
2:     if $\texttt{size}(\mathscr{M}) < M$ then
3:         sample $\alpha^t$ based on gradient search or random search;
4:         update the weights $\mathscr{W}_A(\alpha)$ by normal loss function, and add architecture $\alpha$ into $\mathscr{M}$;
5:     else
6:         sample $\alpha^t$ based on gradient search or random search;
7:         select the architecture $\alpha^m$ that is most similar to $\alpha^t$ from $\mathscr{M}$;
8:         replace $\alpha^m$ with $\alpha^t$ to maximize the diversity of $\mathscr{M}$ based on Algorithm 2;
9:         update the $\mathscr{W}_A(\alpha)$ by our proposed loss function in Eq.(3.6);
10:    end if
11: end for
12: Obtain $\alpha^*$ based on RandomNAS or GDAS.
```

Lemma 1 demonstrates that the NSAS loss function not only contains the WPL but also optimizes learning of the current architecture when the appropriate constraints have been selected. Additionally, when a specific number of constraints for a densely-connected search space are set, the strategy of selecting constraints based on maximizing diversity has the potential to see the two assumptions hold true. Take the search space of NAS-Bench-201[41] (as defined in Section 4.4.2) as an example. When the number of candidate operations in each edge $M = 5$ and the diversity of the constraint subset is maximized, those five constraint architectures should cover all possible operations for all edges (**Assumption** 1), and each edge in each constraint architecture should contain different operations (**Assumption** 2).

3.3.5 One-Shot NAS with Novelty Search based Architecture Selection

We implemented our loss function into two popular one-shot NAS: RandomNAS [86] and GDAS [40]. Like the most weight sharing NAS methods, only a single path is trained in each step of the architecture search phase. Therefore, incorporating NSAS into RandomNAS is relatively easy. However, most gradient-based NAS methods, like DARTS [95], train the whole supernet in each step of the supernet training, which would violate both the assumptions set out above. For this reason, we chose GDAS [40] as the gradient method, which uses the Gumbel-Max trick [65, 103, 140] to relax the discrete architecture distribution so as to be continuous and differentiable. The **argmax** function reparameterizes the architecture distribution and samples only one architecture in each supernet training step during the forward pass. The **softmax** function is applied during the backward pass for architecture learning. Algorithm 3 outlines the one-shot NAS with the **NSAS** loss function, called one-shot NAS-NSAS.

3.4 Experimental Result

To evaluate the performance of **NSAS** loss function, we compared baseline versions of Random-NAS [86] and GDAS [40] with our **NSAS** implementations denoted as **RandomNAS-NSAS** and **GDAS-NSAS**. We considered two different search spaces: a common search space adopted by most state-of-the-art one-shot NAS methods [86, 95], and a second NAS-Bench-201 space [41]. The NAS-Bench-201 dataset was specifically designed for one-shot NAS research, so it comes with a guarantee of fair comparison between one-shot NAS methods. The NAS-Bench-201 search space is much smaller than the common search space, and, accordingly, the best test performances for all candidate architectures on all datasets were reported with this search space, relieving the computational concern in the further analysis of one-shot NAS approaches. We first apply **RandomNAS-NSAS** and **GDAS-NSAS** to search for promising neural architectures in the common search space and compared the results with the two baselines and many other current one-shot NAS algorithms. We then further analyzed **NSAS** loss function with the NAS-Bench-201 benchmark dataset. In the next two subsections, we describe the experimental settings for each of these search spaces.

3.4.1 Experimental Results on Common Search Space

Our first set of experiments was to conduct a neural architecture search on the common DARTS convolutional search space and compare with all methods, including our implementations, the baselines and a range of current and state-of-the-arts methods.

3.4.1.1 Architecture Search on CIFAR-10

With this experiment, we searched for micro-cell structures in the search space and formed the final structure by stacking the cells in series. To compare the performance of one-shot NAS-NSAS with state-of-the-art NAS methods, we follow DARTS's experimental setting in [95]. We conducted the architecture search several times with different random seeds to obtain the architectures, and then retrained them to pick the best architectures based on retraining validation performance. The first block of Table 3.1 contains the NAS methods without weight sharing. The approaches in the second block are the one-shot NAS methods. The comparison results are provided in Table 3.1. As shown, the one-shot NAS methods, which also obtain competitive results, are much more efficient than NAS methods without weight sharing. The

Table 3.1: Results with the existing NAS approaches on CIFAR-10 and CIFAR-100.

Method	Test Error (%)		Param.	FLOPs	Search	Memory	Search
	CIFAR10	CIFAR100	(M)	(M)	Cost	Consumption	Method
NASNet-A [172]	2.65	17.81	3.3	-	1800	Single path	RL
AmoebaNet-A [118]	3.34±0.06	-	3.2	-	3150	Single path	EA
Hierarchical Evo [94]	3.75±0.12	-	15.7	-	300	Single path	EA
PNAS [93]	3.41±0.09	17.63	3.2	-	225	P Single path	SMBO
IRLAS [53]	2.60	-	3.91	-	-	Single path	RL
IRLAS-differential [53]	2.71	-	3.43	-	-	Single path	RL
NAO [99]	3.18	-	10.6	-	1000	Single path	Gradient
NAO-WS [99]	3.53	-	2.5	-	-	Single path	Gradient
SETN (T=1K) [42]	2.69	17.25	4.6	606	1.8	Single path	Gradient
ENAS [114]	2.89	18.91	4.6	-	0.5	Single path	RL
SNAS [140]	2.85±0.02	20.09	2.8	422	1.5	Whole Supernet	Gradient
PARSEC [21]	2.86±0.06	-	3.6	485	0.6	Single path	Gradient
BayesNAS [170]	2.81±0.04	-	3.4	-	0.2	Whole Supernet	Gradient
RENAS [26]	2.88±0.02	-	3.5	-	6	-	RL&EA
MdeNAS [169]	2.55	17.61	3.8	599	0.16	Single path	MDL
DSO-NAS [167]	2.84±0.07	-	3.0	-	1	Whole Superne	Gradient
WPL [11]	3.81	-	-	-	-	Single path	RL
XNAS [109]	**2.57±0.09***	**16.34**	3.7	596	0.3	-	Gradient
PDARTS [25]	**2.50**	**16.63**	3.4	532	0.3	-	Gradient
PC-DARTS [142]	**2.57±0.07**	17.11	3.6	557	0.3	-	Gradient
Random baseline [95]	3.29±0.15	-	3.2	-	4	-	Random
DARTS (1st) [95]	2.94	-	2.9	501	1.5	Whole Supernet	Gradient
DARTS (2nd) [95]	2.76±0.09	17.54	3.4	528	4	Whole Supernet	Gradient
GDAS [40]	2.93	18.38	3.4	519	0.21	Single path	Gradient
GDAS-NSAS	2.75±0.08	18.02±0.05	3.5	528	0.4	Single path	Gradient
GDAS-NSAS-C	2.70±0.07	**16.70±0.08**	3.3	520	0.4	Single path	Gradient
RandomNAS [86]	2.85±0.08	17.63	4.3	612	2.7	Single path	Random
RandomNAS-NSAS	**2.59±0.06**	17.56±0.05	3.1	489	0.7	Single path	Random
RandomNAS-NSAS-C	**2.65±0.05**	**16.69±0.06**	3.5	552	0.7	Single path	Random

“*" indicates the results were reproduced with the best-reported cell structures in the original paper but with the same experimental settings as all the other comparators. Methods with “-” in the CIFAR-100 experiment were not reproduced because either they had different search spaces or did not report their best structures. All models were trained for 600 epochs, and we trained our best-searched architecture with 3 different random seeds to get the statistical results. “P Single path" means that the search space progressively increases during the architecture search, while only a single path is trained at each step of supernet training.

traditional NAS methods requires hundreds or even thousands GPU days, while weigh-sharing NAS methods usually need less than one GPU day, showing that weight-sharing is an efficient way to reduce computational cost. We can also find that, compared to RandomNAS and GDAS, RandomNAS-NSAS and GDAS-NSAS greatly improve the search results, where The **NSAS** loss function decreased the test errors from 2.85% for RandomNAS to 2.59%, from 2.93% for GDAS to 2.75%, demonstrating the effectiveness of NSAS at improving the predictive ability of the supernet for different frameworks. More important, our RandomNAS-NSAS' results were competitive compared to the other NAS methods, with a 2.59% test error and only 489M FLOPs. This is an inspiring result to validate our strategy for overcoming multi-model forgetting. Since our **NSAS** evaluate more architectures during supernet training, the

Table 3.2: Results with manual-designed architectures and NAS approaches on the ImageNet dataset.

Method	Test Error (%) ImageNet	Param. (M)	FLOPs (M)
Inception-v1 [132]	30.2	6.6	1448
MobileNet V2 [121]	25.3	6.9	585
ShuffleNet 2× (V2) [100]	25.1	5	591
NASNet-A [173]	26.0	5.3	564
AmoebaNet-A [118]	25.5	5.1	555
PNAS [93]	25.8	5.1	588
SNAS [140]	27.3	4.3	522
SETN [42]	25.7	5.4	599
PARSEC [21]	26.3	5.5	-
BayesNAS [170]	26.5	3.9	-
MdeNAS [169]	26.8	6.1	595
DSO-NAS [167]	26.2	4.7	571
PC-DARTS [142]	25.7* (25.1†)	5.3	586
DARTS (2nd) [95]	26.7	4.7	574
GDAS [40]	27.5	4.4	497
GDAS-NSAS	26.7	5.1	564
GDAS-NSAS-C	25.9	5.2	565
RandomNAS [86]	27.1	5.4	595
RandomNAS-NSAS	26.1	5.2	581
RandomNAS-NSAS-C	**25.5** (**24.65**†)	5.4	593

† indicates that the architecture evaluation settings are following PC-DARTS [142], with a warm-up linear learning rate scheduler.

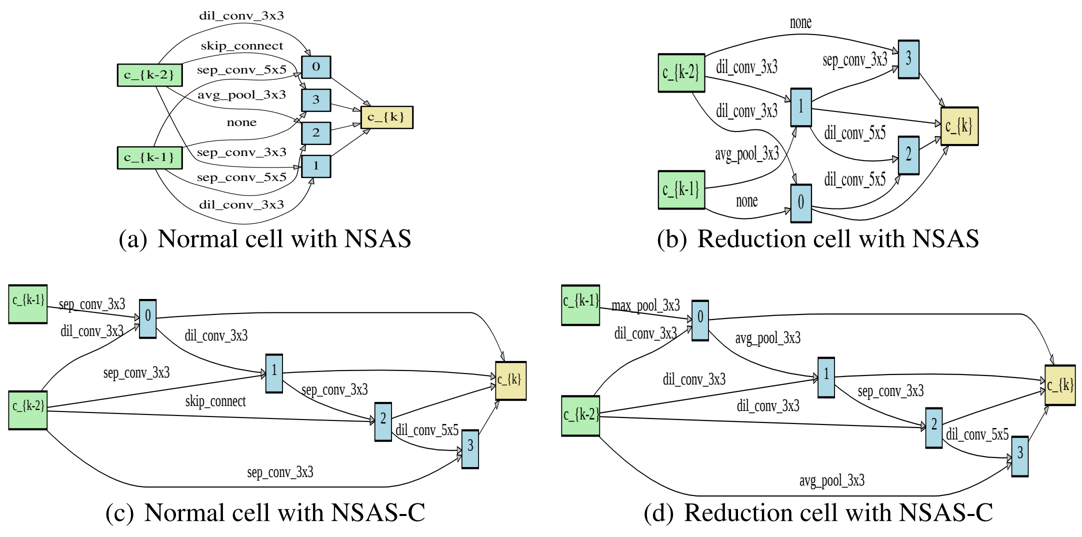
search cost is slightly higher than the baselines. However, it still efficient in the sense that the supernet training in RandomNAS-NSAS only took 0.7 GPU days for and only 0.4 GPU days for GDAS-NSAS.

3.4.1.2 Convolutional Cell Search with Depth Constraint to Improve Transferability

In the next experiments, we transferred the best-found architectures from CIFAR-10 to CIFAR-100 and ImageNet datasets to evaluat their transferability. The results on CIFAR-100 are reported in Table 3.1 and ImageNet are reported in Table 3.2. The first block of Table 3.2 contains manually-designed architectures and the NAS methods without weight sharing. The second block contains one-shot NAS methods. We trained RandomNAS-NSAS with 52 initial channels C and RandomNAS-NSAS-C with $C = 50$, GDAS-NSAS and GDAS-NSAS-C were set to $C = 50$, and the FLOPs were restricted to less than 600M on the ImageNet dataset.

From Table 3.1 and Table 3.2, we can see that the **NSAS** loss function improves the performance of RandomNAS and GDAS with both datasets. However, although the **NSAS** methods yielded remarkable performance with CIFAR-10, the performance was not as impressive with CIFAR-100 and ImageNet. For example, **NSAS** decrease RandomNAS' test error from 2.85% to 2.59% on CIFAR-10, but only from 17.63% to 17.56% with CIFAR-100. Similarly, with ImageNet, the improvement was only 27.1% to 26.1%. More importantly though, the architectures returned by RandomNAS-NSAS on CIFAR-10 were competitive, while XNAS [109] was the superior method with CIFAR100 and ImageNet, thus demonstrating better transferability. XNAS [109] suggests that the architectures with "deeper" cell structures should provide superior performance with the ImageNet dataset. The authors also observe that most NAS methods usually return shallower cells with a larger width after searching CIFAR-10. For example, the architectures found by PC-DARTS and **NSAS** on CIFAR-10 are extremely shallow, which gave excellent results with CIFAR-10 but poor results with ImageNet. Conversely, the architectures found by XNAS, PDARTS, and PC-DARTS on ImageNet were much deeper and the results were impressive. A recent study on neural network optimization [123] gives a hint as to why most NAS methods prefer wider networks. The authors observe that width is a key factor affecting the convergence speed of neural networks, and therefore wider networks are easier to train. Based on this observation, the wider (shallower) architecture in the NAS search space reduces the loss with limited supernet training epochs, and has a higher probability of being chosen.

These findings suggest that encouraging NAS methods to search for "deeper" architectures could improve transferability; hence, our variant of **NSAS-C**, NSAS with depth constraint . The depth constraint in NSAS-C force NAS to search for "deeper" architectures. Simply put, the structure of the architectures are "fixed" to a depth so that the inputs of each node are the outputs of its previous node and the output of the previous cell. Figure 3.3 (c) and (d) show an example. This way, we only need to determine the operation of each edge in an architecture. All remaining experimental settings stay the same. Table 3.1 and 3.2 report the transferability of the models found by **NSAS-C** with CIFAR-100 and ImageNet. Compared to NSAS, the results are excellent. The best found cells by NSAS-C are shown in Fig. 3.3 (c) and (d), with competitive performance of 2.69%, 16.58%, and 25.5% test errors on CIFAR-10, CIFAR-100 and ImageNet, respectively. From Table 3.1, we can see that restricting the architecture depth is a very effective way of improving the transferability of NAS methods, e.g., 16.75% for GDAS-NSAS compared to 18.02% for GDAS with CIFAR-100, and RandomNAS-NSAS from 17.56% to 16.69%. Similarly, as shown in the ImageNet results in Table 3.2, NSAS-C again improves transferability, with improving GDAS-NSAS from 26.7% to 25.9%, and RandomNAS-NSAS from 26.2% to 25.5%.



(a) Normal cell with NSAS (b) Reduction cell with NSAS

(c) Normal cell with NSAS-C (d) Reduction cell with NSAS-C

Figure 3.3: The best found cells with NSAS and NSAS-C on CIFAR-10.

3.4.1.3 Supernet Predictive Ability Comparison

Multi-model Forgetting in One-Shot NAS To demonstrate catastrophic forgetting in a neural architecture search, we conducted experiments with a convolutional cell search task. The results show the differences between weight sharing and a retraining-based architecture ranking strategy. We tracked the validation accuracy of inheriting weights for several fixed sampled architectures with GDAS and also plotted the validation accuracy over 100 epochs when retraining these separate architectures from scratch in Fig. 3.1. From the results, we find that the validation accuracy of the architectures that directly inherit weights from the supernet fluctuate tremendously, making it hard to verify the quality of the architecture. What is worse is that the architecture ranking results completely violate the primary hypothesis of weight sharing NAS, i.e., that architectures with higher validation performance based on weight sharing should yield better retraining performance. It is worth noting that the performance of the architectures that inherited weights gets even worse during the supernet training, as shown in Fig. 3.1.

We also tracked the validation accuracy of weight sharing and retraining during the supernet training with RandomNAS-NSAS and GDAS-NSAS. The results are given in Fig. 3.4. We find that the NSAS loss function substantially alleviates multi-model forgetting with one-shot NAS. The plots of the validation accuracy with the inherited weight methods are much smoother, especially for architectures 2, 3, and 4. Moreover, performance does not decrease during supernet training. This is clearly a more reliable method.

Supernet Predictive Ability Comparison RandomNAS-NSAS and GDAS-NSAS should also alleviate ranking errors. The experiments we conducted to verify the architecture ranking

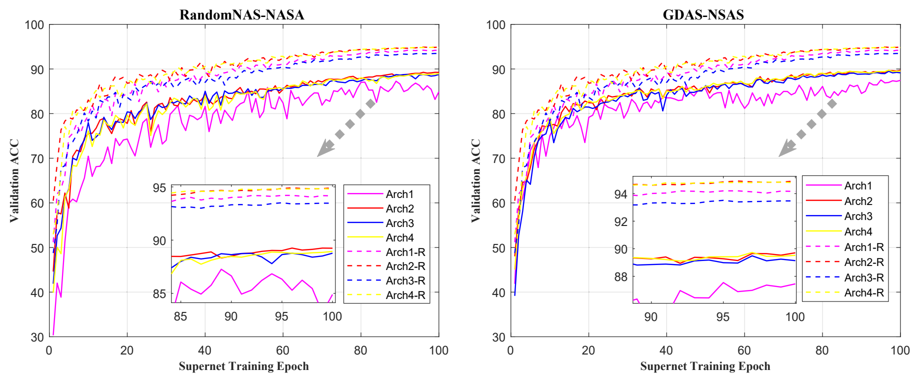
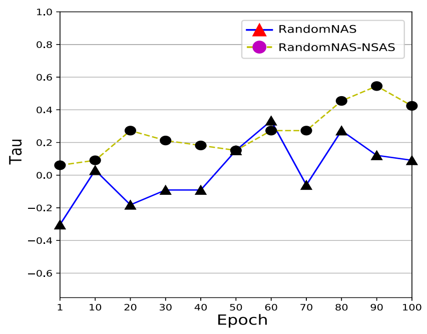
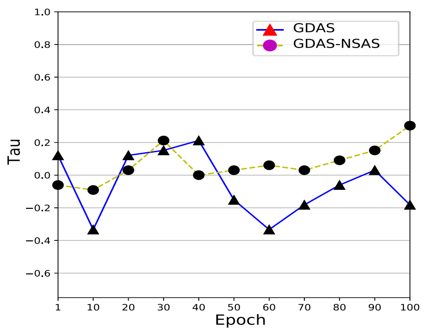


Figure 3.4: The validation accuracy during supernet training for four different architectures with RandomNAS-NSAS and GDAS-NSAS. The solid lines (“Arch") indicate the validation accuracy with weights inherited from the supernet, and the dashed lines (“Arch-R") represent the validation accuracy after retraining.

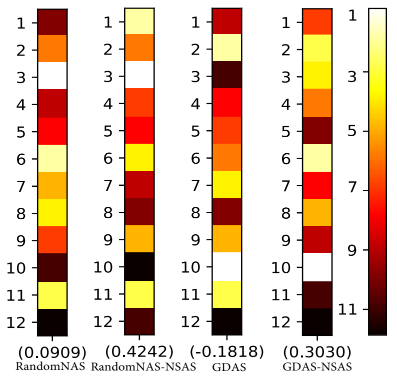


(a) RandomNAS

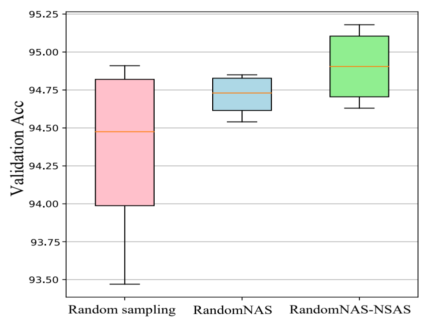


(b) GDAS

Figure 3.5: The Kendall Tau metric (τ) of architecture ranking based on weight sharing and retraining.



(a) Architectures ranking difference after supernet training



(b) Retraining validation accuracy

Figure 3.6: (a) The architecture ranking differences between retraining and inheriting weights from a trained supernet with RandomNAS, RandomNAS-NSAS, GDAS, and GDAS-NSAS (from left to right, respectively). (b) The mean retraining validation accuracy for the architectures found through different methods.

predictions are shown in Figures 3.5 and 3.6. For these experiments, we sampled four of the best architectures over four rounds with RandomNAS and RandomNAS-NSAS, and four randomly sampled from the previous experiment. Then we individually trained these 12 architectures from scratch and calculated the correlation between the architecture ranking and the validation accuracy for each of the weight sharing and retraining approaches. Fig. 3.5 presents the Kendall Tau (τ) metric [70, 169] of the architecture rankings based on weight sharing and retraining. The results show the difference in rankings between the normal cross-entropy loss function and the NSAS loss function. Fig. 3.6 (a) gives the final Kendall Tau (τ) metric values for RandomNAS and GDAS with different loss functions after supernet training. Here, the normal loss function has poor supernet predictive ability, with only $\tau = 0.0909$ and $\tau = -0.1818$ for RandomNAS and GDAS, respectively. Although the supernet trained with the NSAS loss function was not able to provide identical architecture rankings, the positive correlations matched the Kendall Tau metrics ($\tau = 0.4242$ and $\tau = 0.3030$ for RandomNAS-NSAS and GDAS-NSAS, respectively). From this we surmise that a supernet with better predictive ability tends to provide architectures with better retraining performance. Fig. 3.6 (b) plots the mean retraining validation accuracy of the sampled architectures with various methods. We found that RandomNAS-NSAS achieved better results than RandomNAS, further verifying its effectiveness.

3.4.2 Experimental Results on NAS-Bench-201

Evaluating architectures in one-shot NAS is much more computationally intensive than an architecture search, so most state-of-the-art one-shot NAS methods only report the results of their best-found architectures. Comprehensive statistical analyses of the results are usually also overlooked due to computational limitations. Several concurrent studies [41, 75, 150, 155] have tried to address this problem by building benchmark datasets for NAS. With these datasets, researchers can analyze their one-shot NAS methods without evaluating numerous architectures. To analyze our approach in this way, we chose NASBench-201 [41] as a benchmark evaluation set. NAS-Bench-201 is easy to use and can be directly applied to most one-shot NAS algorithms. It also reports the performance of all candidate architectures on CIFAR-10, CIFAR-100, and ImageNet, making it sufficient to evaluate one-shot NAS algorithms. We did not restrict the width of architectures in the NAS-Bench-201 search space because the architectures are densely connected and have the same depth, making that constraint somewhat moot. To verify and further analyze the effectiveness of the NSAS loss function, we conducted three sets of experiments with this search space: 1) a comparison of the baselines; 2) a study of the hyperparameter settings; and 3) a study of the constraint selection strategies.

Table 3.3: Results of one-shot NAS baselines on NAS-Bench-201.

Method	CIFAR-10		CIFAR-100		ImageNet-16-120	
	Valid(%)	Test(%)	Valid(%)	Test(%)	Valid(%)	Test(%)
ENAS [114]	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
DARTS (1st) [95]	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 (2nd) [95]	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
SETN [42]	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
RandomNAS [86]	80.42±3.58	84.07±3.61	52.12±5.55	52.31±5.77	27.22±3.24	26.28±3.09
GDAS [40]	90.00±0.21	93.51±0.13	71.14±0.27	70.61±0.26	41.70±1.26	41.84±0.09
RandomNAS* [86]	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
RandomNAS-NSAS	**89.20±0.31**	**92.61±0.10**	**68.62±1.94**	**68.47±1.73**	**41.17±2.16**	**41.68±1.91**
GDAS* [40]	89.88±0.33	93.40±0.49	70.95±0.78	70.33±0.87	41.28±0.46	41.47±0.21
GDAS-NSAS	**89.99±0.29**	**93.55±0.16**	**71.17±0.44**	**70.69±0.33**	**41.85±1.71**	**42.14±1.40**

“*" indicates that we reproduce the results with same random seeds as our approaches. All results in the first block are from [41]. The hyperparameters M and β were set to 5 and 0.5 for RandomNAS-NSA, 2 and 0.2 for GDAS-NSAS. We run each scenario for 4 independent times with random seed { *0, 1, 100, 101* } following the experimental settings in [41].

Table 3.4: Analysis of one-shot NAS with various settings for β and M on the NAS-Bench-201 dataset.

Method	β	CIFAR-10		CIFAR-100		ImageNet-16-120	
		Valid Acc(%)	Test Acc(%)	Valid Acc(%)	Test Acc(%)	Valid Acc(%)	Test Acc(%)
$(M = 2)$	0	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
	0.2	86.38±4.35	89.58±2.82	63.64±6.36	64.72±4.47	34.68±7.80	34.19±5.72
	0.5	85.13±1.43	88.09±1.23	58.73±6.82	60.77±3.85	31.67±3.58	30.35±4.18
	0.8	86.82±2.44	90.14±2.35	64.41±4.34	64.27±3.26	35.06±4.81	34.82±6.06
$(M = 2)$	0	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
	0.2	88.38±4.35	90.74±1.31	63.64±6.36	64.83±4.05	36.53±6.50	36.08±4.68
	0.5	87.93±1.63	91.23±1.03	66.03±3.46	66.17±4.12	38.14±2.76	38.72±3.11
	0.8	85.58±2.59	88.78±2.23	64.12±4.55	65.06±3.35	34.80±4.24	34.37±5.57
$(M = 2)$	0	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
	0.2	86.73±1.30	90.64±0.99	62.38±4.57	66.42±1.47	36.68±3.80	37.59±3.72
	0.5	87.13±1.43	91.04±0.43	64.43±4.82	64.77±3.61	36.86±3.70	36.35±4.15
	0.8	88.52±0.74	92.04±0.50	67.40±2.22	67.62±1.94	39.91±4.50	40.61±3.51
$(M = 2)$	0	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
	0.2	88.45±0.47	91.36±0.74	65.79±0.59	65.58±0.42	37.69±1.23	37.31±2.42
	0.5	**89.20±0.31**	**92.61±0.10**	**68.62±1.94**	**68.47±1.73**	**41.17±2.16**	**41.68±1.91**
	0.8	88.42±0.30	91.56±0.36	66.77±4.83	66.58±4.99	39.53±4.85	38.64±5.13

3.4.2.1 Empirical Comparison with Baselines

The results of the comparison study are presented in Table 3.3, and all experimental settings follow [41]. The statistical results were calculated from independent searches with four different *random seeds*. We found the NSAS loss function significantly improved the performance of the two baselines. RandomNAS-NSAS, in particular, achieved a test accuracy of 92.61%±0.10 on CIFAR-10 compared to RandomNAS at only 88.14%±0.21. Similarly, GDAS-NSAS yielded a test accuracy of 93.55%±0.16 on CIFAR-10 compared to the 93.40%±0.49 of GDAS. Furthermore, the architectures searched by RandomNAS-NSAS and GDAS-NSAS also performed better when transferred to the larger CIFAR-100 and ImageNet datasets.

3.4.2.2 Hyperparameter Study

As described in Eq.(3.6), the trade-off β and the number of constraints M are important hyperparameters for the NSAS loss function $\mathscr{L}_N$. we studied the impact of β concurrent with M.

First, we considered to fix the number of constraints, and investigated the impact of four different settings of β on multi-model forgetting with one-shot NAS. In this experiment, we took the RandomNAS-NSAS with $M = 5$ as example. The results, shown in the last four rows of Table 3.4, indicate that using constraints to regularize the supernet training can greatly improve test performance and, additionally, that RandomNAS-NSAS is somewhat sensitive to β. More important, with different number of constraints ($M = 2,3,4$), the results all demonstrated our regularization method can enhance the performance, where our RandomNAS-NSAS with different M and β all outperformed the baseline.

We then fixed $\beta = 0.2$ and varied M, also to analyzed the impact on forgetting. Given there are only five candidate operations in the NAS-Bench-201 search space, there are no shared weights among constraints only when $M \leq 5$. The four settings for M in this experiment were 2, 3, 4, and 5. From the results, we found that, again, RandomNAS-NSAS seemed sensitive to the number of constraints, and the larger $M = 5$ gave much better results than the other scenarios for RandomNAS-NSAS with CIFAR10, CIFAR-100, and ImageNet. More interestingly, there was a large performance gain between $M = 4$ and $M = 5$ with RandomNAS-NSAS. One underlying reason may be that $M = 5$ has the potential to ensure the two assumptions hold true, as discussed in Section 3.3.3. Similarly, the tendency also exists in the remain 2 different β, that increasing the number of constrained architectures can enhance the performance. More details on these results can be found in Table 3.4.

Overall, Table 3.4 considered three settings for β and four settings for M, and presented the

Table 3.5: Analysis of the one-shot NAS with constraint selection strategies on CIFAR-10.

METHOD	β	NSAS Test Acc(%)	NSAS-G Test Acc (%)	NSAS-LG Test Acc(%)	RG Test Acc(%)	LoW Test Acc(%)	LoW-R Test Acc (%)
RandomNAS	0.2	89.58±2.82	**91.74±1.16**	91.11±2.16	89.24±2.24	90.13±4.25	89.72±3.64
Test Acc(%)	0.5	88.09±1.23	90.37±2.14	**91.19±0.79**	77.30±19.76	89.86±2.91	86.85±10.29
(88.14±0.21)	0.8	90.14±2.35	**92.52±0.48**	91.18±1.73	89.53±0.24	89.39±3.88	85.54±7.18
GDAS	0.2	**93.55±0.16**	93.37±0.27	93.52±0.30	93.29±0.19	93.51±0.14	93.40±0.26
Test Acc(%)	0.5	93.49±0.24	**93.51±0.14**	93.46±0.27	93.31±0.30	93.47±0.29	93.36±0.21
(93.40±0.49)	0.8	93.32±0.18	93.29±0.29	**93.55±0.20**	93.40±0.28	**93.55±0.20**	**93.55±0.16**

Table 3.6: Analysis of the one-shot NAS with constraint selection strategies on CIFAR-100.

METHOD	β	NSAS Test Acc(%)	NSAS-G Test Acc (%)	NSAS-LG Test Acc(%)	RG Test Acc(%)	LoW Test Acc(%)	LoW-R Test Acc (%)
RANDOMNAS	0.2	64.72±4.47	**67.22±2.20**	66.58±3.43	63.66±3.97	64.06±7.89	60.68±9.04
Test Acc(%)	0.5	60.77±3.85	65.30±3.49	**66.74±2.66**	47.28±27.17	64.27±6.43	59.47±13.13
(63.40±4.52)	0.8	64.27±3.26	**67.83±1.66**	66.01±2.94	64.13±0.56	61.37±8.95	58.32±8.82
GDAS	0.2	70.69±0.33	70.49±0.61	**70.86±0.90**	70.43±0.56	70.53±0.34	70.40±0.51
Test Acc(%)	0.5	70.53±0.30	70.57±0.23	**70.69±0.67**	70.21±0.44	70.10±0.70	70.25±0.38
(70.33±0.87)	0.8	70.33±0.41	70.28±0.58	**70.80±0.55**	70.40±0.60	70.78±0.19	70.38±0.45

Table 3.7: Analysis of the one-shot NAS with constraint selection strategies on ImageNet-16-120.

METHOD	β	NSAS Test Acc(%)	NSAS-G Test Acc (%)	NSAS-LG Test Acc(%)	RG Test Acc(%)	LoW Test Acc(%)	LoW-R Test Acc (%)
RANDOMNAS	0.2	34.19±5.72	**39.58±2.60**	37.79±5.11	34.38±5.02	36.32±8.07	30.64±13.19
Test Acc(%)	0.5	30.35±4.18	35.14±3.33	**39.81±2.81**	31.96±26.53	36.81±5.47	29.11±17.77
(33.83±3.17)	0.8	34.82±6.06	**40.14±2.60**	38.34±4.65	34.94±2.29	33.75±7.91	28.38±9.84
GDAS	0.2	42.14±1.40	**42.26±0.20**	41.71±0.57	41.35±0.13	42.16±1.30	41.68±1.18
Test Acc(%)	0.5	42.20±1.31	42.16±1.30	**42.29±1.00**	42.45±1.07	41.32±1.56	42.20±1.25
(41.47±0.21)	0.8	41.78±0.89	**42.21±0.16**	41.64±1.01	41.68±0.96	41.84±1.01	41.64±1.21

results of RadnomNAS-NSAS on CIFAR-10, CIFAR-100, and ImageNet-16-120. In general, a larger β and a larger M provided the better results. In the next subsection, we discuss the benefits of holding to the two assumptions with NSAS, with respect to the constrained architecture selection strategy.

3.4.2.3 Analysis of Constraints Selection

Although we demonstrate the theoretical benefits of the NSAS loss function in relieving catastrophic forgetting Section 3.3.3, and the experiments in Sections 3.4.2.1 and 5.2.2 support these theories, at least for one-shot NAS, is it still open to debate as to whether these improvements are due to the constraint selection strategy or simply because of the regularization. In this section, we further conduct an ablation study to investigate the impact of different architecture selection strategies.

Directly maximizing the diversity of the constraint subset, as per Section 3.3.4, easily holds **Assumption** 2, but it does not guarantee that **Assumption** 1 will hold. Therefore, we devised two variants of the NSAS loss function, both of which strictly observe the assumptions when selecting constraints. These are **NSAS-G** and **NSAS-LG**. The difference between the two concerns treatment of the last architecture. More specifically, with **NSAS-G**, the constraints are generated randomly, maximizing diversity, but the last constraint θ_M is generated by complementing the operations contained in the current architecture α^t that have not been covered in the previous constraints. This means all selected architectures $\{\theta_1,...,\theta_M\}$ covering all parameters of α_t such that $\theta_t \subseteq \{\theta_1 \cup ... \cup \theta_M\}$. With **NSAS-LG**, however, the last architecture α^{t-1} is first added into the subset, and remaining constraints are generated as following **NSAS-G**. This is to test the common thinking on catastrophic forgetting that the last architecture deteriorates performance the most.

We evaluated all three loss functions - **NSAS**, **NSAS-G**, and **NSAS-LG** - along with three naive architecture selection methods added to the RandomNAS and GDAS baseline to regularize the supernet training. Thus, the six loss functions were:

- **NSAS** - which selects constraints through maximizing diversity.
- **NSAS-G** - a variant of **NSAS** described above.
- **NSAS-LG** - a variant of **NSAS** described above.
- **RG** - randomly generates architectures to form the constraint subspace.
- **LoW** - only adds the last architecture α^{t-1} to the constraint subset.
- **LoW-R** - adds the last architecture α^{t-1} to the constraint subset plus randomly generated constraints.

Table 3.5, 3.6, and 3.7 show the test accuracies for the CIFAR-10, CIFAR-100, and ImageNet-16-120 datasets. We set the number of constraints $M = 2$ in this experiment, to more precisely investigate the effect of architecture selection and quantitatively analyze the constraint selection strategies. The results for one-shot NAS without relieving forgetting are shown in the first column of Table 3.5, 3.6, and 3.7. All NSAS methods improved performance but, interestingly, some of the naive constraint selection methods did as well, which indicates that overcoming catastrophic forgetting is a promising research direction for one-shot NAS. For example, **LoW** improved performance simply by including the last visited architecture in the regularization. However, these results also show the importance of constraint selection strategy, as randomly selecting constraints can reduce performance. We observed that **RG** was the worst method in all cases, with a reduction in test accuracy from 88.14±0.21 to 77.30% ±19.76 for RandomNAS, and from 93.40±0.49 to 93.29±0.19 for GDAS in CIFAR-10. Moreover, the **LoW-R** strategy of adding more randomly generated constraints into the replay buffer yielded even worse results than **LoW** in most cases. These results suggest that randomly selecting constraints does not alleviate multi-model forgetting with one-shot NAS.

As for the **NSAS** and its variants, **NSAS-G** and **NSAS-LG**, all improved performance significantly. It is interesting that **NSAS** and **NSAS-G** achieved similar results with GDAS. This indicates that Assumption 1, which requires the constraints to cover all parameters of α_t, may not be so important for relieving catastrophic forgetting with gradient-based one-shot NAS while holding to this assumption with RandomNAS did help. Overall, **NSAS-G** achieved much better results than **NSAS** and, in most cases, **NSAS-G** and **NSAS-LG** produced the best results. Thus, simultaneously considering the last visited architecture and maximizing the diversity of constraints combined are the two key factors that need to be addressed to relieve catastrophic forgetting with both random sampling-based and gradient-based one-shot NAS.

3.5 Chapter Summary and Discussion

In this chapter, we formulated supernet training as a constrained optimization problem to reduce some of the negative impacts of catastrophic forgetting with one-shot NAS, and multi-model forgetting in particular. Our strategy is to select a representative subset of constraints with a greedy novelty search method. Then the supernet training is regularized in a feasible region with a new novelty search-based architecture selection loss function, i.e., **NSAS** to overcome multi-model forgetting. We implemented **NSAS** into two one-shot NAS baselines - RandomNAS and GDAS - and compared the quality of the architecture selections with and without the new loss function. The results of experiments on the common search space of a neural architecture show **NSAS** and two of its variants improve the predictive ability of the supernet with both convolutional and recurrent cell search. Experiments with the NAS-Bench-201 dataset also suggest that **NSAS** can substantially offset performance degradation due to forgetting with one-shot NAS.

Catastrophic forgetting, which usually exists in the online learning, is also introduced into neural architecture search in this chapter, since the supernet is trained under different paths.

Simply adding a regularization term on the supernet training can greatly improve the predictive ability of the supernet of different NAS frameworks, as analyzed in Lemma 1 and observed from the experimental results. It is clear that the NSAS loss function can easily increase the predictive ability of the supernet, which, in turn, greatly improves the performance of the architectures found by RandomNAS and GDAS. However, supernet training in one-shot NAS is still a problem with much room for further advancements. Devising a more appropriate loss function than the status quo appears to be a promising direction for improving the performance of one-shot NAS methods.

Another interesting finding is that, RandomNAS tends to achieve better performance than GDAS with a DARTS search space, whereas GDAS outperforms RandomNAS with the NASBench-201 space no matter the loss function, showing that the differentiable NAS framework may be more appropriate to small search space. This may be because gradient-based methods typically arrive at the local optimal solution once the supernet is trained. RandomNAS, however, must perform a subsequent model selection process using either a random search or an EA to find a global optimal solution from the trained supernet. Since common search spaces are much more complicated than the one in NAS-Bench-201, a global optimization method will usually outperform a gradient method, while gradient-based NAS is more efficient and effective with simple search spaces.

Part II

Differentiable Neural Architecture Search

CHAPTER 4

Differentiable Neural Architecture Search with Exploration Enhancement

4.1 Introduction

Neural Architecture Search (NAS) has been garnering increasing attention in the deep learning community because it automates the labor-intensive and time-consuming process of neural network design [7, 26, 28, 53, 94]. The downside is that it comes with an extremely high demand for computation power. To mitigate this problem, many recent studies have been devoted to reducing these search costs through a weight-sharing paradigm, also called one-shot NAS [9]. These methods define a supernet that subsumes all possible architectures in the search space. The architectures are then evaluated via a scheme that inherits weights from the supernet. Early weight-sharing approaches to NAS typically adopted a controller that samples the architectures for supernet training [55, 86, 114]. Promising architectures were found by using the trained supernet to search on a discrete search space with a heuristic method. More recent studies have employed continuous relaxation to make the architecture differentiable, which means gradient descent can be used to find the most promising architectures with respect to validation accuracy [20, 40, 58, 62, 95, 99, 140]. This paradigm is sometimes referred to as differentiable NAS, with DARTS being a notable example [95]. DARTS combines both weight sharing [9, 114] and continuous relaxation [20, 40, 95, 140] to reduce the computational cost of a search down to perhaps a few hours on a single GPU.

However, despite its efficiency, the current differentiable NAS approaches are generally considered to be rather unreliable [23, 152]. For example, DARTS does not consistently yield excellent solutions, and the architecture choices get worse and worse as the search proceeds. Performance can even be worse than a random search in some cases [124]. Many refer to this critical weakness as *instability* [152]. From empirical experiments, Zela *et al.* [152] observed that the instability of DARTS is highly correlated to the dominant eigenvalue of the Hessian of the validation loss with respect to an architecture's parameters, and that this dominant eigenvalue increases during the architecture search. Accordingly, they proposed a simple early-stopping criterion based on the dominant eigenvalue to make DARTS more robust. Several recent studies [15, 87, 90, 152] highlight that DARTS tends to choose architectures with more *skip-connections*, which do not always perform well. Liang *et al.*'s [90] solution is to introduce another simple "early stopping" criteria, where the search procedure ends as soon as one cell has two or more *skip-connections*.

One potential reason for the instability of DARTS is that there is no theoretical foundation to show that, with continuous relaxation, an optimization in a continuous latent space is equivalent to an optimization over a discrete space. The lack of injective constraints in a simple continuous relaxation means there can be no guarantee that performing an optimization on a continuous latent space is the equivalent of doing so with a discrete space. With differentiable NAS, this *incongruence* might even increase during the architecture search [23, 152]. In addition, the current differentiable NAS methods use performance rewards as the sole means of determining whether the architecture parameters need to be updated. This method clearly exposes the optimization procedure to the *rich-get-richer problem* [2, 162]. More specifically, the architectures with better performance in the early stages would receive more training, and more frequent updates to their weights. This would, in turn, give those architectures a higher probability of being sampled. Under these conditions, it would be easy to settle on a local optimum.

Another limitation of differentiable NAS is that *catastrophic forgetting problem* can arise during the training process. Differentiable methods assume that learning the inner supernet weights at each step improves the validation performance of all architectures that inherit those supernet weights. However, this assumption may not hold. In practice, each step of supernet training in a weight-sharing NAS usually deteriorates the validation performance of any architecture that partially shares weights with the currently learned architecture [11]. This forgetting problem has not been thoroughly studied with differentiable NAS.

The above limitations faced by existing differentiable NAS approaches are neatly addressed by our proposed algorithm, **E**xploration **E**nhancing **N**eural **A**rchitecture **S**earch with Architecture Complementation (**E^{2}NAS**). The potential for incongruence is addressed by using a variational graph autoencoder with an asynchronous message passing scheme to injectively transform the discrete architectures into an equivalent continuous space. Using an injective approach lends a solid theoretical foundation to the equivalence between performing an optimization in the continuous latent space versus the discrete space [141, 157]. To overcome the *rich-get-richer problem* associated with reward-based gradient methods, we devised a probabilistic enhancement method to encourage intelligent exploration of the latent space during the architecture search. As for the *catastrophic forgetting* problem common to differentiable one-shot NAS, an architecture complementation based continual learning method is further proposed for the supernet training so as to force the supernet to retain its memory of previously visited architectures. To further improve transferability and generalization, the framework incorporates domain knowledge through a regularization strategy, which encourages a deeper search of the cell structure.

From evaluations with **E^{2}NAS** against existing baselines on the benchmark dataset NAS-Bench-201 [41] and the common DARTS convolutional search space [95], we find that **E^{2}NAS** obtains competitive results on the NAS-Bench-201 benchmark dataset [41] outperforming baselines by large margins, and achieves state-of-the-art performance on CIFAR-10, CIFAR-100, and ImageNet datasets in the DARTS [95] search space, with test error 2.37%, 15.77%, and 23.9%, respectively. A summary of our main contributions in this chapter follows.

- This chapter deepens our understanding of exploring differential neural architecture searches in latent space. Our approach involves a variational graph autoencoder that injectively transforms the discrete architecture space into an equivalent continuous latent space, and probabilistic exploration enhancement in the latent space to encourage intelligent exploration during the supernet training.

- We propose an architecture complementation based on continual learning for the supernet training with weight sharing NAS. Further, we theoretically demonstrate the relationship between thethe proposed loss function and the WPL [11] on relieving catastrophic forgetting in weight sharing NAS.

- The framework further includes a novel regularization strategy for searching complicated spaces that integrates expert advice to encourage a deeper search. Experiments on the DARTS convolutional search space demonstrate the increased transferability of the found models.

- Extensive experiments on the benchmark dataset NAS-Bench-201 [41] and a DARTS convolutional search space illustrate the effectiveness of E^{2}NAS in comparison to more than 20 existing differentiable NAS baselines. On NAS-Bench-201, E^{2}NAS outperformed all baselines. With the DARTS space, E^{2}NAS' best-selected architecture returned competitive results on CIFAR-10, CIFAR-100, and the ImageNet datasets, with test 2.37%, 15.77%, and 23.9%, respectively.

This chapter is based on a publication "*Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Zongyuan Ge, Steven Su, Differentiable Neural Architecture Search in Equivalent Space with Exploration Enhancement. In Annual Conference on Neural Information Processing Systems (NeurIPS), 2020*" [158] and a submission "*Miao Zhang, Steven Su, Shirui Pan, Xiaojun Chang, Huiqi Li, Gholamreza Haffari, Differentiable Neural Architecture Search in Equivalent Space with Enhancing Exploration and Relieving Forgetting. Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2021*", where the second one is the extended journal version of the previous one. Miao Zhang conceived the original idea of using a variational graph autoencoder that injectively transforms the discrete architecture space into an equivalent continuous latent space. The E^{2}NAS algorithm was originally devised by Miao Zhang. Steven Su and Shirui Pan helped Miao Zhang to verify all derivations of theoretical parts in this paper. Miao Zhang conducted all experiments, with the help from Shirui Pan. The first version of the paper was written by Miao Zhang with some help from Steven Su. Huiqi Li and Shirui Pan revised the paper many times. The authors Shirui Pan, Xiaojun Chang, Huiqi Li, Gholamreza Haffari, and Zongyuan Ge provided feedback during the writing of the paper.

4.2 Problem Definition and Preliminaries

4.2.1 Weight-Sharing NAS

Weight-sharing NAS [114], also known as one-shot NAS, encodes a search space $\mathscr{A}$ as a supernet $\mathscr{W}_{\mathscr{A}}$. The supernet subsumes all candidate operations and means that all architectures α can directly inherit weights from the supernet for evaluation rather than training them from scratch. The supernet is only trained once during the architecture search, which substantially reduces the search time. The most promising architecture α^* is determined by validating its performance with weights inherited from the supernet:

$$\begin{aligned} &\min_{\alpha\in\mathscr{A}} && \mathscr{L}_{\texttt{val}}(\mathscr{W}^*_{\mathscr{A}}(\alpha)) \\ &\texttt{s.t.} && \mathscr{W}^*_{\mathscr{A}}(\alpha) = \texttt{argmin}\, \mathscr{L}_{\texttt{train}}(\mathscr{W}_{\mathscr{A}}(\alpha)). \end{aligned} \tag{4.1}$$

Eq. (4.1) is a formula for a challenging bilevel optimization. The discrete characteristic of α makes it impossible to use a gradient-based method to alternatively optimize $\mathscr{W}_{\mathscr{A}}$ and α.

Hence, an LSTM controller is used to sample architectures for training the supernet training and validating performance. In [55] the authors adopt a uniform strategy for sampling the architectures, and the best-performing architecture is selected via an evolutionary algorithm. The same overall approach applies in [86] but with a random sampling strategy and a random search.

4.2.1.1 Catastrophic Forgetting in Weight-Sharing NAS

Catastrophic Forgetting [74, 89, 134] usually occurs when sequentially training a model for several tasks. Given a neural network with optimal parameters ω^* on task T_1, its performance on T_1 declines dramatically after the model has been trained on task T_2 because the network weights have been changed to optimize the objectives of T_2.

Catastrophic forgetting with weight-sharing NAS has been observed in quite a few recent studies [11, 87, 124, 160], where performance degrades as the supernet learns new architectures to replace old ones. Sciuto *et al.* [124] make two notable observations in this regard. First, that the models with more shared weights do worse, and, second, that there is no possible positive relationship between the distribution of model weights trained from scratch p_{along} and those inherited from the supernet p_{share}. Benyahia *et al.* [11] describe this problem as *multi-model forgetting*, defining it as when learning the current model deteriorates the performance of previously learned models after several models with shared parameters have been applied to a single dataset $\mathscr{D}$.

Li *et al.* [87] demonstrate that KL-divergence between the true parameter posterior p_{along} and proxy posterior p_{share} also increases during supernet training, making the weight sharing strategy unreliable. They calculate the KL-divergence between the two distributions from a Bayesian standpoint as follows:

$$\begin{aligned}
&D_{\mathscr{KL}}(p_{along}(\mathscr{W} \mid \alpha_k, D) \parallel p_{share}(\mathscr{W} \mid \alpha_k, D)) \\
&\propto \int p_{along}(\mathscr{W} \mid \alpha_k, D) \mathbf{log} \frac{p_{along}(\mathscr{W} \mid \alpha_k, D)}{p_{share}(\mathscr{W} \mid \alpha_k, D)} \mathrm{d}\mathscr{W} \\
&= \int p_{along}(\mathscr{W} \mid \alpha_k, D) \mathbf{log} \frac{p_{along}(\mathscr{W} \mid \alpha_k, D)}{\prod_s p_{share}(\mathscr{W} \mid \alpha_s, D)} \mathrm{d}\mathscr{W} \\
&= -\sum_{k \neq s} \int p_{along}(\mathscr{W} \mid \alpha_k, D)\ \mathbf{log}\ p_{along}(\mathscr{W} \mid \alpha_s, D) \mathrm{d}\mathscr{W}.
\end{aligned} \tag{4.2}$$

Upon the analysis of Eq. (4.2), we find that the KL-divergence is correlated to the sum of the cross-entropy of $p_{along}(\mathscr{W} \mid \alpha_k, D)$ and $p_{along}(\mathscr{W} \mid \alpha_s, D)$, where $k \neq s$ and α_s and α_k partially-shared weights. Further, the KL-divergence increases with the number of architectures containing partially-shared weights. Eq. (4.2) demonstrates that training the supernet increases the distance between the weight distribution when the architecture is trained from scratch and the weight inherited from the supernet. As a result, the supernet becomes unreliable.

This catastrophic forgetting clearly reduces both the predictive ability of the supernet and the overall efficiency of weight-sharing NAS strategies. Three recent studies have put forward ideas to overcome this catastrophic forgetting problem [11, 87]. Li *et al.* [87] propose reducing the KL-divergence by limiting the number of candidate models during each step of the architecture search. They adopt a progressive strategy that shrinks the search space and only looks for part of a model in each step. Zhang *et al.* [160] use the replay-buffer paradigm to overcome forgetting with one-shot NAS. To regularize the supernet training, only the most representative architectures are selected. Benyahia *et al.*'s [11] idea is to use the Weight Plasticity Loss (WPL) loss function to maximize the joint posterior probability and regularize each network parameter based on its importance. WPL can be seen as an online variant of Elastic Weight Consolidation (EWC) [74] in that it only counts the shared parameters between the current architecture and one previous architecture.

4.2.2 Differentiable NAS

Differentiable NAS is built on weight-sharing NAS. Weight-sharing NAS uses a controller to sample discrete architectures from the search space for supernet training. The most promising architecture α^* is obtained from a trained supernet using a heuristic search method. Differentiable NAS further relaxes the discrete architectures into a continuous space $\mathscr{A}_\theta$, and alternatively learns the architecture parameters and supernet weights based on gradient methods [3, 20, 95, 109, 140]. The best architecture in the continuous representation $\mathscr{A}_\theta$ is found via

$$\min_{\alpha_\theta \in \mathscr{A}_\theta} \mathscr{L}_{\texttt{val}}(\operatorname*{argmin}_{\alpha_\theta, \mathscr{W}_\mathscr{A}} \mathscr{L}_{\texttt{train}}(\mathscr{A}_\theta, \mathscr{W}_\mathscr{A})), \tag{4.3}$$

and usually solved through bi-level optimization.

Most state-of-the-art differentiable NAS [20, 95, 140] methods apply a **softmax** function to calculate the magnitude of each operation and relax the discrete architectures into a continuous representation. A discrete architecture is obtained by applying an **argmax** function to the magnitude matrix after the supernet has been trained. Once the discrete architectures have been transformed into a continuous space, continuous optimization is performed to update the continuous architecture representation α_θ along the gradient of validation performance [20, 95, 99]:

$$\alpha_\theta^{i+1} \leftarrow \alpha_\theta^i - \gamma \nabla_{\alpha_\theta} \mathscr{L}_{\texttt{val}}(\alpha_\theta, \mathscr{W}^*), \tag{4.4}$$

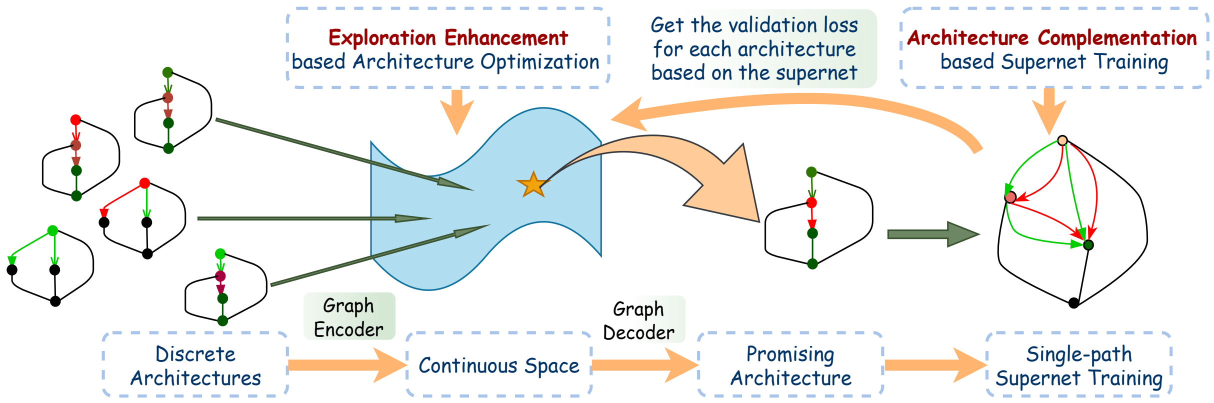


Figure 4.1: The framework of the proposed E^{2}NAS.

where γ is the learning rate, and $\mathscr{W}^*$ is approximated by adapting $\mathscr{W}$ using only a single training step with descending $\nabla_\omega \mathscr{L}_{\mathtt{train}}(\mathscr{W}_{\mathscr{A}}(\alpha_\theta^i))$ [40, 86].

This differentiable method improves the efficiency of weight sharing NAS as it obtains the most promising architecture once the supernet is trained. However, as α_θ reflects the mixing weights of all operations in the supernet, each step of the supernet training in differentiable NAS trains the whole supernet rather than a single path, as is the case with one-shot NAS. As a result, the memory requirements are much higher. Further, as architectures with better performance in the early stages will be trained more frequently, differentiable methods frequently contend with the *rich-get-richer problem* [2, 162] and fall into a local optimum.

4.2.2.1 Incongruence in Differentiable NAS

Another shortcoming with differentiable NAS is that there is no theoretical foundation to show that the optimization in the continuous latent space is equivalent to one done on the discrete space. Chen *et al.* [23] point out that, in most differentiable NAS, incongruence relates to the Hessian norm of the architecture parameters, which constantly increases during supernet training [152]. Taking the most common **softmax-argmax** transformation in DARTS as an example, the architecture parameters α_θ are continuous during the architecture search. And the searched α is obtained through **argmax** argmax after the architecture search, which only contains $\{0,1\}$ values. Obviously, there can be no one-to-one mapping in such a transformation. More importantly, all differentiable NASs assume the validation performance of α_θ^*, $\mathscr{L}_{val}(\alpha_\theta^*, \mathscr{W}*)$ equates, or at least is very informative, to the validation performance of α, $\mathscr{L}_{val}(\alpha^*, \mathscr{W}*)$. However, the incongruence between them may not be negligible.

Our strategy assumes that α_θ^* is in the local optimal, based on the Taylor expansion, we have:

$$\begin{aligned}\mathscr{L}_{val}(\alpha^*, \mathscr{W}*) &\approx \mathscr{L}_{val}(\alpha^*_\theta, \mathscr{W}*) + \nabla_{\alpha_\theta}\mathscr{L}_{val}(\alpha^*_\theta, \mathscr{W}*)(\alpha^* - \alpha^*_\theta) \\ &+ \nabla^2_{\alpha_\theta}\mathscr{L}_{val}(\alpha^*_\theta, \mathscr{W}*)(\alpha^* - \alpha^*_\theta)^2 \\ &= \mathscr{L}_{val}(\alpha^*_\theta, \mathscr{W}*) + (\alpha^* - \alpha^*_\theta)^T\mathscr{H}(\alpha^* - \alpha^*_\theta),\end{aligned} \tag{4.5}$$

where $\nabla_{\alpha_\theta}\mathscr{L}_{val} = 0$ due to the local optimality condition. $\mathscr{H}$ is the Hessian matrix of $\mathscr{L}_{val}(\mathscr{W}^*, \alpha_\theta)$.

We find that incongruence in the performance of the final continuous architecture and the final discrete architecture correlates with the norm of the Hessian matrix of $\mathscr{L}_{val}(\alpha_\theta, \mathscr{W}*)$. Several recent studies on differentiable NAS report that this Hessian norm constantly increases during the architecture search [23, 152]. As a remedy, Zela *et al.* [152] propose an early stopping criterion based on the Hessian. However, based on our observation above, we conclude that the common **softmax-argmax** transformation will not guarantee congruence with differentiable NAS.

4.2.2.2 Single-path Differentiable NAS

Although differentiable NAS requires significantly less computational overhead through weight-sharing and continuous relaxation, it is still a memory hog as the whole supernet must be trained in each step. DARTS, however, gave rise to a number of studies with propositions to further tackle the efficiency issue [20, 40, 140, 149]. Rather than training the whole supernet in each step of supernet training, DARTS-based methods only sample a few paths from the continuous representation with which to train the supernet in each step. For instance, ProxylessNAS [20] transforms the real-valued architecture parameters into binary representations through binary gates. It only activates two paths in each training step.

Several approaches [40, 140, 149] further apply a discrete constraint on the Gumbel-softmax reparametrization [65, 103] to sample a single path for each training step, which reduces the memory requirement to the same level as a single-path one-shot NAS [55]. The validation performance $\mathscr{L}_{val}$ is then calculated based on the discrete architectures, which means incongruence is not an issue. We call this paradigm "single-path differentiable NAS".

One critical component of single-path differentiable NAS is the transformation between the discrete and continuous search spaces. GDAS [40] uses the same continuous relaxation as DARTS but introduces a gradient-based sampler to sample the single path for each training step. NAO [99] uses an LSTM-based autoencoder to transform the discrete architectures into a continuous space. However, there is no injective constraint in these transformations to theoretically guarantee that the optimization in the continuous latent space will be equivalent to one performed on a discrete space. In contrast to the continuous relaxation and LSTM-based autoencoder methods, recent works on graph neural network theoretically show that a variational graph autoencoder is able to injectively transform directed acyclic graphs [141, 157], such as used to represent architectures in a continuous representation in NAS through an asynchronous message passing scheme. With a solid theoretical foundation and a guarantee of equivalence between an optimization in the continuous latent space versus a discrete space, this was an avenue we felt must be pursued.

Hence, in this chapter, we present a single-path differentiable NAS framework called E^{2}NAS that incorporates a variational graph autoencoder for the transformation between the discrete and continuous search spaces. Fig.4.1 sketches the framework. The discrete architectures are mapped to continuous representations α_θ with the trained graph encoder. The architecture search is conducted in the continuous space, with the proposed exploration enhancement based architecture optimization. The optimized α'_θ is fed into the trained graph decoder to get the discrete architecture α', and the supernet is trained accordingly.

As shown, the variational graph autoencoder first transforms the discrete architecture search space into one injective, continuous space where the architecture optimization is conducted. After continuous architecture representation α_θ updates based on the gradient, a decoder derives the discrete architecture and trains a single path of the supernet. A detailed description of E^{2}NAS follows in the next section.

4.3 Exploration Enhancing Neural Architecture Search with Architecture Complementation

As shown in Fig. 4.1, our approach consists of two key components: an exploration enhancement module that overcomes the *rich-get-richer* problem after transforming the discrete architectures into a differentiable and injective space by a graph neural network; and an architecture complementation loss function that reduces the risk of *catastrophic forgetting*. More details follow.

4.3.1 Exploration Enhancement in the Latent Space

4.3.1.1 Differentialable latent space transformation

As previously mentioned, the existing differentiable NAS methods usually adopt a simple continuous relaxation [95] to transform the discrete neural architectures into one continuous search space. Further, the architectures are usually represented as directed acyclic graphs. However, because this approach does not guarantee an injective transformation [141, 157], the continuous results are often incongruent to the original discrete samples. To overcome this problem, E^{2}NAS incorporates an asynchronous message passing scheme, based on a graph neural network (GNN), that encodes the neural architecture into an injective space. However, instead of encoding each discrete architecture into many GNNs, the final output of each architecture, i.e., the computation C, is what is converted into the continuous representation, denoted as $\mathbf{z}$. A function $\mathcal{U}$ is used to update the hidden state of each node based on the hidden states of its neighbors and the type of vertex it is, as calculated by $\mathbf{h}_v = \mathcal{U}(\mathbf{x}_v, \mathbf{h}_v^{\text{in}})$. Here, $\mathbf{h}_v^{\text{in}}$ is obtained by aggregating all the predecessors via $\mathbf{h}_v^{\text{in}} = \mathcal{G}(\{\mathbf{h}_u : u \to v\})$, where $\mathcal{G}$ is an aggregation function. According to the theory of graph neural networks [141, 157], if the aggregation function $\mathcal{G}$ is invariant to the order of input, then the computation encoder will be permutation-invariant. Further, we know that the graph encoder will map C to $\mathbf{z}$ injectively if both the aggregation function $\mathcal{G}$ and the update function $\mathcal{U}$ in the graph encoder are injective.

This injectiveness ensures a one-to-one mapping from a latent representation to an architecture, and vice versa. Therefore, any differentiable optimization on the continuous space will be equivalent to conducting an optimization over the discrete space. Our graph decoder first obtains the initial hidden state $\mathbf{h_0}$ by applying an MLP to the latent vector $\mathbf{z}$. $\mathbf{h_0}$ is then fed into GRU$_d$ to construct a directed acyclic graph node-by-node based on the current state of the graph. Details of the procedure follow.

Following Zhang [157], we also use a gated sum as the aggregate function and a gated recurrent unit (GRU) as the update function.

$$\begin{aligned} \mathbf{h}_v^{\text{in}} &= \sum_{u \to v} g(\text{Concat}(\mathbf{h}_u, \mathbf{x}_{\text{uid}})) \odot m(\text{Concat}(\mathbf{h}_u, \mathbf{x}_{\text{uid}})) \\ \mathbf{h}_v &= \text{GRU}_e(\mathbf{x}_v, \mathbf{h}_v^{\text{in}}). \end{aligned} \tag{4.6}$$

This encoding method is similar to a traditional RNN for sequences. Exactly the same procedure applies after the optimization in the latent continuous space to decode the latent vector $\mathbf{z}$ into a neural architecture. The procedure of the graph autoencoder is generally described as following:

1. Compute v_i's type distribution using an MLP f_{add_vetex} based on current graph state $\mathbf{h}_{\text{G}} := \mathbf{h}_{\text{v}_{i-1}}$
2. Sample v_i's type.
3. Update v_i's hidden state by $\mathbf{h}_{v_i} = \text{GRU}_d(\mathbf{x}_{v_i}, \mathbf{h}_{v_i}^{\text{in}})$
4. For $j = i-1, i-2, ..., 1$: (a) compute the edge probability of (v_j, v_i);(b) sample the edge; (c) if a new edge is added, update $\mathbf{h}_{v_i}$ using previous step.

Zhang *et al.* [157] iteratively apply the above steps to generate new nodes until Step 2) samples the ending type. However, if we want to sample the same number of nodes for each neural network, we could sample the same number of nodes before the end of the iteration, and all nodes without an output edge would be connected by the extra output node.

4.3.1.2 Exploration Enhancement

Contemporary differentiable NAS methods all conduct continuous optimization following Eq.(4.4) to update the continuous architecture representation α_θ, which means that the exploration path only follows the validation performance gradient. This approach easily falls prey to the *rich-get-richer* problem. Hence, with E^{2}NAS, exploration follows the novel gradient in Eq.(4.7):

$$\alpha_\theta^{i+1} \leftarrow \alpha_\theta^i - (1-\gamma)\nabla_{\alpha_\theta^i}\mathscr{L}_{\texttt{val}}(\alpha_\theta^i, \mathscr{W}^*) - \gamma\nabla_{\alpha_\theta^i}\mathscr{L}_N(\alpha_\theta^i, A), \tag{4.7}$$

where $\mathscr{L}_N(\alpha_\theta^i, A)$ measures the novelty of architecture α_i against the archive A that contains the N previously visited architectures. This approach to exploration ensures the search does not fall into a local optimum. After an architecture update, the continuous representation α_θ^{i+1} of promising architectures is fed into the graph decoder, which outputs the discrete architectures α_{i+1}. The weights in the supernet $\omega = \mathscr{W}_{\mathscr{A}}(\alpha_{i+1})$ are updated by descending $\nabla_\omega\mathscr{L}_{\texttt{train}}(\mathscr{W}_{\mathscr{A}}(\alpha_{i+1}))$. While it is intractable to measure the novelty of the architectures in the discrete space, the probability density function of α_θ^i can be calculated. This is done by drawing α_θ^i from the distribution formulated by continuous architectures α_θ in the archive A, also sometimes described to as probabilistic novelty detection in latent space [115].

The trained graph encoder E is a mapping $E : \mathbb{R}^m \to \mathbb{R}^n,\ m > n$, and the decoder D is a mapping $D : \mathbb{R}^n \to \mathbb{R}^m$ that defines a parameterized manifold of dimension n, $\mathscr{M} \equiv D(\mathbb{R}^n)$. As such, it is possible to sample every architecture α_i with noise ξ_i through $\alpha_i = D(\alpha_\theta^i) + \xi_i$, where $\alpha_\theta^i \in \mathbb{R}^n$. Assuming the decoding function D is smooth enough [117, 168], a first-order Taylor expansion used at a given point $\alpha_i \in \mathbb{R}^m$, would give

$$D(\alpha_\theta) = D(\alpha_\theta^i) + J_D(\alpha_\theta^i)(\alpha_\theta - \alpha_\theta^i) + O(\|\alpha_\theta - \alpha_\theta^i\|^2), \tag{4.8}$$

where $J_D(\alpha_\theta^i) \in \mathbb{R}^{m\times n}$ is the Jacobi matrix of D at α_θ^i.

The tangent space of D at α_θ^i can be represented $\mathscr{T}_{\alpha_\theta^i} = \texttt{span}(J_D(\alpha_\theta^i))$. Let $J_D(\alpha_\theta^i) = U^\parallel S V^\top$ be the singular value decomposition (SVD) of the Jacobi matrix at α_θ^i, and we have $\mathscr{T}_{\alpha_\theta^i} = \texttt{span}(J_D(\alpha_\theta^i)) = \texttt{span}(U^\parallel)$ [115, 168]. After defining $U^\perp$ as the orthogonal complement of $U^\parallel$, where $U = [U^\parallel U^\perp]$ is a unitary matrix, wthe data point α_i can be represented with rotated coordinates:

$$w = U^\top \cdot \alpha_i = \begin{bmatrix} U^{\parallel\top} \cdot \alpha_i \\ U^{\perp\top} \cdot \alpha_i \end{bmatrix} = \begin{bmatrix} w^\parallel \\ w^\perp \end{bmatrix}, \tag{4.9}$$

where the component $w^\parallel$ is parallel to $\mathscr{T}$, and $w^\perp$ is orthogonal to $\mathscr{T}$ as the noise ξ.

Lemma 2. *Suppose we have a decoder D with its tangent space represented as $\mathscr{T}$. Given a random variable A formed by a set of architectures, the random variable W is obtained from A rotating the coordinates $W = U^\top \cdot A$. The rotation contains two parts: $W^\parallel$, which is parallel to $\mathscr{T}$, and $W^\perp$, which is orthogonal to $\mathscr{T}$. Defining $p_A(\alpha_i)$ as the probability density function for describing α_i, drawn from A, we have $p_A(\alpha_i) = p_W(w)$, and*

$$\begin{aligned} p_A(\alpha_i) &= p_W(w) = p_W(w^\parallel, w^\perp) = p_{W^\parallel}(w^\parallel) p_{W^\perp}(w^\perp) \\ &\approx \left|\mathtt{det} S^{-1}\right| p_{A_\theta}(\alpha_\theta) \cdot \frac{\Gamma(\frac{m-n}{2})}{2\pi^{\frac{m-n-1}{2}} \left\|\bar{w}^\perp\right\|^{m-n-1}} p_{\|W^\perp\|}(\|\bar{w}^\perp\|). \end{aligned} \tag{4.10}$$

Based on Lemma 2, the novelty of a newly sampled architecture α_j from A can be calculated by measuring the probability that the new sampled architecture will be located in the distribution formed by the archive A: $N(\alpha_j) = -\mathbf{log}(p_A(\alpha_j))$ using a density function. To encourage exploration during the architecture optimization, the exploration enhancement loss term is defined as:

$$\begin{aligned} \mathscr{L}_N(\alpha_\theta, A)) &= \mathbf{log}(p_A(\bar{\alpha})) = \mathbf{log}(\left|\mathtt{det} S^{-1}\right| p_{A_\theta}(\alpha_\theta) \\ &\cdot \frac{2\pi^{\frac{m-n-1}{2}}}{\Gamma(\frac{m-n}{2})} \left\|\bar{w}^\perp\right\|^{m-n-1} p_{\|W^\perp\|}(\|\bar{w}^\perp\|)). \end{aligned} \tag{4.11}$$

With this approach, the enhancement module encourages looking for a novel architecture instead of always sampling a well-trained architecture from a previous round, reducing the chances of falling into a local optimum.

4.3.2 Overcoming Multi-Model Forgetting through Architecture Complementation

As described in Section 4.2, differentiable NAS is built upon weight-sharing NAS, where numerous architectures with partially shared weights are trained on a single dataset. Without loss of generality, this study also considers a typical scenario in which only one architecture (a single path) in the supernet is trained in each step of the architecture search.

Hence, each step of the supernet training is defined as a task with $\mathtt{argmin}\mathscr{L}_{\mathtt{train}}(\alpha_\theta^i, \mathscr{W}_\mathscr{A}) = \mathtt{argmin}\mathscr{L}_{\mathtt{train}}(\mathscr{W}_\mathscr{A}(\alpha_i))$, and the training consists of multiple sequential tasks in a lifelong

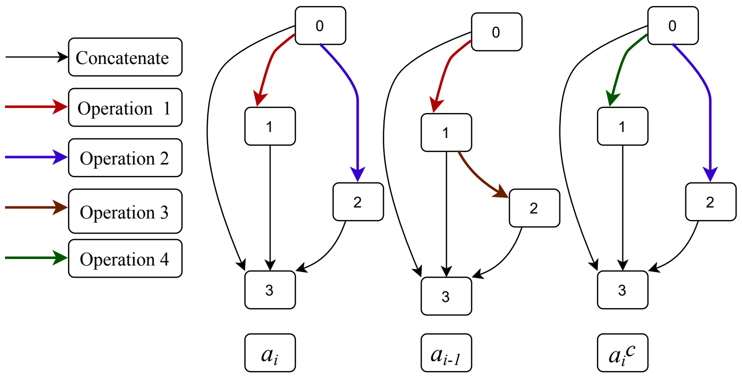


Figure 4.2: Example of obtaining α_i^c through our architecture complementation.

learning setting [29, 120] or an online multi-task learning setting [37]. In either scenario, catastrophic forgetting is an inevitable problem. With differentiable NAS, the supernet is not able to accumulate the newly learned knowledge in a manner consistent with the past experience. Hence, it usually forgets past learned tasks when trained on a new task. As mentioned, this phenomenon is sometimes called *multi-model forgetting* in [11].

A mainstream approach to mitigating this problem is to select and replay several representative tasks from a recent buffer or to apply soft regularization [64]. The tasks selected for replay should not be limited to the most recently experienced tasks. Rather, it is better to maximize the diversity of tasks in the replay buffer [5, 160], to balance the stability and plasticity [112]. Our framework is designed to select two architectures: the most recent architecture α_{i-1} in the replay buffer and another *complementary architecture* α_i^c that is orthogonal to α_{i-1}. Selecting two architectures reduces computational overhead while ensuring some diversity.

Fig. 4.2 gives an example of our architecture complementation. We consider a cell structure, where node 0 is the input node, node 1 and 2 are operation nodes, and node 3 is the output node which concatenates the outputs of all input and operation nodes as the output of the cell. When the input of the operation node of α_i is the same as α_{i-1} (take node 1 as an example), α_i^c randomly select a different operation, but when the input of operation node of α_i is different from α_{i-1} (take node 2 as an example), α_i^c select the same operation as α_i. The way of selecting the complementary architecture ensures the two following conditions hold true: $\omega_i \cap \{\omega_{i-1} \cup \omega_i^c\} = \omega_i$, and $\omega_{i-1} \cap \omega_i^c = \emptyset$, where $\omega_i = \mathscr{W}_{\mathscr{A}}(\alpha_i)$ and $\omega_i^c = \mathscr{W}_{\mathscr{A}}(\alpha_i^c)$.

When the two architectures in the replay buffer are converted into a soft regularization, the loss function for the supernet training in step i becomes:

$$\mathscr{L}_c(\omega_i) = (1-\varepsilon)\mathscr{L}_2(\omega_i) + \varepsilon(\mathscr{L}_2(\omega_i^c) + \mathscr{L}_2(\omega_{i-1})) + \eta\mathscr{R}(\omega_i), \tag{4.12}$$

where $\mathscr{L}_2(\omega_i)$ is the cross-entropy loss for the architectures α_i in the training set, and $\mathscr{R}$ is the

Algorithm 4 E^{2}NAS

Input: Trained encoder E and decoder D, training dataset $\mathscr{D}_{train}$ and validation dataset $\mathscr{D}_{valid}$. Initial architecture archive $A = \emptyset$. Randomly initialize architecture parameter α_θ and supernet weights $\mathscr{W}_{\mathscr{A}}(\alpha)$;

```
1: while not done do
2:     Sample batch of $\mathscr{D}_{train}$;
3:     decode $\alpha_\theta$ to get $\alpha$ based on $D$,
4:     get the complementary architecture $\alpha^c$;
5:     update the supernet weights $\mathscr{W}_{\mathscr{A}}(\alpha)$ based on Eq. (4.12), and add architecture $\alpha$ into $A$;
6:     sample batch of $\mathscr{D}_{valid}$, and update $\alpha_\theta$ based on Eq. (4.7);
7: end while
8: Decode $\alpha_\theta$ to obtain the best $\alpha^*$ based on the trained decoder $D$.
9: Retrain $\alpha^*$ and get the best performance on the test dataset $\mathscr{D}_{test}$.
```

Return: architecture α^* with the best performance.

ℓ_2 regularization term. ε is the trade-off that controls whether to push the weights of current architecture to optimal during training or whether to prevent the performance of the other architectures in the supernet from deteriorating.

The proposed complementation loss function in Eq.(4.12) is related to the WPL loss function [11] for overcoming catastrophic forgetting in one-shot NAS, where WPL tries to maximize the joint posterior probability $p(\omega_{i-1},\omega_i \mid \mathscr{D})$ in each step of supernet training. However, unlike WPL, which only considers one previous architecture α_{i-1} during the supernet training, Eq.(4.12) considers one additional complementary architecture α_i^c that is orthogonal to α_{i-1} during the supernet training. Given two posterior probability $p_1 = p(\omega_{i-1},\omega_i \mid \mathscr{D})$ and $p_2 = p(\omega_i^c,\omega_i \mid \mathscr{D})$ for a dataset $\mathscr{D}$, we have the following Lemma.

Lemma 3. *Given the previous architecture α_{i-1} with parameters ω_{i-1}, the current architecture α_i with the parameters ω_i, and a complementary architecture α_i^c with the parameters ω_i^c, the proposed architecture complementation loss function in Eq.*(4.12) *equates to maximizing $p_1 * p_2$ in each step of the supernet training in a one-shot NAS setting.*

We can observe from Lemma 3 that the proposed loss function $\mathscr{L}_c$ is identical to the WPL loss function when the additional complementary architecture is considered. However, calculating our loss function is more efficient because there is no need to estimate the Fisher information matrix or keep the previous models in optimal points.

4.3.3 Regularization based Differentiable NAS

As noted in several recent studies [15, 87, 90, 127, 152], differentiable NAS tends to choose shallow architectures from a DARTS search space. The reason is that shallow architectures have both a smaller *gradient confusion* [122], and more smooth landscapes [171], which means they can be trained faster. However, some researchers have pointed out that the deeper architectures usually provide superior performance with the larger datasets [25, 109]. This motivated us to wonder whether encouraging differentiable NAS to search for "deeper" architectures might improve transferability.

To test the conjecture, we added a depth regularization term into the architecture optimization loss function in Eq.(4.7) when exploring a DARTS search space. The depth of the architecture is defined as:

$$l_0^{depth}(\alpha) = \sum_{j=1}^{|C|} d(c_j), \quad d(c_j) \in \{0,1\}, \tag{4.13}$$

where $d(c_j) = 1$ when c_j connects node i and $i-1$. Following the constraint-aware differentiable NAS in [85, 143], the depth regularization term when considering the decoder as a depth predictor is defined as:

$$\mathscr{L}^{depth}(\alpha_\theta) = -\mathbb{E}\left[l_0^{depth}(\alpha), \alpha \sim D(\tilde{\alpha}_\theta)\right], \tag{4.14}$$

where D is the decoder in our E^{2}NAS, $\tilde{\alpha}_\theta = \alpha_\theta + \delta$, and δ is a noisy term. Using a batch size of $M = 10$ to sample 10 architectures, we calculated the expectation in Eq.(4.14), resulting in $\mathscr{L}^{depth}(\alpha_\theta) \approx \frac{1}{M}\sum_{m=1}^{M} l_0^{depth}(\alpha),\ \alpha \sim D(\tilde{\alpha}_\theta)$.

This variant of E^{2}NAS for use with DARTS search spaces is denoted as **E^{2}NAS-R**. Its loss function defined as:

$$\mathscr{L}_r = (1-\gamma)\mathscr{L}_{\texttt{val}}(\alpha_\theta, \mathscr{W}^*) + \gamma \mathscr{L}_N(\alpha_\theta, A) + \zeta \mathscr{L}^{depth}(\alpha_\theta), \tag{4.15}$$

where γ and ζ are trade-offs.

In both **E^{2}NAS** and **E^{2}NAS-R**, the encoder E and decoder D are both trained offline to improve efficiency. Following most one-shot NAS methods, only a single-path architecture is trained during each step of the supernet training, and bilevel optimization is used to alternatively learn the architecture parameters and the supernet weights [40, 86]. A simple implementation is given in Algorithm 4.

Table 4.1: Comparison results with state-of-the-art NAS approaches on NAS-Bench-201.

Method	CIFAR-10		CIFAR-100		ImageNet-16-120	
	Valid(%)	Test(%)	Valid(%)	Test(%)	Valid(%)	Test(%)
ENAS [114]	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
RandomNAS* [86]	85.63±0.44	88.58±0.21	60.99±2.79	61.45±2.24	31.63±2.15	31.37±2.51
DARTS (1st) [95]	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 (2nd) [95]	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
SETN [42]	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
NAO* [99]	82.04±0.21	85.74±0.31	56.36±3.14	59.64±2.24	30.14±2.02	31.35±2.21
GDAS* [40]	90.03±0.13	93.37±0.42	70.79±0.83	70.35±0.80	40.90±0.33	41.11±0.13
E^{2}NAS	**90.94±0.83**	**93.89±0.47**	**71.83±1.84**	**72.05±1.58**	**45.44±1.24**	**45.77±1.00**

The hyperparameters of E^{2}NAS are set as ε=0.5 and γ=$\texttt{Sig}_\gamma$(10) in this experiment. The best single run of our E^{2}NAS (with *random seed 0*) achieves 94.22%, 73.13%, and 46.48% test accuracy on CIFAR-10, CIFAR-100, and ImageNet, and the optimal performance on these datasets are 94.37%, 73.51%, and 47.31%, respectively. "*" indicates the results reproduced with the same random seeds with our E^{2}NAS.

4.4 Experimental Result

We assessed our framework in two different search settings. First, we evaluated E^{2}NAS with the benchmark dataset, NAS-Bench-201 [41]. We then tested both E^{2}NAS and E^{2}NAS-R with the common DARTS convolutional search space [95].

4.4.1 Experiments on the Benchmark Dataset

The high computational cost of evaluating architectures is a major obstacle when analyzing and reproducing one-shot NAS methods. For this reason, it can be hard to reproduce results with the current NAS methods under the same experimental settings for a fair comparison. To alleviate this problem, several benchmark datasets have been published in recent studies [41, 150, 154]. The NAS-Bench-201 dataset [41], selected for our experiments, is one such dataset. The search space in NAS-Bench-201 contains four nodes with five associated operations, resulting in 15,625 cell candidates. The search space in NAS-Bench-201 is much simpler than what would commonly be encountered in a NAS search space, but the dataset contains the ground-truth test accuracy for all candidates, which greatly reduces the computational requirements normally associated with a one-shot NAS method. Moreover, using this dataset provides results that can be reproduced relatively easily. It is worth noting that the unified cell structure in the NAS-Bench-201 is densely connected and does not include depth information. Hence, we did not incorporate depth regularization when testing this search space.

4.4.1.1 Reproducible Comparison with Baselines

The results for E^{2}NAS and the NAS comparison baselines on the NAS-Bench-201 set are provided in Table 4.1. This set of experiments involved independent searches with different random seed configurations. E^{2}NAS produced state-of-the-art results on all three datasets, significantly outperforming the other baselines, especially with CIFAR-100 and ImageNet. E^{2}NAS performed only slightly better than GDAS on CIFAR-10. However, the NAS-Bench-201's optimal performance of 94.37% leaves little room for improvement. With random seed 0, E^{2}NAS returned a performance of 94.22% on CIFAR-10, 73.13% on CIFAR-100, and 46.48% on ImageNet, which is almost equal to the optimal test accuracies with the NAS-Bench-201 dataset.

Overall, these results demonstrate the effectiveness of the three innovations in E^{2}NAS, i.e., using an injective transformation to resolve incongruence; enhancing the exploration process to avoid a local optimum; and incorporating an additional complementary architecture to overcome catastrophic forgetting.

In the following, we further investigate the benefits of these three components with differentiable one-shot NAS.

4.4.1.2 Analysis of Continuous Transformation

In this subsection, we conduct comparison experiments to demonstrate the effectiveness of including injective transformation in the framework. We, therefore, set γ in Eq. 4.7 to 0 to remove the exploration enhancement and ε in Eq. 4.12 to 0 to eliminate the complementation, which prevents catastrophic forgetting. As described in Section 4.3.1, we first adopted a graph autoencoder to injectively transform the discrete architectures into an equivalently continuous latent space. We then conducted differentiable optimization over the architecture search space in a differentiable NAS setting. The assessment metrics were the test accuracies of: a) the architecture selected in the last iteration; and b) the best architecture over all iterations, denoted as Best (%).

The first block in Table 4.2 contains several differentiable NAS baselines, all of which use some variation of a continuous transformation method. DARTS, SETN and GDAS all adopt a common continuous relaxation, while NAO uses an LSTM-based autoencoder to transform the discrete architectures into one continuous space. Note that, unlike DARTS, GDAS incorporates uncertainty (exploration) into the architecture sampling via the Gumbel-Max trick. To remove this effect, we prepared a variant of GDAS, GDAS-A, that directly samples architectures through argmax. GDAS-A is a more appropriate baseline for E^{2}NAS ($\gamma = 0$ and $\varepsilon = 0$), because

Table 4.2: Analysis of E^{2}NAS with different γ on NAS-Bench-201.

Method		CIFAR-10 Test(%)	CIFAR-10 Best(%)	CIFAR-100 Test (%)	ImageNet-16-120 Test (%)
DARTS (1st)	[95]	54.30±0.00	54.30±0.00	15.61±0.00	16.32±0.00
DARTS (2st)	[95]	54.30±0.00	54.30±0.00	15.61±0.00	16.32±0.00
SETN	[42]	87.64±0.00	87.64±0.00	59.05±0.24	32.52±0.21
NAO*	[99]	85.74±0.31	86.39±1.31	59.64±2.24	31.35±2.21
GDAS-A		89.73±3.33	93.49±0.24	62.15±6.86	35.34±4.81
GDAS*	[40]	93.37±0.42	93.78±0.16	70.35±0.80	41.11±0.13
E^{2}NAS	0	92.75±0.56	93.79±0.17	68.75±0.35	43.19±2.10
	0.2	82.53±12.07	93.57±0.30	54.26±14.05	19.73±6.43
	0.5	93.34±0.30	93.85±0.09	70.41±0.76	44.43±0.90
	0.8	93.75±0.00	93.77±0.02	70.96±0.00	45.49±0.00
	$\texttt{Sig}_\gamma(1)$	93.82±0.15	93.87±0.07	70.52±1.01	46.10±0.52
	$\texttt{Sig}_\gamma(2)$	93.72±0.30	94.29±0.07	71.63±1.20	45.20±0.24
	$\texttt{Sig}_\gamma(5)$	93.78±0.16	94.19±0.19	70.55±0.44	44.97±0.72
	$\texttt{Sig}_\gamma(10)$	93.43±0.29	94.04±0.08	70.56±0.30	45.07±0.62

The first block shows results of several differentiable NAS baselines, and the second illustrates results of our E^{2}NAS with different γ.

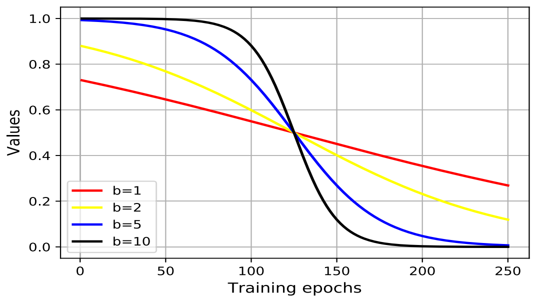


Figure 4.3: Sigmoid-type function for the hyperparameter γ with the training epochs based on Eq.(4.16).

the two are almost the same method. The only difference is that GDAS-A employs a common continuous relaxation while E^{2}NAS uses the proposed continuous transformation method.

From Table 4.2, we can see that E^{2}NAS ($\gamma = 0$ and $\varepsilon = 0$) still outperformed every baseline against both test metrics on the three datasets, even without enhanced exploration and complementation. These results demonstrate the effectiveness of the transformation method and the accuracy with which it injectively transforms discrete architectures into a continuous representation.

4.4.1.3 Analysis of Exploration Enhancement

In the following, we investigate the necessity of exploration with differentiable NAS and whether enhancing the exploration process affects performance. Theoretically, a larger γ should encourage more exploration to avoid a local optimum, while a smaller γ should guarantee better solutions with a higher validation performance. However, for this experiment, we devised a Sigmoid function, defined as $\gamma(t)$, to adapt γ during the supernet training so as to strike a balance between exploration and exploitation. This function should work to avoid a local optimal in the early stages of the search while guaranteeing better solutions with higher validation performance in the later stages. The Sigmoid function $\gamma(t)$ is defined as:

$$\gamma(\mathrm{t}) = 1 - \mathtt{Sigmoid}\big((\frac{\mathrm{t}}{\text{total epochs}} * 2 - 1) * b\big), \tag{4.16}$$

where the $\mathtt{Sigmoid}(x) = \frac{1}{1+e^{-x}}$, and $\mathtt{Sig}_\gamma(b)$ defines that the γ is scheduled with b based on Eq.(4.16).

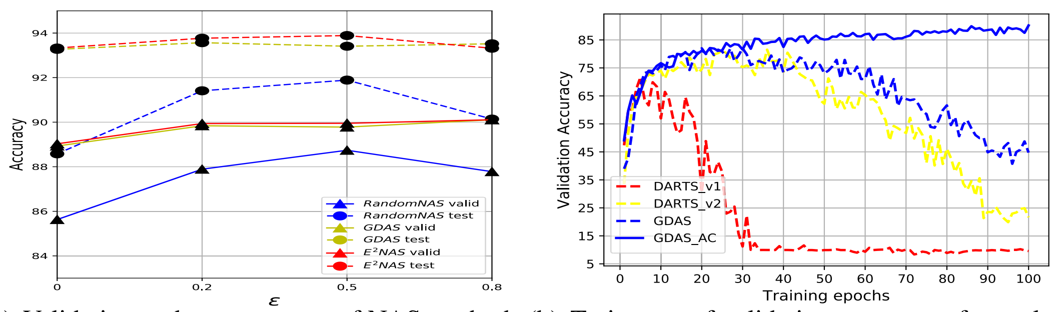
We used eight different settings for γ, including four static settings and four Sigmoid-type settings, as shown in the third block of Table 4.2. Fig.4.3 plots the Sigmoid function $\gamma(t)$ with different γ values.

The results demonstrate that enhancing exploration does indeed help to improve the performance of single-path differentiable NAS. Most settings of γ resulted in improved performance. It also appears that E^{2}NAS is very robust to γ, especially dynamic γ, where all $\mathtt{Sig}_\gamma$ yielded satisfactory results. $\mathtt{Sig}_\gamma(2)$ resulted in the best accuracy for E^{2}NAS at 94.29±0.07%, which represents a great improvement of E^{2}NAS without the exploration enhancement ($\gamma = 0$). GDAS also delivered good performance with this dataset, greatly outperforming its non-exploration enhanced variant GDAS-A. This result offers further support for the benefits of introducing exploration into the architecture search. However, E^{2}NAS with $\mathtt{Sig}_\gamma(10)$ still outperformed GDAS, showing impressive results.

4.4.1.4 Analysis of Architecture Complementation

Section 4.3.2 theoretically demonstrates the benefit of the proposed architecture complementation loss function. The experimental results in Section 4.4.1.1 also verify the effectiveness of our approach. As discussed in Section 4.3.2, ε is the crucial hyperparameter in our novel loss function $\mathscr{L}_c$ to ameliorate catastrophic forgetting with weight sharing NAS. We, therefore, conducted an extensive set of experiments to test its effects. The results are reported in this section.

The baselines compared include E^{2}NAS with a fixed γ=$\mathtt{Sig}_\gamma(10)$, plus two popular one-shot NAS methods: RandomNAS [86] and GDAS [40]. Fig. 4.4 (a) presents the results for four



(a) Validation and test accuracy of NAS methods with different ε on NAS-BENCH-201.

(b) Trajectory of validation accuracy of sampled architecture during supernet training on DARTS search space.

Figure 4.4: Analysis of architecture complementation on NAS-BENCH-201 dataset and DARTS search space [40, 95].

different values of ε. As shown, $\mathscr{L}_c$ significantly improves the search results, not only for E^{2}NAS but also for the other NAS baselines. Compared to a normal cross-entropy loss function (ε=0), $\mathscr{L}_c$ greatly improves the search results for RandomNAS, GDAS, and E^{2}NAS. From this experiment, we recommended a medium-valued ε (ε=0.2 or 0.5) for all three methods.

Fig. 4.4 (b) charts the validation accuracy of the sampled architectures with a common convolutional search space [40, 95]. In “GDAS-AC", we replace the normal loss function during the supernet training with the proposed $\mathscr{L}_c$ defined in Eq. (4.12). Here, all three baselines – DARTS_v1, DARTS_v2, and GDAS, suffered from catastrophic forgetting, but that $\mathscr{L}_c$ relieved the problem, as demonstrated by the curve of GDAS_AC in Fig. 4.4 (b). The performance of architectures by inheriting weights in this curve is getting better with the supernet training, making the assumption in bilevel optimization-based differentiable NAS hold true.

In addition to these tests, we also conducted an ablation study to investigate whether our architecture complementation (**AC**) method would perform better than other naive continual learning methods. For these experiments, we considered **OP(WPL)**, **RP**, and **NA** as baselines. The **OP(WPL)** method, taken from WPL [11], only considers the previous architecture α_{i-1}. **RP** randomly selects an architecture that is not α_{i-1} to place in the replay buffer. NA does not select an architecture, which is the equivalent of setting ε to 0 with a one-shot NAS method. Further, we applied the three different continual learning baselines for each of the methods: RandomNAS, GDAS and E^{2}NAS. Table 4.3 presents the results.

It is clear that overcoming catastrophic forgetting is a promising direction for one-shot NAS since the methods trained with naive continual learning methods showed significant improvement as a result of excellent test accuracies, especially RandomNAS and GDAS. We also observe that architecture selection in the quest to prevent forgetting has a substantial

Table 4.3: Analysis of one-shot NAS with different ε settings on CIFAR-10. We set a fixed γ=$\texttt{Sig}_\gamma(10)$ for our E^{2}NAS in this experiment.

METHOD	ε	AC Test Acc (%)	RP Test Acc(%)	OP Test Acc (%)
RANDOMNAS	0 (NA)	88.58±0.21	88.58±0.21	88.58±0.21
	0.2	**91.41±0.46**	89.72±3.79	90.81±3.37
	0.5	**91.89±0.58**	83.98±11.6	**91.90±0.35**
	0.8	**90.14±2.26**	**90.19±1.58**	88.44±4.24
GDAS	0 (NA)	93.37±0.42	93.37±0.42	93.37±0.42
	0.2	**93.67±0.00**	93.42±0.09	93.42±0.09
	0.5	93.41±0.31	93.42±0.09	**93.58±0.13**
	0.8	**93.52±0.22**	93.42±0.09	**93.52±0.22**
E^{2}NAS	0 (NA)	93.43±0.29	93.43±0.29	93.43±0.29
	0.2	**93.77±0.31**	51.49±58.68	92.50±2.04
	0.5	**93.89±0.47**	51.49±58.68	92.50±2.04
	0.8	93.32±0.48	51.49±58.68	92.03±1.37

influence over performance. Randomly adding just one additional architecture into the replay buffer deteriorated RP's performance to the extent that its results were worse than OP in most cases. By contrast, our architecture complementation method, which selects a complementary architecture, produced the best results in most cases. These results further show the benefits of our architecture complementation loss function.

4.4.1.5 Hyperparameter Study

This subsection presents the results of a hyperparameter study to simultaneously analyze the impact of ε (complementation to address forgetting) and γ (enhanced exploration to prevent convergence on a local optimum). The results with four static settings of ε while varying γ on CIFAR-10 are presented in Table 4.4. From the results, we see that our proposed $\mathscr{L}_c$ generally improves the performance of E^{2}NAS except at $\texttt{Sig}_\gamma(1)$. A mid-range value for ε (ε=0.2 or 0.5) performed best in most scenarios. As with ε, E^{2}NAS is also very robust to the hyperparameter for exploration enhancement, with most values of γ achieving similar results on CIFAR-10. However, unlike with the individual tests, E^{2}NAS was more sensitive to the hyperparameters when both were active in the framework. In general, a Sigmoid-type function with large b for γ combined with a mid-range ε helped E^{2}NAS to yield the best results.

Table 4.4: The CIFAR-10 test accuracy for our E^{2}NAS with different ε and γ settings.

ε \ γ	Sig$_\gamma$(1)	Sig$_\gamma$(2)	Sig$_\gamma$(5)	Sig$_\gamma$(10)
0	**93.82±0.15**	93.72±0.30	92.76±2.06	93.43±0.29
0.2	92.93±0.80	**93.99±0.00**	93.00±0.94	**93.77±0.31**
0.5	92.93±0.80	**93.77±0.31**	**93.87±0.29**	**93.89±0.47**
0.8	92.93±0.80	**93.99±0.00**	93.11±0.78	93.32±0.48

4.4.1.6 Running Time Analysis on NAS-Bench-201

Following most single-path one-shot NAS methods [40, 86], we only trained one single-path architecture during each step of supernet training, using bilevel optimization to alternatively learn the architecture parameters and the supernet weights. Also, the encoder E and decoder D for continuous relaxation in E^{2}NAS are both trained offline to reduce overheads. Algorithm 4 presents a simple implementation of E^{2}NAS. Compared to the GDAS [40] and RandomNAS baselines [86], E^{2}NAS includes an additional architectures in each step of the search. The result is increased diversity in exchange for only a limited increase in the search cost.

E^{2}NAS also needs to calculate the probabilistic novelty of each architecture during the architecture search, which further increases the running time. Table 4.5 illustrates the comparison results of searching time with several differentiable NAS algorithms. GDAS is considered to be the baseline as E^{2}NAS is built on GDAS with the same experimental settings. Note that DARTS trains the whole supernet rather than a single path during the architecture search. Following [41], the training epochs were set to 1/5 of others.

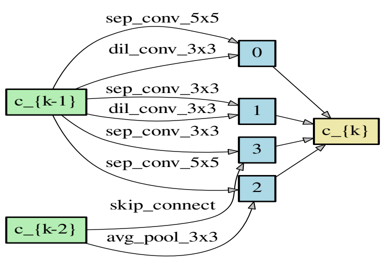
The results show the search time for E^{2}NAS at nearly twice that of GDAS because of the additional architecture that needs to be evaluated in each step and the probabilistic novelty calculation.

Table 4.5: The searching time of differentiable NAS baselines.

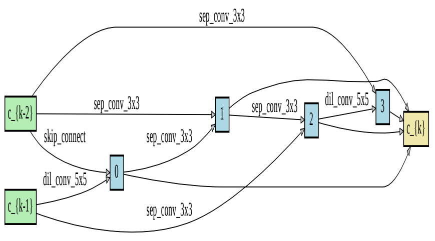
	GDAS	DARTS(1st)	DARTS(2nd)	NAO	SETN	E^{2}NAS
Searching time(s)	62812.6	21472.1	65325.9	74321.5	68132.5	142321.2

4.4.2 Experiments on DARTS Search Space

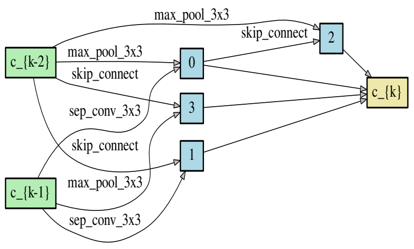
We also apply our proposed approach to a convolutional architecture search in the common DARTS search space [40, 86, 95] in comparison to the current state-of-the-art NAS methods. The candidate operations in the search space contained: 3×3 and 5×5 separable convolutions;



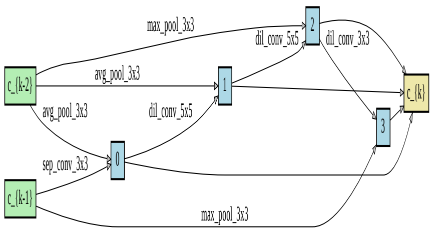
(a) Normal cell with E^{2}NAS



(b) Normal cell with E^{2}NAS-R



(c) Reduction cell with E^{2}NAS



(d) Reduction cell with E^{2}NAS-R

Figure 4.5: Best found cells with E^{2}NAS and E^{2}NAS-R on CIFAR-10.

3×3 and 5×5 dilated separable convolutions; 3×3 average pooling; 3×3 max-pooling; identity; and zero. The search procedure needs to look for two different types of cells with which to build the architecture: normal cells α_{normal} and reduction cells α_{reduce}. Reduction cells are only located in the first and second thirds of the total depth of the network. As described in Section 4.3.3, encouraging the search to look for architectures with greater depth can improve performance and transferability. Hence, E^{2}NAS-R incorporates a depth regularization function into the architecture optimization in the form of Eq.(4.15). For this experiment, we set ε=0.5 and $\gamma=\texttt{Sig}_{\gamma}(10)$ for E^{2}NAS and η=0.05 for E^{2}NAS-R.

Convolutional architecture searches in a DARTS search space generally include three sequential stages: the architecture search, the architecture evaluation, and transfer for evaluation with a larger dataset(s). In the architecture search stage, we stacked eight convolutional cells to form the architecture for the search. The number of initial channels c was set to 16, and the supernet was trained for 100 epochs with a batch size of 32. Once the most promising cell was found, we stacked 20 cells for architecture evaluation with a batch size of 96 for 600 epochs. The number of initial channels was increased to 36, the auxiliary towers were given a weight of 0.4, and the path dropout probability was set to 0.2. The other settings remained the same. The best-found cell structure on CIFAR-10 was subsequently transferred to CIFAR-100 with the same experimental settings.

Table 4.6: Comparison results with state-of-the-art weight-sharing NAS approaches.

Method	Test Error (%)			Param	FLOPs	×+	Search	Supernet
	CIFAR-10	CIFAR-100	ImageNet	(M)	(M)	(M)	Cost	Optimization
NASNet-A [172]	2.65	17.81	26.0 / 8.4	3.3	604	564	1800	RL
PNAS [93]	3.41±0.09	17.63	25.8 / 8.1	3.2	529	588	225	SMBO
AmoebaNet-A [118]	3.34±0.06	-	25.5 / 8.0	3.2	526	555	3150	EA
ENAS [114]	2.89	18.91	-	4.6	-	-	0.5	RL
EN^{2}AS [162]	2.61±0.06	16.45	26.7 / 8.9	3.1	498	506	0.3	EA
RandomNAS [86]	2.85±0.08	17.63	27.1	4.3	604	613	2.7	random
NSAS [160]	2.59±0.06	17.56	25.52 / 8.2	3.1	498	506	2.7	random
NSAS-C [160]	2.65±0.05	16.69	24.65 / 7.5	3.5	557	566	2.7	random
WPL [11]	3.81	-	-	-	-	-	-	RL
NAO-WS [99]	3.53	-	-	-	2.5	-	-	gradient
SNAS [140]	2.85±0.02	20.09	27.3 / 9.2	2.8	438	522	1.5	gradient
PARSEC [21]	2.86±0.06	17.06	26.3 / 8.4	3.6	502	509	0.6	gradient
BayesNAS [170]	2.81±0.04	-	26.5 / 8.9	3.40	-	-	0.2	gradient
MdeNAS [169]	2.55	17.61	25.5 / 7.9	3.61	595	604	0.16	gradient
DSO-NAS [167]	2.84±0.07	-	26.2 / 8.6	3.0	-	-	1	gradient
XNAS* [109]	2.57±0.09*	16.34*	24.7* / 7.5*	3.7	590	599	0.3	gradient
PDARTS [25]	2.50	16.55	24.4 / 7.4	3.4	532	557	0.3	gradient
PC-DARTS [142]	2.57±0.07	17.11	25.1 / 7.8	3.6	557	586	0.3	gradient
PR-DARTS [171]	2.32	16.45	24.1 / 7.3	3.4	-	543	0.17	gradient
DrNAS [24]	2.54±0.03	16.30	24.2 / 7.3	4.0	586	644	3.9	gradient
DARTS (1st) [95]	2.94	17.76	-	2.9	505	513	1.5	gradient
DARTS (2nd) [95]	2.76±0.09	17.54	26.9 / 8.7	3.4	530	574	4	gradient
GDAS [40]	2.93	18.38	27.5 / 8.5	3.4	537	537	0.21	gradient
E^{2}NAS	**2.58±0.09**	16.59	25.1 / 7.8	3.3	545	553	0.8	gradient
E^{2}NAS-R	**2.42±0.07**	**15.77**	**23.9 / 7.1**	3.8	582	591	0.8	gradient

“*" indicates the results reproduced based on the best-reported cell structures with a common experimental setting [95]. We do not reproduce those methods with “-" since those approaches are with different search spaces or do not report their best structures. The Para (M) indicates the model size when applied on CIFAR-10, while the FLOPs (M) and ×+ (M) is calculated when the searched model applied on ImageNet dataset. The best single-run of our E^{2}NAS and E^{2}NAS-R achieves 2.54% and 2.37% test error on CIFAR-10, respectively.

The experimental settings for the evaluation with ImageNet were slightly different from CIFAR-10 in that only 14 cells were stacked, and the number of initial channels was changed to 48. The remaining settings follow [95].

4.4.2.1 Results on CIFAR10

To compare E^{2}NAS with the state-of-the-art NAS methods, we followed the experimental settings prescribed for DARTS just as most other recent works have done. We conducted the architecture search several times with different random seeds to obtain the architectures, and then retrained them to select the best architecture based on the retrained validation performance. The memory consumption of E^{2}NAS is the same as the single-path NAS [40, 55, 86], which is much less than DARTS [95] and other relaxation-based differentiable NAS methods [140, 170]. Since E^{2}NAS adopts single-path, it is very efficient – for example, the whole architecture search phase only took 0.4 GPU days. Compared to existing differentiable NAS baselines, E^{2}NAS and E^{2}NAS-R provide much better results, which, again, verifies the effectiveness of our proposed method. Further, it is inspiring that the best-searched architecture of E^{2}NAS-R performed better than the most compared one-shot NAS under the same experimental settings, with a test error of only 2.42±0.07% on CIFAR-10.

4.4.2.2 Results on CIFAR100

To verify the transferability of our best model, we directly transferred it to CIFAR-100 and executed it without an architecture search. The experimental settings with CIFAR-100 were the same as for CIFAR-10. The results appear in Table 4.6. E^{2}NAS delivered a competitive result with a test error of 16.59%, but not as competitive as its performance on the CIFAR-10 dataset. On CIFAR-100, XNAS yielded an excellent result and PDARTS also performed well, beaten only by the E^{2}NAS-R variant with a test error of 15.77%, outperforming its peers by a large margin.

4.4.2.3 Results on ImageNet

The comparison results with the ImageNet dataset are presented in Table 4.6. All weight-sharing NAS methods transfer the searched cell architecture on CIFAR-10 to ImageNet, and we also follow the mobile setting from [86, 95, 140] with a 224×224 input image size. The number of initial channels was set to 48, and the network was stacked to 14 cells with a batch size of 128 and 250 training epochs.

E^{2}NAS delivered a competitive result with the Top1/Top5 test errors at 25.1%/7.8% given 4.69 million parameters. The FLOPs was only 490 million, which outperforms most SOTA models. Again, a few of the baselines outperformed E^{2}NAS, particularly XNAS and PDARTS, which encourage the search to look for deeper cell structures. The E^{2}NAS-R variant, which also searches for deeper cells, achieved a competitive result of 23.9% test errors. These results, together with the above CIFAR-100 results, demonstrate the importance of a preference for deeper architectures so as to improve transferability.

4.4.2.4 Regularization Analysis

In this section, we analyze the regularization method in E^{2}NAS-R, which encourages the search to look for deeper architectures. Figs. 4.5 (a) and (b) demonstrate the best-found architectures by E^{2}NAS and E^{2}NAS-R. As shown, the cell structure of the architecture found by E^{2}NAS-R is much deeper than the one found by E^{2}NAS. Table 4.6 also compares the results of the two architectures on CIFAR-10, CIFAR-100, and ImageNet datasets. As demonstrated, although the shallow cell searched by E^{2}NAS yielded a satisfactory result with a 2.54% test error on the small CIFAR-10 dataset, its performance on larger datasets, CIFAR-100 and ImageNet, was not as competitive with rates of 16.59% and 25.1%, respectively. However, the architectures found by E^{2}NAS-R still achieves state-of-the-art performance when transferred to the larger datasets, with 15.77% and 23.9% on CIFAR-100 and ImageNet, respectively. These results verify that encouraging NAS to search for deeper architectures can improve transferability in complicated real-world search spaces.

4.5 Chapter Summary and Discussion

The framework presented in this chapter provides a means of efficiently, yet intelligently, searching for differentiable neural architectures in a latent continuous space. A variational graph autoencoder transforms discrete architectures into an equivalent continuous space by injective means, while a probabilistic exploration enhancement method encourages intelligent exploration of the search space during supernet training. In addition, the framework includes a novel architecture complementation loss function to relieve catastrophic forgetting, along with a theoretical demonstration that the proposed loss function provides an identical result to the currently-accepted best-practice methods but is easier to calculate. Plus, to further improve the transferability of the searched architectures, the architecture optimization includes a depth regularization function that controls the depth of architectures to search for. Experiments on a NAS benchmark dataset and the common DARTS convolutional search space show the effectiveness of the proposed framework.

As described, differentiable NAS methods transform the discrete architecture search problem into a continuous optimization problem, while most of them hardly guarantee the optimization in the latent space equals to the discrete space. The final discretization stage of DARTS is still an open and unsolved problem, since the validation performance of a continuous representation architecture can not exactly indicate the performance of a discrete one. There are several recent works also try to relieve this issue, e.g. perturbation-based architecture selection[135]. This chapter points out another promising direction, transforming discrete architectures into an equivalent continuous space by injective means, rather than the commonly used *softmax-argmax*. The exploration enhancement technique in this chapter is related to the Chapter 2, where both of them show that enhancing the exploration can improve the predictive ability of the supernet work. In addition, the architecture complementation part of the proposed E^{2}NAS also helps to overcome catastrophic forgetting, which is related to Chapter 3. The problems, injective transformation, exploration enhancement, and catastrophic forgetting, are three general research directions in the differentiable neural architecture search community.

CHAPTER 5

DIFFERENTIABLE NEURAL ARCHITECTURE SEARCH VIA BAYESIAN LEARNING RULE

5.1 Introduction

Neural Architecture Search (NAS) [30, 82, 119] is attaining increasing attention in the deep learning community by automating the labor-intensive and time-consuming neural network design process. More recently, NAS has achieved the state-of-the-art results on various deep learning applications, including image classification [133], object detection [28], stereo matching [31]. Although NAS has the potential to find high-performing architectures without human intervention, the early NAS methods have extremely-high computational requirements [53, 118, 173]. For example, in [118, 173], NAS costs thousands of GPU days to obtain a promising architecture through reinforcement learning (RL) or evolutionary algorithm (EA). This high computational requirement in NAS is unaffordable for most researchers and practitioners. Since then, more researchers shift to improve the efficiency of NAS methods [54, 86, 114]. Weight sharing NAS, also called One-Shot NAS [9, 114], defines the search space as a supernet, and only the supernet is trained for once during the architecture search. The architecture evaluation is based on inheriting weights from the supernet without retraining, thus significantly reducing the computational cost. *Differentiable architecture search* (DARTS) [95], which is one of the most representative works, further relaxes the discrete search space into continuous space and jointly optimize supernet weights and architecture parameters with gradient descent, to further improve efficiency. Through employing two techniques, weight sharing [9, 114] and continuous relaxation [20, 40, 95, 140], DARTS reformulates the discrete operation selection problem in NAS as a continuous magnitude optimization problem, which reduces the computational cost significantly and completes the architecture search process within several hours on a single GPU.

Despite notable benefits on computational efficiency from differentiable NAS, more recent works find it is still unreliable [23, 152] to directly optimize the architecture magnitudes. For example, DARTS is unable to stably obtain excellent solutions and yields deteriorative architectures during the search proceeds, performing even worse than random search in some cases [124]. This critical weakness is termed as *instability* in differentiable NAS [152]. Zela et al. [152] empirically point out that the instability of DARTS is highly correlated with the dominant eigenvalue of the Hessian of the validation loss with respect to the architectural parameters, where this dominant eigenvalue increases during the architecture search. Accordingly, they proposed a simple early-stopping criterion based on this dominant eigenvalue to robustify DARTS. On the other hand, [24, 87, 127, 161] state that directly optimizing the architecture parameters without exploration easily entails the rich-gets-richer problem, leading to those architectures that converge faster at the beginning while achieve poor performance at the end of training, e.g. architectures with intensive *skip-connections* [33, 91].

Unlike existing works that directly optimize the architecture parameters, we formulate the neural architecture search as a distribution learning problem for differentiable NAS. We investigate differentiable NAS from a Bayesian learning perspective, and introduce the **Ba**yesian **Le**arning rule [72, 73, 106, 111] to the architecture optimization in differentiable **NAS** that considers natural-gradient variational inference (NGVI) methods to optimize the architecture distribution, which we call **BaLeNAS**. We theoretically demonstrate how the framework naturally enhance the exploration for differentiable NAS and improves the stability, and experimental results confirm that our framework enhances the performance for differentiable NAS. Specifically, our approach achieves state-of-the-art performance on NAS-Bench-201 [41] and improves the performance on NAS-Bench-1Shot1 [154] by large margins. To further improve the performance and transferability in the more complicated DARTS search space, we introduce a depth regularization based method to our framework. Specifically, our approach achieves state-of-the-art performance on NAS-Bench-201 [41] and improves the performance on NAS-Bench-1Shot1 [154] by large margins, and obtains competitive results on CIFAR-10, CIFAR-100, and ImageNet datasets in the DARTS [95] search space, with test error 2.37%, 15.72%, and 24.2%, respectively. Our contributions are summarized as follows.

- Firstly, this chapter formulates the neural architecture search as a distribution learning problem and builds a generalized Bayesian framework for architecture optimization in differentiable NAS. We demonstrate that the proposed Bayesian framework is a practical solution to enhance exploration for differentiable NAS and improve stability as a by-product via implicitly regularizing the Hessian norm.

- Secondly, to further improve the transferability, we incorporate the domain knowledge, increasing the depth of the searched architectures, into our framework through regularization strategies. Experiments on the common DARTS search space illustrate the benefits of the proposed method.
- Thirdly, the proposed framework is built based on DARTS and is also comfortable to be extended to other differentiable NAS methods with minimal modifications through leveraging the natural-gradient variational inference (NGVI). Extensive experiments show that our framework consistently improves the baselines with obtaining more competitive results in various search spaces.

This chapter is based on a submission "*Miao Zhang, Steven Su, Shirui Pan, Xiaojun Chang, Li Wang, Gholamreza Haffari, Differentiable Neural Architecture Search via Bayesian Learning Rule. Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2021*". Miao Zhang conceived the original idea of remorfulating the architecture search into a distribution learning problem which is solved by natural-gradient variational inference. The BaLeNAS algorithm was originally devised by Miao Zhang. Steven Su helped Miao Zhang to verify all derivations of theoretical parts in this paper. Miao Zhang conducted all experiments. The first version of the paper was written by Miao Zhang with some help from Shirui Pan. Steven Su and Shirui Pan revised the paper many times. The authors Li Wang, Xiaojun Chang and Gholamreza Haffari provided feedback during the writing of the paper.

5.2 Preliminaries

5.2.1 Differentiable Neural Architecture Search

Differentiable architecture search (DARTS) is built on weight-sharing NAS [9, 114], where the supernet is trained for once per the architecture search cycle. Rather than using the heuristic methods [114, 161] to search for the promising architecture in the discrete architecture space $\mathscr{A}$, DARTS [95] proposes the differentiable NAS framework by applying a continuous relaxation (usually a *softmax*) to the discrete architecture space and enabling gradient descent for architecture optimization. Therefore, architecture parameters α_θ and supernet weights w could be jointly optimized during the supernet training, and the promising architecture parameters

α_θ^* are obtained once the supernet is trained. The bilevel optimization formulation is usually adopted to alternatively learn α_θ and w:

$$\min_{\alpha_\theta \in \mathscr{A}_\theta} \mathscr{L}_{\texttt{val}}\Big(\operatorname*{\texttt{argmin}}_{w} \mathscr{L}_{\texttt{train}}(w(\alpha_\theta), \alpha_\theta)\Big), \tag{5.1}$$

and the best discrete architecture α^* is obtained after applying *argmax* on α_θ^*. Based on the differentiable NAS framework, different relaxation methods can be adopted. For example, NAO [99] utilizes the LSTM based autoencoder to transform the discrete architecture into a continuous space. Several recent works further adopt the graph neural network-based autoencoder to learn the injective representations. Zhang et al. [158] utilize the graph autoencoder to injectively transform the discrete architecture space into a latent space, and equivalently perform optimization in the continuous latent space. Similarly, Yan et al. [145] propose an unsupervised architecture representation learning method to injectively map discrete architectures to unique representations in the latent space, and the empirical results verify its benefits for the downstream architecture search.

As described above, the differentiable NAS first relaxes the discrete architectures into continuous representations to enable the gradient descent optimization, and projects the continuous architecture representation α_θ into discrete architecture α after the differentiable architecture optimization. Taking the DARTS as example, the searched architecture parameters α_θ are continuous, while α is represented with $\{0, 1\}$ after *argmax*. DARTS assumes that the $\mathscr{L}_{val}(w^*, \alpha_\theta^*)$ is a good indicator to the validation performance of α, $\mathscr{L}_{val}(w^*, \alpha^*)$. However, when we conduct the Taylor expansion on the local optimal α_θ^* [23, 24], we have:

$$\begin{aligned} \mathscr{L}_{val}(w^*, \alpha^*) &= \mathscr{L}_{val}(w^*, \alpha_\theta^*) + \nabla_{\alpha_\theta} \mathscr{L}_{val}(w^*, \alpha_\theta^*)^T (\alpha^* - \alpha_\theta^*) \\ &\quad + \frac{1}{2}(\alpha^* - \alpha_\theta^*)^T \mathscr{H} (\alpha^* - \alpha_\theta^*) \\ &= \mathscr{L}_{val}(w^*, \alpha_\theta^*) + \frac{1}{2}(\alpha^* - \alpha_\theta^*)^T \mathscr{H} (\alpha^* - \alpha_\theta^*) \end{aligned} \tag{5.2}$$

where $\nabla_{\alpha_\theta} \mathscr{L}_{val} = 0$ due to the local optimality condition, and $\mathscr{H}$ is the Hessian matrix of $\mathscr{L}_{val}(w^*, \alpha_\theta)$. We can see that the incongruence of the final continuous architecture representation and the final discrete architecture relates to the Hessian matrix's norm. However, as demonstrated by the empirical results in [152], the eigenvalue of this Hessian matrix increases during the architecture search, incurring more incongruence. This phenomenon is also called the instability of differentiable NAS [23]. Keeping the the Hessian matrix's norm in a low level plays a key role in robustifying the performance of differentiable NAS [152].

5.2.2 Distribution learning based NAS

Unlike directly optimizing the architecture parameters, several recent works formulate the differentiable NAS as a distribution learning problem by relaxing architecture parameters into different distributions. SNAS [140] and GDAS [40] formulate the architecture as a discrete distribution with concrete relaxation and utilize the Gumbel-softmax trick to obtain the discrete architecture. DrNAS [24] treats the continuous architecture parameters as random variables being modeled by a learnable Dirichlet distribution. This distribution is parameterized by a concentration parameter β, which controls the sampling behavior and is optimized via pathwise derivative estimators [66]. Zheng et al. [169] consider the whole search space as a joint multinomial distribution and learn the probabilities of candidate operations among all nodes based on the multinomial distribution learning. A common point in these previous methods is that they formulate the architecture parameters as simple distributions in which only one parameter needs to be learned. In this way, these learning paradigms are easy to fit with existing DARTS codebases.

Rather than considering the above distributions, this chapter considers the more general Gaussian distributions for the architecture parameters. By leveraging natural-gradient variational inference (NGVI), the architecture parameter distribution could be learned with by only updating a natural parameter λ during the search. The most relevant work to ours is BayesNAS [170], which also considers the Bayesian learning approach for neural architecture search. BayesNAS models the architecture parameters with hierarchical automatic relevance determination (HARD) priors, while which casts NAS as a model compression problem. The architecture parameters is formulated as $q(\theta) \sim \mathcal{N}(\mu, \psi^{-1})$, where ψ is a hyperparameter to tune rather than a parameter to learn. Furthermore, not only the architecture parameters are formulated as distributions, the supernet in BayesNAS is also formulated as a Bayesian Neural Network, which is hard to train and BayesNAS could only train the supernet for one epoch. Differently, our BaLeNAS only replaces the Adam optimizer with the Variational Adam optimizer for architecture optimization in the DARTS codebase, and keeps the supernet the same. In this way, our BaLeNAS is easy to be applied to most existing differentiable NAS codebases with minimal modifications. After the optimization of BaLeNAS, we learns an optimized Gaussian distribution for the architecture parameters $q(\alpha_\theta^* \mid \mu, \sigma^2)$, which is used to get the optimal architecture α^*. In this paper, we consider a simple and direct method, which utilizes the expectation of α_θ^* to select the best operation for each edge through the *argmax*, where the expectation term is simply the mean μ [24].

5.2.3 Deep Learning with Bayesian Principles

Given a dataset $\mathscr{D} = \{\mathscr{D}_1, \mathscr{D}_1, ..., \mathscr{D}_N\}$ and a deep neural network with parameters θ, the most popular method to learn θ with $\mathscr{D}$ is Empricial Risk Minimization (ERM):

$$\min \bar{\ell}(\theta) := \sum_{i=1}^{N} \ell_i(\theta) + \eta \mathscr{R}(\theta), \tag{5.3}$$

where ℓ_i is a loss function of i'th data example, e.g., $\ell_i = -\log p(\mathscr{D}_i \mid \theta)$ for classification tasks and $\mathscr{R}$ is the regularization term.

In contrast, the Bayesian deep learning estimate the posterior distribution of θ, $p(\theta \mid \mathscr{D}) := p(\mathscr{D} \mid \theta)p(\theta)/p(\mathscr{D})$, where $p(\theta)$ is the prior distribution. However, the normalization constant $p(\mathscr{D}) = \int p(\mathscr{D} \mid \theta)p(\theta)d\theta$ is difficult to compute for large DNNs. The variational inference (VI) [50] resolves this issue in Bayesian deep learning by approximating $p(\theta \mid \mathscr{D})$ with a new distribution $q(\theta)$, and minimizes the Kullback-Leibler (KL) divergence between $p(\theta \mid \mathscr{D})$ and $q(\theta)$,

$$\text{argmin}_\theta \text{KL}(q(\theta) \parallel p(\theta \mid \mathscr{D})). \tag{5.4}$$

When considering both $p(\theta)$ and $q(\theta)$ as Gaussian distributions with diagonal covariances:

$$p(\theta) := \mathscr{N}(\theta \mid \mathbf{0}, \mathbf{I}/\delta), \;\; q(\theta) := \mathscr{N}(\theta \mid \mu, \text{diag}(\sigma^2)), \tag{5.5}$$

where δ is a known precision parameter with $\delta > 0$, the mean μ and deviation σ^2 of q can be estimated by minimizing the negative of evidence lower bound (ELBO) [16]:

$$\begin{aligned} \mathscr{L}(\mu, \sigma) :&= -\sum_{i=1}^{N} \mathbb{E}_q \left[\log p(\mathscr{D}_i \mid \theta)\right] + \text{KL}(q(\theta) \parallel p(\theta)) \\ &= -\mathbb{E}_q \sum_{i=1}^{N} \log p(\mathscr{D}_i \mid \theta) + \mathbb{E}_q \left[\log \frac{q(\theta)}{p(\theta)}\right] \end{aligned} \tag{5.6}$$

A straightforward approach is using the stochastic gradient descent to learn μ and σ^2 along with minimizing $\mathscr{L}$, called as the Bayes by Backprob (BBB) [17]:

$$\mu_{t+1} = \mu_t - \varsigma_t \hat{\nabla}_\mu \mathscr{L}_t, \;\; \sigma_{t+1} = \sigma_t - \varphi_t \hat{\nabla}_\sigma \mathscr{L}_t, \tag{5.7}$$

where ς_t and φ_t are the learning rates, and $\hat{\nabla}_\mu \mathscr{L}_t$ and $\hat{\nabla}_\sigma \mathscr{L}_t$ are the unbiased stochastic gradient estimates of $\mathscr{L}$ at μ_t and σ_t. However, VI remains to be impractical for learning large deep networks. The first obvious issue is that VI introduces more parameters to learn, as it needs to simultaneously optimize two vectors μ and σ to estimate the distribution of θ, so the memory requirement is also doubled. More importantly, existing codebases of differentiable NAS are designed to learn with MLE loss function and require a lot of modifications to fit the VI. For example, we need to re-design the objective function and replace all neural networks weights with random variables.

5.3 Bayesian Learning Rule for Neural Architecture Search (BaLeNAS)

5.3.1 Formulating NAS as Distribution Learning

Differentiable NAS normally considers the architecture parameters α_θ as learnable parameters and directly conducts optimization in this space. Most previous differentiable NAS methods first optimize the architecture parameters based on the gradient of the performance, then update the supernet weights based on the updated architecture parameters. Since architectures with updated supernet weights are supposed to have higher performance, architectures with better performance in the early stage have a higher probability of being selected for the supernet training. The supernet training again improves these architectures' performance. This is to say, directly optimizing α_θ without exploration easily entails the *rich-get-richer problem* [87, 161], leading to suboptimal paths in the search space that converges faster at the beginning but plateaued quickly [24, 127]. Introducing stochasticity and encouraging exploration are the natural ways to resolve this problem [23, 24].

In this chapter, we formulate the architecture search as a distribution learning problem, optimizing the posterior distribution $p(\alpha_\theta \mid \mathscr{D})$ rather than α_θ. Similarly, we also consider both $p(\theta)$ and $q(\theta)$ as Gaussian distributions as Eq.(5.5). Accordingly, the bilevel optimization problem in Eq.(5.1) could be reformulated as the distribution learning based NAS:

$$\begin{aligned}&\min_{\mu,\sigma} \mathbb{E}_{q(\alpha_\theta \mid \mu,\sigma)} \mathscr{L}_{\mathtt{val}}(w^*(\alpha_\theta), \alpha_\theta), \\ &\text{s.t. } w^*(\alpha_\theta) = \underset{w}{\mathtt{argmin}} \mathscr{L}_{\mathtt{train}}(w(\alpha_\theta), \alpha_\theta),\end{aligned} \tag{5.8}$$

where μ and σ are the two learnable parameters for the distribution $q(\alpha_\theta \mid \mu,\sigma) := \mathscr{N}(\alpha_\theta \mid \mu, \text{diag}(\sigma^2))$. Considering the variational inference and Bayesian deep learning, based on Eq.(5.4)-(5.6), the loss function for the outer-loop architecture distribution optimization problem could be defined as:

$$\mathscr{L}(\mu,\sigma) := -\mathbb{E}_q \sum_{i=1}^{N} \log p(\mathscr{D}_i \mid \alpha_\theta) + \mathbb{E}_q \left[\log \frac{q(\alpha_\theta)}{p(\alpha_\theta)}\right] \tag{5.9}$$

Since the architecture parameters α_θ are random variables sampled from the Gaussian distribution $q(\alpha_\theta)$, the distribution learning-based method naturally encourages exploration during the architecture search.

5.3.2 Natural-Gradient Variational Inference for NAS

As describe in Sec.5.2.3, the traditional variational inference has double memory requirement and needs to re-design the object function, making it difficult to fit with the differentiable NAS. Thus, this chapter considers natural-gradient variational inference (NGVI) methods [72, 111] to optimize the architecture distribution $p(\alpha_\theta \mid \mathscr{D})$ in a natural parameter space, which requires the same number of parameters as the traditional learning method.

NGVI parameterizes the distribution $q(\alpha_\theta)$ with a natural parameter λ, considering $q(\alpha_\theta \mid \lambda)$ in a class of minimal exponential family with natural parameter λ [71]:

$$q(\alpha_\theta \mid \lambda) := h(\alpha_\theta)\exp\left[\lambda^T \phi(\alpha_\theta) - A(\lambda)\right], \tag{5.10}$$

where $h(\alpha_\theta)$ is the base measure, $\phi(\alpha_\theta)$ is a vector containing sufficient statistics, and $A(\lambda)$ is the log-partition function.

When $h(\alpha_\theta) \equiv 1$, the distribution $q(\alpha_\theta \mid \lambda)$ could be learned by only updating λ during the training [72, 73], and λ could be learned in the natural-parameter space by:

$$\lambda_{t+1} = (1-\rho_t)\lambda_t - \rho_t \nabla_\mu \mathbb{E}_{q_t}\left[\bar{\ell}(\alpha_\theta)\right], \tag{5.11}$$

where ρ_t is the learning rate, $\bar{\ell}$ is in the form of Eq.(5.3), which is also identical with the normal architecture optimization loss function $\mathscr{L}_{val}$. And q_t is the $q(\alpha_\theta \mid \lambda)$ parameterized by λ_t, $\mu = \mu(\lambda)$ is the expectation parameter of $q(\alpha_\theta \mid \lambda)$, and the derivative $\nabla_\mu \mathbb{E}_{q_t(\alpha_\theta)}\left[\bar{\ell}(\alpha_\theta)\right]$ is taken at $\mu = \mu_t$ with Markov Chain Monte Carlo (MCMC) sampling. This is also called as the Bayesian learning rule [73].

When $p(\alpha_\theta)$ and $q(\alpha_\theta)$ are in the form of Eq.(5.5), the Variational Online-Newton (VON) method proposed by Khan et. al. [72] shows that the NGVI update could be written with the following update:

$$\mu_{t+1} = \mu_t - \beta_t(\hat{\mathbf{g}}(\theta_t) + \tilde{\delta}\mu_t)/(\mathbf{s}_{t+1} + \tilde{\delta}), \tag{5.12}$$

$$\mathbf{s}_{t+1} = (1-\beta_t)\mathbf{s}_t + \beta_t \ \mathrm{diag}[\hat{\nabla}^2 \bar{\ell}(\theta_t)], \tag{5.13}$$

where β_t is the learning rate, $\theta_t \sim \mathscr{N}(\alpha_\theta \mid \mu_t, \sigma_t^2)$ with $\sigma_t^2 = 1/[N(\mathrm{s}_t + \tilde{\delta})]$ and $\tilde{\delta} = \delta/N$. $\hat{\mathbf{g}}$ is the stochastic estimate with respect to q through MCMC sampling that, $\hat{\mathbf{g}}(\theta_t) = \frac{1}{M}\sum_{i\in\mathscr{M}} \nabla_{\alpha_\theta} \bar{\ell}_i(\alpha_\theta)$, and the minibatch $\mathscr{M}$ contains M samples. More details are in [72]. Variational RMSprop (Vprop) [72] further uses gradient magnitude (GM) [18] approximation to reformulate Eq.(5.13) as:

$$\mathbf{s}_{t+1} = (1-\beta_t)\mathbf{s}_t + \beta_t[\hat{\mathbf{g}}(\theta_t) \circ \hat{\mathbf{g}}(\theta_t)], \tag{5.14}$$

with $\hat{\nabla}^2_{j,j}\bar{\ell}(\theta_t) \approx \left[\frac{1}{M}\sum_{i\in\mathcal{M}_t} g_i(\alpha^j_\theta)\right]^2 = [\hat{g}(\theta^j_t)]^2$ [18]. The most important benefit of VON and Vprop is that they only need to calculate one parameter's gradient to update posterior distribution. In this way, this learning paradigm requires the same number of parameters as traditional learning methods and easy to fit with existing codebases.

We implement the proposed BaLeNAS based on the DARTS [95] framework, the most popular differentiable NAS baseline. Similar to DARTS, BaLeNAS also considers an Adam-like optimizer for the architecture optimization, updating the natural parameter λ of $p(\theta \mid \mathscr{D})$ as:

$$\lambda_{t+1} = \lambda_t - \rho_t \nabla_\lambda \mathscr{L}_t + \gamma_t(\lambda_t - \lambda_{t-1}), \tag{5.15}$$

where the last term is the momentum. Based on the Vprop in Eq.(5.12) and (5.14), the update of μ and σ for the Adam-like optimizer with NGVI, also called as Variational Adam (VAdam), could be defined as following:

$$\mu_{t+1} = \mu_t - \beta_t(\hat{\mathbf{g}}(\theta_t) + \tilde{\delta}\mu_t)/(\mathbf{s}_{t+1} + \tilde{\delta}) + \gamma_t \left\lfloor \frac{\mathbf{s}_t + \tilde{\delta}}{\mathbf{s}_{t+1} + \tilde{\delta}} \right\rfloor \circ (\mu_t - \mu_{t-1}), \tag{5.16}$$

$$\mathbf{s}_{t+1} = (1-\beta_t)\mathbf{s}_t + \beta_t[\hat{\mathbf{g}}(\theta_t) \circ \hat{\mathbf{g}}(\theta_t)]. \tag{5.17}$$

where "$\circ$" stands for element-wise product, $\theta_t \sim \mathcal{N}(\alpha_\theta \mid \mu_t, \sigma_t^2)$ with $\sigma_t^2 = 1/[N(\mathrm{s}_t + \tilde{\delta})]$. As pointed out in Sec. 5.2.3 and shown in Eq.(5.16) and Eq.(5.17), the distribution $q(\alpha_\theta) = \mathcal{N}(\alpha_\theta \mid \mu, \sigma^2)$ is now optimized, needing to calculate the gradient of only one parameter.

5.3.3 Implicit Regularization with MCMC Sampling

Several recent works [23, 24, 152] empirically and theoretically show that the performance of differentiable NAS is highly related to the norm of $\mathscr{H}$, the Hessian matrix of $\mathscr{L}_{val}(w^*, \alpha_\theta)$, and keeping this norm in a low level plays a key role in robustifying differentiable NAS. As described before, we know the loss $\mathbb{E}_{q_t(\alpha_\theta)}\left[\bar{\ell}(\alpha_\theta)\right]$ of architecture optimization in BaLeNAS is calculated based on MCMC sampling, showing the naturality of enhancing exploration. Besides, $\mathbb{E}_{q_t(\alpha_\theta)}\left[\bar{\ell}(\alpha_\theta)\right]$ also has the naturality to enhance the stability in differentiable NAS. When conducting the Taylor expansion, the loss function for the architecture parameters update

Algorithm 5 BaLeNAS

```
Input: $\mathscr{D}_{train}$, $\mathscr{D}_{val}$
Create a supernet $w$ with a mixed operation parametrized by $\alpha_\theta$
1: while not converged do
2:     Update $\mu$ and $\sigma^2$ for $q(\alpha_\theta \mid \mu,\sigma^2)$ based on Eq.(5.16) and Eq.(5.17), with VAdam optimizer.
3:     Update supernet weights $w$ based on cross-entropy loss with the common SGD optimizer.
4: end while
5: Obtain discrete architecture $\alpha^*$ through argmax.
```

$\mathbb{E}_{q_t(\alpha_\theta)}\left[\bar{\ell}(\alpha_\theta)\right]$ could be described as:

$$\begin{aligned}\mathbb{E}_{q_t(\alpha_\theta)}\left[\bar{\ell}(\alpha_\theta)\right] =&\mathbb{E}_{q(\alpha_\theta|\mu,\sigma)}\mathscr{L}_{val}(w,\alpha_\theta) = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)}\mathscr{L}_{val}(w,\mu+\epsilon)\\ =&\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)}[\mathscr{L}_{val}(w,\mu)+\nabla_\mu\mathscr{L}_{val}(w,\mu)^T\epsilon+\frac{1}{2}\epsilon^T\mathscr{H}\epsilon]\\ =&\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)}\left[\mathscr{L}_{val}(w,\mu)+\frac{1}{2}\epsilon^T\mathscr{H}\epsilon\right]\\ =&\mathscr{L}_{val}(w,\mu)+\frac{\sigma^2}{2}\mathrm{Tr}\{\mathscr{H}\},\end{aligned} \tag{5.18}$$

where the line 4 in Eq.(5.18) is obtained since $\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)}[\nabla_\mu\mathscr{L}_{val}(w,\alpha_\theta)^T\epsilon] = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)}[\epsilon] * \nabla_\mu\mathscr{L}_{val}(w,\alpha_\theta) = 0$, as $\epsilon\sim\mathcal{N}(0,\sigma^2)$ is a Gaussian distribution with zero mean, and $\mathbb{E}(\epsilon^2)=\sigma^2$. μ is the expectation parameter of $q(\alpha_\theta \mid \mu,\sigma^2)$, and $\mathscr{H}$ is the Hessian matrix of $\mathscr{L}_{val}(w,\mu)$. We can find the loss function that could implicitly control the trace norm of $\mathscr{H}$ similar as [23, 24], helping stabilizing differentiable NAS.

5.3.4 Depth-Aware Regularization for BaLeNAS

Several recent works [109, 127, 171] observe that differentiable NAS prefers those shallower cells with a larger width after searching on CIFAR-10, while these architectures achieve poor generalization performance when transferring to large datasets. A recent study on neural network optimization [122] gives a hint as to why most NAS methods prefer wider networks. The authors define a simple concept as *gradient confusion*, where a smaller *gradient confusion* tends to faster the convergence. They show that increasing the width leads to lower gradient confusion. The theoretical analysis in [127, 171] also states that shallow cells have more smooth landscapes and can be optimized faster than deep architectures, leading DARTS to a bias of shallow structures.

To resolve this bias in the differentiable NAS, we could increase the depth of cells during the architecture search. An intuitive way is to use the l_0 norm to encourage the connection $c^{i,j}$

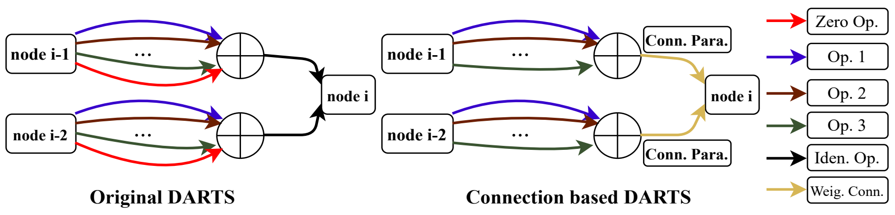


Figure 5.1: Comparison of node connection in original DARTS and *depth-aware* DARTS.

between node i and j during the search:

$$l_0^{depth} = \sum_{j=1}^{|C|} d(c_j), \quad d(c_j) \in \{0,1\}, \tag{5.19}$$

where $d(c_j) = 1$ when c_j connects node i and $i-1$ in the sampled architecture α based on α_θ, and $|C|$ is the number of all candidate connections in a sampled structure α. However, this discrete sampling is non-differentiable [40, 140], and we reparameterize α_θ using the Gumbel trick to relax the discrete architecture distribution to be continuous and differentiable, with $\bar{\alpha}_\theta = \text{Sigmoid}((\log\delta - \log(1-\delta) + \alpha_\theta)/\tau)$. Since the l_0 norm of architecture parameters is still non-differentiable, we could not directly incorporate Eq. (5.19) as regularization term in the loss function $\mathscr{L}$. While, with applying hard-sigmoid gates on the concrete distribution, we could make the expected l_0 regularized objective differentiable with respect to the distribution parameters [97, 171]. Following [97, 171], we first “stretch" $\bar{\alpha}_\theta$ to (γ,ζ) via $\bar{\alpha}_\theta = \gamma + (\zeta-\gamma)\bar{\alpha}_\theta$, where $\gamma < 0$ and $\zeta > 0$. Then a hard threshold gate is applied to obtain the final architecture parameter: $\hat{\alpha}_\theta = \min(1, \max(0, \bar{\alpha}_\theta))$. The loss function for the Eq. (5.19) could be reformulated as:

$$\mathscr{L}_0^{depth} = -\sum_{j=1}^{|C|} \text{Sigmoid}(\alpha_{\theta,j}^{depth} - \tau\log\frac{-\gamma}{\zeta}), \tag{5.20}$$

where $\alpha_{\theta,j}^{depth}$ is the parameter that controls the connection between node i and $i-1$. However, in the DARTS codebase, there is no controlling parameters for the depth connections $c^{i,i-1}$. Thus, we convert the $zero$ operation in the candidate operations as a connection parameter as shown in Fig.5.1. In this way, we could encourage the depth through controlling the connection parameters $\alpha_{\theta,j}^{depth}$ between node i and $i-1$. The final loss function to encourage the deep cell structures is then defined as:

$$\bar{\ell}(\alpha_\theta) = \mathscr{L}_{val}(w^*, \alpha_\theta) + \eta_1 \mathscr{L}_0^{depth}(\alpha_\theta), \tag{5.21}$$

where η_1 is the regularization trade-off. The natural parameter λ is then learned in the natural-parameter space based on Eq.(5.21) and Eq.(5.11)

5.4 Experimental Result

In this section, we consider three different search spaces to analyze the proposed BaLeNAS framework. The first two are NAS benchmark datasets, NAS-Bench-201 [41] and NAS-Bench-1Shot1 [154]. The ground-truth for all candidate architectures in the two benchmark datasets is known. The NAS methods could be evaluated without retraining the searched architectures based on these benchmark datasets, thus greatly relieving the computational burden. The third one is the commonly-used CNN search space in DARTS [95]. The depth regularization is only considered in the DARTS space. We first analyze our proposed BaLeNAS in the two benchmark datasets, then compare BaLeNAS with state-of-the-art NAS methods in the DARTS search space and analyze the proposed regularization-based method for differentiable NAS.

5.4.1 Experiments on Benchmark Datasets

The NAS-Bench-201 [41] has a unified cell-based search space, where the cell structure is densely-connected, containing four nodes with five candidate operations applied on each node, resulting in 15625 architectures. NAS-Bench-201 reports the CIFAR-10, CIFAR-100, and Imagenet performance for all architecture in this search space. The NAS-Bench-1Shot1 [154] is built from the NAS-Bench-101 benchmark dataset [150], through dividing all architectures in NAS-Bench-101 into 3 different unified cell-based search spaces, with containing 6240, 29160, and 363648 architectures, respectively, and the CIFAR-10 performance for all architectures in these three search spaces are reported. The architectures in each search space have the same number of nodes and connections, making the differentiable NAS could be directly applied to each search space.

5.4.1.1 Reproducible comparison on NAS-Bench-201

Table 6.1 summarizes the performance of BaleNAS on NAS-Bench-201 compared with differentiable NAS baselines, where the statistical results are obtained from 4 independent search experiments with four different *random seeds*. As shown in Table 6.1, our BaLeNAS achieves the best results on the NAS-Bench-201 benchmark and greatly outperforms other baselines on all three datasets. BaLeNAS (2nd) even achieves the near-optimal point with *random seed* {100,101}, with a 94.37%, 73.22%, and 46.71% test accuracy on CIFAR-10, CIFAR-100, and ImageNet, respectively. As described, our BaLeNAS is built based on the DARTS framework, with only modeling the architecture parameters into distributions and introducing Bayesian learning rule for optimization. As shown in Table 6.1, our BaLeNAS (1st) and BaLeNAS (2nd) both outperform DARTS with first and second-order approximations by large margins,

Table 5.1: Comparison results with state-of-the-art weight-sharing NAS approaches.

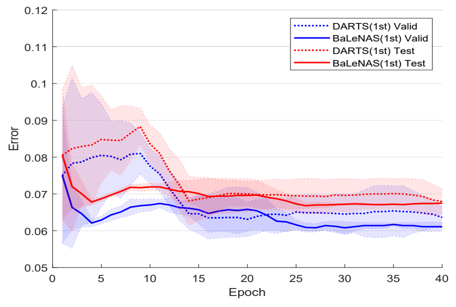
Method	Test Error (%)			Param	FLOPs	Search	Architecture
	CIFAR-10	CIFAR-100	ImageNet	(M)	(M)	Cost	Optimization
NASNet-A [172]	2.65	17.81	26.0 / 8.4	3.3	604	1800	RL
AmoebaNet-A [118]	3.34±0.06	-	25.5 / 8.0	3.2	526	3150	EA
ENAS [114]	2.89	18.91	-	4.6	-	0.5	RL
RandomNAS [86]	2.85±0.08	17.63	27.1	4.3	595	2.7	random
NSAS [158]	2.85±0.08	17.63	25.5 / 8.2	4.3	593	2.7	random
SNAS [140]	2.85±0.02	20.09	27.3 / 9.2	2.8	467	1.5	gradient
SETN [42]	2.69	17.25	25.7 / 8.0	4.6	601	1.8	gradient
BayesNAS [170]	2.81±0.04	-	26.5 / 8.9	3.40	-	0.2	gradient
RENAS [27]	2.88±0.02	-	24.3	3.5	6	RL&EA	
MdeNAS [169]	2.55	17.61	25.5 / 7.9	3.61	500	0.16	gradient
GDAS [40]	2.93	18.38	26.0 / 8.5	3.4	538	0.21	gradient
XNAS* [109]	2.57±0.09	16.34	24.7 / 7.5	3.7	590	0.3	gradient
PDARTS [25]	2.50	16.63	24.4 / 7.4	3.4	543	0.3	gradient
PC-DARTS [142]	2.57±0.07	17.11	25.1 / 7.8	3.6	571	0.3	gradient
DrNAS [24]	2.54±0.03	16.30	24.2 / 7.3	4.0	644	0.4	gradient
DARTS+ [91]	2.50±0.11	16.28	-	3.7	-	0.4	gradient
DARTS (2nd) [95]	2.76±0.09	17.54	26.9 / 8.7	3.4	530	4	gradient
BaLeNAS	**2.43±0.08**	**15.72**	**24.2 / 7.3**	3.86	597	1.3	gradient
BaLeNAS w/o	2.50±0.07	16.84	25.0 / 7.7	3.82	593	1.3	gradient

“BaLeNAS w/o" indicates BaLeNAS without the depth regularization. “Param" is calculated when applied on CIFAR-10, while “FLOPs" is based on the ImageNet. Our best single-run with BaLeNAS on CIFAR-10 achieved **2.37%** test error.

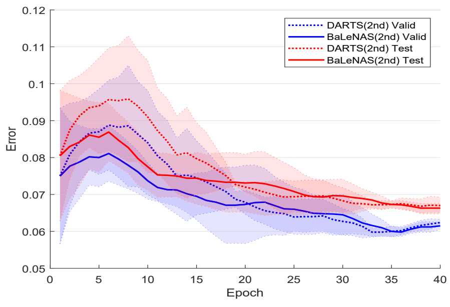
verifying the effectiveness of our method. By formulating the architecture search as a distribution learning problem and introducing the Bayesian learning rule to optimize the posterior distribution, our BaLeNAS can relieve the instability and naturally enhance exploration to avoid local optimum for differentiable NAS.

5.4.1.2 Reproducible comparison on NAS-Bench-1Shot1

We then conduct an ablation study on the NAS-Bench-1Shot1 dataset to verify the effectiveness of our BaLeNAS further. We have compared BaLeNAS with the baseline DARTS on the three search spaces of NAS-Bench-1Shot1 with tracking the validation and test performance of the search architectures in every iteration. As shown in Fig. 5.2, our BaLeNAS generally outperforms DARTS during the architecture search in terms of validation and test error in the most complicated search space 3, both with first and second-order approximation. More specifically, our BaLeNAS significantly outperforms the baseline in the early stage, demonstrating our BaLeNAS could quickly find the superior architectures and is more stable.



(a) Comparison results with baseline using first order approximation



(b) Comparison results with baseline using second order approximation

Figure 5.2: Validation and test error of BaLeNAS and DARTS on the search space 3 of NAS-Bench-1Shot1.

5.4.2 Experiments on DARTS Search Space

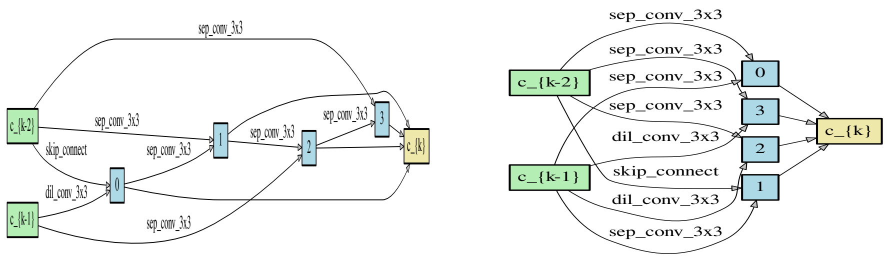
To compare with the state-of-the-art differentiable NAS methods, we applied BaLeNAS to the typical DARTS search space [40, 86, 95] for convolutional architecture search, where all experiment settings are following DARTS [95] for fair comparisons as the same as the most recent works. The architecture search in DARTS space generally contains three stages: The differentiable NAS first searches for micro-cell structures on CIFAR-10, and then stack more cells to form the full structure for the architecture evaluation. The best-found cell on CIFAR-10 is finally transferred to larger datasets to evaluate its transferability.

5.4.2.1 Search Results on CIFAR-10

The comparison results with the state-of-the-art NAS methods are presented in Table 5.1. We set $\eta_1 = 0.05$ for regularization on the DARTS search space. The best architecture searched by our BaLeNAS achieves a 2.37% test error on CIFAR-10, which outperforms state-of-the-art NAS methods. We can also see that BaLeNAS outperforms DARTS by a large margin, demonstrating the effectiveness of the proposed method. Besides, although BaLeNAS introduced MCMC during architecture optimization, it is still efficient in the sense that the whole architecture search phase in BaLeNAS (2nd) only took 1.3 GPU days.

5.4.2.2 Transferability Results Analysis

Following DARTS experimental setting, the best-searched architectures on CIFAR-10 are then transferred to CIFAR-100 and ImageNet to evaluate the transferability. The evaluation setting for CIFAR-100 is the same as CIFAR-10. In the ImageNet dataset, the experiment setting is slightly different from CIFAR-10 in that only 14 cells are stacked, and the number of initial



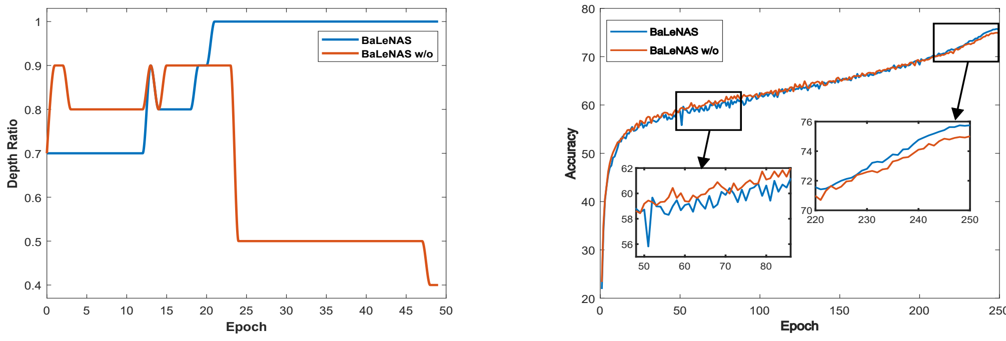
(a) With depth regularization (b) Without depth regularization

Figure 5.3: The best normal cells discovered by BaLeNAS with and without depth regularization.

channels is changed to 48. The comparison results with state-of-the-art differentiable NAS approaches on CIFAR-100 and ImageNet are demonstrated in Table 5.1. As shown in Table2, BaLeNAS achieves a 15.72% test error on the CIFAR-100 dataset, which is a state-of-the-art performance and outperforms peer algorithms by a large margin. On the ImageNet dataset, the best-discovered architecture by our BaLeNAS also achieved a competitive result with 24.2 / 7.3 % top1 / top5 test error, outperforming or on par with all peer algorithms.

5.4.2.3 Effectiveness of Depth Regularization

As described in Sec.5.3.4, we deploy a *depth-aware* regularization method to our BaLeNAS, to search for superior architectures on the complicated DARTS search space. we conduct experiments to validate the effectiveness of *depth-aware* regularization. Fig.5.3 (a) and (b) show the normal cells searched by the proposed BaLeNAS with and without the proposed *depth-aware*. Table 5.1 also gives the results on CIFAR-10, CIFAR-100, and ImageNet for the two architectures. Although the shallow cell searched by BaLeNAS w/o achieves competitive results with 2.50% test error on the small CIFAR-10 dataset, its performance is much worse than the deeper one when transferring to larger datasets, only achieving 16.84% and 25.0% on CIFAR-100 and ImageNet compared with 15.72% and 24.2% of the deeper one. Fig.5.4 (a) tracks the depth ratio of the searched architectures with and without depth regularization. The proposed BaLeNAS without regularization is also likely to select shallow architectures. In contrast, with the proposed *depth-aware* regularization, BaLeNAS is encouraged to search for deep cell structures. Fig.5.4 (b) plots the validation accuracy during the architecture evaluation on the ImageNet dataset for the two architectures. We could observe that the shallow one is easy to be trained as it achieves better results in the early stage, while the deep one is more promising as it obtains better performance after the training is complete. These results suggest that encouraging differentiable NAS to search for “deeper" architectures could improve



(a) Depth of the searched cells (b) Validation accuracy curve

Figure 5.4: (a) The depth of the searched cells during the architecture search with and without depth-aware regularization. (b) The validation performance of searched architectures by BaLeNAS and BaLeNAS w/o on ImageNet.

transferability in the real-world search space.

5.4.3 Ablation Study of MCMC on NAS-Bench-201

As we described in Section 5.3, one key additional hyperparameter in BaLeNAS is the sampling number M in MCMC, and this subsection investigates how this hyperparameter affects the performance of BaLeNAS. Table 5.2 summarizes the performance our BaLeNAS (2nd) with different number of MCMC sampling. As shown, our BaLeNAS is very robust to the number of MCMC sampling, where BaLeNAS achieves excellent results under different scenarios, outperforming most existing NAS baselines. An interesting observation is that the performance of BaLeNAS decreases with multiple samplings $M > 1$ in MCMC, and $M = 1$ achieves the best performance. A detailed explanation can be found in the Section 3.4 of [72]. The VAdam optimizer adopted by BaLeNAS considers a gradient magnitude (GM) approximation to update $\mathbf{s}_{t+1}$ in Eq. (5.14) that:

$$\hat{\nabla}^2_{j,j}\bar{\ell}(\theta_t) \approx \left[\frac{1}{M}\sum_{i\in\mathcal{M}_t} g_i(\alpha_\theta^j)\right]^2 = [\hat{g}(\theta_t^j)]^2. \tag{5.22}$$

However, the Theorem 1 in [72] points out that the GM approximation is an unbiased estimator of the Generalized Gauss-Newton (GGN) approximation only when $M = 1$, and VAdam with $M > 1$ will converge fast while might result in slightly worse estimates.

5.4.4 Ablation Study on the Effect of Exploration

Several recent works [24, 127, 161] point out that directly optimizing architecture parameters without exploration easily entails the rich-gets-richer problem, leading to those architectures

Table 5.2: Comparison results with different MCMC number for BaLeNAS on NAS-Bench-201.

MCMC number	CIFAR-10		CIFAR-100		ImageNet-16-120	
	Valid(%)	Test(%)	Valid(%)	Test(%)	Valid(%)	Test(%)
$M=1$	**91.52±0.09**	**94.33±0.04**	**72.67±0.08**	**72.95±0.27**	**45.39±0.17**	**46.32±0.39**
$M=2$	89.71±2.12	92.75±1.87	70.25±2.92	70.43±2.45	31.63±2.15	43.15±2.95
$M=3$	91.31±0.25	94.03±0.35	72.08±1.27	72.30±1.09	45.72±0.78	45.58±0.78
$M=4$	90.03±0.96	93.04±1.09	68.80±1.46	69.20±1.86	43.09±2.93	43.21±2.88
DARTS (2nd)	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
optimal	91.61	94.37	74.49	73.51	46.77	47.31

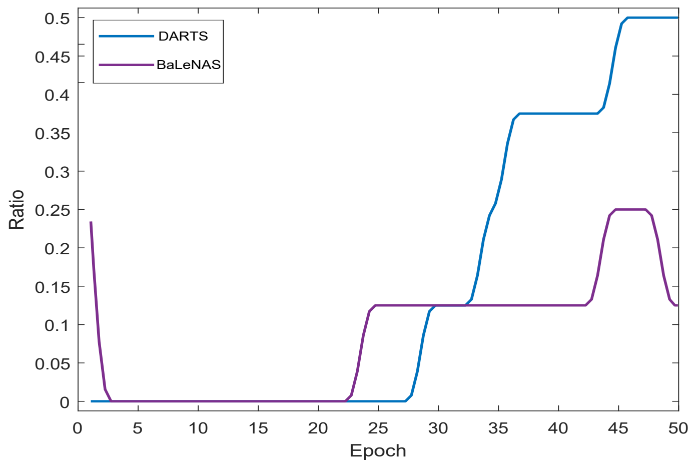


Figure 5.5: The ratio of skip-connection the searched normal cells during the architecture search in the DARTS space.

that converge faster at the beginning while achieve poor performance at the end of training, e.g. architectures with intensive *skip-connections* [33, 91]. However, when the number of *skip-connections* is larger than 3, the architecture's retraining accuracy is usually extremely low [91, 152]. To relieve this issue, we formulate the differentiable neural architecture search as a distribution learning problem. In this subsection, we investigate how the proposed formulation relieves this issue. Fig. 5.5 plots the ratio of *skip-connection* in the searched normal cell for BaLeNAS and DARTS (the total number of operations in a cell is 8). As shown, DARTS is likely to select more than 3 *skip-connection* in the normal cell during the search. In contrast, in the proposed BaLeNAS, the number of *skip-connections* is generally less than 2 in the normal cell during the search for BaLeNAS.

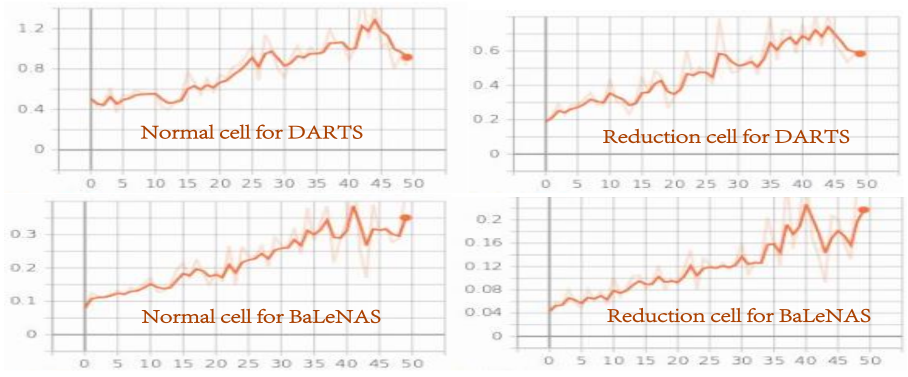


Figure 5.6: Trajectory of the Hessian norm in DARTS space.

5.4.5 Tracking of the Hessian norm

As described in Section 5.2.1, the incongruence between $\mathcal{L}_{val}(w^*, \alpha^*_\theta)$ and $\mathcal{L}_{val}(w^*, \alpha^*)$ is not negligible if we could not maintain the maintains the Hessian norm at a low level during the search. The analysis in Section 5.3.4 and Eq. (5.18) shows that the loss function of the proposed BaLeNAS implicitly controls the trace norm of $\mathcal{H}$ similar as [23, 24], helping stabilizing differentiable NAS. We plot the trajectory of the Hessian norm of BaLeNAS compared with the vanilla DARTS in Fig. 5.6. As show, the Hessian norm in our BaLeNAS is always kept in a low level. Although the Hessian norm of BaLeNAS also increases with the supernet training similar as DARTS, its largest value is still smaller than the Hessian norm of DARTS in the early stage, showing the effectiveness of implicit regularization of our BaLeNAS.

5.5 Chapter Summary and Discussion

This chapter formulates the architecture optimization in the differentiable NAS as a distribution learning problem and introduces a Bayesian learning rule to optimize the architecture parameters posterior distributions. We theoretically demonstrate that the proposed framework can enhance the exploration for differentiable NAS and implicitly impose regularization on the norm of Hessian matrix to improve the stability. We operationalize the framework based on the common differentiable NAS baseline, DARTS, and experimental results on NAS benchmark datasets have verified the proposed framework's effectiveness. To further improve the transferability, we have proposed a depth regularization methods to encourage the depth of the searched cells. Experimental results on the typical DARTS search space validate the advantages of this regularization method.

As we described, although Differentiable Architecture Search (DARTS) has received massive attention in recent years, mainly because it significantly reduces the computational cost through weight sharing and continuous relaxation, more recent works find that existing differentiable NAS techniques struggle to outperform naive baselines, yielding deteriorative architectures as the search proceeds. Rather than directly optimizing the architecture parameters, this chapter formulates the neural architecture search as a distribution learning problem through relaxing the architecture weights into Gaussian distributions. By leveraging the natural-gradient variational inference (NGVI), the architecture distribution can be easily optimized based on existing codebases without incurring more memory and computational consumption. This chapter opens up a new direction for the differentiable neural architecture search, considering architecture search as a distribution learning problem rather than magnitude optimization, since this reformulation can naturally enhance exploration and improve stability.

CHAPTER 6

DIFFERENTIABLE ARCHITECTURE SEARCH WITH STOCHASTIC IMPLICIT GRADIENTS

6.1 Introduction

Neural Architecture Search (NAS) is an efficient and effective method on automating the process of neural network design, with achieving remarkable success on image recognition [82, 84, 133], language modeling [67], and other deep learning applications [28, 31, 119]. The early NAS frameworks are devised via reinforcement learning (RL) [114] or evolutionary algorithm (EA) [118] to directly search on the discrete space. To further improve the efficiency, several recently-proposed works [28, 40, 95, 142] adopts the continuous relaxation to convert the operation selection problem into the continuous magnitude optimization for a set of candidate operations, where a recently proposed *Differentiable ARchiTecture Search* (DARTS) [95] has recently become the mainstream of neural architecture search due to its efficiency and simplicity. With a gradient-based bi-level optimization, DARTS alternately optimizes the inner model weights and the outer architecture parameter in a weight-sharing supernet. A key challenge to the scalability and quality of the learned architectures is the need for differentiating through the inner-loop optimisation. By enabling the gradient descent for the architecture optimization, DARTS significantly reduces the search cost to several GPU hours.

Despite its efficiency, more current works observe that DARTS is somewhat unreliable [23, 86, 124, 152, 159, 161] since it does not consistently yield excellent solutions, performing even worse than random search in some cases. The recent work [152] attributes the failure of DARTS to its supernet training, with empirically observing that the instability of DARTS is highly correlated to the dominant eigenvalue of the Hessian matrix of the validation loss with respect to architecture parameters. Paper [151] also empirically showed that the supernet training negatively impacts the correlation between the validation performance by inheriting wights from supernet and training from the scratch. On the other hand, [136] turn to the magnitude-based architecture selection process, who empirically and theoretically show the magnitude of architecture parameters does not necessarily indicate how much the operation contributes to the supernet's performance. Other researchers [23] observe a precipitous validation loss landscape with respect to architecture parameters, which leads to a dramatic performance drop when discretizing the final architecture for the operation selection. Accordingly, they propose a perturbation based regularization to smooth the loss landscape and improve the stability. Liang et al.'s [90] solution is to introduce another simple "early stopping" criteria, where the search procedure ends as soon as one cell has two or more *skip-connections*.

While there are many variants on improving the DARTS from various aspects, limited research attention has been paid to the approximation of the architecture parameter gradient, which is also called the outer-loop gradient or hypergradient. To fill the gap, this paper focuses on the hypergradient calculation in the differentiable NAS. The main contribution of this work is the development of the differentiable architecture search with stochastic implicit gradients (iDARTS). Specifically, we first revisit the DARTS from the bi-level optimization perspective and utilize the implicit function theorem (IFT) [10, 96], instead of the one-step unroll learning paradigm adopted by DARTS, to calculate the architecture parameter gradient. This IFT based hypergradient depends only on the obtained solution to the inner-loop optimization weights rather than the path taken, thus making the proposed method memory efficient and practical with numerous inner optimization steps. Then, to avoid calculating the inverse of the Hessian matrix with respect to the model weights, we utilize the Neumann series [96] to approximate this inverse and propose an approximated hypergradient for DARTS accordingly. After that, we devise a stochastic approximated hypergradient to relieve the computational burden further, making the proposed method applicable to the differentiable NAS. We theoretically demonstrate that, under some mild assumptions [36, 47, 52] on the inner and outer loss functions, the proposed method is expected to converge to a stationary point with small enough learning rates. Finally, we verify the effectiveness of the proposed approach on two NAS benchmark datasets and the common DARTS search space.

We make the following contributions in this chapter:

- This paper deepens our understanding of the hypergradient calculation in the differentiable NAS. We reformulated the hypergradient in the differentiable NAS with the implicit function theorem (IFT), which can thus gracefully handle many inner optimization steps without increasing the memory requirement.

- To relieve the heavy computational burdens, we consider a Neumann-approximation for the IFT based differentiable NAS. Further, to make the implicit hypergradient practical for differentiable NAS, we formulate a stochastic hypergradient with the Neumann-approximation.

- We provide a theoretical analysis of the proposed method and demonstrate that the proposed method is expected to converge to a stationary point when applied to differentiable NAS. Extensive experiments verify the effectiveness of the proposed method which significantly improves the performance of the differentiable NAS baseline on the NAS-Bench-1Shot1 and the NAS-Bench-201 benchmark datasets and the common DARTS search space.

This chapter is based on a publication "*Miao Zhang, Steven Su, Shirui Pan, Xiaojun Chang, Huiqi Li, Ehsan Abbasnejad, Gholamreza Haffari, iDARTS: Differentiable Architecture Search with Stochastic Implicit Gradients. In International Conference on Machine Learning (ICML), 2021*" [165]. Miao Zhang conceived the original idea of focusing on the hyper-gradient of bi-level optimization framework based DARTS. The iDARTS algorithm was originally proposed by Miao Zhang. Steven Su helped Miao Zhang to verify all derivations of theoretical parts in this paper. Miao Zhang conducted all experiments. The first version of the paper was written by Miao Zhang with some help from Gholamreza Haffari and Shirui Pan. Gholamreza Haffari and Shirui Pan revised the paper many times. The authors Xiaojun Chang, Huiqi Li, Ehsan Abbasnejad, and Gholamreza Haffari provided feedback during the writing of the paper.

6.2 Preliminaries: Hypergradient Approximation in DARTS

Existing differentiable NAS methods mostly leverage the weight sharing and continuous relaxation to enable the gradient descent for the discrete architecture search, significantly improving the search efficiency. DARTS [95] is one of the most representative differentiable NAS methods, which utilizes the continuous relaxation to convert the discrete operation selection into the magnitude optimization for a set of candidate operations. Typically, NAS searches for cells to stack the full architecture, where a cell structure is represented as a directed acyclic graph (DAG) with N nodes. NAS aims to determine the operations and corresponding connections for each node, while DARTS applies a **softmax** function to calculate the magnitude of each operation, transforming the operation selection into a continuous magnitude optimization problem:

$$\mathbf{X}_n = \sum_{0\le s<n} \sum_{o=1}^{|\mathcal{O}|} \bar{\alpha}_o^{(s,n)} o(\mathbf{X}_s), \qquad \bar{\alpha}_o^{(s,n)} = \frac{\exp(\alpha_o^{s,n})}{\sum_{o'\in\mathcal{O}} \exp(\alpha_{o'}^{s,n})},$$

where $\mathbf{X}_n$ is the output of node n, $\mathcal{O}$ contains all candidate operations, and the output of each node is the weighted sum of its previous nodes' outputs affiliated with all possible operations. In this way, DARTS transforms the discrete architecture search into optimizing the continuous magnitude $\hat{\alpha}_o^{s,n}$, enabling gradient descent for the architecture optimization. A discrete architecture is obtained by applying an **argmax** function to the magnitude matrix after the differentiable architecture optimization.

The optimization in DARTS is based on the bi-level optimization formulation [34, 95]:

$$\begin{aligned} &\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha) \\ &\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha), \end{aligned} \tag{6.1}$$

where α is the continuous architecture representation and w is the supernet weights. We indicate the $\mathcal{L}_{val}$ as $\mathcal{L}_2$ and the $\mathcal{L}_{train}$ as $\mathcal{L}_1$ in the remaining text for convenience. The nested formulation in DARTS is the same as the gradient-based hyperparameter optimization with bi-level optimization [46, 102, 113], where the inner-loop is to train the network parameter w and the outer-loop is to optimize the architecture parameter α. The gradient of the outer-loop for DARTS is then calculated as:

$$\nabla_\alpha \mathcal{L}_2 = \left(\frac{\partial \mathcal{L}_2}{\partial \alpha} + \frac{\partial \mathcal{L}_2}{\partial w}\frac{\partial w^*(\alpha)}{\partial \alpha}\right). \tag{6.2}$$

DARTS considers the one-step unroll learning paradigm [95, 116] for the hypergradient calculation. This is done by taking a single step in optimising w instead of the optimal w^*.

Different from the majority of existing works that attributes the failure of DARTS to its supernet optimization [11, 152], or the final discretization with **argmax** [23, 135], this paper revisits DARTS from the perspective of the hypergradient calculation $\nabla_\alpha \mathcal{L}_2$. Rather than considering the one-step unroll learning paradigm [44, 95], this paper utilizes the implicit function theorem (IFT) [10, 96] to reformulate the hypergradient calculation in DARTS. In the following subsection, we first recap the hypergradient calculation with different paradigms for DARTS.

6.2.1 One-step Unrolled Differentiation

The one-step unroll learning paradigm, as adopted by DARTS, is commonly used in the bi-level optimization based applications, including meta-learning [44], hyperparameter optimization

[98], generative adversarial networks [108], and neural architecture search [95], as it simplifies the hypergradient calculation and makes the bi-level optimization formulation practical for large-scale network learning. As described above, the one-step unroll learning paradigm restricts the inner-loop optimization with only one step training. Differentiating through the inner learning procedure with one step $w^*(\alpha) = w - \gamma \nabla_w \mathcal{L}_1$, and obtaining $\frac{\partial w^*(\alpha)}{\partial \alpha} = -\gamma \frac{\partial^2 \mathcal{L}_1}{\partial \alpha \partial w}$, DARTS calculates the hypergradient as:

$$\nabla_\alpha \mathcal{L}_2^{DARTS} = \frac{\partial \mathcal{L}_2}{\partial \alpha} - \gamma \frac{\partial \mathcal{L}_2}{\partial w} \frac{\partial^2 \mathcal{L}_1}{\partial \alpha \partial w}, \tag{6.3}$$

where γ is the inner-loop learning rate for w.

6.2.2 Reverse-mode Back-propagation.

Another direction of computing hypergradient is the reverse-mode [45, 125], which trains the inner-loop with enough steps to reach the optimal points for the inner optimization. This paradigm assumes T-step is large enough to adapt $w(\alpha)$ to $w^*(\alpha)$ in the inner-loop. Defining Φ as a step of inner optimization that $w_t(\alpha) = \Phi(w_{t-1}, \alpha)$, and defining $Z_t = \nabla_\alpha w_t(\alpha)$, we have:

$$Z_t = A_t Z_{t-1} + B_t,$$

where $A_t = \frac{\partial \Phi(w_{t-1}, \alpha)}{\partial w_{t-1}}$, and $B_t = \frac{\partial \Phi(w_{t-1}, \alpha)}{\partial \alpha}$.

Then the hypergradient of DARTS with the reverse model could be formulated as:

$$\nabla_\alpha \mathcal{L}_2^{Reverse} = \frac{\partial \mathcal{L}_2}{\partial \alpha} + \frac{\partial \mathcal{L}_2}{\partial w_T} (\sum_{t=0}^{T} B_t A_{t+1} ... A_T). \tag{6.4}$$

Although the reverse-mode bi-level optimization is easy to implement, the memory requirement linearly increases with the number of steps T [45] as it needs to store all intermediate gradients, making it impractical for deep networks. Rather than storing the gradients for all steps, a recent work [125] only uses the last K-step ($K << T$) gradients to approximate the exact hypergradient, which is called the truncated back-propagation. Based on the K-step truncated back-propagation, the hypergradient for DARTS could be described as:

$$h_{T-K} = \frac{\partial \mathcal{L}_2}{\partial \alpha} + \frac{\partial \mathcal{L}_2}{\partial w_T} Z_T = \frac{\partial \mathcal{L}_2}{\partial \alpha} + \frac{\partial \mathcal{L}_2}{\partial w_T} (\sum_{t=T-K+1}^{T} B_t A_{t+1} ... A_T). \tag{6.5}$$

The lemmas in [125] show that h_{T-K} is a sufficient descent direction for the outer-loop optimization.

Lemma 4. *[125]. For all $K \geq 1$, with T large enough and γ small enough, h_{T-K} is a sufficient descent direction that, i.e. $h_{T-K}^\top \nabla_\alpha \mathcal{L}_2 \geq \Omega(\|\nabla_\alpha \mathcal{L}_2\|^2)$.*

6.3 Differentiable Architecture Search with Stochastic Implicit Gradients

6.3.1 iDARTS: Implicit gradients differentiation.

Although h_{T-K} significantly decreases memory requirements, it still needs to store K-step gradients, making it impractical for differentiable NAS. In contrast, by utilizing the implicit function theorem (IFT), the hypergradient can be calculate without storing the intermediate gradients [10, 96]. The IFT based hypergradient for DARTS could be formulated as the following lemma.

Lemma 5. *Implicit Function Theorem: Consider $\mathscr{L}_1$, $\mathscr{L}_2$, and $w^*(\alpha)$ as defined in Eq.*(6.1)*, and with $\frac{\partial \mathscr{L}_1(w^*,\alpha)}{\partial w} = 0$, we have*

$$\nabla_\alpha \mathscr{L}_2 = \frac{\partial \mathscr{L}_2}{\partial \alpha} - \frac{\partial \mathscr{L}_2}{\partial w}\left[\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{-1} \frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w}. \tag{6.6}$$

This is also called as implicit differentiation theorem [96]. However, for a large neural network, it is hard to calculate the inverse of Hessian matrix in Eq.(6.6), and one common direction is to approximate this inverse. Compared with Eq.(6.6), the hypergradient of DARTS [95] in Eq.(6.3), which adopts the one-step unrolled differentiation, simply uses an identity to approximate the inverse $\left[\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{-1} = \gamma I$. This naive approximation is also adopted by [8, 98, 110]. In contrast, [113, 116] utilize the conjugate gradient (CG) to convert the approximation of the inverse to solving a linear system with δ-optimal solution, with applications to the hyperparameter optimization and meta-learning.

Recently, the Neumann series is introduced to approximate the inverse in the hyperparameter optimization [96] for modern and deep neural networks since it is a more stable alternative to CG and useful in stochastic settings. This paper thus adopts the Neumann series for the inverse approximation and proposes an approximated hypergradient for DARTS accordingly. A stochastic approximated hypergradient is further devised to fit with differentiable NAS and relieve the computational burden, which is called Differentiable Architecture Search with Stochastic Implicit Gradients (**iDARTS**). We theoretically show the proposed method converges in expectation to a stationary point for the differentiable architecture search with small enough learning rates. A detailed description and analysis of **iDARTS** follow in the next section.

6.3.2 Stochastic Approximations in iDARTS

As described, our **iDARTS** utilizes the Neumann series to approximate the inverse of the Hessian matrix for the hypergradient calculation in the IFT-based bi-level optimization of NAS. We further consider a stochastic setting where the Neumann approximation is computed based on minibatch samples, instead of the full dataset, enabling scalability to large datasets, similar to standard-practice in deep learning.

This section starts by analyzing the bound of the proposed hypergradient approximation, and then shows the convergence property of the proposed stochastic approximated hypergradient for differentiable NAS.

Before our analysis, we give the following common assumptions in the bi-level optimization.[1]

Assumption 3. *For the outer-loop function $\mathscr{L}_2$:*

1. *For any w and α, $\mathscr{L}_2(w,\cdot)$ and $\mathscr{L}_2(\cdot,\alpha)$ are bounded below.*
2. *For any w and α, $\mathscr{L}_2(w,\cdot)$ and $\mathscr{L}_2(\cdot,\alpha)$ are Lipschitz continuous with constants $L_2^w > 0$ and $L_2^\alpha > 0$.*
3. *For any w and α, $\nabla_w \mathscr{L}_2(w,\cdot)$ and $\nabla_\alpha \mathscr{L}_2(\cdot,\alpha)$ are Lipschitz continuous with constants $L_2^{\nabla_w} > 0$ and $L_2^{\nabla_\alpha} > 0$ with respect to w and α.*

Assumption 4. *For the inner-loop function $\mathscr{L}_1$*

1. *$\nabla_w \mathscr{L}_1$ is Lipschitz continuous with respect to w with constant $L_1^{\nabla_w} > 0$.*
2. *The function $w : \alpha \to w(\alpha)$ is Lipschitz continuous with constant $L_w > 0$, and has Lipschitz gradient with constant $L_{\nabla_\alpha w} > 0$.*
3. *$\|\nabla^2_{w\alpha}\mathscr{L}_1\|$ is bounded that $\|\nabla^2_{w\alpha}\mathscr{L}_1\| \le C_{\mathscr{L}_1^{w\alpha}}$ for some constant $C_{\mathscr{L}_1^{w\alpha}} > 0$.*

6.3.2.1 Hypergradient approximation based on Neumann Series

In this subsection, we describe how to use the Neumann series to reformulate the hypergradient in DARTS.

Lemma 6. *Neumann series [96]: With a matrix A that $\|I - A\| < 1$, $A^{-1} = \sum_{k=0}^{\infty}(I-A)^k$.*

[1]Similar assumptions are also considered in [36, 47, 51, 52].

Based on Lemma 6, the Eq. (6.6) for the IFT based DARTS is formulated by Eq. (6.7) as described in Corollary 1.

Corollary 1. *With small enough learning rate $\gamma < \frac{1}{L_1^{\nabla w}}$, the hypergradient in DARTS can be formulated as:*

$$\begin{aligned}\nabla_\alpha \mathscr{L}_2 &= \frac{\partial \mathscr{L}_2}{\partial \alpha} - \frac{\partial \mathscr{L}_2}{\partial w}\left[\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{-1} \frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w} \\ &= \frac{\partial \mathscr{L}_2}{\partial \alpha} - \gamma \frac{\partial \mathscr{L}_2}{\partial w} \sum_{j=0}^{\infty}\left[I - \gamma \frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{j} \frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w}.\end{aligned} \tag{6.7}$$

As shown in the Corollary 1, the approximated hypergradient for DARTS, denoted by $\nabla_\alpha \tilde{\mathscr{L}}_2$ could be obtained by only considering the first K terms of Neumann approximation without calculating the inverse of Hessian [96, 125] as,

$$\nabla_\alpha \tilde{\mathscr{L}}_2 = \frac{\partial \mathscr{L}_2}{\partial \alpha} - \gamma \frac{\partial \mathscr{L}_2}{\partial w} \sum_{k=0}^{K}\left[I - \gamma \frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{k} \frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w}. \tag{6.8}$$

As shown, we could observe the relationship between the proposed $\nabla_\alpha \tilde{\mathscr{L}}_2$ and the hypergradient of DARTS in Eq(6.3), which is the same as $\nabla_\alpha \tilde{\mathscr{L}}_2$ when $K = 0$. In the following theorem, we give the error bound between our approximated hypergradient $\nabla_\alpha \tilde{\mathscr{L}}_2$ and the exact hypergradient $\nabla_\alpha \mathscr{L}_2$.

Theorem 1. *Suppose the inner optimization function $\mathscr{L}_1$ is twice differentiable and is μ-strongly convex with w around $w^*(\alpha)$. The error between the approximated gradient $\nabla_\alpha \tilde{\mathscr{L}}_2$ and $\nabla_\alpha \mathscr{L}_2$ in DARTS is bounded with $\left\|\nabla_\alpha \mathscr{L}_2 - \nabla_\alpha \tilde{\mathscr{L}}_2\right\| \leqslant C_{\mathscr{L}_1^{w\alpha}} C_{\mathscr{L}_2^{w}} \frac{1}{\mu}(1-\gamma\mu)^{K+1}$.*

Theorem 1 states that the approximated hypergradient approaches to the exact hypergradient as K increases. As described, the form of $\nabla_\alpha \tilde{\mathscr{L}}_2$ is similar to the K-step truncated back-propagation in Eq. (6.5), while the memory consumption of our $\nabla_\alpha \tilde{\mathscr{L}}_2$ is only $\frac{1}{K}$ of the memory needed to compute h_{T-K}, as we only store the gradients of the final solution w^*. In the following corollary, we describe the connection between the proposed approximated hypergradient $\nabla_\alpha \tilde{\mathscr{L}}_2$ and the approximation based on the truncated back-propagation h_{T-K} [125].

Corollary 2. *When we assume w_t has converged to a stationary point w^* in the last K steps, the proposed $\nabla_\alpha \tilde{\mathscr{L}}_2$ is the same as the truncated back-propagation h_{T-K}.*

6.3.3 Stochastic Approximation of Hypergradient

The Lemma 4 and Corollary 2 show that the approximated hypergradient $\nabla_\alpha \tilde{\mathscr{L}_2}$ has the potential to be a sufficient descent direction. However, it is not easy to calculate the implicit gradients for DARTS based on Eq. (6.8) as it needs to deal with large-scale datasets in which the loss functions are large sums of error terms:

$$\mathscr{L}_2 = \frac{1}{R}\sum_{i=1}^{R} \mathscr{L}_2^i; \qquad \mathscr{L}_1 = \frac{1}{J}\sum_{j=1}^{J} \mathscr{L}_1^j,$$

where J is the number of minibatches of the training dataset $\mathscr{D}_{train}$ for the inner supernet training $\mathscr{L}_1$, and R is the number of minibatches of the validation dataset $\mathscr{D}_{val}$ for the outer architecture optimization $\mathscr{L}_2$. It is apparently challenging to calculate the gradient based on the full dataset in each step. We therefore utilize the stochastic gradient based on individual minibatches in practice. That is, we consider the following stochastic approximated hypergradient,

$$\nabla_\alpha \hat{\mathscr{L}}_2^i(w^j(\alpha),\alpha) = \frac{\partial \mathscr{L}_2^i}{\partial \alpha} - \gamma \frac{\partial \mathscr{L}_2^i}{\partial w} \sum_{k=0}^{K} \left[I - \gamma \frac{\partial^2 \mathscr{L}_1^j}{\partial w \partial w} \right]^k \frac{\partial^2 \mathscr{L}_1^j}{\partial \alpha \partial w}. \tag{6.9}$$

where $\mathscr{L}_2^i$ and $\mathscr{L}_1^j$ correspond to loss functions calculated by randomly sampled minibatches i and j from $\mathscr{D}_{val}$ and $\mathscr{D}_{train}$, respectively. This expression can be computed using the Hessian-vector product technique without explicitly computing the Hessian matrix (see Appendix B). Before analyzing the convergence of the proposed $\nabla_\alpha \hat{\mathscr{L}}_2(w^j(\alpha),\alpha)$, we give the following lemma to show function $\mathscr{L}_2 : \alpha \to \mathscr{L}_2(w,\alpha)$ is differentiable with a Lipschitz continuous gradient [36].

Lemma 7. *Based on the Assumption 3 and 4, we have the function $\mathscr{L}_2 : \alpha \to \mathscr{L}_2(w,\alpha)$ is differentiable with Lipschitz continuous gradient and Lipschitz constant $L_{\nabla_\alpha \mathscr{L}_2} = L_2^{\nabla_\alpha} + L_2^{\nabla_w} L_w^2 + L_2^w L_{\nabla_\alpha w}$.*

Then we state and prove the main convergence theorem for the proposed stochastic approximated hypergradient $\nabla_\alpha \hat{\mathscr{L}}_2^i(w^j(\alpha),\alpha)$ for the differentiable NAS.

Theorem 2. *Based on several assumptions, we could prove the convergence of the proposed stochastic approximated hypergradient for differentiable NAS. Suppose that:*

1. *All assumptions in Assumption 3 and 4 and Corollary 1 are satisfied;*
2. *$\exists D > 0$ such that $E\left[\|\varepsilon\|^2\right] \leq D \|\nabla_\alpha \mathscr{L}_2\|^2$;*

Algorithm 6 iDARTS

```
Input: $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$. Initialized supernet weights $w$ and operations magnitude $\alpha_\theta$.
1: while not converged do
2:     ⋆ Sample batches from $\mathcal{D}_{train}$. Update supernet weights $w$ based on cross-entropy loss with $T$ steps.
3:     ⋆ Get the Hessian matrix $\frac{\partial^2 \mathcal{L}_1}{\partial w \partial w}$.
4:     ⋆ Sample batch from $\mathcal{D}_{val}$. Calculate hypergradient $\nabla_\alpha \hat{\mathcal{L}}_2^i(w^j(\alpha), \alpha)$ based on Eq.(6.9), and update $\alpha$ with $\alpha \leftarrow \alpha - \gamma_\alpha \nabla_\alpha \hat{\mathcal{L}}_2^i(w^j(\alpha), \alpha)$.
5: end while
6: Obtain $\alpha^*$ through argmax.
```

3. *$\forall i > 0$, γ_{α_i} satisfies $\sum_{i=1}^{\infty} \gamma_{\alpha_i} = \infty$ and $\sum_{i=1}^{\infty} \gamma_{\alpha_i}^2 < \infty$.*

4. *The inner function $\mathcal{L}_1$ has the special structure: $\mathcal{L}_1^j(w, \alpha) = h(w, \alpha) + h_j(w, \alpha)$, $\forall j \in 1, ..., J$, that h_j is a linear function with respect to w and α.*

With small enough learning rate γ_α for the architecture optimization, the proposed stochastic hypergradient based algorithm converges in expectation to a stationary point, i.e. $\lim_{m \to \infty} E\left[\|\nabla_\alpha \hat{\mathcal{L}}_2^i(w^j(\alpha_m), \alpha_m)\|\right] = 0.$

The ε is defined as the noise term between the stochastic gradient $\nabla_\alpha \mathcal{L}_2^i(w^j(\alpha), \alpha)$ and the true gradient $\nabla_\alpha \mathcal{L}_2$ as:

$$\varepsilon_{i,j} = \nabla_\alpha \mathcal{L}_2 - \nabla_\alpha \mathcal{L}_2^i(w^j(\alpha), \alpha).$$

where $\nabla_\alpha \mathcal{L}_2^i(w^j(\alpha), \alpha)$ is the non-approximate version of Eq. (6.9) when $K \to \infty$.

Theorem 2 shows that the proposed stochastic approximated hypergradient is also a sufficient descent direction, which leads the differentiable NAS converges to a stationary point. The conditions 2-4 in Theorem 2 are common assumptions in analyzing the stochastic bi-level gradient methods [36, 47, 51, 52]. We assume that $\mathcal{L}_1$ in Eq. (6.7) is μ-strongly convex with w around w^*, which can be made possible by appropriate choice of learning rates [116, 125]. Another key assumption in our convergence analysis is the Lipshitz differentiable assumptions for $\mathcal{L}_1$ and $\mathcal{L}_2$ in Assumption 3 and 4, which also received considerable attention in recent optimization and deep learning literature [51, 68, 96, 101, 116].

6.3.4 Differentiable Architecture Search with Stochastic Implicit Gradients

In this subsection, we give a whole picture of the proposed iDARTS. Different from DARTS that alternatively optimizes both α and w with only one step in each round, iDARTS is

Table 6.1: Comparison results with NAS baselines on NAS-Bench-201.

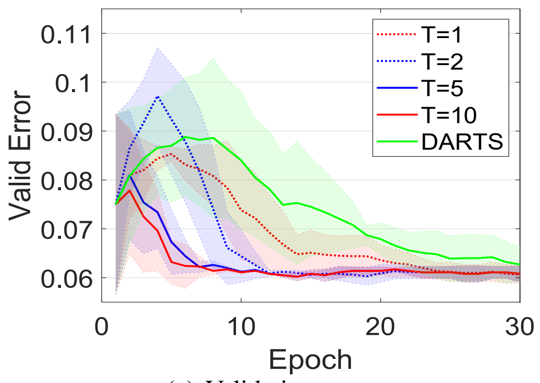
Method	CIFAR-10		CIFAR-100		ImageNet-16-120	
	Valid(%)	Test(%)	Valid(%)	Test(%)	Valid(%)	Test(%)
ENAS [114]	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
RandomNAS [86]	80.42±3.58	84.07±3.61	52.12±5.55	52.31±5.77	27.22±3.24	26.28±3.09
SETN [42]	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
GDAS [40]	89.88±0.33	93.40±0.49	70.95±0.78	70.33±0.87	41.28±0.46	41.47±0.21
DARTS [95]	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
iDARTS	89.86±0.60	93.58±0.32	70.57±0.24	70.83±0.48	40.38±0.593	40.89±0.68
optimal	91.61	94.37	74.49	73.51	46.77	47.31

iDARTS's best single run achieves **93.76%**, **71.11%**, and **41.44%** test accuracy on CIFAR-10, CIFAR-100, and ImageNet, respectively.

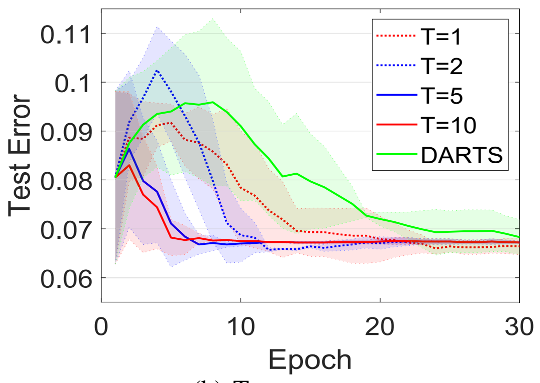
supposed to train the supernet with enough steps to make sure the $w(\alpha)$ is near $w^*(\alpha)$ before optimizing α. As described in Sec.6.3.1, we utilize the implicit function theorem to formulate the hypergradient, where the inverse Hessian matrix $\left[\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{-1}$ is approximated through the Neumann series, and the approximated hypergradient is called as implicit gradients. In the deep learning community, it is apparently challenging to calculate the gradient based on the full dataset in each step, and we therefore utilize the stochastic gradient based on individual minibatches in practice. In this way, the final approximated hypergradient in our iDARTS is termed as stochastic implicit gradients. The framework of our iDARTS is sketched in Algorithm 1. Generally, it is impossible to consider a very large T for each round of supernet weights w optimization, as the computational cost increase linear with T. Fortunately, empirical experiments show that, with the weight sharing, the differentiable NAS can adapt $w(\alpha)$ to $w^*(\alpha)$ with a small T in the later phase of architecture search.

6.4 Experimental Result

In Section 6.3, we have theoretically shown that our iDARTS can asymptotically compute the exact hypergradient and lead to a convergence in expectation to a stationary point for the architecture optimization. In this section, we conduct a series of experiments to verify whether the iDARTS leads to better results in the differentiable NAS with realistic settings. We consider three different cases to analyze iDARTS, including two NAS benchmark datasets, NAS-Bench-1Shot1 [154] and NAS-Bench-201 [41], and the common DARTS search space [95]. We first analyze the iDARTS on the two NAS benchmark datasets, along with discussions of hyperparameter settings. Then we compare iDARTS with state-of-the-art NAS methods on



(a) Validation error



(b) Test errors

Figure 6.1: Validation and test errors of iDARTS with different T and DARTS on the search space 3 of NAS-Bench-1Shot1.

the common DARTS search space.

6.4.1 Reproducible Comparison on NAS-Bench-1Shot1

To study the empirical performance of iDARTS, we run the iDARTS on the NAS-Bench-1Shot1 dataset with different *random seeds* to report its statistical results, and compare with the most closely related baseline DARTS [95]. The NAS-Bench-1Shot1 is built from the NAS-Bench-101 benchmark dataset [150], through dividing all architectures in NAS-Bench-101 into 3 different unified cell-based search spaces. The architectures in each search space have the same number of nodes and connections, making the differentiable NAS could be directly applied to each search space. The three search spaces contain 6240, 29160, and 363648 architectures with the CIFAR-10 performance, respectively. We choose the third search space in NAS-Bench-1Shot1 to analyse iDARTS, since it is much more complicated than the remaining two search spaces and is a better case to identify the advantages of iDARTS.

Figure 6.1 plots the mean and standard deviation of the validation and test errors for iDARTS and DARTS, with tracking the performance during the architecture search on the NAS-Bench-1Shot1 dataset. As shown, our iDARTS with different T generally outperforms DARTS during the architecture search in terms of both validation and test error. More specifically, our iDARTS significantly outperforms the baseline in the early stage, demonstrating that our iDARTS could quickly find superior architectures and is more stable.

As described, one significant difference from DARTS is that iDARTS can conduct more than one training step in the inner-loop optimization. Figure 6.1 also analyzes the effects of the inner optimization steps T, plotting the performance of iDARTS with different T on the

NAS-Bench-1Shot1. As shown, the inner optimization steps positively affect the performance of iDARTS, where increasing T helps iDARTS converge to excellent solutions faster. One underlying reason is that increasing T could adapt w to a local optimal w^*, thus helping iDRTS approximate the exact hypergradient more accurately. We should notice that the computational cost of iDARTS also increases with T, and our empirical findings suggest a $T = 5$ achieves an excellent compute and performance trade-off for iDARTS on NAS-Bench-1shot1. More interesting, iDARTS with $T = 1$ is similar as DARTS which both conduct the inner optimization with only one step, with the difference that iDARTS adopts the Neumann approximation while DARTS considers the unrolled differentiation. We could observe that iDARTS still outperforms DARTS by large margins in this case, showing the superiority of the proposed approximation over DARTS.

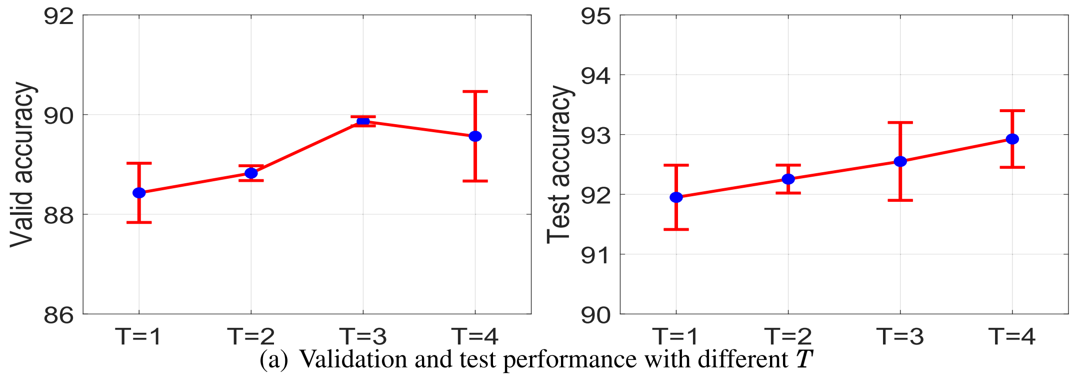
6.4.2 Reproducible Comparison on NAS-Bench-201

The NAS-Bench-201 dataset [41] is another popular NAS benchmark dataset to analyze differentiable NAS methods. The search space in NAS-Bench-201 contains four nodes with five associated operations, resulting in 15,625 cell candidates. The search space of NAS-Bench-201 is much simpler than NAS-Bench-1Shot1, while it contains the performance of CIFAR-100, CIFAR-100, and ImageNet for all architectures in this search space.

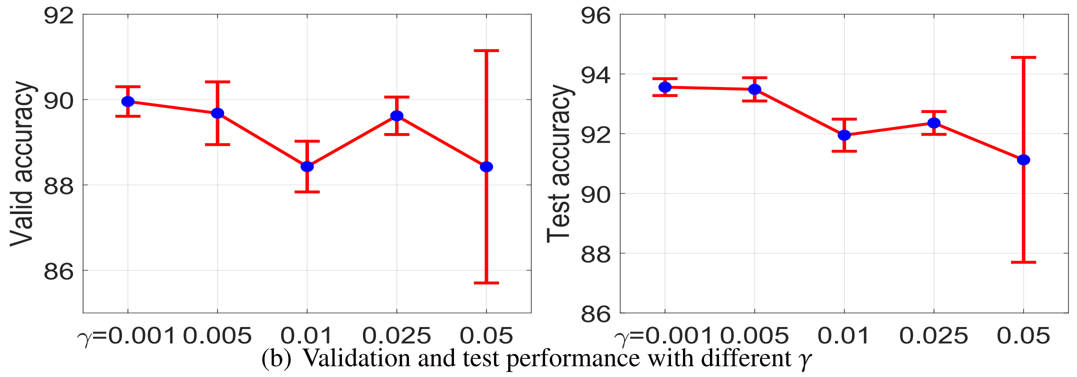
Table 6.1 summarizes the performance of iDARTS on NAS-Bench-201 compared with differentiable NAS baselines, where the statistical results are obtained from independent search experiments with different *random seeds*. As shown, our iDARTS achieved excellent results on the NAS-Bench-201 benchmark and significantly outperformed the DARTS baseline, with a 93.76%, 71.11%, and 41.44% test accuracy on CIFAR-10, CIFAR-100, and ImageNet, respectively. As described above, iDARTS is built based on the DARTS framework, with only reformulating the hypergradient calculation. These results in Table 6.1 verified the effectiveness of our iDARTS, which outperforms DARTS by large margins.

Similar to the experiments in the NAS-Bench-1Shot1, we also analyze the importance of hyperparameter T in the NAS-Bench-201 dataset. Figure 6.2 (a) summaries the performance of iDARTS with different number of inner optimization steps T on the NAS-Bench-201. As demonstrated, the performance of iDARTS is sensitive to the hyperparameter T, and a larger T helps iDARTS to achieve better results while also increases the computational time, which is also in line with the finding in the NAS-Bench-1Shot1. We empirically find that $T = 4$ is enough to achieve competitive results on NAS-Bench-201.

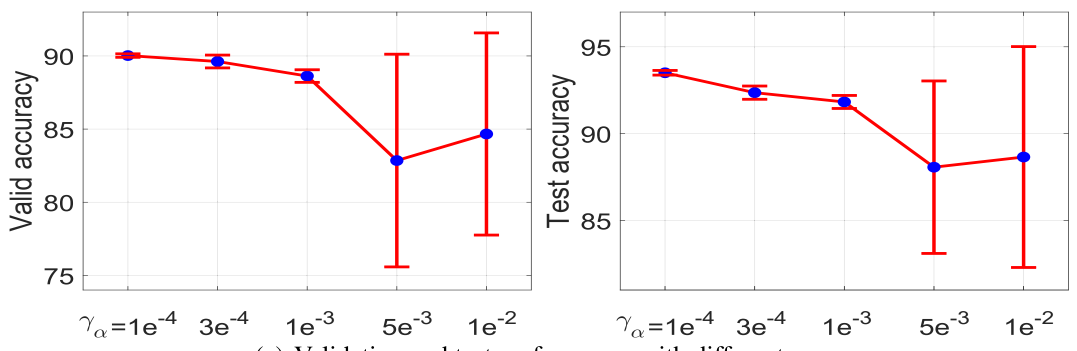
The hypergradient calculation of iDARTS is based on the Neumann approximation in Eq.(6.7), and one underlying condition is that the learning rates γ for the inner optimization



(a) Validation and test performance with different T



(b) Validation and test performance with different γ



(c) Validation and test performance with different γ_α

Figure 6.2: Hyperparameter analysis of iDARTS on the NAS-Bench-201 benchmark dataset.

should be small enough to make $\left\|I-\gamma\frac{\partial^2\mathscr{L}_1}{\partial w\partial w}\right\|<1$. We also conduct an ablation study to analyze how this hyperparameter affects our iDARTS, where Figure 6.2 (b) plots the performance of iDARTS with different learning rates γ for the inner optimization on the NAS-Bench-201. As shown, the performance of iDARTS is sensitive to γ, and a smaller γ is preferred, which also offers support for the Corollary 1 and Theorem 1.

During the analysis of the convergence of iDARTS, the learning rate γ_α plays a key role in the hypergradient approximation for the architecture optimization. Figure 6.2 (c) also summaries the performance of iDARTS with different initial learning rate γ_α on the NAS-Bench-201. As

Table 6.2: Comparison results with state-of-the-art weight-sharing NAS approaches.

Method	Test Error (%)			Param	+×	Architecture
	CIFAR-10	CIFAR-100	ImageNet	(M)	(M)	Optimization
NASNet-A [172]	2.65	17.81	26.0 / 8.4	3.3	564	RL
PNAS [93]	3.41±0.09	17.63	25.8 / 8.1	3.2	588	SMBO
AmoebaNet-A [118]	3.34±0.06	-	25.5 / 8.0	3.2	555	EA
ENAS [114]	2.89	18.91	-	4.6	-	RL
EN^{2}AS [162]	2.61±0.06	16.45	26.7 / 8.9	3.1	506	EA
RandomNAS [86]	2.85±0.08	17.63	27.1	4.3	613	random
NSAS [158]	2.59±0.06	17.56	25.5 / 8.2	3.1	506	random
PARSEC [21]	2.86±0.06	-	26.3	3.6	509	gradient
SNAS [140]	2.85±0.02	20.09	27.3 / 9.2	2.8	474	gradient
SETN [42]	2.69	17.25	25.7 / 8.0	4.6	610	gradient
MdeNAS [169]	2.55	17.61	25.5 / 7.9	3.6	506	gradient
GDAS [40]	2.93	18.38	26.0 / 8.5	3.4	545	gradient
XNAS* [109]	2.57±0.09	16.34	24.7 / 7.5	3.7	600	gradient
PDARTS [25]	2.50	16.63	24.4 / 7.4	3.4	557	gradient
PC-DARTS [142]	2.57±0.07	17.11	25.1 / 7.8	3.6	586	gradient
DrNAS [24]	2.54±0.03	16.30	24.2 / 7.3	4.0	644	gradient
DARTS [95]	2.76±0.09	17.54	26.9 / 8.7	3.4	574	gradient
iDARTS	**2.37±0.03**	**16.02**	**24.3 / 7.3**	3.8	595	gradient

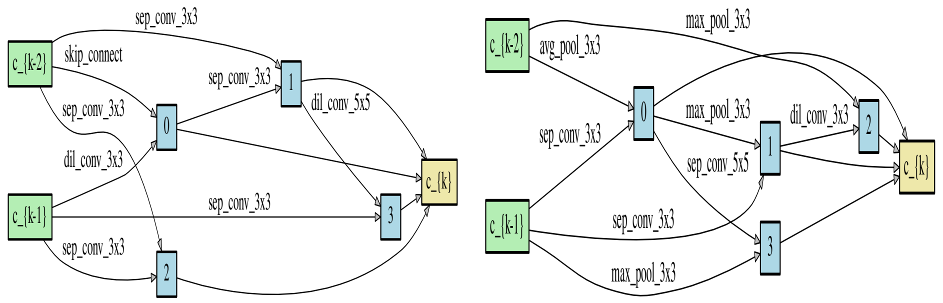
“*" indicates the results reproduced based on the best-reported cell structures with a common experimental setting [95]. “Param" is the model size when applied on CIFAR-10, while “+×" is calculated based on the ImageNet dataset.

shown in Figure 6.2 (c), the performance of iDARTS is sensitive to γ_α, where a smaller γ_α is recommended, and a large γ_α is hardly able to converge to a stationary point. An underlying reason may lay in the proof of Theorem 2, that choosing a small enough γ_α guarantees that the iDARTS converges to a stationary point.

6.4.3 Experiments on DARTS Search Space

We also apply iDARTS to a convolutional architecture search in the common DARTS search space [95] to compare with the state-of-the-art NAS methods, where all experiment settings are following DARTS for fair comparisons. The search procedure needs to look for two different types of cells on CIFAR-10: normal cell α_{normal} and reduction cell α_{reduce}, to stack more cells to form the final structure for the architecture evaluation. The best-found cell on CIFAR-10 is then transferred to CIFAR-100 and ImageNet datasets to evaluate its transferability.

The comparison results with the state-of-the-art NAS methods are presented in Table 6.2,



(a) normal cell (b) reduction cell

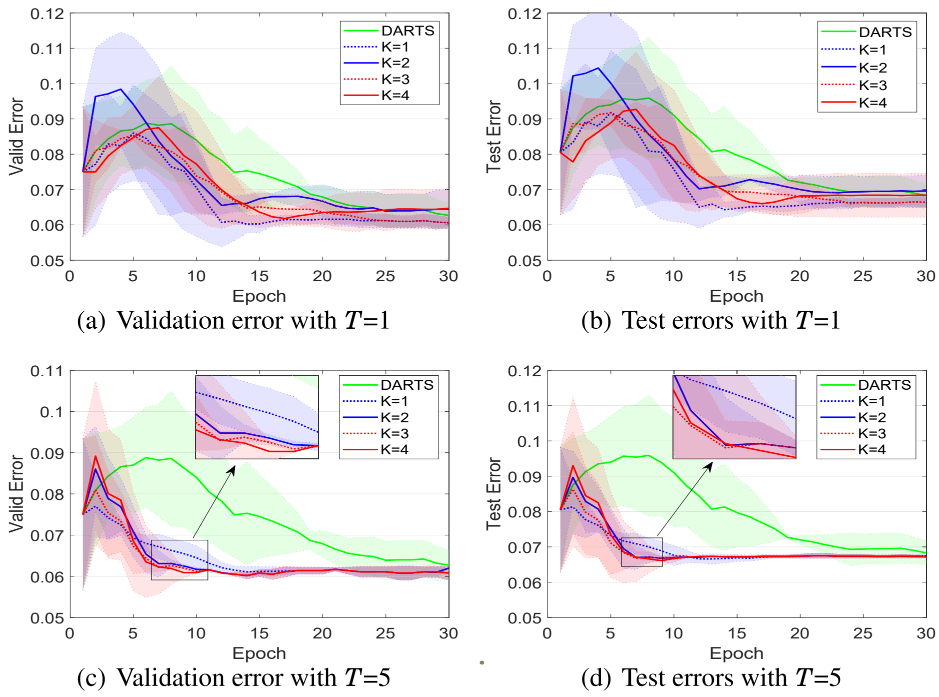
Figure 6.3: The best cells discovered by iDARTS on the DARTS search space.

and Figure 6.3 demonstrates the best-found architectures by iDARTS. As shown in Table 6.2, iDARTS achieves a 2.37±0.03 % test error on CIFAR-10 (where the best single run is 2.35%), which is on par with the state-of-the-art NAS methods and outperforms the DARTS baseline by a large margin, again verifying the effectiveness of the proposed method.

Following DARTS experimental setting, the best-searched architectures on CIFAR-10 are then transferred to CIFAR-100 and ImageNet to evaluate the transferability. The evaluation setting for CIFAR-100 is the same as CIFAR-10. In the ImageNet dataset, the experiment setting is slightly different from CIFAR-10 in that only 14 cells are stacked, and the number of initial channels is changed to 48. We also follow the mobile setting in [95] to restrict the number of multiply-add operations ("$+\times$") to be less than 600M on the ImageNet. The comparison results with state-of-the-art differentiable NAS approaches on CIFAR-100 and ImageNet are demonstrated in Table 6.2. As shown, iDARTS deliveres a competitive result with 16.02% test error on the CIFAR-100 dataset, which is a state-of-the-art performance and outperforms peer algorithms by a large margin. On the ImageNet dataset, the best-discovered architecture by our iDARTS also achieves a competitive result with 24.4 / 7.3 % top1 / top5 test error, outperforming or on par with all peer algorithms. Please note that, although DrNAS achieved outstanding performance on ImageNet, the number of multiply-add operations of its searched model is much over 600 M, violating the mobile-setting.

6.4.4 Ablation study on the number of approximation terms

As we described before, there are two additional hyperparameters in our practical iDARTS, the inner optimization steps T and the number of terms for the approximation in Eq.(6.8). We have analyzed T in previous experiments. In this section, we analyze another hyperparameter K on the NAS-Bench-1Shot1 benchmark dataset. In the first experiment, we set a default hyperparameter $T = 1$ the same as DARTS for the inner supernet training to remove the bias



(a) Validation error with T=1 (b) Test errors with T=1
(c) Validation error with T=5 (d) Test errors with T=5

Figure 6.4: Ablation study on K for iDARTS with $T = 1$ and $T = 5$ on NAS-Bench-1Shot1.

from T. From Eq.(6.8) and (6.3), we could further find that the hypergradient calculation in our iDARTS with $T = 1$ and $K = 0$ is the same as DARTS. Figure 6.4 (a) (b) plots the performance of iDARTS with different K on the NAS-Bench-1Shot1. As shown, our iDARTS is very robust to K with limited training steps $T = 1$, where iDARTS with different K all outperform the DARTS baseline with the same inner training steps $T = 1$, showing the superiority of the proposed approximation over DARTS. Another interesting finding is that, our iDARTS with $K = 1$ and $T = 1$ even achieve slightly more competitive results than $K > 1$. An underlying reason is that, when the inner training step is too small, it is hard to achieve the local optimal w^* and the corresponding hypergradient is not accurate.

To further investigate the effectiveness of the proposed approximation, we consider setting enough inner training steps with $T = 5$, and Figure 6.4 (c) (d) plots the performance of iDARTS with different K on the NAS-Bench-1Shot1 under $T = 5$. The first impression from Figure 6.4 is that increasing inner training steps could significantly improve the performance, where all cases with $T = 5$ generally outperform $T = 1$. Another interesting finding is that, with enough inner training steps, the number of approximation terms K has a positive impact on the performance of iDARTS. As shown in Figure 6.4 (c) (d), increasing K also helps iDARTS converge to excellent solutions faster, verifying that the proposed $\nabla_\alpha \hat{\mathcal{L}}_2^i$ could asymptotically approach to the exact hypergradient $\nabla_\alpha \mathcal{L}_2^i$ with the increase of approximation term K. Besides,

Table 6.3: Ablation study on K for iDARTS with on NAS-Bench-201.

Method	CIFAR-10		CIFAR-100		ImageNet-16-120	
	Valid(%)	Test(%)	Valid(%)	Test(%)	Valid(%)	Test(%)
DARTS($T = 1, K = 0$)	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
iDARTS($T = 1, K = 1$)	86.85±0.93	89.67±1.31	64.09±2.92	64.17±3.26	36.26±5.71	36.11± 5.77
iDARTS($T = 4, K = 0$)	87.31±1.33	90.36±1.79	64.76±2.54	64.43±2.47	32.53±1.31	32.42±1.54
iDARTS($T = 4, K = 1$)	89.30±1.47	92.44±1.14	67.88±1.86	68.17±2.81	37.11±7.79	36.61±7.47
iDARTS($T = 4, K = 2$)	89.86±0.60	93.58±0.32	70.57±0.24	70.83±0.48	40.38±0.59	40.89±0.68
iDARTS($T = 4, K = 3$)	89.35±0.03	92.29±0.26	68.51±0.77	68.58±1.18	42.37±0.48	42.26±0.41

we can find that, $K = 2$ is large enough to result in competitive performance for our iDARTS on NAS-Bench-1shot1, which results in similar performance as $K \geq 3$.

We also conduct an ablation study on NAS-Bench-201 dataset to analyse the hyperparameter K, and Table 6.3 summarizes the performance of iDARTS on NAS-Bench-201 with a different number of approximation term K. The results in Table 6.3 are similar to those on the NAS-Bench-1Shot1 dataset, also showing that K has a positive impact on the performance of iDARTS. Firstly, we can find that, with the same inner training steps $T = 1$ as DARTS baseline, our iDARTS ($T = 1, K = 1$) with one approximation term outperform DARTS by large margins in this case, verifying the superiority of the proposed approximation over DARTS. Secondly, the results in Table 6.3 also demonstrate that considering more approximation terms does indeed help improve our iDARTS to a certain degree. With enough inner training steps, the performance of iDARTS increases with K from 0 to 2. Another interesting finding is that the performance of iDARTS does not always increase with the K, and there is a decrease for $K \geq 3$. One underlying reason may be that, the iDARTS with smaller K brings more noises into the hypergradient, which in turn enhances the exploration. Several recent works [23, 158] show the importance of the exploration in the differentiable NAS, where adding more noises into the hypergradient could improve the performance. Our experimental results suggest that a $K = 2$ achieves an excellent trade-off between the accuracy of hypergradient and the exploration, thus achieving the competitive performance on the NAS-Bench-201 dataset.

6.5 Chapter Summary and Discussion

Although *Differentiable ARchiTecture Search* (DARTS) has recently become the mainstream of neural architecture search (NAS) due to its efficiency and simplicity, DARTS adopts a lot of inappropriate approximations in the estimation of architecture gradient, a.k.a. hypergradient. With a gradient-based bi-level optimization, DARTS alternately optimizes the inner model weights and the outer architecture parameter in a weight-sharing supernet. A key challenge to the scalability and quality of the learned architectures is the need for differentiating through the inner-loop optimisation. While much has been discussed about several potentially fatal factors in DARTS, the hypergradient has received less attention.

This chapter opens up a promising research direction for NAS by focusing on the hypergradient approximation in the differentiable NAS. We introduced the implicit function theorem (IFT) to reformulate the hypergradient calculation in the differentiable NAS, making it practical with numerous inner optimization steps. To avoid calculating the inverse of the Hessian matrix, we utilized the Neumann series to approximate the inverse.

With utilizing the implicit function theorem, we can tackle the hypergradient computation in DARTS only depends on the obtained solution to the inner-loop optimization and agnostic to the optimization path. To further reduce the computational requirements, a stochastic hypergradient approximation is formulated for differentiable NAS, and we theoretically show that the architecture optimization with the proposed method, named iDARTS, is expected to converge to a stationary point when applied to a differentiable NAS. We based our framework on DARTS and performed extensive experimental results to verify the proposed framework's effectiveness. Comprehensive experiments on two NAS benchmark search spaces and the common NAS search space verify the effectiveness of our proposed method. It leads to architectures outperforming, with large margins, those learned by the baseline methods. While we only considered the proposed stochastic approximated hypergradient for differentiable NAS, iDARTS can in principle be used with a variety of bi-level optimization applications, including in meta-learning and hyperparameter optimization, opening up several interesting avenues for future research.

Chapter 7

Conclusions and Future Work

In this chapter, we first summarize the entire thesis, then we discuss the limitations of this tehsis and provide some possible directions for future research.

7.1 Summary of Thesis

Automated Deep Learning (AutoDL) aims to build a better deep learning model in a data-driven and automated manner, compensating for the lack of deep learning experts and lowering the threshold of various areas of deep learning to help all the amateurs to use deep learning without any hassle. This thesis focuses on the neural architecture search (NAS) in the AutoDL, with addressing several issues in one-shot NAS and differentiable NAS.

Specifically, in Chapter 2, we originally focus on resolving the “rich-get-richer" problem in supernet training for the weight-sharing neural architecture search, where a novelty search is proposed to enhance the exploration for architecture sampling during the supernet training. In particular, a novelty search mechanism is developed to efficiently find the most abnormal architecture, and the single-path model is adopted to greatly reduce computational and memory demand. Experimental results show the proposed approach could find the state-of-the-art or competitive CNN and RNN models, and also improve the predictive ability of the supernet in one-shot NAS. Rather than considering the validation performance-based indicator for architecture sampling, this chapter simply devises a diversity-based indicator to sample diversified architectures for the supernet training in the one-shot NAS. As we can observe from the experimental results, the proposed architecture sampling method obtains to a better predictive ability for the supernet in one-shot NAS, showing in the Sec.2.4.3. The underline reason is that, the performance-based indicator usually leads to the “rich-get-richer" problem while the diversity-based indicator has the potential to fairly train the supernet. This phenomenon suggests that the validation performance by inheriting the weights from the supernet is deceptive, and devising a more effective indicator for the architecture sampling or selection is a promising research direction, which is also in line with several concurrent works [22, 105, 166].

In Chapter 3, we formulate supernet training as a constrained optimization problem to reduce some of the negative impacts of catastrophic forgetting with one-shot NAS, and multi-model forgetting in particular. Our strategy is to select a representative subset of constraints with a greedy novelty search method. Then the supernet training is regularized in a feasible region with a new novelty search-based architecture selection loss function, i.e., **NSAS** to overcome multi-model forgetting. We implemented **NSAS** into two one-shot NAS baselines - RandomNAS and GDAS - and compared the quality of the architecture selections with and without the new loss function. The results of experiments on the common search space of a neural architecture and the NAS-Bench-201 dataset show **NSAS** and two of its variants improve the predictive ability of the supernet with both convolutional and recurrent cell search. Catastrophic forgetting, which usually exists in the online learning, is also introduced into neural architecture search in this chapter, since the supernet is trained under different paths. Simply adding a regularization term on the supernet training can greatly improve the predictive ability of the supernet of different NAS frameworks, as analyzed in Lemma 1 and observed from the experimental results. It is clear that the NSAS loss function can easily increase the predictive ability of the supernet, which, in turn, greatly improves the performance of the architectures found by RandomNAS and GDAS. However, supernet training in one-shot NAS is still a problem with much room for further advancements. Devising a more appropriate loss function than the status quo appears to be a promising direction for improving the performance of one-shot NAS methods.

Chapter 4 provides a means of efficiently, yet intelligently, searching for differentiable neural architectures in a latent continuous space. A variational graph autoencoder transforms discrete architectures into an equivalent continuous space by injective means, while a probabilistic exploration enhancement method encourages intelligent exploration of the search space during supernet training. In addition, the framework includes a novel architecture complementation loss function to relieve catastrophic forgetting, along with a theoretical demonstration that the proposed loss function provides an identical result to the currently-accepted best-practice methods but is easier to calculate. Experiments on a NAS benchmark dataset and the common DARTS convolutional search space show the effectiveness of the proposed framework. As we known, differentiable NAS methods transform the discrete architecture search problem into a continuous optimization problem, while most of them hardly guarantee the optimization in the latent space equals to the discrete space. The final discretization stage of DARTS is still an open and unsolved problem, since the validation performance of a continuous representation architecture can not exactly indicate the performance of a discrete one. There are several recent works also try to relieve this issue, e.g. perturbation-based architecture selection[135]. This chapter points out another promising direction, transforming discrete architectures into an equivalent continuous space by injective means, rather than the commonly used *softmax-argmax*.

In Chapter 5, we formulate the architecture optimization in the differentiable NAS as a distribution learning problem and introduces a Bayesian learning rule to optimize the architecture parameters posterior distributions. We theoretically demonstrate that the proposed framework can enhance the exploration for differentiable NAS and implicitly impose regularization on the norm of Hessian matrix to improve the stability. We operationalize the framework based on the common differentiable NAS baseline, DARTS, and experimental results on NAS benchmark datasets have verified the proposed framework's effectiveness. To further improve the transferability, we have proposed a depth regularization methods to encourage the depth of the searched cells. Experimental results on the typical DARTS search space validate the advantages of this regularization method. Although Differentiable Architecture Search (DARTS) has received massive attention in recent years, mainly because it significantly reduces the computational cost through weight sharing and continuous relaxation, more recent works find that existing differentiable NAS techniques struggle to outperform naive baselines, yielding deteriorative architectures as the search proceeds. Rather than directly optimizing the architecture parameters, this chapter formulates the neural architecture search as a distribution learning problem through relaxing the architecture weights into Gaussian distributions. By leveraging the natural-gradient variational inference (NGVI), the architecture distribution can be easily optimized based on existing codebases without incurring more memory and computational consumption. This chapter opens up a new direction for the differentiable neural architecture search, considering architecture search as a distribution learning problem rather than magnitude optimization, since this reformulation can naturally enhance exploration and improve stability.

Chapter 6 introduces the implicit function theorem (IFT) to reformulate the hypergradient calculation in the differentiable NAS, making it practical with numerous inner optimization steps. To avoid calculating the inverse of the Hessian matrix, we utilized the Neumann series to approximate the inverse, and further devised a stochastic approximated hypergradient to relieve the computational cost. We have theoretically analyzed the convergence and proved that the proposed method, called iDARTS, is expected to converge to a stationary point when applied to a differentiable NAS. This chapter opens up a promising research direction for NAS by focusing on the hypergradient approximation in the differentiable NAS. We introduced the implicit function theorem (IFT) to reformulate the hypergradient calculation in the differentiable NAS, making it practical with numerous inner optimization steps. We based our framework on DARTS and performed extensive experimental results to verify the proposed framework's effectiveness. Comprehensive experiments on two NAS benchmark search spaces and the common NAS search space verify the effectiveness of our proposed method. It leads to architectures outperforming, with large margins, those learned by the baseline methods. While we only considered the proposed stochastic approximated hypergradient for differentiable NAS, iDARTS can in principle be used with a variety of bi-level optimization applications, including in meta-learning and hyperparameter optimization, opening up several interesting avenues for future research.

Generally, the thesis involves two popular paradigms in NAS, one-shot NAS and differentiable NAS, where one-shot NAS performs architecture search in a discrete space while differentiable NAS in a continuous space. In particular, we proposed a novelty driven sampling method and formulate the supernet training as a constrained continual learning optimization problem, to address the “rich-get-richer" problem and multi-model forgetting issue existing in one-shot NAS. As to the differentiable NAS, we leverage a variational graph autoencoder to relieve the non-negligible incongruence, formulating the neural architecture search as a distribution learning problem to enhance exploration, and propose the differentiable architecture search with stochastic implicit gradients to enable multi-step inner optimization. Despite the two paradigms work in different spaces, they also share several common challenges. As we described in the first part of this thesis, “rich-get-richer" and catastrophic forgetting are two critical issues in one-shot NAS, while they also exist in the differentiable NAS where the techniques proposed in the second part could also alleviate them. For example, the EN^{2}AS proposed in Chapter 2 utilized novelty search to enhance the exploration, while the BaLeNAS proposed in Chapter 5 reformulated the architecture search into distribution learning which naturally enhances exploration. Chapter 3 targeted on the catastrophic forgetting in one-shot NAS and devises the NSAS loss function to overcome multi-model forgetting. More interesting, the experimental results in Chapter 4 also showed that relieve the catastrophic forgetting in differential NAS through an architecture complementation strategy could also improve the performance. Rather than only focusing on the neural architecture search, Chapter 4 tried to reveal some critical issues in the mathematic bilevel-optimization formulation of the differentiable NAS, where Chapter 4 therefore proposed the an implicit function theorem based strategy, iDARTS, to calculate the hypergradient more accurately. More importantly, apart from neural architecture search, iDARTS can in principle be used with a variety of bi-level optimization applications, including in meta-learning and hyperparameter optimization, opening up several interesting avenues for future research.

7.2 Limitations of Thesis and Future Work

The advancements of AutoDL and NAS in recent years make more people believe the AutoDL system can beat experts in designing deep neural networks, and these advancements also raise concerns about AutoDL hubris. Although recent several years witness the rapid development of neural architecture search, NAS is still in the preliminary research stage, and it is still very challenging to use it for practical applications, compared with the rapid development and numerous application of deep neural network designed by human experts. In addition to our study in this thesis, there are several possible pursuits on the neural architecture search. In the following, we will describe several limitations in existing NAS community with the discussion of possible research directions on the neural architecture search that we can make breakthroughs in the future.

NAS with Human Knowledge. Existing NAS methods select neural networks only based on the validation performance, making it hard to guarantee the searched architectures with excellent transferability. Expert knowledge plays a key role in the manual-designed neural networks, which determines goodness of architecture rather only based on the performance. More important, the NAS methods could only find the best architecture in the search space at most, while it is hard to determine the potential of the search space. For example, although DARTS search space includes most human-designed neural network structures, it is hard to know whether the desired neural network structure has been included in the search space. In our future work, we will try to incorporate human knowledge into NAS for searching architectures with better performance and transferability, where we focus on how to design a better search space to help NAS methods to find more competitive architectures with efficiency.

NAS without Weight-Sharing Paradigm. In this thesis, the two NAS paradigms, one-shot NAS and differentiable NAS, are based on the weight-sharing. While weight-sharing paradigm is an efficient way to reduce the computational cost in NAS, more recent work find that this strategy usually leads a sub-optimal architecture, making it impossible to find the optimal structures for those weight-sharing paradigm based NAS methods. For example, DARTS is unable to stably obtain excellent solutions and yields deteriorative architectures with the search proceeding, performing even worse than random search in some cases. Furthermore, the validation performance by inheriting weights from the supernet could hardly indicate the performance by training from scratch. In the future work, we will focus on other paradigms to reduce the computational cost rather than weight-sharing, e.g., the neural architecture search based on the weights generation and performance prediction.

NAS without Supernet Training nor Labels. More recent works found that supernet training seems to deteriorate the performance with search progresse, and the validation performance through inheriting weights is not a good indicator for architecture search. In addition, although most NAS alternatively optimize the architecture parameters and model weights by the supervision signal, (e.g., training and validation accuracy), the recent emerged unsupervised NAS without label information [1, 83, 92, 144, 166] has been validated to achieve comparable performance to supervised NAS methods. The above observations inspire us to discard the supernet training and abandon the supervision signal in DARTS, with raising the question: can we find high-quality architectures without training nor the labels? In future research, we will focus on this interesting direction on neural architecture search.

More Applications. In Chapter 6, we only considered the proposed stochastic approximated hypergradient for differentiable NAS, while it can in principle be used with a variety of bi-level optimization applications, including meta-learning, graph structure learning, and hyperparameter optimization, opening up several interesting avenues for future research. For example, the meta-learning method can simulate the continuous learning process of experts in designing the deep learning models. By introducing the meta-information, the performance of neural architecture search can be further improved. Furthermore, rather than searching on the benchmark datasets, it will be valuable if we apply the proposed NAS methods to real-world applications, such as social media analysis, medical image processing, action recognition, and so on.

In summary, Automatic Deep Learning (AutoDL) aims to build a better deep learning model in a data-driven and automated manner, compensating for the lack of deep learning experts and lowering the threshold of various areas of deep learning to help all the amateurs to use machine learning without any hassle. AutoDL is an up-and-coming tool to take advantage of the extracted data to find the solutions automatically. These days, the emergence of neural architecture search (NAS) is an exciting development, which automatically builds Deep Neural Networks for solving various tasks, including computer vision, natural language processing, autonomous driving, and so on. However, it is worth notice that NAS is in its infancy, where more additional theoretical guidance and experimental analysis are required. Finding out which NAS designs can greatly improve the performance of NAS is still an open and critical research problem. There is still a long way to go.

APPENDIX A

APPENDIX

A.1 Proof of Lemma 1

Lemma 1 describes the relationship between WPL and the proposed NSAS loss function. Since the weights of current architecture θ_b are shared by the constraints described in **Assumption 1**, $\theta_b^{(e)} \subseteq \{\theta_1^{(e)} \cup ... \cup \theta_M^{(e)}\}$, and for every edge e in α_t, we have $\theta_b^p = \emptyset$. Further, θ_i and θ_j $(i, j = 1...M)$ are independent as the architectures are trained separately without shared weights, as described in **Assumption 2**. Now the posterior probability in the WPL can be written as:

$$\begin{aligned} p(\theta \mid \mathscr{D}) &= p(\theta_1...\theta_M, \theta_b \mid \mathscr{D}) = \frac{p(\theta_1^p...\theta_M^p, \theta_1^s...\theta_M^s, \mathscr{D})}{p(\mathscr{D})} \\ &= \frac{p(\theta_1...\theta_M, \mathscr{D})}{p(\mathscr{D})} = \frac{p(\theta_1 \mid \theta_2...\theta_M, \mathscr{D}) p(\theta_2...\theta_M, \mathscr{D})}{p(\mathscr{D})} \\ &= \prod_{i=1:M} p(\theta_i \mid \mathscr{D}) \propto \prod_{i=1:M} p(\mathscr{D} \mid \theta_i) p(\theta_i) \\ &= p(\theta) \prod_{i=1:M} p(\mathscr{D} \mid \theta_i) = p(\theta^t) \prod_{i=1:M} p(\mathscr{D} \mid \theta_i), \end{aligned} \tag{A.1}$$

where θ_i is the weights of architecture α^i in the constraint subset. As only architecture α^t is trained, $p(\theta) = p(\theta^t)$, where θ^t is the weights of the current architecture α^t, and θ is all the considered weights. Eq.(A.1) derives the posterior probability without the assumption that θ^s in the previous step is optimal. Hence, now, the WPL to optimize the posterior probability $p(\theta \mid \mathscr{D})$ can be expressed as:

$$\mathscr{L}_{WPL}(\mathscr{W}_{\mathscr{A}}(\alpha^t)) = \epsilon \mathscr{R}(\mathscr{W}_{\mathscr{A}}(\alpha^t)) + \sum_{i=1:M} \mathscr{L}_c(\mathscr{W}_{\mathscr{A}}(\alpha^i)), \tag{A.2}$$

where ϵ is the trade-off factor. And the proposed **NSAS** loss function with the WPL can be expressed as:

$$\begin{aligned}\mathcal{L}_N(\mathcal{W}_{\mathcal{A}}(\alpha^t)) &= (1-\beta)(\mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha^t)) + \lambda\mathcal{R}(\mathcal{W}_{\mathcal{A}}(\alpha^t))) \\ &+ \frac{\beta}{M}\sum_{i=1:M}(\mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha^i)) + \lambda\mathcal{R}(\mathcal{W}_{\mathcal{A}}(\alpha^i))) \\ &= \mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha^t)) + \gamma\mathcal{L}_{WPL}(\mathcal{W}_{\mathcal{A}}(\alpha^t)).\end{aligned} \tag{A.3}$$

A.2 Proof of Lemma 2

Lemma 2 operates on the assumption that the noise ξ is orthogonal to $\mathcal{T}$ and statistically independent from the manifold. Given a new data point $\bar{\alpha}$, rather than directly calculating the probability that the new points located in the distribution of the random variable A, the probability of $\bar{w}$ drawn from random variable W after coordinates rotation $W = U^\top \cdot A$ is calculated instead:

$$p_A(\bar{\alpha}) = p_W(\bar{w}) = p_{W^\parallel}(\bar{w}^\parallel)p_{W^\perp}(\bar{w}^\perp). \tag{A.4}$$

In this way, we could calculate the two parts $p_{W^\parallel}(\bar{w}^\parallel)$ and $p_{W^\perp}(\bar{w}^\perp)$ to obtain the $p_A(\bar{\alpha})$. Beginning with

$$\bar{w}^\parallel = {U^\parallel}^\top \bar{\alpha} = {U^\parallel}^\top(\bar{\alpha} - \bar{\alpha}^\parallel) + {U^\parallel}^\top \bar{\alpha}^\parallel = {U^\parallel}^\top \bar{\alpha}^\parallel, \tag{A.5}$$

assume the noise ξ orthogonal to $\mathcal{M}$, and we have $U^\parallel(\bar{\alpha} - \bar{\alpha}^\parallel) \approx 0$. Based on Eq.(4.8) and Eq.(A.5), we have

$$\begin{aligned}\bar{w}^\parallel &= {U^\parallel}^\top D(\bar{\alpha}_\theta) + {U^\parallel}^\top U^\parallel S V^\top(\alpha_\theta - \bar{\alpha}_\theta) + O(\|\alpha_\theta - \bar{\alpha}_\theta\|^2) \\ &= {U^\parallel}^\top D(\bar{\alpha}_\theta) + S V^\top(\alpha_\theta - \bar{\alpha}_\theta) + O(\|\alpha_\theta - \bar{\alpha}_\theta\|^2)\end{aligned}. \tag{A.6}$$

Then $p_{W^\parallel}(\bar{w}^\parallel) = p_{A_\theta}({U^\parallel}^\top D(\bar{\alpha}_\theta) + SV^\top(\alpha_\theta - \bar{\alpha}_\theta))$. Based on a linear transformation of the probability density, we have

$$p_{W^\parallel}(\bar{w}^\parallel) = \left|\mathtt{det}S^{-1}\right| p_{A_\theta}(\alpha_\theta) \tag{A.7}$$

since V is a unitary matrix.

The next step is to calculate the second part of Eq.(4.10), which is also a component of the noise ξ. As $p_{W^\perp}(\bar{w}^\perp)$ is in $m-n$-dimensional Euclidean space, this calculation can be made by approximating it with its average over a hypersphere $\mathcal{S}^{m-n-1}$ of radius $\|w^\perp\|$, where the noise with the given intensity is assumed to be equally present in every direction. This should give

$$\int \frac{2\pi^{\frac{m-n-1}{2}}}{\Gamma(\frac{m-n}{2})} \left\|\bar{w}^\perp\right\|^{m-n-1} p_{W^\perp}(\bar{w}^\perp) d(\|\bar{w}^\perp\|) = 1, \tag{A.8}$$

and also

$$\int p_{\|W^{\perp}\|}(\|\bar{w}^{\perp}\|)d(\|\bar{w}^{\perp}\|) = 1. \tag{A.9}$$

Let

$$p_{\|W^{\perp}\|}(\|\bar{w}^{\perp}\|) = \frac{2\pi^{\frac{m-n-1}{2}}}{\Gamma(\frac{m-n}{2})} \left\|\bar{w}^{\perp}\right\|^{m-n-1} p_{W^{\perp}}(\bar{w}^{\perp}), \tag{A.10}$$

where $\Gamma(\cdot)$ is the gamma function. Thus,

$$p_{W^{\perp}}(\bar{w}^{\perp}) = \frac{\Gamma(\frac{m-n}{2})}{2\pi^{\frac{m-n}{2}} \left\|\bar{w}^{\perp}\right\|^{m-n-1}} p_{\|W^{\perp}\|}(\|\bar{w}^{\perp}\|). \tag{A.11}$$

Therefor Eq.(4.10) is proved.

A.3 Proof of Lemma 3

WPL [11] regularizes learning the current architecture by maximizing $p(\theta_v, \theta_i \mid \mathscr{D})$. However, with E^{2}NAS, two posterior probabilities need to be maximized, $p_1 * p_2 = p(\omega_{i-1}, \omega_i \mid \mathscr{D}) * p(\omega_i^c, \omega_i \mid \mathscr{D})$ in each step of supernet training. Hence, we need to prove that the proposed complementation loss function in Eq.(4.12) is the same as maximizing $p_1 * p_2$.

Similar to WPL [11], the shared weights between ω_i^c and ω_i are depicted as ω_s^c, and the private weights for the two architectures are defined as ω_c^p and ω_i^c. The shared weights between ω_{i-1} and ω_i are also depicted as ω_s^{i-1}, and the private weights for the two architectures are defined as ω_{i-1}^p and ω_i^{i-1}. Further, we have $\omega_i \cap \{\omega_{i-1}, \omega_i^c\} = \omega_i$, and $\omega_{i-1} \cap \omega_i^c = \emptyset$. From

Bayes' theorem, we have:

$$
\begin{aligned}
p_1 * p_2 &= p(\omega_{i-1},\omega_i \mid \mathscr{D}) * p(\omega_i^c,\omega_i \mid \mathscr{D}) = p(\omega_{i-1}^p,\omega_i^{i-1},\omega_s^{i-1} \mid \mathscr{D}) * p(\omega_c^p,\omega_i^c,\omega_s^c \mid \mathscr{D}) \\
&= \frac{p(\omega_{i-1}^p \mid \omega_i^{i-1},\omega_s^{i-1},\mathscr{D}) p(\omega_i^{i-1},\omega_s^{i-1},\mathscr{D})}{p(\mathscr{D})} * \frac{p(\omega_c^p \mid \omega_i^c,\omega_s^c,\mathscr{D}) p(\omega_i^c,\omega_s^c,\mathscr{D})}{p(\mathscr{D})} \\
&= \frac{p(\omega_{i-1}^p \mid \omega_s^{i-1},\mathscr{D}) p(\omega_i^{i-1},\omega_s^{i-1},\mathscr{D})}{p(\mathscr{D})} * \frac{p(\omega_c^p \mid \omega_s^c,\mathscr{D}) p(\omega_i^c,\omega_s^c,\mathscr{D})}{p(\mathscr{D})} \\
&\propto \frac{p(\omega_{i-1}^p,\omega_s^{i-1},\mathscr{D}) p(\omega_i^{i-1},\omega_s^{i-1},\mathscr{D})}{p(\omega_s^{i-1},\mathscr{D})} * \frac{p(\omega_c^p,\omega_s^c,\mathscr{D}) p(\omega_i^c,\omega_s^c,\mathscr{D})}{p(\omega_s^c,\mathscr{D})} \\
&= \frac{p(\omega_{i-1},\mathscr{D}) p(\omega_i,\mathscr{D})}{p(\omega_s^{i-1},\mathscr{D})} * \frac{p(\omega_i^c,\mathscr{D}) p(\omega_i,\mathscr{D}))}{p(\omega_s^c,\mathscr{D})} \\
&= \frac{p(\omega_{i-1},\mathscr{D}) p(\omega_i,\mathscr{D}) p(\omega_i^c,\mathscr{D}) p(\omega_i,\mathscr{D}))}{p(\omega_s^{i-1},\mathscr{D}) p(\omega_s^c,\mathscr{D})} \\
&= \frac{p(\omega_{i-1},\mathscr{D}) p(\omega_i,\mathscr{D}) p(\omega_i^c,\mathscr{D}) p(\omega_i,\mathscr{D}))}{p(\omega_i,\mathscr{D})} \\
&= p(\omega_{i-1}) p(\mathscr{D} \mid \omega_{i-1}) p(\omega_i^c) p(\mathscr{D} \mid \omega_i^c) p(\omega_i) p(\mathscr{D} \mid \omega_i) \\
&= p(\omega_i)^2 p(\mathscr{D} \mid \omega_i^c) p(\mathscr{D} \mid \omega_{i-1}) p(\mathscr{D} \mid \omega_i),
\end{aligned}
\tag{A.12}
$$

where the conditional independence assumption $p(\omega_1 \mid \omega_2,\omega_s,\mathscr{D}) = p(\omega_1 \mid \omega_s,\mathscr{D})$ applies as each discrete architecture is trained independently in Line 4.

As with WPL [11], the parameters $(\omega_s^{i-1},\omega_s^c)$ in Line 4 are presupposed to be independent, and, in Line 5 $\{\omega_s^{i-1},\omega_s^c\} = \omega_i$ as $\omega_i \cap \{\omega_{i-1},\omega_i^c\} = \omega_i$; $p(\omega_{i-1})p(\omega_i^c) = p(\omega_i)$ since only weights ω_i of architecture α^i are trained in step i.

The loss function can be derived directly from Eq.(A.12)), which is the same as Eq.(4.12). Therefore, Lemma 3 is proved.

A.4 Proof of Lemma 5

This subsection describes the proof of Lemma 5. Based on the implicit function theorem [96], or we simply set $\frac{\mathscr{L}_1(w^*,\alpha)}{\partial w} = 0$ since the model weights w achieved the local optimal in the training set with α, we have:

$$
\frac{\partial \mathscr{L}_1(w^*(\alpha),\alpha)}{\partial w} = 0, \tag{A.13}
$$

and we have

$$
\begin{aligned}
\frac{\partial}{\partial \alpha}\left(\frac{\partial \mathscr{L}_1(w^*(\alpha),\alpha)}{\partial w}\right) &= 0, \\
\frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w} + \frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\frac{\partial (w^*(\alpha))}{\partial \alpha} &= 0, \\
\frac{\partial (w^*(\alpha))}{\partial \alpha} &= -\left[\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{-1}\frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w}.
\end{aligned} \tag{A.14}
$$

In this way, the hypergradient could be formulated as

$$
\nabla_\alpha \mathscr{L}_2 = \frac{\partial \mathscr{L}_2}{\partial \alpha} - \frac{\partial \mathscr{L}_2}{\partial w}\left[\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{-1}\frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w}. \tag{A.15}
$$

A.5 Proof of Corollary 1

The key in the Corollary 1 is to use the Neumann series to approximate the $\left[\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{-1}$.

Based on the Neumann series approximation, for $\|I - A\| < 1$, we have:

$$
A^{-1} = \sum_{k=0}^{\infty}(I-A)^k. \tag{A.16}
$$

Based Assumption 4.1, we have $\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w} < L_1^{\nabla_w}$. With $\gamma < \frac{1}{L_1^{\nabla_w}}$, we have $\left\|I - \gamma\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right\| < 1$ [96, 125]. When we conduct the Neumann series approximation for $\left[\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{-1}$ in the optimal point, we have:

$$
\left[\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^{-1} = \gamma(I - I + \gamma\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w})^{-1} = \gamma\sum_{j=0}^{\infty}\left[I - \gamma\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^j. \tag{A.17}
$$

So that:

$$
\nabla_\alpha \mathscr{L}_2 = \frac{\partial \mathscr{L}_2}{\partial \alpha} - \gamma\frac{\partial \mathscr{L}_2}{\partial w}\sum_{j=0}^{\infty}\left[I - \gamma\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^j\frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w}. \tag{A.18}
$$

A.6 Proof of Theorem 1

Based on the Eq. (6.8) and (6.7), we have

$$
\nabla_\alpha \mathscr{L}_2 - \nabla_\alpha \tilde{\mathscr{L}}_2 = \gamma\frac{\partial \mathscr{L}_2}{\partial w}\sum_{j=K+1}^{\infty}\left[I - \gamma\frac{\partial^2 \mathscr{L}_1}{\partial w \partial w}\right]^j\frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w}. \tag{A.19}
$$

Since the $\mathscr{L}_1$ is μ-strongly convex, and $\gamma\mu I \preceq \gamma\frac{\partial^2\mathscr{L}_1}{\partial w\partial w} \preceq I$, we have

$$\sum_{j=K+1}^{\infty}\left[I-\gamma\frac{\partial^2\mathscr{L}_1}{\partial w\partial w}\right]^j \le \sum_{j=K+1}^{\infty}[I-\gamma\mu]^j. \tag{A.20}$$

Based on the sum of geometric sequence, we have

$$\sum_{j=K+1}^{\infty}\left[I-\gamma\frac{\partial^2\mathscr{L}_1}{\partial w\partial w}\right]^j \le \frac{1}{\gamma\mu}(1-\gamma\mu)^{K+1}. \tag{A.21}$$

Since $\frac{\partial\mathscr{L}_2}{\partial w}$ and $\frac{\partial^2\mathscr{L}_1}{\partial\alpha\partial w}$ are bounded, we have

$$\left\|\nabla_\alpha\mathscr{L}_2-\nabla_\alpha\tilde{\mathscr{L}}_2\right\| \leqslant C_{\mathscr{L}_1^{w\alpha}}\, C_{\mathscr{L}_2^{w}}\,\frac{1}{\mu}(1-\gamma\mu)^{K+1}. \tag{A.22}$$

Therefore, Theorem 1 is proved.

A.7 Proof of Corollary 2

Based on the definitions, the hypergradient of truncated back-propagation and the proposed Neumann approximation based hypergradient are defined in Eq.(6.4) and Eq.(6.8). When we assume that w_t has converged to a stationary point w^* in the last K steps, we have

$$\begin{aligned}
w_i(\alpha)=w_j(\alpha)=w^*(\alpha), &\qquad for\ all\ i,j\in[T-K+1,T];\\
\frac{\partial\Phi(w_i,\alpha)}{\partial w_i}=\frac{\partial\Phi(w_j,\alpha)}{\partial w_j}=\frac{\partial\Phi(w^*(\alpha),\alpha)}{\partial w^*(\alpha)}=A_T, &\qquad for\ all\ i,j\in[T-K+1,T];\\
\frac{\partial\Phi(w_i,\alpha)}{\partial\alpha}=\frac{\partial\Phi(w_j,\alpha)}{\partial\alpha}=\frac{\partial\Phi(w^*(\alpha),\alpha)}{\partial\alpha}=B_T, &\qquad for\ all\ i,j\in[T-K+1,T].
\end{aligned} \tag{A.23}$$

Now the truncated back-propagation could be formulated as:

$$\begin{aligned}
h_{T-K}&=\frac{\partial\mathscr{L}_2}{\partial\alpha}+\frac{\partial\mathscr{L}_2}{\partial w_T}\Big(\sum_{t=T-K+1}^{T}B_tA_{t+1}...A_T\Big)\\
&=\frac{\partial\mathscr{L}_2}{\partial\alpha}+\frac{\partial\mathscr{L}_2}{\partial w_T}\Big(\sum_{t=0}^{K}B_TA_T^t\Big).
\end{aligned} \tag{A.24}$$

We have

$$\begin{aligned}
A_T&=\frac{\partial\Phi(w^*(\alpha),\alpha)}{\partial w^*(\alpha)}=\frac{\partial(w^*-\eta\frac{\partial\mathscr{L}_1}{\partial w})}{\partial w^*}=I-\gamma\frac{\partial^2\mathscr{L}_1(w^*)}{\partial w\partial w},\\
B_T&=\frac{\partial\Phi(w^*(\alpha),\alpha)}{\partial\alpha}=\frac{\partial(w^*-\eta\frac{\partial\mathscr{L}_1}{\partial w})}{\partial\alpha}=-\gamma\frac{\partial^2\mathscr{L}_1(w^*)}{\partial\alpha\partial w}.
\end{aligned} \tag{A.25}$$

From the above, we have

$$
\begin{aligned}
h_{T-K} &= \frac{\partial \mathscr{L}_2}{\partial \alpha} + \frac{\partial \mathscr{L}_2}{\partial w_T}(\sum_{t=0}^{K} B_T A_T^t) \\
&= \frac{\partial \mathscr{L}_2}{\partial \alpha} - \gamma \frac{\partial \mathscr{L}_2}{\partial w} \sum_{j=0}^{K} \left[I - \gamma \frac{\partial^2 \mathscr{L}_1}{\partial w \partial w} \right]^j \frac{\partial^2 \mathscr{L}_1}{\partial \alpha \partial w} \\
&= \nabla_\alpha \tilde{\mathscr{L}}_2.
\end{aligned}
\tag{A.26}
$$

Therefore, Corollary 2 is proved.

A.8 Proof of Lemma 7

First, for $\forall(\alpha, \alpha')$, we have

$$
\begin{aligned}
&\left\| \nabla_\alpha \mathscr{L}_2(w,\alpha) - \nabla_\alpha \mathscr{L}_2(w,\alpha') \right\| \\
&= \left\| \nabla_\alpha \mathscr{L}_2(\cdot,\alpha) - \nabla_\alpha \mathscr{L}_2(\cdot,\alpha') + \nabla_\alpha \mathscr{L}_2(w(\alpha),\cdot) - \nabla_\alpha \mathscr{L}_2(w(\alpha'),\cdot) \right\| \\
&= \left\| \nabla_\alpha \mathscr{L}_2(\cdot,\alpha) - \nabla_\alpha \mathscr{L}_2(\cdot,\alpha') + \nabla_w \mathscr{L}_2(w(\alpha),\cdot)\nabla_\alpha w(\alpha) - \nabla_w \mathscr{L}_2(w(\alpha'),\cdot)\nabla_\alpha w(\alpha') \right\| \\
&\le \left\| \nabla_\alpha \mathscr{L}_2(\cdot,\alpha) - \nabla_\alpha \mathscr{L}_2(\cdot,\alpha') \right\| + \left\| \nabla_w \mathscr{L}_2(w(\alpha),\cdot)\nabla_\alpha w(\alpha) - \nabla_w \mathscr{L}_2(w(\alpha'),\cdot)\nabla_\alpha w(\alpha') \right\|.
\end{aligned}
\tag{A.27}
$$

Then we divide Eq.(A.27) to two parts. For the first part, based on the Assumption 3.2, we have:

$$
\left\| \nabla_\alpha \mathscr{L}_2(\cdot,\alpha) - \nabla_\alpha \mathscr{L}_2(\cdot,\alpha') \right\| \le L_2^{\nabla_\alpha}(\alpha - \alpha'). \tag{A.28}
$$

And for the second part of Eq.(A.27), we have

$$
\begin{aligned}
&\left\| \nabla_w \mathscr{L}_2(w(\alpha),\cdot)\nabla_\alpha w(\alpha) - \nabla_w \mathscr{L}_2(w(\alpha'),\cdot)\nabla_\alpha w(\alpha') \right\| \\
&= \left\| \nabla_w \mathscr{L}_2(w(\alpha),\cdot)\nabla_\alpha w(\alpha) - \nabla_w \mathscr{L}_2(w(\alpha'),\cdot)\nabla_\alpha w(\alpha) - \nabla_w \mathscr{L}_2(w(\alpha'),\cdot)\nabla_\alpha w(\alpha') + \nabla_w \mathscr{L}_2(w(\alpha'),\cdot)\nabla_\alpha w(\alpha) \right\| \\
&\le \left\| \nabla_w \mathscr{L}_2(w(\alpha'),\cdot) - \nabla_w \mathscr{L}_2(w(\alpha'),\cdot) \right\| \left\| \nabla_\alpha w(\alpha) \right\| + \left\| \nabla_w \mathscr{L}_2(w(\alpha'),\cdot) \right\| \left\| \nabla_\alpha w(\alpha) - \nabla_\alpha w(\alpha') \right\|.
\end{aligned}
\tag{A.29}
$$

Based Assumption 3.3, we have

$$
\left\| \nabla_w \mathscr{L}_2(w(\alpha'),\cdot) - \nabla_w \mathscr{L}_2(w(\alpha'),\cdot) \right\| \le L_2^{\nabla_w} \left\| w(\alpha) - w(\alpha') \right\|, \tag{A.30}
$$

and based Assumption 4.2 that we have

$$
\left\| w(\alpha) - w(\alpha') \right\| \le L_w \left\| \alpha - \alpha' \right\|, \quad \text{and} \quad \left\| \nabla_\alpha w(\alpha) - \nabla_\alpha w(\alpha') \right\| \le L_{\nabla_\alpha w} \left\| \alpha - \alpha' \right\|. \tag{A.31}
$$

Based on Assumption 3.3, we know $\nabla_w \mathscr{L}_2(w(\alpha'),\cdot)$ is bounded that $\nabla_w \mathscr{L}_2(w(\alpha'),\cdot) \le L_2^w$. $\nabla_\alpha w(\alpha)$ is also bounded by $\|\nabla_\alpha w(\alpha)\| \le L_w$. In this way, Eq.(A.29) could be rephrased as:

$$
\left\| \nabla_w \mathscr{L}_2(w(\alpha),\cdot)\nabla_\alpha w(\alpha) - \nabla_w \mathscr{L}_2(w(\alpha'),\cdot)\nabla_\alpha w(\alpha') \right\| \le L_2^{\nabla_w} L_w^2 \left\| \alpha - \alpha' \right\| + L_2^w L_{\nabla_\alpha w} \left\| \alpha - \alpha' \right\|. \tag{A.32}
$$

Based on Eq. (A.27), Eq. (A.28) and (A.32) we have

$$\left\|\nabla_\alpha \mathscr{L}_2(w,\alpha) - \nabla_\alpha \mathscr{L}_2(w,\alpha')\right\| \le (L_2^{\nabla_\alpha} + L_2^{\nabla_w} L_w^2 + L_2^w L_{\nabla_\alpha w}) \left\|\alpha - \alpha'\right\|. \tag{A.33}$$

Therefore, Lemma 7 is proved.

A.9 Proof of Theorem 2

We first define the noise term between the stochastic estimate $\nabla_\alpha \mathscr{L}_2^i$ and the true gradient $\nabla_\alpha \mathscr{L}_2$ as:

$$\varepsilon_i = \nabla_\alpha \mathscr{L}_2 - \nabla_\alpha \mathscr{L}_2^i, \tag{A.34}$$

and the error between the approximated hypergradient $\nabla_\alpha \tilde{\mathscr{L}}_2$ and the exact hypergradient $\nabla_\alpha \mathscr{L}_2$ as:

$$e_m = \nabla_\alpha \mathscr{L}_2(w^*(\alpha_m),\alpha_m) - \nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m). \tag{A.35}$$

We then prove that $\nabla_\alpha \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m)$ is an unbiased estimate of $\nabla_\alpha \mathscr{L}_2(w^*(\alpha_m),\alpha_m)$ that:

$$E[\nabla_\alpha \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m) \mid \alpha_m] = \nabla_\alpha \mathscr{L}_2(w^*(\alpha_m),\alpha_m). \tag{A.36}$$

Based on IFT in Eq.(6.7), we have

$$\nabla_\alpha \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m) = \frac{\partial \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m)}{\partial \alpha} - \frac{\partial \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m)}{\partial w}\left[\frac{\partial^2 \mathscr{L}_1^j(w^*(\alpha_m),\alpha_m)}{\partial w \partial w}\right]^{-1}\frac{\partial^2 \mathscr{L}_1^j(w^*(\alpha_m),\alpha_m)}{\partial \alpha \partial w}. \tag{A.37}$$

So that

$$\begin{aligned} &E\left[\nabla_\alpha \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m) \mid \alpha_m\right] \\ =&E\left[\frac{\partial \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m)}{\partial \alpha} - \frac{\partial \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m)}{\partial w}\left[\frac{\partial^2 \mathscr{L}_1^j(w^*(\alpha_m),\alpha_m)}{\partial w \partial w}\right]^{-1}\frac{\partial^2 \mathscr{L}_1^j(w^*(\alpha_m),\alpha_m)}{\partial \alpha \partial w} \mid \alpha_m\right]. \end{aligned} \tag{A.38}$$

Based on the linear assumption for $\mathscr{L}_1^j$ in the condition 4 of the Theorem 2, we have $\frac{\partial^2 \mathscr{L}_1^j(w^*(\alpha_m),\alpha_m)}{\partial w \partial w} = \frac{\partial^2 \mathscr{L}_1(w^*(\alpha_m),\alpha_m)}{\partial w \partial w}$, and

$$\begin{aligned} &E\left[\nabla_\alpha \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m) \mid \alpha_m\right] \\ &= \frac{1}{R}\sum_{i=1}^{R}\frac{\partial \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m)}{\partial \alpha} - \frac{1}{R}\sum_{i=1}^{R}\frac{\partial \mathscr{L}_2^i(w^*(\alpha_m),\alpha_m)}{\partial w}\left[\frac{\partial^2 \mathscr{L}_1(w^*(\alpha_m),\alpha_m)}{\partial w \partial w}\right]^{-1}\frac{1}{J}\sum_{j=1}^{J}\frac{\partial^2 \mathscr{L}_1^j(w^*(\alpha_m),\alpha_m)}{\partial \alpha \partial w} \\ &= \frac{\partial \mathscr{L}_2(w^*(\alpha_m),\alpha_m)}{\partial \alpha} - \frac{\partial \mathscr{L}_2(w^*(\alpha_m),\alpha_m)}{\partial w}\left[\frac{\partial^2 \mathscr{L}_1(w^*(\alpha_m),\alpha_m)}{\partial w \partial w}\right]^{-1}\frac{\partial^2 \mathscr{L}_1(w^*(\alpha_m),\alpha_m)}{\partial \alpha \partial w} \\ &= \nabla_\alpha \mathscr{L}_2(w^*(\alpha_m),\alpha_m). \end{aligned} \tag{A.39}$$

Based on the Lemma 7, we know that $\nabla_\alpha \mathscr{L}_2(w^*(\alpha_m),\alpha_m)$ is Lipschitz continuous with $L_{\nabla_\alpha \mathscr{L}_2} = L_2^{\nabla_\alpha} + L_2^{\nabla_w} L_w^2 + L_2^w L_{\nabla_\alpha w}$. Based on Lipschitz condition, we have

$$\begin{aligned}
&E\left[\mathscr{L}_2(w^*(\alpha_{m+1}),\alpha_{m+1}) \mid \alpha_m\right] \le E\left[\mathscr{L}_2(w^*(\alpha_m),\alpha_m) \mid \alpha_m\right] \\
&+ E\left[\left\langle \nabla_\alpha \mathscr{L}_2(w^*(\alpha_m),\alpha_m), \alpha_{m+1}-\alpha_m \right\rangle \mid \alpha_m\right] + \frac{L_{\nabla_\alpha \mathscr{L}_2}}{2} E\left[\|\alpha_{m+1}-\alpha_m\|^2\right] \\
&= \mathscr{L}_2(w^*(\alpha_m),\alpha_m) + \left\langle E\left[\nabla_\alpha \mathscr{L}_2(w^*(\alpha_m),\alpha_m)\right], -\gamma_{\alpha_m} E\left[\nabla_\alpha \mathscr{L}_2^{i'}(w^*(\alpha_m),\alpha_m) \mid \alpha_m\right]\right\rangle \\
&+ \frac{L_{\nabla_\alpha \mathscr{L}_2}}{2} \gamma_{\alpha_m}^2 E\left[\left\|\nabla_\alpha \mathscr{L}_2^{i'}(w^*(\alpha_m),\alpha_m)\right\|^2\right].
\end{aligned} \tag{A.40}$$

From our definitions, we have

$$\begin{aligned}
E\left[\nabla_\alpha \mathscr{L}_2(w^*(\alpha_m),\alpha_m)\right] &= E\left[\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m) + e_m\right] = E\left[\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right] + E\left[e_m\right], \\
E\left[\nabla_\alpha \hat{\mathscr{L}}_2^i(w^*(\alpha_m),\alpha_m) \mid \alpha_m\right] &= E\left[\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m) - \varepsilon_m \mid \alpha_m\right] = E\left[\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right], \\
E\left[\left\|\nabla_\alpha \hat{\mathscr{L}}_2^i(w^j(\alpha_m),\alpha_m)\right\|^2 \mid \alpha_m\right] &= E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m) - \varepsilon_m\right\|^2\right] = E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right\|^2\right] + E\left[\|\varepsilon_m\|^2\right],
\end{aligned} \tag{A.41}$$

since $E(\varepsilon_m) = 0$. In this way, we have

$$\begin{aligned}
&E\left[\mathscr{L}_2(w^*(\alpha_{m+1}),\alpha_{m+1}) \mid \alpha_m\right] \le E\left[\mathscr{L}_2(w^*(\alpha_m),\alpha_m) \mid \alpha_m\right] - \gamma_{\alpha_m} E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right\|^2\right] \\
&- \gamma_{\alpha_m} E\left\langle e_m, \nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right\rangle + \frac{L_{\nabla_\alpha \mathscr{L}_2}}{2} \gamma_{\alpha_m}^2 E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right\|^2\right] \\
&+ \frac{L_{\nabla_\alpha \mathscr{L}_2}}{2} \gamma_{\alpha_m}^2 E\left[\|\varepsilon_m\|^2\right].
\end{aligned} \tag{A.42}$$

Based on Theorem 1, we have $\|e_m\| \leqslant C_{\mathscr{L}_1^{w\alpha}} C_{\mathscr{L}_2^w} \frac{1}{\mu}(1-\gamma\mu)^{K+1}$. In this way, for all $\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)$, we have

$$\begin{aligned}
\left\langle e_m, \nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right\rangle &\ge -C_{\mathscr{L}_1^{w\alpha}} C_{\mathscr{L}_2^w} \frac{1}{\mu}(1-\gamma\mu)^{K+1} \left\|\nabla_\alpha \tilde{\mathscr{L}}_2\right\| \\
&= -\frac{C_{\mathscr{L}_1^{w\alpha}} C_{\mathscr{L}_2^w} (1-\gamma\mu)^{K+1}}{\mu \left\|\nabla_\alpha \tilde{\mathscr{L}}_2\right\|} \left\|\nabla_\alpha \tilde{\mathscr{L}}_2\right\|^2 \\
&= -P \left\|\nabla_\alpha \tilde{\mathscr{L}}_2\right\|^2,
\end{aligned} \tag{A.43}$$

where $P = \frac{C_{\mathscr{L}_1^{w\alpha}} C_{\mathscr{L}_2^w} (1-\gamma\mu)^{K+1}}{\mu \|\nabla_\alpha \tilde{\mathscr{L}}_2\|}$. In this way, we have:

$$\begin{aligned}
E\left[\mathscr{L}_2(w^*(\alpha_{m+1}),\alpha_{m+1})\right] &\le E\left[\mathscr{L}_2(w^*(\alpha_m),\alpha_m)\right] - \gamma_{\alpha_m}(1-P) E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2\right\|^2\right] \\
&+ \frac{L_{\nabla_\alpha \mathscr{L}_2}}{2} \gamma_{\alpha_m}^2 (1+D) E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2\right\|^2\right] \\
&\le E\left[\mathscr{L}_2(w^*(\alpha_m),\alpha_m)\right] - \gamma_{\alpha_m}[(1-P) - \frac{L_{\nabla_\alpha \mathscr{L}_2}}{2} \gamma_{\alpha_m}(1+D)] E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2\right\|^2\right].
\end{aligned} \tag{A.44}$$

If we choose γ_{α_m} to make $(1-P)-\frac{L_{\nabla_\alpha \mathscr{L}_2}}{2}\gamma_{\alpha_m}(1+D)>0$, we have $\gamma_{\alpha_m}<\frac{(1-P)}{\frac{L_{\nabla_\alpha \mathscr{L}_2}}{2}(1+D)}$. In addition, since the learning rate should be positive, we should make that $1-P>0$, which could be reached by choose appropriate γ and K that $\frac{C_{\mathscr{L}_1^{w\alpha}} C_{\mathscr{L}_2^{w}} (1-\gamma\mu)^{K+1}}{\mu\|\nabla_\alpha \tilde{\mathscr{L}}_2\|}<1$, where $0<1-\gamma\mu\leq 1$. In this way, we could find that $\mathscr{L}_2$ is decreasing with α_m, and we know that with sufficiently large m, $\mathscr{L}_2$ will decrease and converge since $\mathscr{L}_2$ is bounded.

Furthermore, we have:

$$\begin{aligned}&E\left[\mathscr{L}_2(w^*(\alpha_m),\alpha_m)\right]-E\left[\mathscr{L}_2(w^*(\alpha_{m+1}),\alpha_{m+1})\right]\\ \geq&\gamma_{\alpha_m}[(1-P)-\frac{L_{\nabla_\alpha \mathscr{L}_2}}{2}\gamma_{\alpha_m}(1+D)]E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right\|^2\right].\end{aligned} \tag{A.45}$$

By telescoping sum, we can show that

$$E\left[\mathscr{L}_2(w^*(\alpha_0),\alpha_0)\right]-E\left[\mathscr{L}_2(w^*(\alpha_m),\alpha_m)\right]\geq\sum_{k=0}^{K} q_t E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right\|^2\right], \tag{A.46}$$

where $q_t=\gamma_{\alpha_m}[(1-P)-\frac{L_{\nabla_\alpha \mathscr{L}_2}}{2}\gamma_{\alpha_m}(1+D)]>0$. Since $\mathscr{L}_2$ is bounded, we have $\sum_{\substack{k=0\\K\to\infty}}^{K} q_t E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right\|^2\right]<\infty$. In addition, based on condition 3, we have $\sum_{\substack{k=0\\K\to\infty}}^{K} q_t=\infty$, which imply that $\lim_{k\to\infty} E\left[\left\|\nabla_\alpha \tilde{\mathscr{L}}_2(w^*(\alpha_m),\alpha_m)\right\|\right]=0$, so as $\lim_{m\to\infty} E\left[\left\|\nabla_\alpha \hat{\mathscr{L}}_2^i(w^j(\alpha_m),\alpha_m)\right\|\right]=0$.

Therefore, Theorem 2 is proved.

BIBLIOGRAPHY

[1] M. S. ABDELFATTAH, A. MEHROTRA, Ł. DUDZIAK, AND N. D. LANE, *Zero-cost proxies for lightweight nas*, in ICLR, 2021.

[2] G. ADAM AND J. LORRAINE, *Understanding neural architecture search techniques*, arXiv preprint arXiv:1904.00438, (2019).

[3] Y. AKIMOTO, S. SHIRAKAWA, N. YOSHINARI, K. UCHIDA, S. SAITO, AND K. NISHIDA, *Adaptive stochastic natural gradient method for one-shot neural architecture search*, in Proceedings of International Conference on Machine Learning (ICML), 2019, pp. 171–180.

[4] ALIBABA, *Shortening machine learning development cycle with automl*, 2018.

[5] R. ALJUNDI, M. LIN, B. GOUJAUD, AND Y. BENGIO, *Gradient based sample selection for online continual learning*, in Advances in Neural Information Processing Systems (NeurIPS), 2019.

[6] BAIDU, *Ezdl*, 2018.

[7] B. BAKER, O. GUPTA, R. RASKAR, AND N. NAIK, *Accelerating neural architecture search using performance prediction*, arXiv preprint arXiv:1705.10823, (2017).

[8] Y. BALAJI, S. SANKARANARAYANAN, AND R. CHELLAPPA, *Metareg: Towards domain generalization using meta-regularization*, in Advances in Neural Information Processing Systems (NeurIPS), 2018, pp. 998–1008.

[9] G. BENDER, P.-J. KINDERMANS, B. ZOPH, V. VASUDEVAN, AND Q. LE, *Understanding and simplifying one-shot architecture search*, in Proceedings of International Conference on Machine Learning (ICML), 2018, pp. 550–559.

[10] Y. BENGIO, *Gradient-based optimization of hyperparameters*, Neural computation, 12 (2000), pp. 1889–1900.

[11] Y. BENYAHIA, K. YU, K. B. SMIRES, M. JAGGI, A. C. DAVISON, M. SALZMANN, AND C. MUSAT, *Overcoming multi-model forgetting*, in Proceedings of International Conference on Machine Learning (ICML), 2019, pp. 594–603.

[12] J. BERGSTRA AND Y. BENGIO, *Random search for hyper-parameter optimization*, Journal of Machine Learning Research, 13 (2012), pp. 281–305.

[13] J. BERGSTRA, D. YAMINS, AND D. D. COX, *Making a science of model search*, (2012).

[14] J. S. BERGSTRA, R. BARDENET, Y. BENGIO, AND B. KÉGL, *Algorithms for hyper-parameter optimization*, in Advances in Neural Information Processing Systems (NeurIPS), 2011, pp. 2546–2554.

[15] K. BI, C. HU, L. XIE, X. CHEN, L. WEI, AND Q. TIAN, *Stabilizing darts with amended gradient estimation on architectural parameters*, arXiv preprint arXiv:1910.11831, (2019).

[16] D. M. BLEI, A. KUCUKELBIR, AND J. D. MCAULIFFE, *Variational inference: A review for statisticians*, Journal of the American statistical Association, 112 (2017), pp. 859–877.

[17] C. BLUNDELL, J. CORNEBISE, K. KAVUKCUOGLU, AND D. WIERSTRA, *Weight uncertainty in neural network*, in Proceedings of International Conference on Machine Learning (ICML), 2015, pp. 1613–1622.

[18] L. BOTTOU, F. E. CURTIS, AND J. NOCEDAL, *Optimization methods for large-scale machine learning*, Siam Review, 60 (2018), pp. 223–311.

[19] A. BROCK, T. LIM, J. M. RITCHIE, AND N. J. WESTON, *Smash: One-shot model architecture search through hypernetworks*, in International Conference on Learning Representations (ICLR), 2018.

[20] H. CAI, L. ZHU, AND S. HAN, *Proxylessnas: Direct neural architecture search on target task and hardware*, in International Conference on Learning Representations (ICLR), 2019.

[21] F. P. CASALE, J. GORDON, AND N. FUSI, *Probabilistic neural architecture search*, arXiv preprint arXiv:1902.05116, (2019).

[22] W. CHEN, X. GONG, AND Z. WANG, *Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective*, in ICLR, 2021.

[23] X. Chen and C.-J. Hsieh, *Stabilizing differentiable architecture search via perturbation-based regularization*, in Proceedings of International Conference on Machine Learning (ICML), 2020.

[24] X. Chen, R. Wang, M. Cheng, X. Tang, and C.-J. Hsieh, *Drnas: Dirichlet neural architecture search*, arXiv preprint arXiv:2006.10355, (2020).

[25] X. Chen, L. Xie, J. Wu, and Q. Tian, *Progressive differentiable architecture search: Bridging the depth gap between search and evaluation*, in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2019, pp. 1294–1303.

[26] Y. Chen and G. Meng, *Renas: Reinforced evolutionary neural architecture search*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 4787–4796.

[27] Y. Chen, G. Meng, Q. Zhang, S. Xiang, C. Huang, L. Mu, and X. Wang, *Renas: Reinforced evolutionary neural architecture search*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 4787–4796.

[28] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun, *Detnas: Backbone search for object detection*, in Advances in Neural Information Processing Systems (NeurIPS), 2019, pp. 6642–6652.

[29] Z. Chen and B. Liu, *Lifelong machine learning*, Synthesis Lectures on Artificial Intelligence and Machine Learning, 10 (2016), pp. 1–145.

[30] X. Cheng, Y. Zhong, M. Harandi, Y. Dai, X. Chang, T. Drummond, H. Li, and Z. Ge, *Hierarchical Neural Architecture Search for Deep Stereo Matching*, in Advances in Neural Information Processing Systems (NeurIPS), 2020.

[31] X. Cheng, Y. Zhong, M. Harandi, Y. Dai, X. Chang, T. Drummond, H. Li, and Z. Ge, *Hierarchical neural architecture search for deep stereo matching*, in NeurIPS, 2020.

[32] P. Chrabaszcz, I. Loshchilov, and F. Hutter, *A downsampled variant of imagenet as an alternative to the cifar datasets*, arXiv preprint arXiv:1707.08819, (2017).

[33] X. Chu, B. Zhang, R. Xu, and J. Li, *Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search*, arXiv preprint arXiv:1907.01845, (2019).

[34] B. Colson, P. Marcotte, and G. Savard, *An overview of bilevel optimization*, Annals of operations research, 153 (2007), pp. 235–256.

[35] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. Stanley, and J. Clune, *Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents*, in Advances in Neural Information Processing Systems (NeurIPS), 2018, pp. 5027–5038.

[36] N. Couellan and W. Wang, *On the convergence of stochastic bi-level gradient methods*, Optimization, (2016).

[37] O. Dekel, P. M. Long, and Y. Singer, *Online learning of multiple tasks with a shared loss*, Journal of Machine Learning Research, 8 (2007), pp. 2233–2264.

[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *Imagenet: A large-scale hierarchical image database*, in 2009 IEEE conference on computer vision and pattern recognition, IEEE, 2009, pp. 248–255.

[39] T. Domhan, J. T. Springenberg, and F. Hutter, *Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves*, in Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2015.

[40] X. Dong, *Searching for a robust neural architecture in four gpu hours*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 1761–1770.

[41] X. Dong, *Nas-bench-201: Extending the scope of reproducible neural architecture search*, in International Conference on Learning Representations (ICLR), 2020.

[42] X. Dong and Y. Yang, *One-shot neural architecture search via self-evaluated template network*, in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2019, pp. 3681–3690.

[43] T. Elsken, J. H. Metzen, and F. Hutter, *Neural architecture search: A survey*, arXiv preprint arXiv:1808.05377, (2018).

[44] C. Finn, P. Abbeel, and S. Levine, *Model-agnostic meta-learning for fast adaptation of deep networks*, in Proceedings of International Conference on Machine Learning, 2017, pp. 1126–1135.

[45] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, *Forward and reverse gradient-based hyperparameter optimization*, in Proceedings of International Conference on Machine Learning, 2017, pp. 1165–1173.

[46] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, *Bilevel programming for hyperparameter optimization and meta-learning*, in Proceedings of International Conference on Machine Learning (ICML), PMLR, 2018, pp. 1568–1577.

[47] S. Ghadimi and M. Wang, *Approximation methods for bilevel programming*, arXiv preprint arXiv:1802.02246, (2018).

[48] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, *Google vizier: A service for black-box optimization*, in Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, 2017, pp. 1487–1495.

[49] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT press, 2016.

[50] A. Graves, *Practical variational inference for neural networks*, in Advances in Neural Information Processing Systems (NeurIPS), 2011, pp. 2348–2356.

[51] R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo, *On the iteration complexity of hypergradient computation*, in Proceedings of International Conference on Machine Learning (ICML), PMLR, 2020, pp. 3748–3758.

[52] R. Grazzi, M. Pontil, and S. Salzo, *Convergence properties of stochastic hypergradients*, arXiv preprint arXiv:2011.07122, (2020).

[53] M. Guo, Z. Zhong, W. Wu, D. Lin, and J. Yan, *Irlas: Inverse reinforcement learning for architecture search*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 9021–9029.

[54] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, *Single path one-shot neural architecture search with uniform sampling*, 2019.

[55] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, *Single path one-shot neural architecture search with uniform sampling*, in Proceedings of the European Conference on Computer Vision (ECCV), Springer, 2020, pp. 544–560.

[56] D. Ha, A. Dai, and Q. V. Le, *Hypernetworks*, arXiv preprint arXiv:1609.09106, (2016).

[57] S. Haykin and N. Network, *A comprehensive foundation*, Neural networks, 2 (2004), p. 41.

[58] C. He, H. Ye, L. Shen, and T. Zhang, *Milenas: Efficient neural architecture search via mixed-level reformulation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 11993–12002.

[59] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[60] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, in NIPS Deep Learning and Representation Learning Workshop, 2015.

[61] M. W. Hoffman and Z. Ghahramani, *Predictive entropy search for efficient global optimization of black-box functions*, in International Conference on Neural Information Processing Systems, 2014, pp. 918–926.

[62] A. Hundt, V. Jain, and G. D. Hager, *sharpdarts: Faster and more accurate differentiable architecture search*, arXiv preprint arXiv:1903.09900, (2019).

[63] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning*, Springer, 2019.

[64] D. Isele and A. Cosgun, *Selective experience replay for lifelong learning*, in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[65] E. Jang, S. Gu, and B. Poole, *Categorical reparameterization with gumbel-softmax*, in International Conference on Learning Representations (ICLR), 2017.

[66] M. Jankowiak and F. Obermeyer, *Pathwise derivatives beyond the reparameterization trick*, in Proceedings of International Conference on Machine Learning (ICML), 2018, pp. 2235–2244.

[67] Y. Jiang, C. Hu, T. Xiao, C. Zhang, and J. Zhu, *Improved differentiable architecture search for language modeling and named entity recognition*, in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and

the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019, pp. 3576–3581.

[68] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, *How to escape saddle points efficiently*, in Proceedings of International Conference on Machine Learning (ICML), PMLR, 2017, pp. 1724–1732.

[69] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, *Neural architecture search with bayesian optimisation and optimal transport*, in Advances in Neural Information Processing Systems (NeurIPS), 2018, pp. 2020–2029.

[70] M. G. Kendall, *The treatment of ties in ranking problems*, Biometrika, 33 (1945), pp. 239–251.

[71] M. Khan and W. Lin, *Conjugate-computation variational inference: Converting variational inference in non-conjugate models to inferences in conjugate models*, in Artificial Intelligence and Statistics, PMLR, 2017, pp. 878–887.

[72] M. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava, *Fast and scalable bayesian deep learning by weight-perturbation in adam*, in Proceedings of International Conference on Machine Learning (ICML), 2018, pp. 2611–2620.

[73] M. E. Khan and H. Rue, *Learning-algorithms from bayesian principles*, arXiv preprint arXiv:2002.10778, (2020).

[74] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al., *Overcoming catastrophic forgetting in neural networks*, Proceedings of the national academy of sciences, 114 (2017), pp. 3521–3526.

[75] A. Klein and F. Hutter, *Tabular benchmarks for joint architecture and hyperparameter optimization*, arXiv preprint arXiv:1905.04970, (2019).

[76] A. Krizhevsky et al., *Learning multiple layers of features from tiny images*, (2009).

[77] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in International Conference on Neural Information Processing Systems, 2012, pp. 1097–1105.

[78] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[79] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, *Efficient backprop*, Neural Networks Tricks of the Trade, 1524 (1998), pp. 9–50.

[80] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, *Overcoming catastrophic forgetting by incremental moment matching*, in Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 4652–4662.

[81] J. Lehman and K. O. Stanley, *Abandoning objectives: Evolution through the search for novelty alone*, Evolutionary computation, 19 (2011), pp. 189–223.

[82] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, *Blockwisely supervised neural architecture search with knowledge distillation*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020, 2020, pp. 1986–1995.

[83] C. Li, T. Tang, G. Wang, J. Peng, B. Wang, X. Liang, and X. Chang, *Bossnas: Exploring hybrid cnn-transformers with block-wisely self-supervised neural architecture search*, arXiv preprint arXiv:2103.12424, (2021).

[84] C. Li, G. Wang, B. Wang, X. Liang, Z. Li, and X. Chang, *Dynamic slimmable network*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[85] G. Li, M. Xu, S. Giancola, A. Thabet, and B. Ghanem, *Lc-nas: Latency constrained neural architecture search for point cloud networks*, arXiv preprint arXiv:2008.10309, (2020).

[86] L. Li and A. Talwalkar, *Random search and reproducibility for neural architecture search*, arXiv preprint arXiv:1902.07638, (2019).

[87] X. Li, C. Lin, C. Li, M. Sun, W. Wu, J. Yan, and W. Ouyang, *Improving one-shot nas by suppressing the posterior fading*, arXiv preprint arXiv:1910.02543, (2019).

[88] X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong, *Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting*, in Proceedings of International Conference on Machine Learning (ICML), 2019, pp. 3925–3934.

[89] Z. Li and D. Hoiem, *Learning without forgetting*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 40 (2017), pp. 2935–2947.

[90] H. Liang and S. Zhang, *Darts+: Improved differentiable architecture search with early stopping*, arXiv preprint arXiv:1909.06035, (2019).

[91] H. Liang, S. Zhang, J. Sun, X. He, W. Huang, K. Zhuang, and Z. Li, *Darts+: Improved differentiable architecture search with early stopping*, arXiv preprint arXiv:1909.06035, (2019).

[92] C. Liu, P. Dollár, K. He, R. Girshick, A. Yuille, and S. Xie, *Are labels necessary for neural architecture search?*, in European Conference on Computer Vision, Springer, 2020, pp. 798–813.

[93] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, *Progressive neural architecture search*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 19–34.

[94] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, *Hierarchical representations for efficient architecture search*, in International Conference on Learning Representations (ICLR), 2018.

[95] H. Liu, K. Simonyan, and Y. Yang, *Darts: Differentiable architecture search*, in International Conference on Learning Representations (ICLR), 2019.

[96] J. Lorraine, P. Vicol, and D. Duvenaud, *Optimizing millions of hyperparameters by implicit differentiation*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 1540–1552.

[97] C. Louizos, M. Welling, and D. P. Kingma, *Learning sparse neural networks through l_0 regularization*, in International Conference on Learning Representations (ICLR), 2018.

[98] J. Luketina, M. Berglund, K. Greff, and T. Raiko, *Scalable gradient-based tuning of continuous regularization hyperparameters*, in Proceedings of International Conference on Machine Learning (ICML), 2016, pp. 2952–2960.

[99] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, *Neural architecture optimization*, in Advances in Neural Information Processing Systems (NeurIPS), 2018, pp. 7816–7827.

[100] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, *Shufflenet v2: Practical guidelines for efficient cnn architecture design*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 116–131.

[101] M. Mackay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse, *Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions*, in International Conference on Learning Representations (ICLR), 2018.

[102] D. Maclaurin, D. Duvenaud, and R. Adams, *Gradient-based hyperparameter optimization through reversible learning*, in Proceedings of International Conference on Machine Learning (ICML), PMLR, 2015, pp. 2113–2122.

[103] C. J. Maddison, A. Mnih, and Y. W. Teh, *The concrete distribution: A continuous relaxation of discrete random variables*, in International Conference on Learning Representations (ICLR), 2017.

[104] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, *The penn treebank: Annotating predicate argument structure*, in Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994, 1994.

[105] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, *Neural architecture search without training*, arXiv preprint arXiv:2006.04647, (2020).

[106] X. Meng, R. Bachmann, and M. E. Khan, *Training binary neural networks using the bayesian learning rule*, arXiv preprint arXiv:2002.10778, (2020).

[107] S. Merity, C. Xiong, J. Bradbury, and R. Socher, *Pointer sentinel mixture models*, in ICLR, 2017.

[108] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, *Unrolled generative adversarial networks*, in International Conference on Learning Representations (ICLR), 2017.

[109] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik-Manor, *Xnas: Neural architecture search with expert advice*, in Advances in Neural Information Processing Systems (NeurIPS), 2019.

[110] A. Nichol, J. Achiam, and J. Schulman, *On first-order meta-learning algorithms*, arXiv preprint arXiv:1803.02999, (2018).

[111] K. Osawa, S. Swaroop, M. E. E. Khan, A. Jain, R. Eschenhagen, R. E. Turner, and R. Yokota, *Practical deep learning with bayesian principles*, in Advances in Neural Information Processing Systems (NeurIPS), 2019, pp. 4287–4299.

[112] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, *Continual lifelong learning with neural networks: A review*, Neural Networks, (2019).

[113] F. Pedregosa, *Hyperparameter optimization with approximate gradient*, in Proceedings of International Conference on International Conference on Machine Learning, 2016, pp. 737–746.

[114] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, *Efficient neural architecture search via parameters sharing*, in Proceedings of International Conference on Machine Learning (ICML), 2018, pp. 4095–4104.

[115] S. Pidhorskyi, R. Almohsen, and G. Doretto, *Generative probabilistic novelty detection with adversarial autoencoders*, in Advances in Neural Information Processing Systems (NeurIPS), 2018, pp. 6822–6833.

[116] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, *Meta-learning with implicit gradients*, in Advances in Neural Information Processing Systems (NeurIPS), 2019, pp. 113–124.

[117] N. Ratliff, *Multivariate calculus ii: The geometry of smooth maps*, lecture notes: Mathematics for Intelligent Systems series, (2014).

[118] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, *Regularized evolution for image classifier architecture search*, in Proceedings of the AAAI conference on Artificial Intelligence, vol. 33, 2019, pp. 4780–4789.

[119] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, *A comprehensive survey of neural architecture search: Challenges and solutions*, arXiv preprint arXiv:2006.02903, (2020).

[120] M. Rostami, S. Kolouri, and P. K. Pilly, *Complementary learning for overcoming catastrophic forgetting using experience replay*, in Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-19), 2019.

[121] M. SANDLER, A. HOWARD, M. ZHU, A. ZHMOGINOV, AND L.-C. CHEN, *Mobilenetv2: Inverted residuals and linear bottlenecks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[122] K. A. SANKARARAMAN, S. DE, Z. XU, W. R. HUANG, AND T. GOLDSTEIN, *The impact of neural network overparameterization on gradient confusion and stochastic gradient descent*, arXiv preprint arXiv:1904.06963, (2019).

[123] K. A. SANKARARAMAN, S. DE, Z. XU, W. R. HUANG, AND T. GOLDSTEIN, *Analyzing the effect of neural network architecture on training performance*, in Proceedings of International Conference on Machine Learning (ICML), 2020.

[124] C. SCIUTO, K. YU, M. JAGGI, C. MUSAT, AND M. SALZMANN, *Evaluating the search phase of neural architecture search*, arXiv preprint arXiv:1902.08142, (2019).

[125] A. SHABAN, C.-A. CHENG, N. HATCH, AND B. BOOTS, *Truncated back-propagation for bilevel optimization*, in The 22nd International Conference on Artificial Intelligence and Statistics, PMLR, 2019, pp. 1723–1732.

[126] B. SHAHRIARI, K. SWERSKY, Z. WANG, R. P. ADAMS, AND N. DE FREITAS, *Taking the human out of the loop: A review of bayesian optimization*, Proceedings of the IEEE, 104 (2015), pp. 148–175.

[127] Y. SHU, W. WANG, AND S. CAI, *Understanding architectures learnt by cell-based neural architecture search*, in International Conference on Learning Representations (ICLR), 2020.

[128] K. SIMONYAN AND A. ZISSERMAN, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, (2014).

[129] P. SINGH, T. JACOBS, S. NICOLAS, AND M. SCHMIDT, *A study of the learning progress in neural architecture search techniques*, arXiv preprint arXiv:1906.07590, (2019).

[130] J. SNOEK, H. LAROCHELLE, AND R. P. ADAMS, *Practical bayesian optimization of machine learning algorithms*, in Advances in Neural Information Processing Systems (NeurIPS), 2012, pp. 2951–2959.

[131] K. SWERSKY, J. SNOEK, AND R. P. ADAMS, *Freeze-thaw bayesian optimization*, arXiv preprint arXiv:1406.3896, (2014).

[132] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[133] M. Tan and Q. Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, in Proceedings of International Conference on Machine Learning (ICML), 2019, pp. 6105–6114.

[134] G. M. van de Ven and A. S. Tolias, *Three scenarios for continual learning*, arXiv preprint arXiv:1904.07734, (2019).

[135] R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh, *Rethinking architecture selection in differentiable nas*, in International Conference on Learning Representations (ICLR), 2021.

[136] ——, *Rethinking architecture selection in differentiable nas*, in International Conference on Learning Representations (ICLR), 2021.

[137] W. Wang, D. Tran, and M. Feiszli, *What makes training multi-modal networks hard?*, arXiv preprint arXiv:1905.12681, (2019).

[138] C. White, W. Neiswanger, and Y. Savani, *Bananas: Bayesian optimization with neural architectures for neural architecture search*, arXiv preprint arXiv:1910.11858, (2019).

[139] M. Wistuba, A. Rawat, and T. Pedapati, *A survey on neural architecture search*, arXiv preprint arXiv:1905.01392, (2019).

[140] S. Xie, H. Zheng, C. Liu, and L. Lin, *Snas: stochastic neural architecture search*, in International Conference on Learning Representations (ICLR), 2019.

[141] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, *How powerful are graph neural networks?*, arXiv preprint arXiv:1810.00826, (2018).

[142] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, *Pc-darts: Partial channel connections for memory-efficient architecture search*, in International Conference on Learning Representations (ICLR), 2020.

[143] Y. Xu, L. Xie, X. Zhang, X. Chen, B. Shi, Q. Tian, and H. Xiong, *Latency-aware differentiable neural architecture search*, arXiv preprint arXiv:2001.06392, (2020).

[144] S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang, *Does unsupervised architecture representation learning help neural architecture search?*, Advances in Neural Information Processing Systems, 33 (2020).

[145] S. Yan, Y. Zheng, W. Ao, and M. Zhang, *Does unsupervised architecture representation learning help neural architecture search*, in Advances in Neural Information Processing Systems (NeurIPS), 2020.

[146] A. Yang, P. M. Esperança, and F. M. Carlucci, *Nas evaluation is frustratingly hard*, in International Conference on Learning Representations (ICLR), 2020.

[147] Z. Yang, Y. Wang, X. Chen, B. Shi, C. Xu, C. Xu, Q. Tian, and C. Xu, *Cars: Continuous evolution for efficient neural architecture search*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[148] Q. Yao, M. Wang, Y. Chen, W. Dai, H. Yi-Qi, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Y. Yang, *Taking human out of learning applications: A survey on automated machine learning*, arXiv preprint arXiv:1810.13306, (2018).

[149] Q. Yao, J. Xu, W.-W. Tu, and Z. Zhu, *Efficient neural architecture search via proximal iterations*, in AAAI Conference on Artificial Intelligence (AAAI), 2020.

[150] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, *Nas-bench-101: Towards reproducible neural architecture search*, in Proceedings of International Conference on Machine Learning (ICML), 2019, pp. 7105–7114.

[151] K. Yu, R. Ranftl, and M. Salzmann, *How to train your super-net: An analysis of training heuristics in weight-sharing nas*, arXiv preprint arXiv:2003.04276, (2020).

[152] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, *Understanding and robustifying differentiable architecture search*, in International Conference on Learning Representations (ICLR), 2020.

[153] A. Zela, A. Klein, S. Falkner, and F. Hutter, *Towards automated deep learning: Efficient joint neural architecture and hyperparameter search*, arXiv preprint arXiv:1807.06906, (2018).

[154] A. ZELA, J. SIEMS, AND F. HUTTER, *Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search*, in International Conference on Learning Representations (ICLR), 2020.

[155] A. ZELA, J. SIEMS, AND F. HUTTER, *Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search*, in arXiv preprint arXiv:2001.10422, 2020.

[156] C. ZHANG, M. REN, AND R. URTASUN, *Graph hypernetworks for neural architecture search*, arXiv preprint arXiv:1810.05749, (2018).

[157] M. ZHANG, S. JIANG, Z. CUI, R. GARNETT, AND Y. CHEN, *D-vae: A variational autoencoder for directed acyclic graphs*, arXiv preprint arXiv:1904.11088, (2019).

[158] M. ZHANG, H. LI, S. PAN, X. CHANG, Z. GE, AND S. SU, *Differentiable neural architecture search in equivalent space with exploration enhancement*, in Advances in Neural Information Processing Systems (NeurIPS), 2020.

[159] M. ZHANG, H. LI, S. PAN, X. CHANG, AND S. SU, *Overcoming multi-model forgetting in one-shot nas with diversity maximization*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 7809–7818.

[160] M. ZHANG, H. LI, S. PAN, X. CHANG, AND S. S. SU, *Overcoming multi-model forgetting in one-shot nas with diversity maximization*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, 2020.

[161] M. ZHANG, H. LI, S. PAN, X. CHANG, C. ZHOU, Z. GE, AND S. W. SU, *One-shot neural architecture search: Maximising diversity to overcome catastrophic forgetting*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2020).

[162] M. ZHANG, H. LI, S. PAN, T. LIU, AND S. SU, *One-shot neural architecture search via novelty driven sampling*, in Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, International Joint Conferences on Artificial Intelligence Organization, 2020.

[163] M. ZHANG, H. LI, S. PAN, J. LYU, S. LING, AND S. SU, *Convolutional neural networks based lung nodule classification: A surrogate-assisted evolutionary algorithm for hyperparameter optimization*, IEEE Transactions on Evolutionary Computation, (2021).

[164] M. Zhang, H. Li, and S. Su, *High dimensional bayesian optimization via supervised dimension reduction*, in Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2019, pp. 4292–4298.

[165] M. Zhang, S. Su, S. Pan, X. Chang, E. Abbasnejad, and R. Haffari, *idarts: Differentiable architecture search with stochastic implicit gradients*, in Proceedings of International Conference on Machine Learning (ICML), 2021.

[166] X. Zhang, P. Hou, X. Zhang, and J. Sun, *Neural architecture search with random labels*, arXiv preprint arXiv:2101.11834, (2021).

[167] X. Zhang, Z. Huang, and N. Wang, *You only search once: Single shot neural architecture search via direct sparse optimization*, arXiv preprint arXiv:1811.01567, (2019).

[168] Z. Zhang and H. Zha, *Principal manifolds and nonlinear dimensionality reduction via tangent space alignment*, SIAM journal on scientific computing, 26 (2004), pp. 313–338.

[169] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, and Q. Tian, *Multinomial distribution learning for effective neural architecture search*, in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2019, pp. 1304–1313.

[170] H. Zhou, M. Yang, J. Wang, and W. Pan, *Bayesnas: A bayesian approach for neural architecture search*, in Proceedings of International Conference on Machine Learning (ICML), 2019, pp. 7603–7613.

[171] P. Zhou, C. Xiong, R. Socher, and S. C. Hoi, *Theory-inspired path-regularized differential network architecture search*, in Advances in Neural Information Processing Systems (NeurIPS), 2020.

[172] B. Zoph and Q. V. Le, *Neural architecture search with reinforcement learning*, (2017).

[173] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, *Learning transferable architectures for scalable image recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 8697–8710.